Seminar Schema Evolution

Schema Evolution für XML und Web Services

Bearbeiter: Enrico Hörnig
Betreuerin: Sabine Maßmann
Dozent: Prof. Dr. Erhard Rahm

Verschiedene Ansätze für die Evolution von DTD- oder XML-Schema-basierten XML-Dokumenten und von Web Services.

Version: 09.01.2006, 23:00:55

0.1 Statistik

Seiten:	38	Wörter:	5398
Tabellen:	6	Zeichen:	39147
Grafiken:	16		
Erstellt am:	25.11.2005	Autor:	Enrico Hörnig (Hör)
Geändert am:	09.01.2006		
Dateiname:	DBSemi XWSE.odt	Erstellt mit OpenOffice 2.0	
Titel:	Schema Evolution für XML und Web Services		
Thema:	Seminar Schema Evolution		

Dieses Dokument ist verfügbar unter

http://dbs.uni-leipzig.de

0.2 Vorwort

Diese Ausarbeitung ist im Rahmen des Seminars „Schema Evolution" im Wintersemester 2005/06 am Lehrstuhl von Prof. Rahm, der Abteilung Datenbanksysteme entstanden. Dieser Lehrstuhl ist Teil des Institutes für Informatik der Universität Leipzig.

Besonders möchte ich mich bei Frau Sabine Maßmann bedanken, die mir während der Anfertigung dieses Beitrags mit Rat und Tat zur Seite gestanden hat.

0.3 Zusammenfassung

Diese Ausarbeitung versucht, grundlegende Eigenschaften der Evolution von XML-Schemata vorzustellen. Dabei wird auf DTD und XML-S eingegangen. Es werden einige konkrete Ansätze behandelt.

0.4 Inhaltsverzeichnis

0.5 Tabellenverzeichnis

0.6 Abbildungsverzeichnis

1. Einführung

Dieses Kapitel bietet einen kurzen Überblick über XML und Web-Services, und setzt beides in Bezug zu relationalen Datenbanken. XML und seine Applikationen sind ein sehr aktuelles Feld. Es bietet die Möglichkeit, Daten Struktur zu geben, die normalerweise nicht in Datenbanken gespeichert werden. Web-Services hingegen bieten maschinellen, betriebssystemunabhängigen und wohlstandardisierten Zugriff auf Informationen im Internet.

1.1 XML und Schemata

XML ist eine echte Teilmenge von SGML (Standard Generalized Markup Language) und dazu gedacht, maschinenlesbar, strukturiert, präzise, knapp und formal korrekt (im Gegensatz zu HTML) zu sein [W3C04]. Ein XML-Dokument beschreibt den Aufbau eines Baumes aus einzelnen XML-Elementen, dessen Signatur spezifiziert werden kann: Durch eine DTD (siehe [W3C04]) mit einer eigenen, BNF-ähnlichen Syntax, oder durch ein XML-Schema (siehe [W3C04a]), mit XML-Syntax und mehr Datentypen. Die folgende Abbildung zeigt das XML Modell, auf das in ähnlicher Weise DOM (Document Object Model, siehe [W3C98]) auf XML-Daten zugreift.

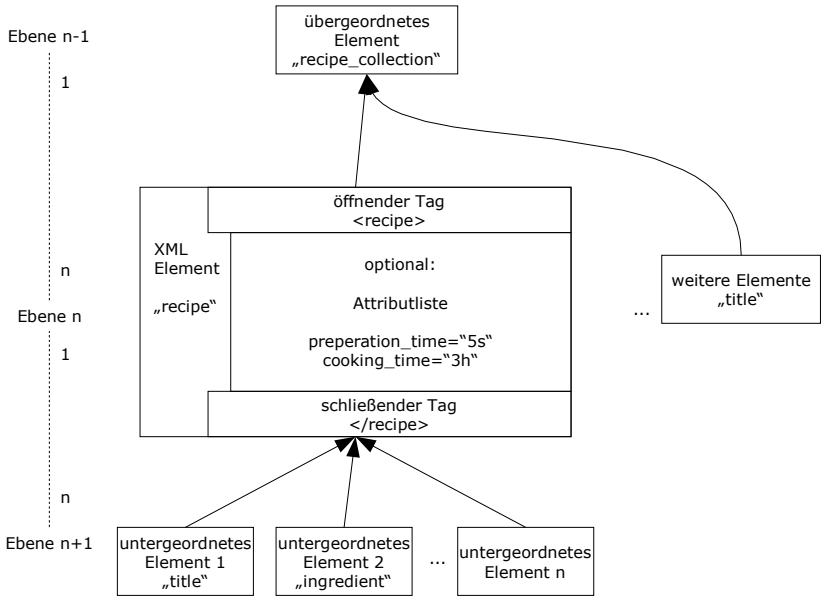


Abbildung 1: XML Modell (generalisiert aus [MS04], Seite 1025)

Jedes Element besteht aus genau drei Einheiten und ist rekursiv:

- Bezeichner: Bezeichnet ein XML-Element eindeutig (z.B. „recipe“).
- Attributmenge: Eine leere oder nichtleere ungeordnete Menge von Tupeln der Art `Bezeichner="Wert"`. Die Bezeichner sind disjunkt.
- Elementmenge: Leere oder nichtleere geordnete Menge von Elementen.

Das Beispiel aus Abbildung 1 kann mit der folgenden XML-Syntax geschrieben werden:

```
<recipe_collection title="Omas Geheimrezepte">
	<recipe name="kaesekuchen" preperation_time="5s" cooking_time="3h">
		<title>Käsekuchen</title>
		<ingredient amount="1000" unit="g">Quark</ingredient>
		<ingredient amount="125" unit="g">Margarine</ingredient>
		<ingredient amount="400" unit="g">Zucker</ingredient>
		<ingredient amount="10" unit="Stück">Eier</ingredient>
		<ingredient amount="1" unit="Paket">Backpulver</ingredient>
	</recipe>
</recipe_collection>
```

Die zugehörige DTD:

```
<!ELEMENT recipe_collection (recipe*)>
<!ELEMENT recipe (title,ingredient*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT ingredient (#PCDATA)>

<!ATTLIST recipe_collection
	title                 ID      #REQUIRED>
<!ATTLIST recipe
	name                  ID      #REQUIRED
	preperation_time      CDATA   #IMPLIED
	cooking_time          CDATA   #IMPLIED>
<!ATTLIST ingredient
	amount                CDATA   #REQUIRED
	unit                  IDREF   #REQUIRED>
```

Und als XML-S:

```
<element name="recipe_collection" type="Type_recipe_collection" />

<complexType name="Type_recipe_collection">
	<sequence>
		<element name="recipe" type="Type_recipe"
			minOccurs="0" maxOccurs="unbounded" />
	</sequence>
	<attribute name="title" type="ID" use="required" />
</complexType>

<complexType name="Type_recipe">
	<sequence>
		<element name="title" type="string"
			minOccurs="1" maxOccurs="1" />
		<element name="ingredient" type="Type_ingredient"
			minOccurs="0" maxOccurs="unbounded" />
	</sequence>
	<attribute name="name" type="ID" use="required" />
	<attribute name="preperation_time" type="time" use="optional" />
	<attribute name="cooking_time" type="time" use="optional" />
</complexType>

<complexType name="Type_ingredient">
	<attribute name="amount" type="Float" use="required" />
	<attribute name="unit" type="IDREF" use="required" />
</complexType>
```

Sind XML-Dokumente syntaktisch korrekt, heißen sie wohlgeformt („well-formed"). Erfüllen sie ein XML-Schema, sind sie zusätzlich valide („valid").

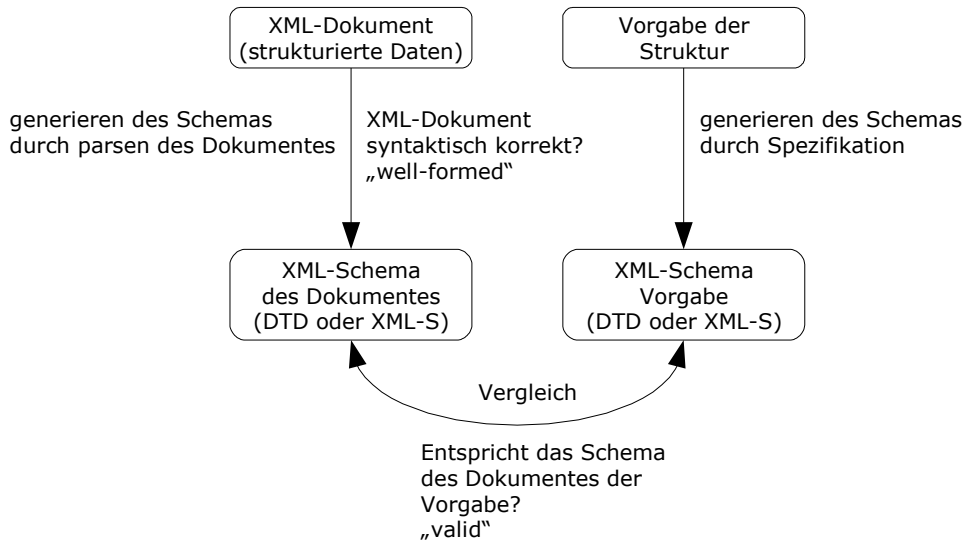


Abbildung 2: Parsen von XML-Dokumenten und Schema-Vergleich

Im XML-Schema wird auch festgelegt, wie oft ein Element/Attribut vorkommen darf. Die folgende Tabelle gibt dazu einen Überblick:

	Elemente		***Attribute***	
Kard.	***DTD***	***XML-S***	***DTD***	***XML-S***
1	(implizit)	minOccurs="1", maxOccurs="1"	#IMPLIED	implied
0, 1	?	minOccurs="0", maxOccurs="1"	#REQUIRED	required
0...n	*	minOccurs="0", maxOccurs="unbounded"		
1...n	+	monOccurs="1", maxOccurs="unbounded"		

Tabelle 1: Kardinalitäten von Elementen und Attributen

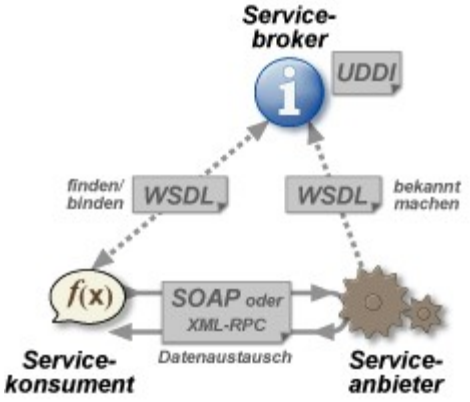
Die Entscheidung über die Validität fällt der Parser Anhang der Kardinalitäten der erlaubten und verwendenten Elemente. Dabei exisitieren die folgenden Teilmengen:

$$Kard(1) \subset Kard(1..n) \subset Kard(0..n),\ Kard(0,1) \subset Kard(0..n)$$

1.2 Web Services

Ein Web-Service ist ein Software-System, welches zur Unterstützung der Kommunikation von Rechnern über ein Netzwerk von Computern dient.

Heutzutage wird eine World-Wide-Web-basierte Service-Architektur favourisiert: Ein Web-Service besitzt eine maschinenlesbare Schnittstelle (WSDL, spezifiziert in [W3C01]) und ist an einer eindeutigen Position (URL, siehe [RFC 1738]) auffindbar (UDDI, siehe [OA04]). Andere Systeme interagieren mit dem Web-Service auf von der Schnittstelle vorgegebene Weise durch SOAP-Nachrichten (siehe [W3C03]). Dies geschieht typischerweise durch Serialisierung mittels HTML und XML. Auch eine Kommunikation über XML-RPC ist möglich.



1.3 XML in Datenbanken

Moderne objektorientierte und relationale Datenbanksysteme unterstützen das Speichern von XML-Daten. Beispiele hierfür wären DB2, Oracle, PostgreSQL, MS SQL als relationale Vertreter. Eine ausführliche Auflistung findet sich in [Bou04].

Im Folgenden wird nur auf relationale Datenbanken Bezug genommen. /BEGRÜNDUNG/

In relationalen DBS werden die XML-Daten innerhalb von Tabellenspalten gespeichert, entweder als Text („textual fidelity" nach [Bey05]) mittels CLOB oder speziellen XML-Datentyp („relational fidelity"). In DB2 wäre dies zum Beispiel „xml" (laut [Bey05]).

XML-Schemas können in der Datenbank registriert werden. Bei Einfüge- und Änderungsoperationen können die Daten dann gegen das Schemata geprüft werden, etwa mittels spezieller SQL-Erweiterungen ([Bey05]).

Der Zugriff auf XML-Daten kann mittels SQL/XML geschehen (spezifiziert in [SX04]). Allerdings wurde die Spezifikation seit 2004 nicht mehr gepflegt. Aktueller ist XQuery [W3C05], welches auf XPath [W3C99] aufbaut und von einer Reihe von Datenbanken, wie DB2 und Oracle, unterstützt wird.

Mit Hilfe von Views kann eine zusätzliche Abstraktionsebene zwischen Anwendungen und Daten geschaffen werden. So unterstützt DB2 nach [Bey05] Views aus XML-Anfragen. Diese können

Oracle [Se05] wie auch DB2 [Bey05] unterstützen Indizes auf XML-Elemente. Dies erlaubt, wie die Quellen versichern, einen Zugriff auf XML-Daten ohne Performanceverluste im Vergleich zu relationalen Zugriffen.

Beispiel: XML in DB2 (aus [Bey05])

```
-- Schema registrieren
register XML-Schema '<url>' from 'new_schema.xsd' as SchemaV1 complete;

-- Tabelle mit XML-Spalte erstellen
create table Interest ( id int not null, doc xml not null );

-- XQuery mit Aufruf von SQL (gekürzt)
for $i in
       db2-fn:sqlquery(„select doc from Interest ... „)
return $i/i:interest/i:prime/data(.)
```

XML ist semi-strukturiert. Verbindent man nun XML mit wohlstrukturierten Datenbanken, werden diese zwangsläufig aufgeweicht. [Bey05] sieht darin eine Chance, Datenbanksysteme an die reale „schmutzige" Welt anzupassen. Hierbei bietet XML einen Zugriff auf Daten, der Toleranter gegenüber Änderungen des Schemas ist als reine SQL-Zugriffe.

2. XML und Web Service Evolution

Kein Datenbankschema ist perfekt – kein XML-Schema ist perfekt. Wie Datenbanken Evolution erfahren, erfahren auch XML-Schemata Evolution.

David Lorge Parnas (in [Par94]): „Software aging will occur in all successful products."

2.1 Evolution

Für XML-Daten ergeben sich nach [Kle05] zwei Arten der Evolution. Auf der einen Seite die Evolution der XML-Daten an sich, auf der anderen die Evolution des Schemas von XML-Daten.

Abbildung 4: Evolution von Daten

Da die Evolution der XML-Daten das Schema unverändert lässt, sollten Anwendungen, die auf dieses Schema aufbauen, keine Probleme mit dieser Art der Evolution haben.

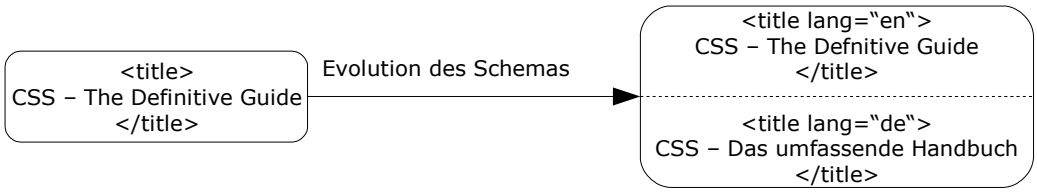


Abbildung 5: Evolution des Schemas

Anwendungen, die die alte Version des Schema benutzen, müssen unter Umständen angepasst werden, da sie das neue Schema von sich aus nicht kennen können und das alte eventuell inkompatibel wird.

Diese Ausarbeitung befasst sich mit der Evolution des Schemas von XML-Daten.

Nach [Kle05] gibt es folgende Gründe für die Notwendigkeit der Anpassung von XML-Schemas:

- unzureichendes Design: Das vorhandene Schema erfüllt nicht notwendige Anforderungen und muss überarbeitet werden. Falls bereits Daten integriert sind (zum Beispiel bei inkrementeller Entwicklung), muss Schema Evolution durchgeführt werden.
- neue Anforderungen: Das vorhandene Schema erfüllt nicht neu hinzugekommene

Anforderungen.

- Änderungen in der Software-Umgebung: Neue Anwendungen sollen unterstützt werden. Ein Beispiel wäre die Migration auf eine neues DBMS.

Wie auch bei relationalen Datenbanksystemen, gibt es einige grundlegende Axiome, um die Evolution von XML-Schemata zu erleichtern. Laut [Bey05] sind dies:

- kein Zugriff auf Tabellen über „select *" oder „//": Falls eine Informationserweiterung stattgefunden hat (hinzufügen neuer Elemente oder Attribute), würden nun deutlich mehr Daten abgerufen und diese für die Anwendung möglichweise unbrauchbar.
- benutzen von Views, um eine zusätzlichen Abstraktionsebene zu erhalten: Die Anwendungen sollen auf Views (virtuelle Sichten) arbeiten, die das tatsächliche Datenbank oder XML-Schema abstrahieren. Diese Views können, wenn das tatsächliche Schema geändert werden muss, nach Bedarf angepasst werden.

[Se05] erkennt, das Evolution teuer ist: Im Sinne von Rechenzeit und ökonomischen Faktoren:

- Abhängigkeiten von anderen Anwendungen: Es müssen unter Umständen andere Anwendungen oder Schemata angepasst werden.
- Kontaktieren der Entwickler: Bei manuellen Eingriffen kann es nötig werden, Entwickler mit Anpassungsaufgaben zu betrauen, falls der DBA bestimmte eingriffe nicht durchführen darf oder kann.
- Arbeitszeit: Nichtautomatische Evolution kostet Arbeitszeit. Besonders bei externem Personal ist dies ein bedeutender Faktor.

Bei Evolutionsschritten muss darüberhinaus aus zwei Extrema das Optimum ermittelt werden:

- Die Datenbank wird für Zugriffe abgeschaltet. Schema-Updates können dann eingefügt werden.
- Die Schema-Updates werden während der Laufzeit der Datenbank durchgeführt. Dies führt zu einer Performanceminderung und bei hoher Last zur Abweisung von Anfragen.

In Abhängigkeit vom Einsatzgebiet der Datenbank ist nun das Optimum zu finden.

2.2 XML-Schema-Evolution

Die XML-Schema-Evolution verändert ein Schema von XML-Daten. Dies kann entweder als DTD [W3C04] vorliegen oder als XML-Schema [W3C04a], was im Folgenden als XML-S bezeichnet wird. Sind beide Ansätze gemeint, wird der Begriff XML-Schema verwendet.

Die Evolution eines XML-Schemas umfasst zwei Teileinheiten: Schema-Evolution und Evolution der XML-Dokumente.

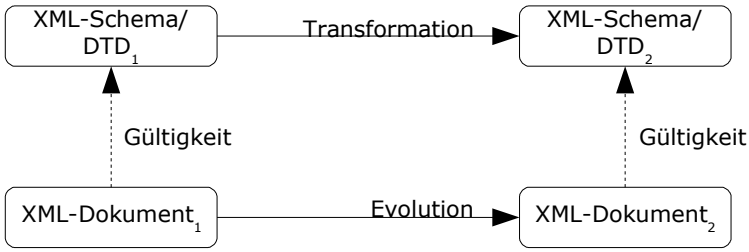


Abbildung 6: XML-Schema-Evolution

[KT05] unterscheidet vier Arten von Veränderungen von Schemata. Diese wurden aus dem Bereich der objektorientierten Datenbanken übernommen ([Tre94]).

Art der Informationsänderung	***Beispiele***
erhaltend	
erweiternd	setzen von „maxOccurs" von „1" auf „unbounded" [1]
reduzierend	setzen von „maxOccurs" von „unbounded" auf „1"
ändernd	<name>Deutschland</name> zu <cname>BRD</cname>

Tabelle 2: Informationsänderungen von XML-Schemas

Bei den fettgedruckten Änderungsarten muss beachtet werden, das XML-Daten, die zum alten Schema kompatibel waren, nun einer Anpassung bedürfen können [KT05].

1 maxOccurs legt in XML-S fest, wie oft ein Element vorkommen darf. unbounded bedeutet beliebig oft. Eine DTD nutzt hierfür die Operatoren *, + und ?.

Weiterhin lässt sich nach [KT05] die Evolution von XML-Schemata in vier Schritte unterteilen:

1. **Ausführen des Evolutionsschrittes auf dem Schema**: Mittels einer Updatesprache werden die Änderungen am Schema durchgeführt.
2. **Berechnen der Constraints**: Ermittlung der Bedingungen der Elemente und Attribute wie erlaubte Häufigkeit des Vorkommens (minOccurs, required, ...).
3. **Inkrementelle Validierung**: Prüfen, ob die XML-Dokumente, die gemäß dem alten Schema valide waren, es auch noch sind.
4. **Generieren von XML-Updates**: Anpassen der XML-Dokumente, falls nötig.

2.3 Web-Service-Evolution

Die Web-Service-Evolution bezeichnet die Veränderung der Schnittstelle eines Web Services.

Man unterscheidet zwei Arten von Veränderungen:

1. kompatible Änderungen
2. inkompatible Änderungen

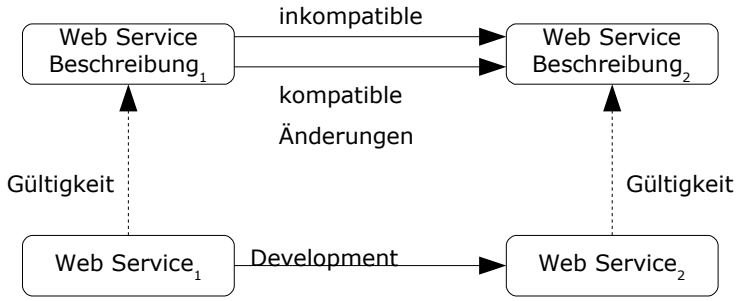


Abbildung 7: Web-Service-Evolution

Unter kompatiblen Änderungen versteht man diejenigen Änderungen, die Funktionalität hinzufügen. Ein Beispiel für eine Parameterliste wäre wie folgt:

```
<country id="BRD">
        <name>Deutschland</name>
</country>
```

Dieses wird geändert in:

```
<country id="BRD">
        <name>Deutschland</name>
        <lname>Bundesrepublik Deutschland</lname>
</element>
```

Ein Zugriff, etwa mit einer Anfragesprache wie XPath oder XQuery, auf das Element „name" bleibt hier unverändert möglich. Ein Zugriff auf „lname" wird möglich. Ein Zugriff auf „BRD" liefert anders formatierte Daten zurück – ein Verarbeitung dieser Daten ist für einen nicht-aktualisierten Benutzer nicht möglich.

Inkompatible Änderungen hingegen reduzieren Funktionalität oder stellen sie anders dar. Und zwar in der Hinsicht, das es existierenden Applikationen nicht mehr möglich ist, auf eine Schnittstelle in gleicher Weise wie vorher zuzugreifen.

Andere Darstellung:

```
<country id="BRD">
       <de_name>Deutschland</de_name>
       <en_name>Germany</en_name>
</country>
```

Weniger Information:

```
<country id="BRD"/>
```

Applikationen, die auf einen Web Service mit dieser Schnittstelle zugreifen, müssen geändert werden, da nötige Informationen nicht mehr zur Verfügung stehen.

2.4 XML-Update-Architekturen

XML-Systeme erhalten Anfragen die das XML-Schema der integrierten Daten verändern würden, falls diese akzeptiert werden. Anhand der Reaktion des Systems kann eine Klassifikation erfolgen (nach [KMH05]):

Typ	***Erläuterung***	***Vorteile***	***Nachteile***
IGNORE	kein Schematest, jedes Update wird angenommen	alle Updates können durchgeführt werden; keine Validierung nötig; passt sich dynamisch an	Schema-verletzende Updates werden durchgeführt, andere Applikationen können hierbei unbenutzbar werden
REJECT	inkompatible Updates werden verworfen, wenn das Schema verändert werden würde	XML-Schema bleibt stets erhalten, Applikationen bleiben benutzbar	Transaktionskonzept muss implementiert werden zur Vermeidung inkompatibler Zwischenzustände, starres Schema
REDO	inkompatible Updates werden verworfen, der Benutzer wird benachrichtigt, er kann das Schema manuell anpassen und das Update wiederholen	Schema-Anpassung kann kontrolliert erfolgen, Applikationen bleiben nach Bedarf benutzbar	Anpassung muss manuell durchgeführt werden; zeitaufwändig; fehleranfällig
EVOLVE (siehe unten)	erlaubt alle Updates, bei inkompatiblen Updates wird das Schema angepasst, teilweise automatisch	implizite Benutzerverwaltung; XML-Schema bleibt nach Wunsch erhalten – oder nicht	komplex; Änderungen sind nicht offensichtlich für den Benutzer und können weitreichende Änderungen des Schemas bewirken

Tabelle 3: Übersicht XML-Architekturen

Außerdem kann nach der Gruppe des Benutzer unterschieden werden, der die Anfrage abgesendet hat. So kann sich ein XML-System bei Schema-Änderungen einem Administrator gegenüber wie IGNORE verhalten, einem Benutzer gegenüber wie REJECT.

[KMH05] schlägt ferner die EVOLVE-Architektur vor. Wird eine inkompatible Änderung durchgeführt, entscheidet sich das System für eine der folgenden Änderungsoperationen. Dies geschieht in Abhängigkeit von Benutzer- und Systemkontext:

1. **REJECT**: Alle inkompatiblen Änderungen werden verworfen.
2. **EVOLVE**: Inkompatible Änderungen haben eine Schema-Evolution zur Folge.
3. **USER-DEFINED**: Der Benutzer (bzw. Administrator) entscheidet, ob Methode 1 oder 2 durchgeführt wird.
4. **SEMI-AUTOMATIC**: Generieren von möglichen Evolutionsschritten. Nach Auswahl durchführen der Evolution oder verwerfen.

3. XML und Web Service Evolution: Einige Ansätze

Dieses Kapitel stellt einige konkrete Ansätze zur Schema-Evolution vor. Dabei wurde versucht, eine rote Linie in der Form herzustellen, das mit einfacheren, das heißt manuellen, Ansatz begonnen und mit einem kompletten Framework für XML-Schemata und Web-Services geschlossen wird.

3.1 XSLT: XML Transformation

XSLT [W3C99a] ist eine XML-Transformationssprache für XML-Dokumente. Es handelt sich um ein System, mit dessen Hilfe es möglich wird, XML-Dokumente von einer Version in die nächste zu überführen. Es wird dabei ausdrücklich nicht auf das Schema des Dokumentes Bezug genommen – es ist Aufgabe des Benutzers, eine Transformation zu entwickeln, die gültige Dokumente des Ausgangsschemas in gültige Dokumente des Zielschemas umwandelt. Interessant ist hierbei, das es genügt, dass die betreffenden Schemen implizit vorhanden sind.

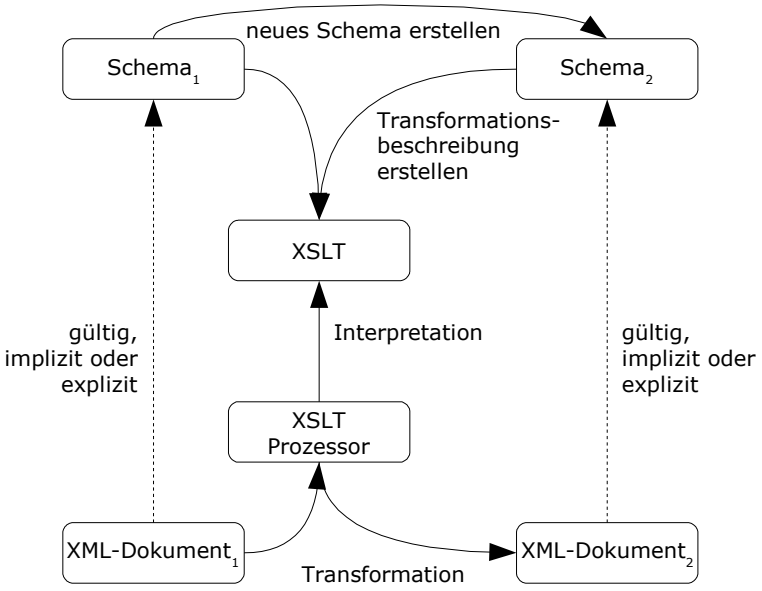


Abbildung 8: XSLT Evolutionsschritt

Leider sind bei XSLT Endlosschleifen nicht ausgeschlossen. Auch ist der Zugriff nur auf Teile einer DTD möglich, sowie der Zugriff auf bereits geparste Entitäten [Zei05].

In [Zei05] schlägt Zeitz ein System vor, das aufbauend auf abstrakten Änderungen der DTD einen XSLT-Prozessor bedient, der XML-Daten umwandelt.

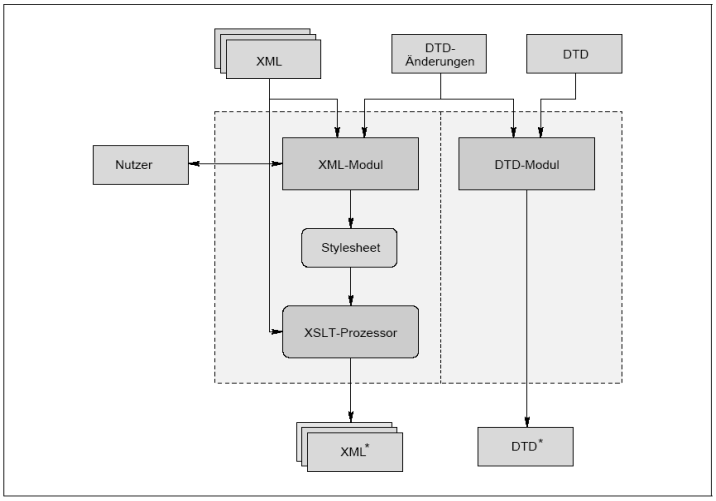


Abbildung 9: Systemaufbau nach Zeitz

3.2 XEM: XML Evolution Management

Kramer entwickelt in [Kra01] (in [SKR02] publiziert) ein System von Operatoren, um XML-Daten sowie XML-Schemata umzuformen. Beide Arten werden getrennt voneinander gesehen. Es wird zwar auf XML-S eingangen, die Umformungen beziehen sich jedoch explizit auf DTDs.

Die folgende Tabelle zeigt die eingeführten Operatoren:

Bereich	***Operator***	***Beschreibung***
DTD	createDTDElement()	Erstellen eines Blattes
	destroyDTDElement()	Löschen eines Blattes
	insertDTDElement()	Einfügen eines Elementes
	removeContentParticle()	Löschen eines Elementes
	changeQuant()	Ändern eines Quantifizierers
	convertToGroup()	Gruppieren von Unterelementen
	flattenGroup()	Gruppierung von Unterelementen aufheben
	addDTDAttr()	Hinzufügen eines Attributes
	destroyDTDAttr()	Löschen eines Attributes
XML-Daten	createDataElement(e,v)	Erstellen eines Blattes
	addDataElement(e,i)	Hinzufügen eines Elementes
	destroyDataElement()	Löschen eines Blattes
	addDataAttr(s,v)	Hinzufügen eines Attributes
	destroyDataAttr(s)	Löschen eines Attributes

Tabelle 4: XEM Operatoren

Es müssen einige Vorbedingungen beachtet werden. Für *destroyDataElement* wäre dies zum Beispiel: Das Element muss optional sein.

Die exakten Definitionen aller Operatoren entnimmt sich der geneigte Leser aus [Kra01] - die formale Definition für *addDataElement* ist wie folgt:

```
Primitive: addDataElement
Syntax: n.addDataElement(ElemDef e,DataPosition i)
Semantics: Add a new element e to be the i_th subelement of element n.
Preconditions: A new element instance is allowed to be added only in two cases. In case
1, the type of the new element instance is a repeatable content particle. In case 2, the
type of the new element instance is an optional content particle and no instance of this
content particle exists before.
Resulting Data Changes: The element instance e will be added to the children list of n
as the i_th child.
```

Im Beispiel wird ein Teilbaum eingefügt:

```
<article>
  <title>XML Evolution Manager</title>
  <author id = "dk">
    <name>
      <first>Diane</first>
      <last>Kramer</last>
    </name>
  </author>
  <author id = "er">
    <name>
      <first>Elke</first>
      <last>Rundensteiner</last>
    </name>
  </author>
  <affiliation>WPI</affiliation>
  <related>
    <monograph>
       <title>Modern database systems</title>
       <editor name = "Won Kim"></editor>
    </monograph>
  </related>
</article>
```

(a)

```
<article>
  <title>XML Evolution Manager</title>
  <author id = "dk">
    <name>
      <first>Diane</first>
      <last>Kramer</last>
    </name>
  </author>
  <author id = "er">
    <name>
      <first>Elke</first>
      <last>Rundensteiner</last>
    </name>
  </author>
  <affiliation>WPI</affiliation>
  <related>
    <monograph>
       <title>Modern database systems</title>
       <editor name = "Won Kim"></editor>
    </monograph>
    <monograph>
       <title>XML</title>
       <editor></editor>
    </monograph>
  </related>
</article>
```

(b)

Abbildung 10: Operator addDataElement (aus [Kra01], Seite 20)

Desweiteren stellt Kramer fest, dass die hier eingeführten Operatoren vollständig sind und alle möglichen DTD-Umformungen unterstützt.

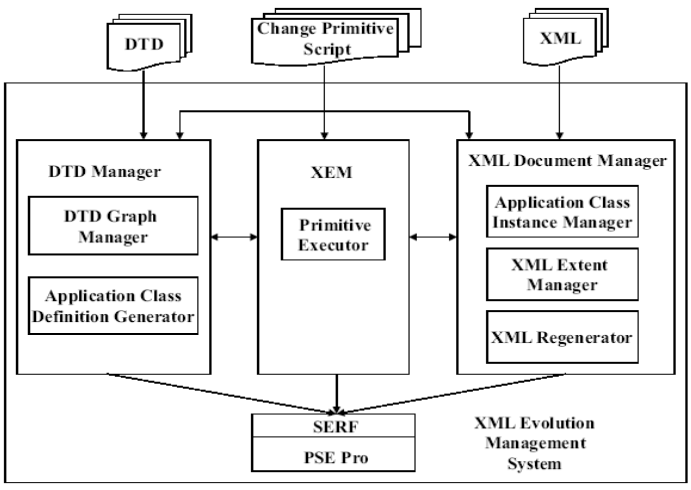


Abbildung 11: Architektur des XEM-Prototyps (aus [Kra01], Seite 20)

Ein Prototypsystem wurde entwickelt. Dessen Architektur ist in der obigen Abbildung ersichtlich. Dieses System benötigt ein Script „Change Primitive Script" aus den bereits erwähnten Umformungsoperationen. Dieses wird nicht maschinell erstellt. Der Ansatz ähnelt daher eher XSLT.

3.3 XQuery XML Evolution

Tatarinov entwickelt in [Tat01] eine Erweiterung für XQuery zur Manipluation von XML-Dokumenten. Klettke fasst dies in [KMH05] auf, um dessen Auswirkungen auf DTDs wie XML-S zu studieren.

Operator	***Beschreibung***
DELETE *child*	*child* löschen
INSERT *content* **BEFORE** *ref*	*content* einfügen, genau vor *ref*
INSERT *content* **AFTER** *ref*	*content* einfügen, genau nach *ref*
RENAME *child* **TO** *name*	*child* umbennen in *name*
REPLACE *child* **WITH** *content*	*child* ersetzen mit content

Tabelle 5: Update-Operatoren für XQuery

Die Operatoren werden in einer Update-Klausel innerhalb der XQuery-Anfrage geschrieben. Ein Beispiel (aus [KMH05]) hierzu:

```
FOR $f IN document(„books.xml“)/book/fm,
    $a IN $f/author[last()]
WHERE $f/title = „XML and Databases“
UDPATE $f {
    RENAME $a TO „editor“
}
```

Das letzte Author-Element in einer XML-Struktur wird umbenannt in „editor“.

Mit der Definition der Operatoren, stellt [KMH05] fest, ist die Update-Operation aber noch nicht abgeschlossen: Das XML-kompatible System muss daraufhin prüfen, ob diese zulässig ist. Etwa setzt die *DELETE*-Operation voraus, das das zu löschende Element als ? oder * in einer DTD (bzw. minOccurs=“0“ in XML-S) definiert war. Andererseits muss das XML-Schema angepasst werden.

Daraus folgen die möglichen Evolutionsschritte für XML-Schemata:

- optionale Attribute oder Elemente hinzufügen
- Attributdeklarationen ändern von benötigt nach optional
- minOccurs verringern
- maxOccurs erhöhen

Es wird ersichtlich, das diese Änderungsoperationen allesamt informationserweiternd wirken. Das heißt: Alle mit diesem XML-Schema verbundene Elemente bleiben gültig.

Weiterreichende Änderungen bedürfen der Änderung der mit dem Schema verbundenen XML-Daten. Der Reaktion des Systems auf solche Anfragen erfolgt in Abhängigkeit von seiner Architektur (siehe Abschnitt XML-Update-Architekturen).

Desweitern wird festgestellt, das bei der Benutzung von XML-S auch auf eine Evolution von Datentypen zu achten ist. Ist der zuendernde Datentyp eine Teilmenge des Zieldaten-typs (etwa von SmallInt zu Int), ist die Evolution trivial. Bei Verkleinerungen hingegen nur bedingt möglich.

3.4 XSEL: XML Schema Evolution Language

XSEL (XML-Schema Evolution Language) ist eine von Tiedt in [Tie05] entwickelte XML-Applikation, die Evolution von XML-Schemata ermöglicht. XSEL versucht, möglichst allgemein zu bleiben, um auch auf XML-Daten arbeiten zu können.

Vorausgesetzt wird, dass das XML-Schema in der von Tiedt in [Tie03] entwickelten Normalform vorliegt: „Die Normalform schreibt vor, dass alle Typdefinitionen global erfolgen müssen." (aus [Tie05], Seite 62).

Als XML-Datenmodell wird die Baum-Repräsentation aus DOM angewendet. Dabei wird bei den Knoten nicht zwischen Elementen, Attributen und Text unterschieden. Es muss daher nur mit einem eindimensionalen Baum gearbeitet werden.

Operationen

Es werden eine Reihe von Operationen definiert, mit denen Baum-Strukturen manipuliert werden:

- **`add(k*,n)`**: Einfügen eines Knotens k unterhalb eines anderen n.
- **`insert_before(k*,n)`**: Einfügen eines Knotens k auf gleicher Ebene vor einem anderen Knoten n.
- **`insert_after(k*,n)`**: Einfügen eines Knotens k auf gleicher Ebene nach einem anderen Knoten n.
- **`move(b,n)`**: Verschieben eines Teilbaums b unterhalb eines anderen Knoten n.
- **`move_before(b,n)`**: Verschieben eines Teilbaums b auf gleiche Ebene vor einem anderen Knoten n.
- **`move_after(b,n)`**: Verschieben eines Teilbaums b auf gleiche Ebene nach einem anderen Knoten n.
- **`replace(k*,l)`**: Ersetzen eines Knotens k durch einen neuen Knoten l.
- **`delete(k*)`**: Löschen eines Knotens k.
- **`rename(k*,s)`**: Umbenennen eines Knotens k in s.
- **`change(k*,s)`**: Ändern des Wertes eines Knotens k in s.

Ein * gibt an, dass hier beliebig viele Knoten selektiert werden können. Ist dieser weggelassen, kann nur jeweils ein Knoten selektiert werden. Die move-Operationen verschieben Teilbäume – also implizit auch mehrere Knoten.

Diese Operatoren bilden laut Tiedt alle nötigen Evolutionsschritte ab. Als Syntax sei hier

ein Beispiel für eine Add-Operation gegeben:

```
<add
        select="//xs:complexType[@name='kontakt']/xs:sequence"
        content="&lt;xs:element
                name=&quot;Nachname&quot;
                type=&quot;xs:string&quot;/&gt;"
/>
```

Die Anfrage graphisch:

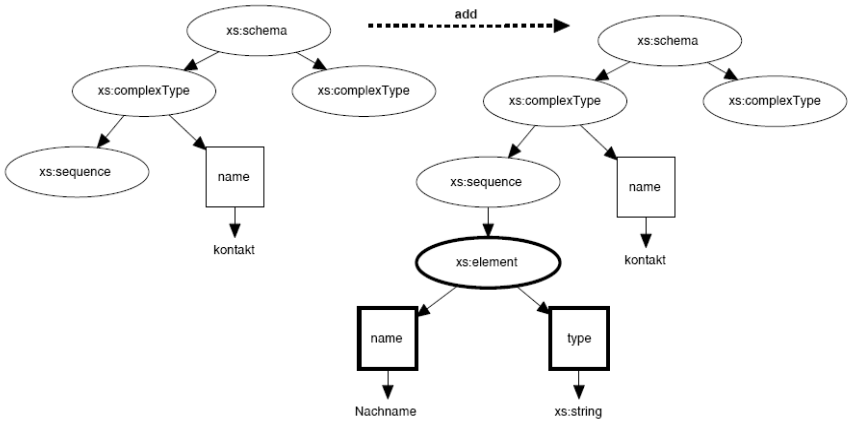


Abbildung 12: Add-Operation

Beim Ausführen solcher Operationen wird weiterhin darauf geachtet, das nur syntaktisch korrekte Schemata erzeugt werden. So können Attribute lediglich an Elemente gehängt werden.

XSEL unterstützt die Gruppierung von Anfragen als „Queries" und „Transaktionen". Damit wird eine Reihe von Umforumungen atomar. Queries sind lose Gruppierungen von Anfragen, deren Arbeit mit Hilfe eines einzelnen Rollbacks rückgängig gemacht werden kann. Die Validität des Schemas wird nach jedem Schritt geprüft. Transaktionen hingegen lassen nichtvalide Zwischenzustände zu. Transaktionen können in Queries eingebettet werden.

Die Evolution der XML-Dokumente geschieht mittels Inkrementeller Validierung. Diese generiert XSEL-Operationen, die die Dokumente dann ändern.

Zur XSEL wurde ein modularer Anfrageprozessor entwickelt. Als Basis dafür dient Java 1.4.2 und Xerces.

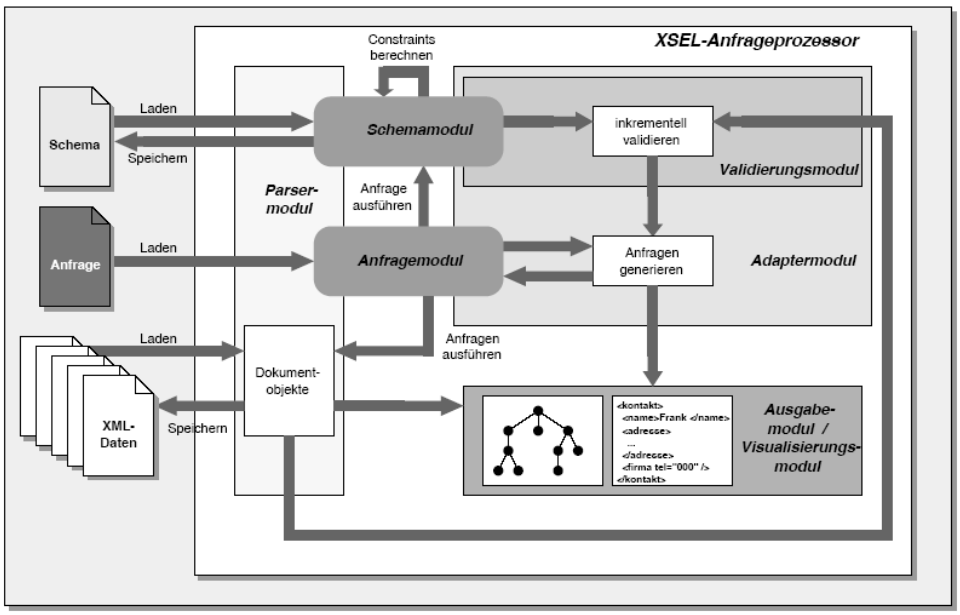


Abbildung 13: XSEL-Prozessor (aus [Tie05], Seite 74)

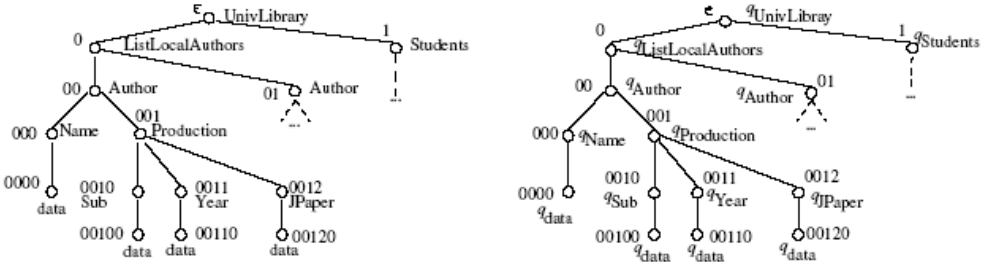
3.5 GREC: Manuelle, graphenbasierte Evolution

Bouchou et. al. schlägt in [Bou04] eine informationserhaltende Methode vor, in der mittels einem eigens entwickelten Algorithmus, GREC, Änderungen an dem Graphen eines endlichen Automaten vorgenommen werden. Dabei wird bei Aufnahme von inkompatiblen XML-Daten das XML-Schema angepasst anstatt die XML-Daten anzupassen. Es wird bei Änderungsoperationen zwischen Benutzern und Administratoren unterschieden:

Benutzer	***Update-Operation[2]***	
Benutzer	kompatibel	wird durchgeführt
	inkompatibel	wird abgelehnt
Administrator	kompatibel	wird durchgeführt
	inkompatibel	wird durchgeführt mit Schema-Änderung

Tabelle 6: Benutzerabhängige Änderungen an XML-Schemata

Der vorgestellte Algorithmus schlägt Evolutions-Möglichkeiten vor, von denen der Administrator wählen kann. Damit richtet er sich ausdrücklich nicht an Informatik-Wissenschaftler sondern an Administratoren, die entscheiden können, zu wählen, in welche Richtung das existierende Schema abgewandelt werden soll.



Der Algorithmus GREC sieht das Schema eines XML-Dokumentes als bennanten Baum wie in Abbildung 14 verdeutlicht. Seine Ausführung auf diesem Baum bewirkt eine Überprüfung der Gültigkeit des XML-Dokumentes [Bou04]. Bei einem Update schlägt der Algorithmus fehl und bietet dem Aufrufer eine Auswahl an regulären Ausdrücken, mit denen er den Baum abändern lassen kann.

2 kompatible (valide XML-Daten bleiben valide) oder inkompatible Änderungen

Ein Beispiel [Bou04]:

Einfügen des Elementes *ConfPaper* an Position 0013 des Baumes aus Abbildung 14. Dies geschieht rechts von 0012.

- Überprüfen, ob der übergeorndete Knoten 001 eine solche Erweiterung zulässt
- (1) Erweiterung: $q_{Sub}q_{Year}q_{ConfPaper}$
- (2) aktueller Zustand: $(q_{Year}q_{JPaper}+)*$
- Ziel: Zustand, der (1) und (2) akzeptiert
- Wenn nicht, ist es ein invalides Update (es verändert das Schema)
    - Der Baum A aus Abbildung 14 wird nun ersetzt duch einen Baum A', zu dem alle Dokumente valide sind, die auch zu A valide waren; A' unterscheidet sich von A nur in einer der möglichen Änderung:
        1. Einfügen von einem oder mehreren *ConfPaper*, nicht nach Jahren organisiert.
            1. $(q_{Sub}(q_{Year}\ q_{JPaper}+)*q_{ConfPaper}?)$
            2. $(q_{Sub}(q_{Year}q_{JPaper}+)*(q_{ConfPaper})*)$
        2. Einfügen von einem oder mehreren *ConfPaper* zu einer, nach Jahren organisiert.
            1. $(q_{Sub}(q_{Year}q_{JPaper}+q_{ConfPaper}?)*)$
            2. $(q_{Sub}(q_{Year}q_{JPaper}+q_{ConfPaper}*)*)$
        3. Einfügen von mehreren *ConfPaper* zu jedem Jahr mit der Einschränkung, das *JPaper* vor *ConfPaper* vorkommt, und ohne.
            1. $(q_{Sub}(q_{Year}(q_{JPaper}q_{ConfPaper}?)+)*)$
            2. $(q_{Sub}(q_{Year}(q_{JPaper}q_{ConfPaper}*)+)*)$
        4. JPaper und ConfPaper gibt es separat, es muss eines von beiden vorkommen.
            1. $(q_{Sub}(q_{Year}(q_{JPaper}|q_{ConfPaper})+)*)$
- der DBA entscheidet nun, welche Änderung am Schema durchgeführt wird.

Der Algorithmus erhält diese regulären Ausdrücke, indem er ein endlicher Glushkov-Graph durchgeht.

```
(1) procedure GREC(G_A, O_A, s_nl, s_nr, s_new) {
(2)  if graph G_A has only one node
(3)    then stop
(4)    else{
(5)     R_i := ChooseRule(G_A, O_A);
(6)     for each (G_B,O_B):=LookForGraphAlternative(G_A,O_A,R_i,s_nl,s_nr,s_new) do
(7)         GraphToRegExp(G_B, O_B);
(8)     G_C := ApplyRule(R_i, G_A);
(9)     GREC(G_C,O_A,s_nl,s_nr,s_new);
(10)   }      }
```

Abbildung 15: GREC-Algorithmus

3.6 WSDL/UDDI: Web-Service-Evolution

WSDL und UDDI sind zwar nicht zur Evolution von Web Services gedacht, bieten aber eine begrenzte Unterstützung, die sich aus ihrer Spezifikation ergibt.

WSDL

WSDL [W3C01] ist die gängige Beschreibungssprache für Web-Services. Mithilfe von WSDL wird die Schnittstelle eines Web-Services definiert. Mittels SOAP [W3C03] wird auf diesen danach zugegriffen.

Eine kompatible Änderung

```
<message name="getInterestRequest">
        <part name="asOf" type="xs:dateTime" />
</message>

<message name="getInterestRequest2">
        <part name="asOf" type="xs:dateTime" />
        <part name="country" type="xs:string" />
</message>

<message name="getInterestResponse">
        <part name="prime" type="xs:decimal" />
</message>

<portType name="glossaryTerms">
        <operation name="getInterest">
                <input message="getInterestRequest" />
                <input message="getInterestResponse" />
        </operation>

        <operation name="getInterest2">
                <input message="getInterestRequest2" />
                <input message="getInterestResponse" />
        </operation
</portType>
```

Zugreifende Applikationen können durch parsen der Datei nun feststellen, das eine Informationserweiterung stattgefunden hat. Sie können jedoch nicht feststellen, das die Operation „getInterest2" eine aktualisierte Version von „getInterest" ist.

Über Web-Server-Statistiken kann dann entschieden werden, wann die alte Version des Web-Services abgeschaltet werden kann.

UDDI

Diese bei WSDL beschriebene Herangehensweise ist analog auch auf UDDI übertragbar. Statt auf verschiedenen Operationen agiert man an dieser Stelle aber auf verschiedenen Web Services.

3.7 DB2 Web Service Evolution

DB2 unterstützt laut [Bey05] das rationale Abspeichern von XML-Daten in Tabellenspalten (Datentyp „xml"). Navigation ist mittels SQL/XML ([ISO05]) und XQuery ([W3C05]) möglich, beide Anfragesprachen können sich gegenseitig aufrufen. XML-Schemas werden untertstützt und erlauben optionales Validieren von XML-Dokumenten mit „xmlValidate". Dazu besitzt DB2 eine Schema-Sammlung, die XML-Schemas nativ als XML-Dokumente speichert. Desweiteren werden Indizes von XPath-Ausdrücken [W3C99] unterstützt.

Die hier vorgestellte Herangehensweise von Beyer et al [Bey05] nutzt diese Merkmale, um ein konkretes Beispiel zu modellieren, die „Teenee Bank", die jeden Tag aktuelle Zinssätze über einen Web Service einholen will. Dieser Ansatz unterscheidet explizit zwischen kompatiblen und inkompatiblen Änderungen.

Web Service Evolution

Die durch den Web-Service gelieferten Daten sind gegenüber einem bestimmten XML-Schema valide. Im Beispiel wäre dies:

```
schema:        interest-v1.xsd
interest
               asOf    dateTime
               prime   decimal
```

Wird der Web-Service abgefragt, liefert er zwei Dokumente: Ein Evolve-Dokument und sowie den aktuellen Zinssatz gültig gemäß dem aktuellen Schema.

```
<interest>
       <asOf>2004-12-13T07:00:00-05:00</asOf>
       <prime>5.501</prime>
</interest>
```

Ist nun dieses Schema überholt, so liefert der Web-Service eine Evolve-Aufforderung zurück:

```
<evolve>
       <service>bogus://fed.gov/interest-v1.php</service>
       <deprecated>
              <expires>2005-03-23T20:59:59-08:00</expires>
              <replacedBy>bogus://fed.gov/interest-v2.php</replacedBy>
              <reason>new-required-tags</reason>
              <detailsAt>bogus://fed.gov/new-in-v2.html</detailsAt>
       </deprecated>
</evolve>
```

An diesem Punkt wird unterschieden zwischen kompatiblen und inkompatiblen Änderungen.

1. Kompatible Änderungen (hier durch `<reason>new-required-tags</reason>` impliziert) machen keine Änderungen am bisherigen Zugriff auf die Daten nötig – sie sind informationserweiternd. Anfragen, die Wildcards wie „/insert//*", enthalten,

wurden im Vorhinein verboten. Diese Anfragen würden nun mehr Daten liefern und das resultierende Schema wäre dem Benutzer unbekannt. Es ändert sich allerdings das Schema der gelieferten Daten. Dies muss nun mit der Datenbank registriert werden.

2. Inkompatible Änderungen werden dagegen manuell durchgeführt. Dabei wird das neue Schema registriert und die Datensammlung damit weitergeführt. Um Applikationen, die mit dem alten Schema vertraut sind, das Weiterarbeiten zu ermöglichen, wird deren View angepasst (informationsändernd). Damit geschieht ein solche Schema-Wechsel für die Applikation transparent. Nach Bedarf, können Applikationen danach einzeln umgestellt oder ersetzt werden.
3. Ist dies nicht möglich (informationsreduzierend), so muss die Applikation angepasst werden.

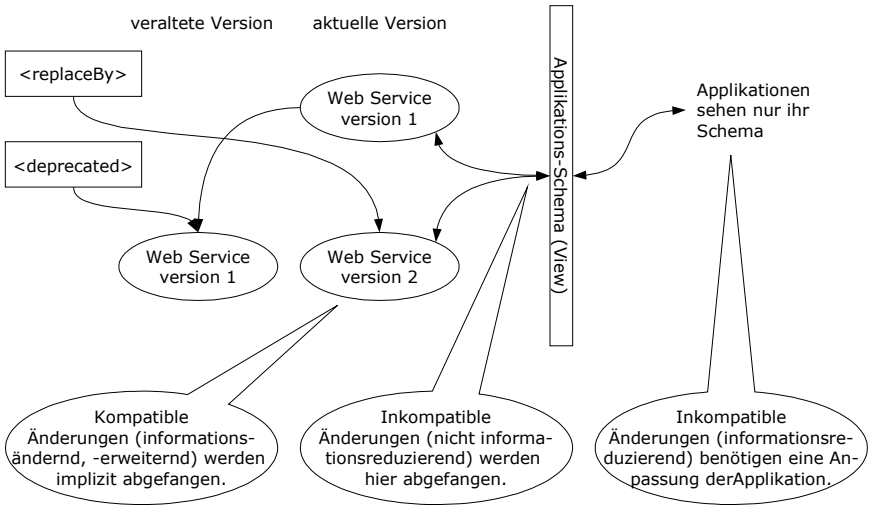


Abbildung 16: (In-)kompatible Änderungen bei Web Services

Laut [Bey05] sollte die Mehrheit der Änderungen kompatible Änderungen sein und damit automatisch evolutionierbar.

Multiple Versions Query

Eine Anfrage über mehrere Versionen von Schemata sind laut [Bey05] schwieriger zu evolutionieren als Anfragen in einer Version. Es werden zwei Lösungsstrategien vorgeschlagen:

1. Für jede beteiligte Version wird ein View erstellt, welcher jede Version in ein universelles Dokument umwandelt.
2. Jede beteiligte Version muss manuell angepasst werden, wenn nötig. Kompatible Versionen bleiben hierbei ungeberührt.

3.8 Weitere

An dieser Stelle einige Ansätze zur Schema Evolution, die in dem letzten Kapitel nicht weiter vertieft wurden.

Microsoft SQL Server 2005

Auch der Microsoft SQL Server 2005 bietet laut [Rys04] eine grundlegende Unterstützung für XML-Schema-Evolution, was durch [PFD05] bestätigt wird:

1. Änderungen von XML-Schemata, die das bestehende Schema nicht invalide machen würden, werden akzeptiert.
2. Änderungen von XML-Schemata, die das bestehende Schema invalide machen würden, müssen manuell durchgeführt werden: Zuerst wird das bestehende XML-Schema fallen gelassen. Daraufhin werden die Ändernungen manuell, etwa per XSLT [W3C99a], durchgeführt. Zuletzt werden die Daten wieder mit dem neuen XML-Schema typisiert.

[Rys04] erkennt dabei, das die 2. Art sehr komplex werden kann und dies daher vom Benutzer vorher in Betracht gezogen werden muss.

4. Fazit

Die vorgestellten Ansätze aus Kapitel 3 unterlegen die in Kapitel 2 gemachte Unterscheidung zwischen kompatiblen und inkompatiblen Updates. XML, in Kombination mit dessen Anfragesprachen, erlaubt es, kompatible Änderungen an Schemata und Web-Service-Interfaces ohne Änderungen der Zugriffe zu gestatten. Demhingegen müssen inkompatible Änderungen immer noch besonders behandelt werden: Hier ist eine manuelle oder algorithmengestützte Anpassung der Zugriffe von Nöten.

Interessierte Leser sollten sich nun die eine oder andere Quelle zu Gemüte führen, da eine wirklich erschöpfende Erörterung aller Einzelheiten den Rahmen dieser Ausarbeitung bei weitem sprengen würde. Darüberhinaus ist das Feld der XML-Schema-Evolution sehr jung. Es ist zu erwarten, das in unmittelbarer Zukunft noch einige interessante Ideen auf uns zukommen. Viele in dieser Seminararbeit benutzten Quellen sind erst im Laufe des Jahres 2005, in welchem diese Arbeit fertiggestellt wurde, entstanden.

Beyer et. al. proklamieren in [Bey05] für XML-fähige Datenbanken: Nicht mehr korrekt auf das verwendete Schema zu pochen, sondern auch Daten ohne festgelegtes Schema zu akzeptieren. Dies bedeutet, ein Stück weit Kontrolle an die Anwendungen abzugeben. Die Anwendungen müssen also lernen, damit umgehen zu können.

4.1 Literaturverzeichnis

[Bey05] Beyer, Özcan, Saiprasad, Van der Linden: DB2/XML - Designing for Evolution, SIGMOD 2005.

[Bou04] Bouchou, Duarte, Alves, Laurent, Musicante: Schema Evolution for XML: A Consistency-preserving Approach, 2004.

[ISO05] ISO: SQL: XML-Related Specifications (SQL/XML), 2005

[Kle05] Meike Klettke: XML-Updates und Schemaevolution, 2005.

[KMH05] Klettke, Meyer, Hänsel: Evolution - The Other Side of the XML Update Coin, 2005. URL: http://www.xml-und-datenbanken.de/sada.html

[Kra01] Diane Kramer: XEM: XML Evolution Management, 2001. URL: http://www.wpi.edu/

[KT05] Meike Klettke, Tobias Tiedt: XML-Schemaevolution und inkrementelle Validierung, 2005. URL: http://www.xml-und-datenbanken.de/sada.html

[MS04] Christoph Meinel, Harald Sack: WWW, 2004. ISBN: 3540442766

[OA04] OASIS: UDDI 3.0.2 Specification, 2004. URL: http://uddi.org/pubs/uddi_v3.htm

[Par94] David Lorge Parnas: Software aging, IEEE Computer Society Press 1994.

[PFD05] Pal, Fussell, Dolobowsky: XML Support in Microsoft SQL Server 2005, 2005. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/sql2k5xml.asp

[RFC 1738] Network Working Group: RFC 1738 - URLs , 1994. URL: http://www.ietf.org/rfc/rfc1738.txt

[Rys04] Michael Rys: XML Schema Evolution in SQL Server 2005, 2004. URL: http://www.sqljunkies.com/WebLog/mrys/archive/2004/05/11/2438.aspx

[Se05] Eric Sedlar: Managing Structure in Bits and Pieces: The Killer Use Case for XML, SIGMOD 2005 2005.

[SKR02] Su, Kramer, Rundensteiner: XEM: XML Evolution Management, 2002. URL: http://www.wpi.edu

[SX04] ISO: SQL Part 14: SQL/XML (9075-14), 2004. URL: http://www.sqlx.org/

[Tat01] Tatarinov, Ives, Halevy, Weld: Updating XML, 2001.

[Tie03] Tobias Tiedt: Normalform für XML-Schema, 2003. URL: http://www.xml-und-datenbanken.de/sada/

[Tie05] Tobias Tiedt: Schemaevolution und Adaption von XML-Dokumenten, 2005. URL: http://www.xml-und-datenbanken.de/sada/

[Tre94] Markus Tresch: Dynamische Evolution in Objektdatenbanken. Ulm, Teubner Text zur Mathematik, 1994.

[W3C01] W3C: WSDL TR 1.1, 2001. URL: http://www.w3.org/TR/wsdl

[W3C03] W3C: SOAP 1.2 TR, 2003. URL: http://www.w3.org/TR/soap/

[W3C04] W3C: XML TR 1.0 3rd Ed., 2004. URL: http://www.w3.org/TR/REC-xml

[W3C04a] W3C: XMLS TR 1.0 2nd Ed., 2004. URL: http://www.w3.org/XML/Schema

[W3C05] W3C: XQuery Specification 1.0, 2005. URL: http://www.w3.org/TR/2005/CR-xquery-20051103/

[W3C98] W3C: Document Object Model, 1998. URL: http://www.w3.org/DOM/DOMTR

[W3C99] W3C: XPath 1.0 TR, 1999. URL: http://www.w3.org/TR/xpath

[W3C99a] W3C: XSL Transformations, 1999. URL: http://www.w3.org/TR/xslt

[Zei05] Andre Zeitz: Evolution von XML-Dokumenten, 2005. URL: http://www.xml-und-datenbanken.de/sada/