Studienarbeit

Ableitung von Schemaevolutionsschritten aus XML-Updateoperationen

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von	:	Christian Will
geboren am	:	08.05.1979 in Neubrandenburg
Gutachter	:	Prof. Dr. Andreas Heuer
Betreuerin	:	Dr. Meike Klettke
Abgabedatum	:	31.01.2006

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Neubrandenburg, den 30. Januar 2006

Zusammenfassung

Anders als bei Datenbanken können Updates auf XML-Dokumente bewirken, dass das zugehörige Schema nach der Ausführung verletzt ist. Es gibt vier Möglichkeiten dies zu behandeln: 1. Ablehnen dieser Updates, 2. Akzeptieren dieser Updates und zukünftiges Ignorieren des Schemas, 3. Manuelle Schemaevolution und 4. Automatische Schemaevolution.
Im Rahmen dieser Arbeit wird der vierte Fall genauer untersucht. Es wird gezeigt, welche Updateoperationen das Schema eines gegebenen XML-Dokumentes ungültig machen. Für diese Fälle werden mögliche Schemaevolutionschritte generiert, welche die Gültigkeit des Schemas erhalten und damit das Update realisierbar machen.

Abstract

In comparison with the database approach XML updates can violate the structure which is defined by an XML Schema. There are 4 different possibilities to solve this: 1. deny these updates, 2. accept these updates and ignore the schema for the future, 3. manual schema evolution, and 4. automatically schema evolution.
This paper introduces the fourth case. Update operations that violates the associated schema will be selected. If it's possible schema evolution steps will be generated to maintain the validity and to realize these updates.

CR-Klassifikation

E.2	DATA STORAGE REPRESENTATIONS
H.2.1	Logical Design
I.7	DOCUMENT AND TEXT PROCESSING

Schlüsselwörter

XML, XML-Schema, Schema Evolution, XQuery, Anfragesprache, Änderungssprache, inkrementelle Validierung, Dokumentanpassung

Keywords

XML, XML-Schema, Schema Evolution, XQuery, Query Language, Update Language, Incrementeal Validation, Document Adaption

Inhaltsverzeichnis

1 Einleitung **1**

2 XML-Updatesprachen **7**
2.1 Anforderungen an eine XML-Updatesprache 7
2.1.1 Allgemeine Anforderungen 7
2.1.2 XML-spezifische Anforderungen 9
2.2 Existierende Updatesprachen 11
2.2.1 DOM 11
2.2.2 XUpdate 14
2.2.3 XQuery 16
2.2.4 XSL/XSLT 19
2.2.5 Weitere Updatesprachen 21
2.3 Vergleich 21

3 Analyse der ausgewählten Updatesprache **25**
3.1 Operationen 25
3.1.1 INSERT 25
3.1.2 DELETE 27
3.1.3 REPLACE 28
3.1.4 RENAME 28
3.1.5 Bedingte Updates 29
3.1.6 FLW-Update Anweisung 29

4 Schemasprachen und -evolution **31**
4.1 Einführung 31
4.2 XML Schemasprachen 32
4.2.1 DTD 32
4.2.2 XML-Schema 32
4.2.3 RELAX NG 33
4.2.4 Schematron 33
4.3 Datenmodell von XML-Schema 34

4.3.1 Primäre Komponenten 34
4.3.2 Hilfskomponenten 36
4.3.3 Sekundäre Komponenten 37
4.4 Sprachen zur Schemaevolution 37

5 Ableitung der Schemaevolutionsschritte 39
5.1 Updates auf Attributebene 39
5.1.1 Insert Attribute 40
5.1.2 Delete 44
5.1.3 Replace 46
5.1.4 Rename 47
5.2 Updates auf Elementebene 47
5.2.1 Insert Into | Preceding | Following 47
5.2.2 Delete 49
5.2.3 Replace 50
5.2.4 Rename 50
5.3 Besonderheiten 51
5.3.1 Updates von Kommentaren und PIs 51

6 Implementierungsdetails und Ergebnisse 53
6.1 Aufgabe 53
6.2 Verwendete Technologien 53
6.3 Architektur 54
6.4 Beispiel 56
6.5 Ergebnisse 60

7 Verwandte Arbeiten 63

8 Schlussbetrachtung 67
8.1 Zusammenfassung 67
8.2 Ausblick und weitere Ideen 67

Verzeichnis der Abkürzungen 70

Verzeichnis der Abbildungen 71

Literaturverzeichnis 76

Kapitel 1

Einleitung

Es sind bereits 7 Jahre vergangen, seitdem das *W3C (World Wide Web Consortium)* die erste Empfehlung von *XML (Extensible Markup Language)* veröffentlicht hat. Aber was ist eigentlich XML? Das W3C hat die Aktivitäten derzeit in 7 Gruppen eingeteilt, welche in Abbildung 1.1 ersichtlich sind. Dabei ist der eigentliche Sprachentwurf nur ein Bestandteil des XML Kerns (XML Core Working Group). Da die Sprachdefinition von XML die Basis aller ihr umgebenden Technologien ist, werden folgend die wichtigsten Eigenschaften kurz zusammengefasst.

XML (früher XML-lang) ist keine Neuerfindung, sondern eine Teilmenge der bereits im Jahre 1986 von der ISO (International Organization for Standardization) standardisierten Sprache SGML. Sie ist eine Meta-Auszeichnungssprache und erlaubt damit das Definieren einer Menge ähnlich aussehender Auszeichnungssprachen. Sie besitzt eine menschen- und maschinenlesbare Darstellung in Textform, welche jedoch nicht die einzige Darstellungsform bleiben wird. Sie besitzt ein Datenmodell bestehend aus Knoten verschiedener Typen, welche mit Vater-/Kindbeziehungen eine Hierarchie bilden.

Die Sprache allein hätte bestimmt nicht den Erfolg gehabt ohne die Technologien, welche den anderen Arbeitsgruppen zugeordnet wurde. Dazu gehört z.B. die XSL (Extensible Stylesheet Language) Arbeitsgruppe. Sie entwickelt die Transformationssprache XSLT (XSL Transformation), die Achsennavigationssprache XPath (XML Path Language) und die Seitenbeschreibungssprache XSL-FO (XSL Formating Objects).

Seit September 2003 gibt es die Arbeitsgruppe *XML Binary Characterization*. Da durch die textuelle Darstellung und die Redundanz, die z.B. durch die Tagnamen entsteht, die Performance bei der Übertragung und Verarbeitung beeinträchtigt wird, werden binäre Darstellungsformen entwickelt.

Die Erweiterbarkeit von XML zeigt sich auch an der Arbeitsgruppe *XML*

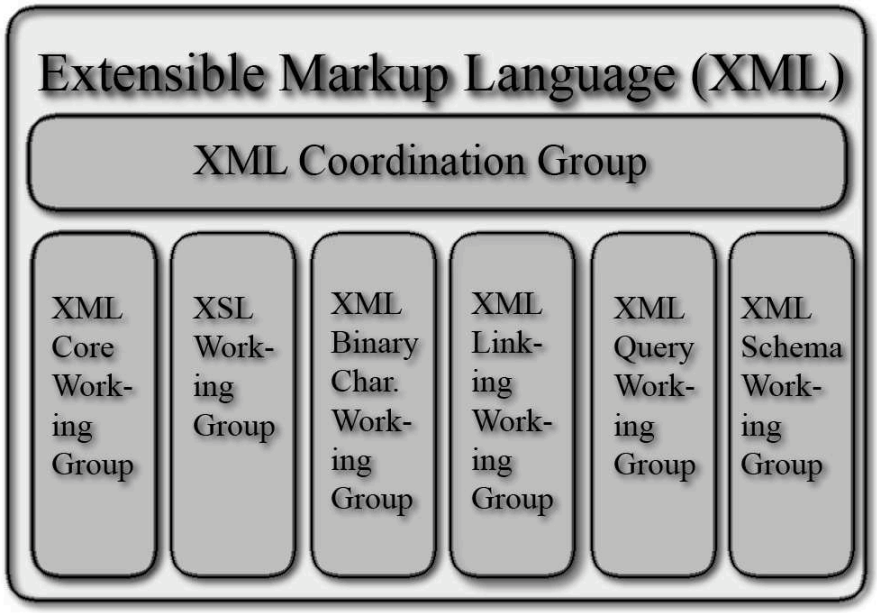


Abbildung 1.1: Aufbau der XML Arbeitsgruppen

Schema. Sie entwickelt Mechanismen zur Definition und Beschreibung der Struktur und des Inhaltes. Damit stehen dem Entwickler mehr als die von SGML geerbte DTD Sprache (Data Type Definition) zur Verfügung. Neben den vom W3C entwickelten XML Schema Sprachen DTD und XML-Schema gibt es auch bereits eine Reihe weiterer, wie z.B. Relax NG und Schematron.

Eine Schlüsselrolle für die Verarbeitung der XML Dokumente wird die Arbeitgruppe *XML Query* haben. Sie arbeitet an einer Anfragesprache, welche auch Updatefunktionalität beinhalten wird.

Die Einfachheit, Erweiterbarkeit, Offenheit der Sprache XML sowie die Akzeptanz der führenden Software Firmen haben mit Sicherheit einen großen Anteil an dem Erfolg der Sprache gehabt.

Geschichtliche Entwicklung von XML

Die Entstehung von XML geht auf das Jahr 1996 zurück. Zu dieser Zeit hatte sich HTML zum erfolgreichsten elektronischen Dokumentenformat entwickelt. HTML selber ist eine durch SGML definierte Auszeichnungssprache, welche seine Stärken in der Darstellung und Verknüpfung von Dokumenten hat. Jede bedeutsame Organisation zeigte großes Interesse, HTML nach seinen Bedürfnissen zu erweitern. Jedoch wäre gerade damit die Einfachheit und

Abbildung 1.2: Beziehungen der Sprachen

Integration, welche HTML so erfolgreich machte, in Gefahr gewesen. Aus diesem Grund startete das W3C im Mai 1996 die Entwicklung der Metasprache XML. Die Idee war es, jede durch SGML definierte Sprache über das Web übertragen zu können und somit HTML als eine von vielen Sprachen anzusehen. Die Wurzeln von SGML gehen auf das Ende der 60iger Jahre zurück. Damals entwickelte Charles Goldfarb eine Metasprache unter dem Namen GML (Generalized Markup Language). 1974 begann Goldfarb die Sprache unter dem Namen SGML weiterzuentwickeln, bis sie schließlich als Standard von der ISO 1986 verabschiedet wurde. SGML selber wurde entworfen, um komplexe und vor allem technische Dokumente darstellen zu können. Für das Web waren die Anforderungen jedoch verschieden. Die Aspekte der Datenorientiertheit und Einfachheit standen im Vordergrund. Die Komplexität von SGML wurde verringert, indem in der Praxis selten genutzte Bestandteile entfernt wurden. Außerdem wurde die Strukturierung der Daten einfacher und strenger. Die daraus hervorgegangene Sprache ist eine vereinfachte Teilmenge von SGML. Sie wurde das erste Mal im November 1996 als Entwurf unter dem Namen *XML-lang* veröffentlicht. Die Beziehungen der Sprachen werden in der Abbildung 1.2 nochmals verdeutlicht.

Folgend sind die bedeutendsten Zeitpunkte der Entwicklung von *XML* aus [W3C00], [W3C98] und [Gol96] zusammengefasst:

- **1969 -** Charles Goldfarb, Ed Mosher und Ray Lorie entwickelten bei IBM die Sprache GML (Generalized Markup Language)
- **1980 -** Erster Entwurf von SGML wurde von der ANSI veröffentlicht
- **1986 -** SGML wurde als Standard von der ISO verabschiedet
- **1989-90 -** Tim Bernes-Lee schlug ein SGML-basiertes Hypertext System vor und empfahl das von ihm entwickelte HTML, HTTP und URL

- **Oktober 1994 -** Das W3C (World Wide Web Consortium) wurde gegründet
- **November 1995 -** HTML 2.0 (Hypertext Markup Language) wurde verabschiedet
- **Juni 1996 -** Die *Generic SGML Activity* wurde gestartet
- **August 1996 -** Das *SGML Editorial Review Board* und die *SGML Working Group* wurden offiziell gegründet
- **November 1996 -** Erster Entwurf von *XML-lang* wird auf der *SGML '96* in Boston vorgestellt
- **April 1997 -** Erster Entwurf (Working Draft) von *XML-link* und zweiter Entwurf von *XML-lang* wird veröffentlicht
- **Juli 1997 -** *XML-lang* wird zu *XML* und *XML-link* zu *XLL* umbenannt
- **Juli 1997 -** Aus dem *SGML Editorial Review Board* und die *SGML Working Group* entstanden die *XML Working Group* und die *XML Sepcial Interest Group*, welche durch das W3C Konsortium organisiert werden
- **Dezember 1997 -** *XML* 1.0 wird auf der *SGML/XML '97 Konferenz* als Empfehlung freigegeben
- **Februar 1998 -** *XML* 1.0 wird zu einer Empfehlung
- **Januar 2000 -** *XHTML* 1.0 wird zu einer Empfehlung

Ziel und Aufbau der Arbeit

Derzeitig hat die Anfragesprache *XQuery* noch den Status eines Entwurfs. Auch wird in der ersten Empfehlung der Anfragesprache die Updatefunktionalität fehlen. Es gibt jedoch schon eine Reihe von Vorschlägen, wie z.B. die von [Leh01], die *XQuery* um Updatefunktionalität erweitern. Schwerpunkt dieser Arbeit ist es zu untersuchen, welche Auswirkungen die Updateoperationen auf die Gültigkeit des XML Dokumentes haben. Ein XML Dokument ist gültig, wenn es der Struktur, welche durch ein zugehöriges XML-Schema vorgegeben ist, entspricht. Falls die Struktur verletzt wurde wird gezeigt,

welche Schemaevolutionsschritte abgeleitet werden können, um die Struktur anzupassen und damit das Update zu ermöglichen.

Der Aufbau der Arbeit stellt sich wie folgt da:

- **Kapitel 1**
  Da es zum Entstehungszeitpunkt der Arbeit noch keine standardisierte Updatesprache gab, widmen wir uns im ersten Kapitel den existierenden Ansätzen und vergleichen diese.

- **Kapitel 2**
  Im zweiten Kapitel wird auf eine aus dem ersten Kapitel ausgewählte Sprache näher eingegangen. Dazu werden auch Syntax und Semantik genau beschrieben.

- **Kapitel 3**
  Da es eine Menge von XML Schemasprachen gibt, wird im dritten Kapitel ein kurzer Überblick über die Bekanntesten gegeben. Es wird dabei eine Schemasprache näher erläutert. Weiterhin wird sich der Evolution gewidmet. Die Bedeutung der Evolution wird verdeutlicht.

- **Kapitel 4**
  Im vierten Kapitel werden die Auswirkungen auf die Struktur der Updateoperationen untersucht. Wenn möglich werden Schemaevolutionsschritte abgeleitet.

- **Kapitel 5**
  Das fünfte Kapitel beschreibt die prototypische Umsetzung und zeigt die daraufhin gewonnenen Ergebnisse.

- **Kapitel 6**
  Das sechste Kapitel gibt einen Überblick über Verwandte Arbeiten.

- **Kapitel 7**
  Das siebente Kapitel dient der Zusammenfassung. Außerdem wird ein Ausblick gegeben und weitere Ideen zusammengetragen.

Kapitel 2

XML-Updatesprachen

Derzeit gibt es noch keine standardisierte Datenmanipulationssprache für XML. Aus diesem Grund wird in diesem Kapitel ein Überblick über die existierenden Ansätze gegeben. Dazu werden im ersten Abschnitt Anforderungen zusammengetragen, die für eine XML-spezifische Updatesprache von Bedeutung sind. Weiterhin wird auf ausgewählte Sprachansätze eingegangen. Abschließend werden die existierenden Ansätze anhand ausgewählter Anforderungen verglichen.

2.1 Anforderungen an eine XML-Updatesprache

In diesem Abschnitt werden die Anforderungen an XML-spezifische Datenmanipulationssprachen (DML) zusammengetragen.

Datenmanipulationssprachen bezeichnen im Allgemeinen Anfragesprachen, welche um Änderungsoperationen erweitert wurden. Der eigentliche Kern, die Anfragesprache, wird dabei nicht zur Datenextraktion, sondern zur Selektion der zu ändernden Daten verwendet. Ein Teil der Anforderungen kann somit aus den Anforderungen an Anfragesprachen abgeleitet werden.

Die folgenden beiden Abschnitte unterteilen die Anforderungen in allgemeine und XML-spezifische.

2.1.1 Allgemeine Anforderungen

Allgemeine Anforderungen können teilweise aus den Forderungen an Anfragesprachen aus dem Datenbankbereich abgeleitet werden. Folgende Kriterien wurden in [HS00] für (relationale) Anfragesprachen vorgestellt. Sie können weitestgehend übernommen werden.

- **Ad-hoc-Formulierung** - Anfragen sollen ohne das Schreiben eines kompletten Programms formuliert werden können.
- **Deskriptivität** - Die Anfrage beschreibt, was in dem Ergebnis stehen soll, jedoch nicht wie dieses zustande kommt.
- **Abgeschlossenheit** - Jedes Ergebnis kann als Eingabe einer weiteren Anfrage benutzt werden. Für XML bedeutet dies, dass das Ergebnis der Anfrage eine Kollektion bestehend aus wohlgeformten XML-Dokumenten ist.
- **Adäquatheit** - Alle Konstruktoren des zugehörigen Datenmodells werden unterstützt.
- **Orthogonalität** - Alle Anfragekonstrukte sind beliebig miteinander kombinierbar.
- **Optimierbarkeit** - Für die Menge der Grundoperationen gibt es Optimierungsregeln, welche die Ausführungszeit verbessern.
- **Sicherheit** - Keine Anfrage darf unendlich viel Zeit brauchen oder ein unendliches Ergebnis liefern.
- **Vollständigkeit** - Die Anfragesprache muss mindestens die Anfragen ausdrücken können, die eine Standardsprache stellen kann.
- **Eingeschränktheit** - Die Anfragesprache darf nicht die Komplexität einer Programmiersprache haben.
- **Effizienz** - Jede Grundoperation ist effizient ausführbar.
- **Mengenorientiertheit** - Jede Operation soll gleichzeitig Mengen von Daten bearbeiten und nicht über einzelne Objekte navigieren.
  Dieses Kriterium muss für XML-spezifische Anfragesprachen eingeschränkt werden.

  Im Vergleich zum Relationalen Modell, wo mit einer Menge von Tupeln gearbeitet wird, besitzt das Datenmodell von XML eine Ordnung. Jedes Element besitzt eine Liste, welche Elemente, Verarbeitungseinheiten, Entitäten, Zeichendaten und Kommentare enthalten kann. Im Abschnitt 1.1.2 wird deshalb das Kriterium der **Ordnungserhaltung** eingeführt.

  Mengenorientiert kann jedoch trotzdem gearbeitet werden. Die Attribute eines Elementes besitzen keine Ordnung. Auch Kollektionen weisen

auf eine Menge von XML-Dokumenten. Wenn die Anwendung nicht auf den Erhalt der Ordnung angewiesen ist, können die Listen der Elemente als Mengen verarbeitet werden.

Folgende Kriterien spielen bei der Datenmanipulation eine Rolle.

- **Transaktionsverwaltung** - Transaktionen sind im Mehrbenutzerbetrieb unerlässlich. Dabei sollte jede Transaktion folgende ACID-Eigenschaften erfüllen.
  - **Atomar -** Die Transaktion wird ganz oder gar nicht ausgeführt. Bei einem Abbruch werden keine Zwischenzustände hinterlassen.
  - **Konsistenz -** Die Integritätsbedingungen müssen auch nach der Ausführung der Transaktion erfüllt sein.
  - **Isolation -** Alle Transaktionen laufen isoliert voneinander ab.
  - **Dauerhaftigkeit -** Das Ergebnis einer erfolgreich ausgeführten Transaktion ist dauerhaft.

2.1.2 XML-spezifische Anforderungen

In diesem Abschnitt werden Kriterien zusammengetragen, die für eine XML-spezifische Updatesprache von Bedeutung sind.

Die wichtigsten Kriterien aus [CR05] und [Mar00] sind:

- **Syntaxform -** Es *kann* mehr als eine Syntaxform für die Updatesprache geben. Davon *muss* mindestens eine Syntaxform für den Benutzer leicht verständlich sein. Außerdem *muss* eine Syntaxform in XML gehalten sein, welche die Struktur der Operationen widerspiegelt.
- **Protokollunabhängigkeit -** Die Updatesprache *muss* unabhängig von allen benutzten Protokollen definiert sein.
- **Fehlerzustände -** Die Updatesprache *muss* für jeden Fehler, der während der Ausführung auftreten kann, einen Fehlerzustand besitzen.
- **Statische Typüberprüfung -** Die Updatesprache *sollte* statische Typüberprüfung ermöglichen.
- **Datenmodell -** Nach der Definition vom W3C *muss* die Spracherweiterung XQuery Update das gleiche Datenmodell wie die Anfragesprache XQuery besitzen.

Folgende Kriterien beschreiben die Funktionalität der Updatesprache.

- **Löschen -** Es *muss* möglich sein, Knoten zu löschen.
- **Einfügen -** Es *muss* möglich sein, neue Knoten an bestimmten Positionen einzufügen.
- **Ersetzen -** Es *muss* möglich sein, einen bestimmten Knoten zu ersetzen.
- **Änderung der Werte -** Es *muss* möglich sein, den Wert eines selektierten Knotens auf Basis des Datentyps zu verändern.
- **Änderung der Eigenschaften -** Es *sollte* möglich sein, die Eigenschaften eines Knotens zu verändern. Zu diesen Eigenschaften gehören der Name, Type, Inhaltstyp, Nullwertfähigkeit, etc.
- **Verschieben von Knoten -** Es *sollte* möglich sein, einen bestimmten Knoten zu einer anderen Position zu verschieben.
- **Bedingte Updates -** Die Datenmanipulationssprache *muss* die Möglichkeit bieten, die Ausführung von Updates an bestimmte Bedingungen zu knüpfen.
- **Iterative Updates -** Es *muss* möglich sein, über Knoten zu iterieren, um Update auszuführen.
- **Validierung -** Um die Gültigkeit des Knoten zu sichern, *sollte* die Spracherweiterung eine Operation zur expliziten Validierung besitzen.
- **Komposition -** Es *muss* möglich sein, Updateoperationen miteinander zu kombinieren. Für jeden XQuery-Ausdruck *sollte* es möglich sein, eine Updateoperation zu benutzen.
- **Ziel** - Die XML Updatesprache *muss* es erlauben, ein Teildokument, das gesamte Dokument und eine Menge von Dokumenten zu bearbeiten.
- **Anfrage** - Die Spezifikation *muss* eine einfache Anfragefunktionalität bieten.
- **Logische Operatoren** - Die Spezifikation *muss* eine Menge von logischen Operatoren enthalten. Dabei *sollten* die Operatoren 'and', 'or' und 'not' nicht fehlen.

- **Bibliotheken** - Die Spezifikation *sollte* eine minimale Menge von standardisierten Funktionsbibliotheken enthalten, z.B. mit mathematische Funktionen oder Funktionen zur Datumsmanipulation.

Folgende Kriterien wurden in der genannten Literatur nicht betrachtet und sollten noch ergänzt werden.

- **Kollektionen** - Eine XML-Kollektion besteht aus einer Menge von Verweisen auf XML-Dokumente. Die Anfragesprache sollte die Möglichkeit bieten, Kollektionen zu bilden, auf die Anfrage- und Änderungsoperationen ausgeführt werden können.

- **Ordnungserhaltung** - Die durch das XML-Dokument gegebene Ordnung muss durch die Ausführung jeder Änderungsoperation erhalten bleiben. Das Ergebnis jeder Anfrageoperation sollte die Ordnung des XML-Dokumentes widerspiegeln. Aus Effizienzgründen sollte es die Möglichkeit geben auf die Ordnungserhaltung zu verzichten, wenn es die Anwendung erlaubt.

2.2 Existierende Updatesprachen

2.2.1 DOM

Das vom W3C entwickelte *Document Object Model* (DOM)[DOM04] ist neben der *Simple API for XML* (SAX)[Meg98] die am meisten benutzte Programmierschnittstelle zur Verarbeitung von XML-Dokumenten. Die aktuelle Version DOM Level 3 hat seit April 2004 den Status einer Empfehlung. Zur Definition der Schnittstelle wurde die von der *Object Management Group* (OMG) entwickelten Sprache IDL (Interface Definition Language) benutzt. Das macht DOM zu einer plattform- und programmiersprachenunabhängigen Schnittstelle.

Im Gegensatz zu SAX, welches eine ereignisorientierte Verarbeitung der Dokumente benutzt, handelt es sich bei DOM um eine Projektion des XML-Datenmodells [CT04]. Dabei wird jedes XML-Dokument beim Parsen in einen hierarchischen Baum überführt. Auf dieser Baumstruktur kann dann mit Hilfe verschiedener Schnittstellen wie z.B. XPath[XPA99] navigiert werden. Neben der Navigation und dem Auslesen der Informationen bietet DOM auch einen ausgereiften Mechanismus zur Datenmanipulation. Dabei können neue Dokumente erstellt oder existierende beliebig manipuliert werden.

Weiterhin bietet DOM die Unterstützung zur Validierung der Dokumente gegen DTDs (Data Type Definition) und XML-Schemata[SCH04b]. Doku-

mente können vor dem Manipulieren überprüft werden, ob das zugehörige Schema die Änderung zulässt.

Wie anfangs beschrieben ist DOM keine echte Updatesprache, sondern eine Programmierschnittstelle. Jedoch gab die weite Entwicklung und Verbreitung den Ausschlag dieses Modell mit in dem Vergleich aufzunehmen.

In den folgenden beiden Abschnitten wird ein kurzer Überblick über die geschichtliche Entwicklung und den Bestandteilen von DOM gegeben.

Geschichte

Die Entwicklung von DOM geht auf das Jahr 1995 zurück. Damals entwickelte Netscape die Sprache Javascript, um statische HTML-Seiten mit dynamischen Komponenten zu erweitern. DOM war dabei ein Bestandteil von Javascript, welcher es ermöglichte, auf Teile des HTML-Dokumentes zuzugreifen. Microsoft^{TM}kopierte anfangs noch die Version von Netscape, begann jedoch später eine eigene Version zu entwickeln. Dies führte schließlich dazu, dass die verschiedenen Browser inkompatibel bei der Verarbeitung von dynamischen Seiten waren. Das W3C begann die Entwicklung einer neuen DOM Version. Das Ziel war es, die Verarbeitung von XML für das Web zu ermöglichen. Microsoft^{TM}und Netscape wirkten bedeutsam bei der Standardisierung mit. Im Oktober 1998 wurde DOM Level 1 [DOM98] zu einer Empfehlung vom W3C. Die Empfehlung bestand aus 2 Komponenten, dem DOM Kern (engl. Core) und DOM HTML. Der DOM Kern stellt die Basis dar, welche die Abbildung der XML-Dokumente auf eine baumartige Struktur beschreibt. Außerdem wird beschrieben wie Teile des Baums gelesen und geschrieben werden können. DOM HTML erweitert den DOM Kern mit Funktionen und Eigenschaften, um die Kompatibilität der von Javascript bekannten DOM Version zu erhalten.

Überblick der verschiedenen Bestandteile

- **DOM Kern (Core)** - beschreibt die Abbildung des XML-Dokumentes und definiert eine Schnittstelle zum Auslesen und Ändern der Daten.

- **DOM Sichten (Views)** - beschreibt eine Schnittstelle, welche es ermöglicht, auf verschiedene Sichten eines Dokumentes zuzugreifen. Zum Beispiel vor und nach der Verarbeitung einer CSS (Cascading Style Sheet) Stilvorlage.

- **DOM Ereignisse (Events)** - beschreibt eine Schnittstelle zur Definition eigener Eventhändlern. Auf Ereignisse wie z.B. Tastatureingaben,

Mausbewegungen oder die Änderung der Dokumentstruktur kann somit reagiert werden.

- **DOM Stilvorlagen (Stylesheets)** - beschreibt ein Interface zur Verarbeitung von Stilvorlagen (Sylesheets), wie z.B. CSS (Cascading Style Sheets).
- **DOM Durchlauf und Auswahl (Traversal and Range)** - beschreibt eine Schnittstelle, die es ermöglicht anhand definierter Filter die Dokumentstruktur zu durchlaufen.
- **DOM HTML** - beschreibt eine Schnittstelle mit HTML-spezifischen Funktionen und Eigenschaften.
- **DOM Laden und Speichern (Load and Save)** - beschreibt eine Schnittstelle zum einheitlichen Laden und Speichern der Dokumente.
- **DOM Validierung (Validation)** - beschreibt ein Schnittstelle mit deren Hilfe abgefragt werden kann, ob durch die Ausführung der Änderungsoperation die Gültigkeit des Dokumentes erhalten bleibt.

Updateoperationen

- *Node* **appendChild**(*in Node newChild*);
- *Node* **insertBefore**(*in Node newChild, in Node refChild*);
- *Node* **removeChild**(*in Node oldChild*);
- *Node* **replaceChild**(*in Node newChild, in Node oldChild*);
- *Node* **cloneNode**(*in boolean deep*);

Die Klasse *Element* ist eine Unterklasse der Klasse *Node* und erlaubt die Manipulation der Attribute.

- *void* **setAttribute**(*in DOMString name, in DOMString value*);
  void **setAttributeNS**(*in DOMString namespaceURI, in DOMString qualifiedName, in DOMString value*);
- *Attr* **setAttributeNode**(*in Attr newAttr*);
  Attr **setAttributeNodeNS**(*in Attr newAttr*);
- *void* **setIdAttribute**(*in DOMString name, in boolean isId*);
  void **setIdAttributeNS**(*in DOMString namespaceURI, in DOMString localName, in boolean isId*);

- *void* **setIdAttributeNode(***in Attr idAttr, in boolean isId***)**;
- *void* **removeAttribute(***in DOMString name*);
  void **removeAttributeNS(***in DOMString namespaceURI, in DOMString localName*);
- *Attr* **removeAttributeNode(***in Attr oldAttr*);

2.2.2 XUpdate

XUpdate ist ein von der *XML:DB Initiative for XML Databases* veröffentlichter Entwurf (engl. Working Draft) für eine Updatesprache. Unverkennbar auf den Sprachentwurf ist der Einfluss der Standards XSLT[Cla99] und XPath[XPA99]. Die Syntax der Sprache ist in XML gehalten, welches zwar die Verarbeitung erleichtert, es aber für den Menschen schwierig macht, Anweisungen zu schreiben. Zur Selektierung der Knoten wird die Sprache XPath v1.0 eingesetzt. Weiterhin sind gewisse Operationen definiert, mit denen die selektierten Knoten manipuliert werden können. Das zugrunde liegende Datenmodell ist zum jetzigen Zeitpunkt eine Untermenge des XML-Datenmodells.

Aktuelle Implementationen sind z.B. Ozone[1], Xindice[2] und Jaxup[3]. In dem frühen Stadium der Sprachentwicklung ist jedoch fraglich, inwieweit Implementation und Entwurf übereinstimmen.

In den folgenden beiden Abschnitten wird ein Überblick über die geschichtliche Entwicklung gegeben und es wird der Sprachvorschlag näher erklärt.

Geschichte

Die Aufgabe der Erstellung der XML Standards liegt aus traditionaler Sicht beim *World Wide Web Consortium*. Durch die Notwendigkeit einer Sprache zur Datenmanipulation gründeten die Firmen *SMB GmbH*, *the dbXML Group L.L.C* und die *OpenHealth Care Group* bereits im Jahre 2000 die *XML:DB Initiative for XML Databases*[4]. Das Ziel der *XML:DB* ist es eine Plattform für die gemeinsame Entwicklung von Standards für XML Datenbanken, sowie Technologien für die Datenmanipulation zu schaffen. Die Mitgliedschaft ist kostenfrei und zählt bereits mehr als 20 Firmen aus der Industrie. Die Organisation erhofft sich eine schnellere Entwicklung, sowie

[1] http://www.ozone-db.org
[2] http://www.xindice.org
[3] http://sourceforge.net/projects/jaxup
[4] http://xmldb-org.sourceforge.net

größere Akzeptanz der Vorschläge. Weiterhin ist geplant, diese Vorschläge an eine Internationale Organisation zur Standardisierung zu übermitteln.

Die Sprache XUpdate ist ein Vorschlag der *XML:DB Initiative for XML Databases*. Der Vorschlag hat den Status eines Entwurfes (engl. *Working Draft*) und die letzte Veröffentlichung gab es im August 2000.

Sprachvorschlag

In diesem Abschnitt wird der Entwurf der Sprache XUpdate aus [LM00] dargestellt.

Jede Update Anweisung ist ein nach [XML04] wohlgeformtes XML-Dokument und besteht aus dem Container-Element mit dem Namen *xupdate:modifications*. Dieses Container-Element kann weiterhin folgende Elemente, welche die Operationen widerspiegeln, enthalten.

- **insert-before**, **insert-after** und **append** - Fügt einen neuen Element-, Attribut-, Text-, PI- oder Kommentarknoten ein. Anhand der drei Operationen wird die Position unterschieden, an die der neue Knoten eingefügt wird.
- **update** - Verändert die selektierten Knoteninhalte.
- **remove** - Löscht die selektierten Knoten.
- **rename** - Benennt die selektierten Knoten um.
- **variable** - Bindet einen Knoten an eine Variable.
- **value-of** und **if** - Wurden im Sprachvorschlag genannt, jedoch nicht genauer spezifiziert.

Jedes dieser Elemente besitzt ein Attribut mit dem Namen *select*. Diese Attribut enthält einen XPath-Ausdruck, welcher die Knoten selektiert.

Fazit ist, dass der Sprachvorschlag sich noch in einem sehr frühen und unvollständigen Stadium befindet. Anforderungen die von der *XML:DB* aufgestellt wurden, werden durch den eigentlichen Sprachvorschlag nicht gehalten.

Folgende Mängel sind dabei aufgefallen:

- Die Semantik existierender Sprachkonstrukte wie z.B. *xupdate:value-of* und *xupdate:if* wurden nicht beschrieben.
- Es fehlen die geforderten Logischen Operatoren und die Funktionsbibliotheken.

- Es fehlt die Möglichkeit der Komposition von Anweisungen beschrieben.
- Der Mechanismus für Bedingte Anweisungen wurde nicht definiert.
- Gefordert war das auch Operationen über eine Menge von Dokumenten möglich ist. Die verwendete XPath Version 1.0 enthält noch kein Mechanismus für Kollektionen, demnach kann nicht mehr als ein Dokument selektiert werden.
- Die Sprache soll konform zu XPointer sein. Jedoch ist XPointer eine Erweiterung von XPath. Da aber maximal XPath zur Selektierung der Knoten benutzt werden kann, fehlt die Unterstützung von XPointer.
- Die Sprache ist nicht adäquat, da die Sprache nur eingeschränkt Konstruktoren für das Datenmodell von XPath besitzt.
- Es fehlt eine Tranksaktionsverwaltung und die Möglichkeit der Validierung gegen XML Schemata.

2.2.3 XQuery

Im Jahre 1998 gründete das W3C die *XML Query Workinggroup*, welche den Auftrag bekam, eine Anfragesprache für XML zu entwickeln. Der erste Entwurf (engl. Working Draft) der XML Anfragesprache *XQuery* erschien im Februar 2001. Die aktuelle Version [XQU05b] wurde im November 2005 veröffentlicht und besitzt den Status *Candidate Recommendation*. Dies bedeutet, dass sich das Dokument in Revision befindet und es eine Empfehlung wird, wenn keine Änderungsvorschläge in diesem Zeitraum gemacht werden.

Im weiteren wird ein Überblick über die wichtigsten Eigenschaften der Sprache gegeben.

XQuery ist eine Weiterentwicklung der XML Anfragesprache Quilt, welche wiederum Ideen und Eigenschaften der Sprachen XPath 1.0[XPA99], XQL[XQL98], XML-QL[XQL98], SQL und OQL enthält.

Sie ist eine funktionale Sprache, welche eine Menge von Ausdrücken besitzt, die beliebig miteinander verschachtelt und kombiniert werden können. Zum Selektieren der gewünschten Teile des XML-Dokumentes wird XPath [XPA05] verwendet. Beide Sprachen XQuery 1.0 und XPath 2.0 besitzen ein gemeinsames Datenmodell[XQU05c], welches auf dem XML-Infosets[CT04] basiert. Dabei wurde das XML-Infoset um folgende Eigenschaften erweitert:

- Die Unterstützung von typisierten atomaren Werten

- Die Unterstützung des durch XML Schema definierten Typsystems
- Die Darstellung von Kollektionen von Dokumenten und komplexen Werten
- Die Unterstützung von geordneten und heterogenen Sequenzen

Das Typsystem von XQuery ist das vom W3C entwickelte XML-Schema [SCH04b]. Neben der im Sprachvorschlag definierten Syntaxform, welche für das Schreiben und Verstehen von Menschen optimiert ist, entwickelt das W3C eine weitere Syntaxform unter dem Namen XQueryX. XQueryX[XQU05a] ist eine XML-basierte Syntaxform und eignet sich gut zur Erstellung und Verarbeitung mit existierenden XML-Prozessoren.

Ein besonderes Merkmal ist die Turingvollständigkeit, deren Beweis in der Arbeit [Kep04] erbracht wurde. Demnach ist es möglich komplexe Anfragen zu schreiben, deren Ausgang oder Dauer nicht endlich sind.

Die FLWOR-Anweisung

Der markanteste Ausdruck verbirgt sich hinter dem Kürzel FLWOR (*flower* gesprochen). Es handelt sich dabei um eine Abkürzung der Wörter FOR, LET, WHERE, ORDER und RETURN.

Mit den Ausdrücken FOR und LET wird eine Variable an eine Menge von Ausdrücken gebunden. Mit Hilfe der WHERE Klausel können bestimmte Tupel mit Hilfe von Prädikaten ausgefiltert werden. Die ORDER Klausel bietet die Möglichkeit, die Ausdrücke zu sortieren. Die RETURN Klausel generiert die Ausgabe der Anweisung. Dabei wird für alle durch die WHERE Klausel gefilterten Titel die RETURN Klausel angewendet.

```
for $var in fn:doc("books.xml")/book
where $var/year > 1999
return $var/name
```

Abbildung 2.1: Beispiel eine FLWOR-Anweisung

Die Anweisung ähnelt sehr der von SQL bekannten SELECT FROM WHERE Anweisung, wobei die Reihenfolge der Konstrukte invertiert betrachtet wird.

Weiterhin bietet XQuery die Möglichkeit, dem Ergebnis neu generierte Elemente hinzuzufügen. Es gibt Möglichkeiten, Anweisungen an Bedingungen zu knüpfen oder eigene Funktionen zu definieren. Für eine genauere Beschreibung wird auf die W3C Dokumente verwiesen.

Updateeigenschaften

Updatefunktionalität ist vorgesehen, wird jedoch noch nicht in der ersten Version von XQuery enthalten sein. Jedoch hat die *XML Query Workinggroup* bereits ein Dokument [CR05] mit Anforderungen für eine XML-spezifische Updatesprache erstellt.

Es gibt bereits mehrere Ansätze XQuery um Updatefunktionalität zu erweitern. Der Vorschlag aus [Leh01] wird im zweiten Kapitel näher vorgestellt.

Es existieren auch bereits Systeme, deren XQuery Implementierungen Updatefunktionalität besitzen, wie z.B. der *Microsoft SQL Server 2005*[5] oder *eXist*[6]. Jedoch ist fraglich, inwieweit dies bereits dem zukünftigen Standard entspricht.

Überblick

Die folgenden Aufzählung gibt einen Überblick über die wichtigsten Spezifikationen, die von der *XML Query Workingroup* entwickelt werden.

- **XQuery 1.0: An XML Query Language -** Das Dokument [XQU05b] beschreibt die Syntax der Anfragesprache.
- **XQuery 1.0 and XPath 2.0 Data Model -** In [XQU05c] wird das für XQuery 1.0 und XPath 2.0 zugrunde liegende Datenmodell definiert.
- **XML Syntax for XQuery 1.0 (XQueryX) -** Dieses Dokument [XQU05a] definiert eine XML-basierte Syntax für XQuery und beschreibt deren Abbildung.
- **XQuery 1.0 and XPath 2.0 Formal Semantics -** Dieses Dokument [XQU05d] wurde anfangs als *XML Query Algebra* betitelt und beschreibt die formale Semantik von XQuery und XPath.
- **XQuery 1.0 and XPath 2.0 Functions and Operators -** Dieses Dokument [XQU05f] definiert die in XPath, XQuery und XSLT enthaltenen Funktionen und Operatoren.
- **XML Path Language (XPath) 2.0 -** Dieses Dokument definiert die Navigationssprache XPath[XPA05].
- **XQuery 1.0 and XPath 2.0 Full-Text -** In [XQU05e] wird eine Spracherweiterung und deren formale Semantik zur Volltextsuche in XML definiert.

[5] http://www.microsoft.com/sql
[6] http://exist.sourceforge.net

- **XSLT 2.0 and XQuery 1.0 Serialization -** In dem Dokument [XQU05g] wird beschrieben, wie aus den Ergebnissen von XQuery-Abfragen und XSLT-Tranformationen XML-Dokumente entstehen.

2.2.4 XSL/XSLT

Hinter der Abkürzung XSL (engl. Extensible Sytlesheet Language) verbirgt sich eine Sammlung von Sprachen zur Transformation und Darstellung von XML Dokumenten. Zum jetzigen Zeitpunkt besteht die Sammlung aus XSLT (XSL-Transformation) und XSL-FO (XSL-Formatting Objects).

XSL-FO[XSL01] ist eine Formatierungssprache und definiert ein Vokabular zur Spezifikation der Darstellung. Die aktuelle Version 1.0 ist seit Oktober 2001 eine Empfehlung vom W3C.

XSLT[Cla99] ist eine baumorientierte Transformationssprache und dient zum Restrukturieren von XML-Dokumenten. Die aktuelle Version 2.0 ist seit November 2005 ein Kandidat für eine Empfehlung vom W3C. XSLT besitzt ein Vokabular, das die Transformation eines XML-Dokumentes in ein Zieldokument beschreibt. Das Format des Zieldokumentes ist dabei nicht auf XML beschränkt und kann z.B. HTML, PDF oder Text sein. XSLT unterliegt seit der Version 2.0 dem gleichen Datenmodell[XQU05c] wie auch XPath 2.0 und XQuery 1.0.

Die Regeln zur Transformation der XML-Dokumente werden in XSLT Stilvorlagen (engl. Stylesheets) definiert. Eine Stilvorlage ist ein XML-Dokument, das entweder im Quelldokument eingebettet ist oder separat vorliegt.

Templates

Eine XSLT Stilvorlage besteht aus einer Menge von Schablonen (engl. Templates). Jedes Template besitzt eine Maske (engl. Pattern), welche eine Menge von Knoten des Quelldokumentes selektiert. Der XSLT-Prozessor durchläuft das Quelldokument und wendet auf jeden Knoten, der durch ein Pattern markiert wird, die im Template definierten Regeln an. Das Ergebnis der Transformation wird im Zieldokument gespeichert.

Die Abbildung 2.2 zeigt ein Beispiel für ein XSLT-Stylsheet. Das Ergebnis des Stylesheets ist ein neues XML-Dokument, indem Elemente mit Namen 'Price', die ein Vaterelement mit dem Namen 'Buch' haben, zu 'Preis' umbenannt wurden.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="Buch/Price">
        <Preis>
        <xsl:apply-templates />
        </Preis>
    </xsl:template>
    <xsl:template match="* | text() | attribute()">
    <xsl:copy>
        <xsl:apply-templates select="* | text() | attribute()" />
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Abbildung 2.2: Beispiel eines XSLT-Stylesheets

Updateeigenschaften

XSLT ist eine Transformationssprache zur Restrukturierung der Instanzdokumente. XSLT ist wie XQuery turingvollständig, unterliegt dem gleichen Datenmodell, Typsystem und besitzt den gleichen Funktions- und Operationenumfang. Demzufolge ist XQuery gegenüber XSLT nicht eingeschränkt. Aber ist XSLT eine XML-Updatesprache? Folgende Probleme traten dabei auf.

- **Zieldokument -** Das Ergebnis einer XSL Transformation ist immer ein neues Instanzdokument. Das Quelldokument wird nicht verändert. Eine Änderung der Implementierung des XSLT- Prozessors könnte dieses Problem jedoch lösen.
- **Syntax -** Die Syntax ist in XML und damit schwer für den Menschen schreib- und lesbar.
- **Templates -** Das Schreiben von Templates für Updates ist für den Menschen sehr aufwendig. Wenn z.B. ein Knoten gelöscht werden soll, müssen 2 Templates erstellt werden. Das 1. Template selektiert den zu löschenden Knoten und enthält keine Anweisung. Das 2. Template selektiert alle restlichten Knoten und erzeugt eine Kopie dieser. Updateoperationen wie Delete, Insert, Change und Replace gibt es nicht und müssen kompliziert mit Templateregeln umschrieben werden.

- **Turingvollständigkeit -** Die Ausführungszeit eines Updates ist ungewiss.

2.2.5 Weitere Updatesprachen

Zur Vollständigkeit werden in diesem Abschnitt weitere, nicht genauer betrachtete Updatesprachen kurz vorgestellt.

LOREL

Lorel[AQM+97] ist eine Anfragesprache, die für das Datenbanksystem Lore[7] entwickelt wurde. Das Datenmodell von Lore war ursprünglich das OEM (Object Exchange Model), welches später durch die Nachfrage zu XML geändert wurde. Ihren Ursprung hat Lorel von der Sprache OQL (Object Query language), die ein Bestandteil des ODMG-Datenmodells ist. Lorel hat eine SQL-ähnliche Syntax und erlaubt auch das Manipulieren der Daten.

XML-GL

Grundlage der Sprache XML-GL[XML99] ist eine graphische Repräsentation des XML-Dokumentes. Da es sich um eine graphische Anfragesprache handelt, ist sie gut für benutzerfreundliche Programmoberflächen geeignet. Eine Anfrage besteht aus zwei azyklischen Graphen. Mit dem ersten Graphen wird die Selektion und mit dem zweiten Graphen die Projektion bestimmt.

2.3 Vergleich

In diesem Abschnitt werden die in Kapitel 1 vorgestellten Sprachen verglichen. Anhand der Ergebnisse wird entschieden, welche Anfragesprache als Grundlage der Studienarbeit dient.

Als Grundlage für den Vergleich sollen die vom W3C aufgestellten Anforderungen [CR05] verwendet werden. Es enthält alle Eigenschaften, die ein XQuery Updatesprache besitzen sollte. Dabei wird die Notwendigkeit der Eigenschaft mit Hilfe der Schlüsselwörter 'muss', 'sollte' und 'vielleicht' bestimmt.

Syntaxform Es *sollte* mehr als eine Syntaxform geben. Davon *muss* eine gut von Menschen les- und schreibbar sein. Eine weitere *muss* in Form von XML sein, welche die Struktur der Operationen wiedergibt.

[7] http://www-db.stanford.edu/lore/home/index.html

XQuery genügt als einzige Sprache diesen Anforderungen. Eine menschlesbare Syntax wird in der XQuery-Empfehlung [XQU05b] beschrieben. In [XQU05a] wird eine Syntaxform in XML beschrieben.

DOM Level 3 ist eine Programmierschnittstelle und hat weder eine menschlesbare Sytnax noch eine XML.

Die Syntax der Sprachen XSLT und XUpdate sind in XML-Form. XSLT ist zur Beschreibung von Transformationen entwickelt worden. Atomare Updates mit XSLT zu beschreiben, ist kompliziert und deshalb ist es keine menschenlesbare Sytnax. XUpdate ist eine reine XML-Updatesprache und obwohl die Syntax XML-Form hat, ist sie menschenlesbar.

Datenmodell Das zugrunde liegende Datenmodell *muss* das XQuery 1.0 und XPath 2.0 Datenmodell[XQU05c] sein.

Für die vom W3C entwickelte Sprache XQuery trifft diese Anforderung zu.

DOM ist eine frühere Entwicklung und besitzt ein eigenes Datenmodell, was eine Erweiterung des XML-Infosets[CT04] ist.

Die Entwicklung der Sprache XSLT läuft parallel zur Sprache XQuery. Beide Sprachen unterliegen dem gleichen Datenmodell.

XUpdate ist eine der ersten Entwicklungen einer Updatesprache für XML. Das Datenmodell der Sprache basiert auf eine Untermenge des XML-Infosets. Damit besitzt die Sprache nicht die Eigenschaft der Vollständigkeit.

Operationen Nach den Anforderungen vom W3C muss eine Updatesprache Operationen zum *Löschen*, *Einfügen*,*Ersetzen* und zum Ändern der Werte bieten.

Das W3C hat bisher noch keinen Vorschlag für eine Updateerweiterung für XQuery veröffentlicht.

DOM biete für alle Operationen eine Umsetzung in Form der API.

XSLT ist zur Transformation von XML-Dokumenten entwickelt worden. Alle gewünschten Operationen müssen mit Hilfe von Schablonen (engl. Templates) simuliert werden.

XUpdate bietet Operationen zum Löschen, Einfügen und zum Ändern der Werte. Das Ersetzen muss durch die Operationen Löschen und Einfügen simuliert werden.

Transaktion Die Anforderungen an eine Updatesprache umfassen auch Transaktionen. Dabei sind die Eigenschaften Atomar, Konsistent und Dauerhaft ein *muss*. Wenn möglich, *sollen* die Operationen isoliert voneinander

arbeiten. Ein Konzept einer Transaktionsverwaltung bietet keine der vorgestellten Updatesprachen.

Ergebnis

Grundlage dieser Arbeit wird die Updatesprache XQuery Update. Derzeit gibt es noch keinen Vorschlag von dem W3C. Aus diesem Grund wird der Vorschlag aus der Diplomarbeit [Leh01] verwendet.

Die Programmierschnittstelle DOM und die Transformationssprache XSLT werden in der Praxis am häufigsten verwendet, aber eignen sich nicht als XML-Updatesprachen.

Der Sprachvorschlag für XUpdate ist beinahe 6 Jahre alt und noch sehr unvollständig. Die Entwicklung der Sprache scheint eingestellt zu sein.

Kapitel 3

Analyse der ausgewählten Updatesprache

Die Anfragesprache XQuery, welche bereits im Kapitel 1 näher beschrieben wurde, hat zum jetzigen Zeitpunkt noch den Status eines Entwurfs. Es wird noch einige Zeit in Anspruch nehmen, bis es eine Empfehlung wird. Außerdem wird die erste Version nicht die Spracherweiterung zur Manipulation der XML Instanzen enthalten. Aus diesen Gründen liegt dieser Arbeit der Vorschlag aus [Leh01] zugrunde.

Dieses Kapitel beschreibt die Syntax und Semantik der Spracherweiterung.

3.1 Operationen

Jede Updateanweisung beginnt mit dem Schlüsselwort *UPDATE*. Weiterhin folgt eins der Schüsselwörter *INSERT*, *DELETE*, *REPLACE* und *RENAME*, wodurch die Art der Modifikation bestimmt wird. Jeder dieser Ausdrücke identifiziert eine Menge von Knoten, welche unabhängig von der Operation und deren Auswirkung selektiert werden. Auf jeden dieser Knoten wird unabhängig voneinander die Updateoperation angewandt. Weiterhin können mehrere Updateanweisungen in einer Anfrage existieren. Die Reihenfolge der Ausführung ist dabei nicht vorbestimmt.

Anschließend wird auf jede Operation näher eingegangen.

3.1.1 INSERT

Mit Hilfe der Insert-Anweisung wird die Instanz mit neuem Inhalt erweitert. Die Klauseln *INTO*, *PRECEDING* und *FOLLOWING* bestimmen die

Position, wo der neue Inhalt eingefügt wird.

Die Ausdrücke *INSERT PRECEDING* und *INSERT FOLLOWING* unterscheiden sich semantisch nur in einem Punkt und werden deshalb gemeinsam erklärt.

Syntax:
INSERT %1 **(PRECEDING|FOLLOWING)** %2

Parameter:
%1,%2 - Element-, Kommentar-, PI-, Textknoten oder eine Menge dieser

Beispiel: Das Buch bekommt einen neuen zweiten Autor.

```
UPDATE INSERT <author>Name</author>
FOLLOWING //book[@name='XML']/author[1]
```

Das Ergebnis dieser Anweisung ist, dass alle durch %1 bestimmten Knoten kopiert und als Schwesterknoten bevor bzw. nach dem durch %2 identifizierten Knoten eingefügt werden. Falls durch %2 das Root-Element selektiert wurde, darf %1 nur Kommentar- und PI-Knoten beinhalten, da in jeder XML-Instanz die Existenz maximal eines Root-Elementes erlaubt ist.

Syntax:
INSERT %1 **INTO** %2

Parameter:
%1 - Element-, Attribute-, Kommentar-, PI-, Textknoten oder eine Menge dieser
%2 - Element-, Dokumentknoten oder eine Menge dieser

Beispiel: Das Element Buch bekommt ein Tochterelement, welches den Preis angibt.

```
UPDATE INSERT <price>$20</price> INTO //book[@title='XML']
```

Falls der Ausdruck *INSERT INTO* verwendet wird, werden alle durch %1 bestimmten Knoten kopiert und als Tochterknoten in die durch %2 identifizierten Knoten eingefügt. Dabei werden die neuen Knoten immer am Ende der Liste angefügt.

Syntax:
INSERT ATTRIBUTE %1 **INTO** %2

Parameter:
%1 - Attributknoten oder eine Menge dieser
%2 - Element-, Dokumentknoten oder eine Menge dieser

Beispiel: Das Element Buch bekommt ein neues Attribut 'year' mit dem Inhalt 2004.

```
UPDATE INSERT ATTRIBUTE @year{2004} INTO //book[@title='XML']
```

Um ein neues Attribut hinzuzufügen, wird der Ausdruck *INSERT ATTRIBUTE INTO* verwendet. Hierbei werden die durch %1 selektieren Knoten mit dem neuen Attribut erweitert.

3.1.2 DELETE

Mit Hilfe der Delete-Anweisung werden Inhalte aus der XML Instanz entfernt.

Syntax:
DELETE %1

Parameter:
%1 - Element-, Attribut-, Kommentar-, PI-, Text-, Dokumentknoten oder eine Menge dieser

Beispiel: Das Tochterelement, welches den Preis angibt, wird aus dem Buch entfernt.

```
DELETE //book[@title='SGML']/price
```

Die Delete-Anweisung besitzt genau einen Ausdruck. Durch diesen kann jede Art von Knoten selektiert werden. Durch die Ausführung werden alle selektieren Knoten einschließlich ihrer Unterknoten aus der Instanz entfernt. Eine Ausnahme gibt es jedoch. Wenn durch den Ausdruck der Wurzelknoten des Dokumentes selektiert wird, darf die Anweisung nicht ausgeführt werden, da in [XML04] für die Wohlgeformtheit die Existenz genau eines Wurzelelementes gefordert wird. Falls der Dokumentknoten selektiert ist, wird die gesamte Instanz aus der Kollektion entfernt.

3.1.3 REPLACE

Die Replace-Anweisung dient dem Ersetzen bestimmter Knoten. Semantisch ist diese Anweisung eine Kombination einer Insert- und Delete-Anweisung. Wenn durch die Anweisung ein Attributknoten selektiert wird, wäre es ein *Delete* gefolgt von einem *Insert Into*. Für alle anderen Knotentypen wird zuerst ein *Insert Preceding* gefolgt von einem *Delete* ausgeführt.

Syntax:
REPLACE %1 **WITH** %2

Parameter:
%1,%2 - Element-, Attribut-, Kommentar-, PI-, Textknoten oder eine Menge dieser

Beispiel: Das XML Buch wird durch das SGML Buch ersetzt.

```
UPDATE
REPLACE /me/rental/book[@title='XML'] with <book title='SGML'/>
```

Besonderheiten:

Attributknoten können nur durch Attribute ersetzt werden. Außerdem muss geprüft werden, ob die Menge der Attribute eindeutig ist, da in einem Element nach [XML04] jedes Attribut genau einmal vorkommen darf. Das Wurzelelement kann nur durch genau ein Element und nicht eine Menge von Elementen ersetzt werden.

3.1.4 RENAME

Durch die Rename-Anweisung werden alle selektierten Knoten umbenannt.

Syntax:
RENAME %1 **AS** %2

Parameter:
%1 - Element-, Attribut-, Dokumentknoten oder eine Menge dieser
%2 - Textknoten

Beispiel: Der Attributname 'price' wird zu 'Preis' geändert.

```
UPDATE RENAME /Buch[@Name='XML']/@price AS Preis
```

Es können Element-, Attribut- oder Dokumentknoten selektiert werden. Durch die Ausführung wird der Name aller selektierten Knoten mit Ausdruck %2 umbenannt. Hier ist wieder darauf zu achten, dass nicht Elemente entstehen, die zwei Attribute gleichen Namens besitzen.

3.1.5 Bedingte Updates

Bedingte Updates werden durch die Anweisung *IF/THEN/ELSE* beschrieben. Anders als wie bei der Sprache XQuery ist hier die Angabe des *ELSE* Ausdruckes optional.

Syntax:
UPDATE IF Bedingung THEN Anweisung1 (ELSE Anweisung2)

Beispiel:

```
UPDATE IF (//arbeiter/gehalt < 1000) THEN
REPLACE //arbeiter/gehalt WITH <gehalt>1000</gehalt>
```

3.1.6 FLW-Update Anweisung

Die Funktionsweise der FLW-Update Anweisung ähnelt der FLWOR Anweisung der Sprache XQuery. Die 'OrderBy' Klausel, welche der Sortierung der Ergebnismenge dient, wurde entfernt. Außerdem wurde die 'Return' Klausel durch die Angabe der Update-Anweisung ersetzt.

Syntax:

UPDATE FOR expr1 LET expr2 WHERE expr3 UPDATE-Operation

Beispiel: Jede Abteilung mit mehr als 20 Angestellen bekommt
2 Praktikanten.

```
UPDATE
FOR $d IN /abteilung
LET $e := /angestellte/angestellter[abteilung_nr = $d/nr]
WHERE fn:count($e) > 20
REPLACE $d/praktikanten WITH <praktikanten>2</praktikanten>
```

Die FLW-Update Anweisung selektiert mit Hilfe der 'FOR' Schleife alle Knoten, welche die mit der 'WHERE' Klausel angegeben Bedingung erfüllen. Auf allen selektieren Knoten wird dann die angegebene Update-Anweisung ausgeführt.

Kapitel 4

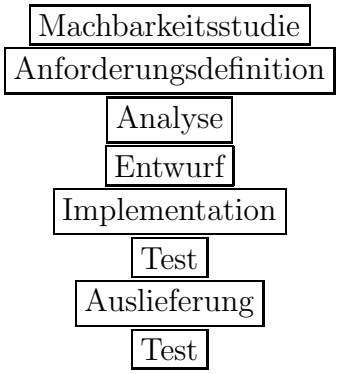
Schemasprachen und -evolution

Schwerpunkt dieses Kapitels sind Schemasprachen und -evolution. Dazu wird kurz die Bedeutung der Evolution erklärt und der Bezug auf die semi- und strukturierten Daten gezeigt. Da es für XML mehrere Schemasprachen gibt, wird eine ausgewählt und näher beschrieben. Das der Sprache zugrunde liegende Datenmodell wird vorgestellt.

4.1 Einführung

Definition(im Rahmen der Evolutionslehre): **Evolution** (aus dem Lateinischen evolvere abwickeln, entwickeln) ist das fortlaufende Entstehen neuer und das Wachsen bereits entstandener Muster in Richtung aufsteigende Komplexität und Vernetzung von Bereichen der Wirklichkeit.[WIK05]

Dieser Begriff lässt sich gut auf den Bereich der Verarbeitung semi- und strukturierter Daten übertragen. Evolution steht in diesem Zusammenhang für die Anpassung der Struktur der Daten, welche durch ein Schema vorgegeben wird.



Wasserfallmodell [Royce1970]

Bezogen auf das Wasserfallmodell wird ein Schema zwischen den ersten vier Phasen definiert. Spätere Änderungen haben Auswirkungen auf die darunter liegenden Phasen. Existierende und die mit Software erstellten oder verarbeiteten Dokumente müssen auch zu dem neuem Schema gültig sein. Demzufolge darf durch eine Schemaevolution die Kapazität des Schemas nur erweitert werden.

Mögliche Gründe für eine Schemaevolution, bzw. die Änderung des Schemas, entstehen z.B. durch geänderte Anforderungen oder Fehler in der Analyse oder Entwurfsphase. Hierzu ein Beispiel für geänderte Anforderungen. Vor nicht all zu langer Zeit, wo das Internet noch in den Kinderschuhen steckte, gab es kaum jemanden der eine E-Mailadresse hatte bzw. wusste was es ist. Demnach gab es auch keine Anforderungen diese zu speichern. Heutzutage dagegen hat beinahe jede Person eine E-Mailadresse und demnach existieren auch die Anforderungen diese zu speichern.

Da wahrscheinlich nie ausgeschlossen werden kann, dass sie die Anforderungen angepasst werden müssen, sollte man die Schemaevolution auch als fortlaufenden Prozess ansehen der weitläufige Veränderungen mit sich bringt.

4.2 XML Schemasprachen

Es existieren zurzeit mehr als 5 XML Schemasprachen. Darunter gehören DTD, XML-Schema, RELAX NG und Schematron zu den Bekanntesten.

4.2.1 DTD

DTD ist die älteste XML Schemasprache. Sie ist aus der Sprache SGML DTD hervorgegangen und ist eine Empfehlung vom W3C. In Zukunft wird sie durch ihren Nachfolger XML-Schema ersetzt werden. Die Syntax ist nicht in XML, was die Erstellung und Verarbeitung der Schemainstanzen erschwert. DTD besitzt ein Datenmodell mit 10 vordefinierten Typdefinitionen und erlaubt es nicht diese zu erweitern. Von dem Vorgänger SGML DTD wurde auch der ID/IDREF-Mechanismus übernommen, der es ermöglicht Schlüssel-/Fremdschlüsselbeziehungen zwischen Elementen darzustellen.

4.2.2 XML-Schema

XML-Schema ist eine Weiterentwicklung von DTD und ist ebenfalls eine Empfehlung vom W3C. Die Syntax ist in XML, welche es ermöglicht die

Instanz mit einem XML-Parser zu verarbeiten. Die Sprache besitzt ein ausgereiftes Datenmodell mit 45 vordefinierten Typdefinitionen und einen Vererbungsmechanismus für Attribut, Element und Typdefinition. Benutzerdefinierte Typdefinitionen können durch Erweiterung, Einschränkung oder Neuerstellung definiert werden. Außerdem bietet XML-Schema einen erweiterten Mechanismus zur Definition von Schlüssel-/Fremdschlüsselbeziehungen. Dabei kann z.B. ein Schlüssel über mehrere Attribut- und Elementwerte definiert sein. Auch der Gültigkeitsbereich der Schlüsselbeziehung kann im Gegensatz zu dem ID/IDREF Mechanismus eingeschränkt werden. Ein weiteres signifikantes Merkmal ist die Trennung von Deklaration und Definition.

4.2.3 RELAX NG

RELAX NG[REL01] ist eine Schemasprache, die auf den Sprachen RELAX (Regular Language Description for XML) von Murata Makotos und TREX von James Clarks basiert. Seit Dezember 2003 ist die Sprache ein Standard der ISO. Ein Vorteil gegenüber XML-Schema ist die vereinfachte und menschenlesbarere Syntax. RELAX NG besitzt kein ausgeprägtes Typsystem wie XML-Schema, kann jedoch auf fremde Datentypbibliotheken wie dem von XML-Schema verweisen. Neben der in XML gefassten Syntax, biete RELAX NG auch eine kompakte DTD-ähnliche Syntax. Derzeit hat RELAX NG eine Außenseiterrolle.

4.2.4 Schematron

Schematron[SCH04a] ist eine Schemasprache, die zur Validierung der Struktur entwickelt wurde. Meist wird die Sprache auch zur Ergänzung anderer XML Schemasprachen angesehen. Anders als das W3C XML-Schema und RELAX NG ähnelt Schemtron der Syntax von XSLT. Ein Schemtron-Schema besteht aus einer Menge von Mustern (engl. pattern). Jedes Muster enthält eine Menge von Regeln (engl. rules). Und jede Regel besitzt einen XPath-Ausdruck, um einen Teil eines XML-Dokumentes zu selektieren. Weiterhin besitzt jede Regel eine Menge von Behauptungen (engl. asserts) und Reports. Die Behauptungen beschreiben bestimmte Zustände. Falls diese nicht zutreffen, erscheint die im Report definierte Nachricht in dem Ergebnis der Validierung. Zur Validierung wird nur ein XSLT-Prozessor benötig. Laut der Website von Schematron wird angestrebt die Sprache zu einem ISO-Standard zu erheben.

4.3 Datenmodell von XML-Schema

Das abstrakte Datenmodell besteht aus einer Menge von Bausteinen, welche im Weiteren als Schemakomponenten bezeichnet werden. Es gibt 13 verschiedene Komponenten, welche in drei Gruppen eingeteilt werden.

- Primäre Komponenten
  - Definitionen einfacher und komplexer Typen
  - Attribut- und Element-Deklarationen
- Sekundäre Komponenten
  - Attributgruppen- und Elementgruppen-Definitionen
  - Identitätsbeschränkungs-Definitionen
  - Notations-Deklarationen
- Hilfskomponenten
  - Attribut-Verwendungen
  - Elementgruppen und Partikel
  - Wildcards
  - Anmerkungen

4.3.1 Primäre Komponenten

Die Abbildung 4.1 verdeutlicht die Beziehung der Primären Komponenten. Sehr deutlich ist die Trennung zwischen Typdefinitionen und den Deklarationen. Das Datenmodell hat 2 verschiedene Typdefinitionen: einfache und komplexe.

Einfache Typdefinition

Zu den einfachen Typen gehören alle atomaren Datentypen, sowie Listen und Unions bestehend aus den einfachen Typen. Mit Unions meinen wir eine nicht-leere Folge von einfachen Datentypen. Einfache Typdefinition können keine Attribute oder Element enthalten.

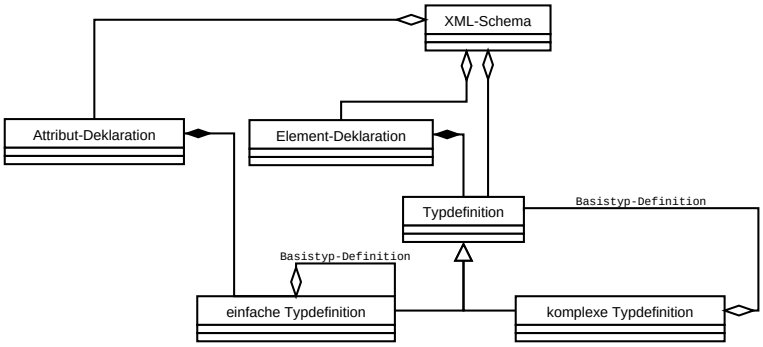


Abbildung 4.1: Darstellung der Primären Komponenten

Komplexe Typdefinition

Komplexe Typendefinition können Attribut-Deklaration besitzen.
Jede Komplexe Typdefinition hat eines der Inhaltsmodelle:

- **Leeren Inhalt**
  Verbietet dem Element jeglichen Inhalt (Elemente, Zeichen).
- **Einfachen Inhalt**
  Ist eine Erweiterung einer einfachen Typdefinition mit Attributen.
- **Element**
  Das Element darf Unterelemente besitzen.
- **Gemischert Inhalt**
  Das Element darf Unterelemente und Textinhalt besitzen.
- **AnyType**
  Das Element darf beliebigen Inhalt haben. Der XML-Parser prüft nur die Wohlgeformtheit des Elementes.

Hierarchie der Typdefinitionen

Das Typsystem besitzt 45 vordefinierte Typen. Es gibt eine Urtyp-Definition *anyType*. Alle anderen Typedefinitionen entstehen durch Einschränkung und Erweiterung.

Element-Deklaration

Eine Element-Deklaration ist eine Verknüpfung von einem Namen mit einer Typdefinition.

Attribut-Deklaration

Eine Attribut-Deklaration ist eine Verknüpfung von einem Namen mit einer einfachen Typdefinition.

4.3.2 Hilfskomponenten

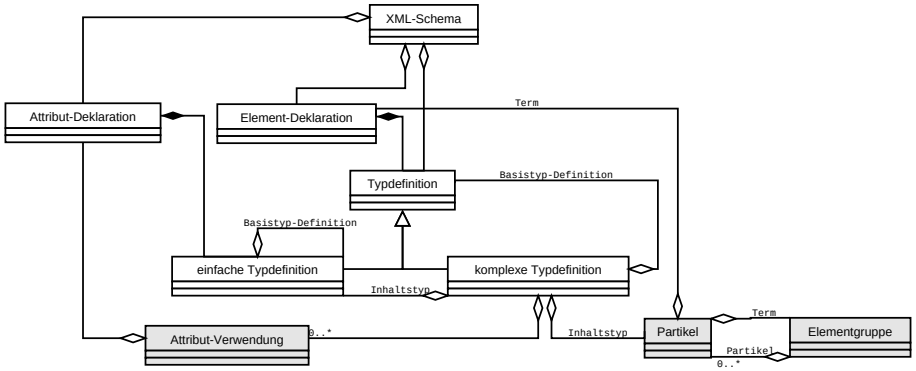


Abbildung 4.2: Darstellung der Hilfskomponenten

Zu der Gruppe der Hilfskomponenten gehören:

Elementgruppe

Partikel

Ein Partikel ist Bestandteil einer komplexen Typdefinition und dient zur Einschränkung der Element-Informationseinheit.

Attribut-Verwendung

Jede Komplexe Typdefinition besitzt für jede Attribut-Deklaration eine Attribut-Verwendung. Die Attribut-Verwendung bestimmt, ob das Attribut optional oder erforderlich ist. Zusätzlich können auch vordefinierte oder feste Attributwerte definiert werden.

Wildcard

Mit Hilfe von Wildcard kann die Form der Validierung von Attributen und Element-Informationseinheiten bestimmt werden.

4.3.3 Sekundäre Komponenten

Zu der Gruppe der sekundären Komponenten gehören:

Attributgruppen-Definition

Mit einer Attributgruppen-Definition wird eine Menge aus Attributen zusammengefasst und einen Namen zugewiesen.

Elementgruppen-Definition

Ähnlich wie bei der Attributgruppen-Definition wird mit einer Elementgruppen-Definition eine Menge aus Elementen zusammengefasst und einen Namen zugewiesen.

Identitätsbeschränkungs-Definitionen

Mit Identitätsbeschränkungsdefinitionen können Schlüssel-/Fremdschlüsselbedingungen definiert werden.

Notations-Deklaration

Notations-Deklarationen dienen nicht der Validierung der XML-Dokumente. Notationen werden benutzt, um zusätzliche Informationen über das Schema dem Menschen oder einer Applikation zur Verfügung zustellen.

4.4 Sprachen zur Schemaevolution

Derzeit gibt es verschiedene Möglichkeiten eine Schemaevolution für XML-Schema zu beschreiben.

XML-Updatesprache

Für XML-Schema gibt es eine Repräsentationsform in XML. Demzufolge liegt es nahe das Schema mit der gleichen Updatesprache, wie auch die XML Instanzdokumente, zu bearbeiten.

Transformation mit XSLT

Eine weitere Möglichkeit ist der Verwendung der Transformationssprache XSLT. In der Arbeit [Zei01] wird XSLT zur Anpassung der Instanzdokumente benutzt.

XSEL

In der Arbeit [Tie05] wird die Schemaevolutionssprache XSEL vorgestellt. Zur Repräsentation wird das XML-Dokument des Schemas auf einen Baum abgebildet. Die Operationen sind beinahe identisch einer XML-Updatesprache. Es gibt Operationen zum Hinzufügen, Löschen, Ersetzen und Löschen von Knoten. Zur Selektion der zu ändernden Knoten wird XPath verwendet.

Zusammenfassung

Leider ist keine der benannten Möglichkeiten zur Beschreibung der Schemaevolutionen geeignet. Ein Problem stellt das Auffinden der Definitionen und Deklarationen dar. Z.B. kann eine Typdefinition in einem Schema global oder auf lokal definiert sein. Es ist außerdem möglich, dass die zu ändernden Eigenschaften von einer anderen Typdefinition geerbt wurden. Die Notwendigkeit erst viele Informationen zu erfragen, um den gewünschten Knoten im Schema-Dokument zu finden, ist unumgänglich.

Der Umfang der Operationen ist unzureichend. Z.B. gibt es keine Operation, um eine einfache Typendefinition in eine komplexe Typdefinition umzuwandeln oder eine lokale Definition global zu definieren.

Eine Schemaevolutionssprache für XML-Schemata sollte das abstrakte Datenmodell von XML-Schema verwenden und nicht die Repräsentationsform in XML. Das Auffinden von Typdefinitionen und Deklarationen sollte anhand des Namens oder anhand der Elemente der Instanz möglich sein. Es sollten Operationen zum Ändern aller Eigenschaften vorhanden sein. Dazu gehört auch das Umformen einer einfachen Typdefinition zu einer komplexen Typdefinition.

Kapitel 5

Ableitung der Schemaevolutionsschritte

In diesem Kapitel werden wir uns den Teil der Update-Operationen näher anschauen, durch welchen die Gültigkeit der Struktur verletzt wird. Anhand der Update-Operation, der Struktur und der XML-Instanz wird untersucht, inwieweit Schemaevolutionsschritte abgeleitet werden können, welche die Struktur anpassen und damit die Gültigkeit der alten und neuen XML-Instanzen bestehen bleibt.

5.1 Updates auf Attributebene

Dieser Abschnitt widmet sich der Manipulation von Attributen.

Nach der Definition des XML-Infosets[CT04] besitzt jedes Element eine Menge mit Attributinformationseinheiten. Attribute mit gleichen Namen sind nicht erlaubt. Neben dem Attributwert besitzt jede Attributinformationseinheit einen Namen und Namensraum zur Identifikation.

Nach der Definition aus [SCH04c] besitzt jedes Attribut eine einfache Typdefinition. Im XML-Schema wird der Deklarations- und Definitionsteil voneinander getrennt definiert. Ein Schema besteht danach aus einer Attributdeklaration, welche mindestens einen Namen hat und eine Referenz auf eine einfache Typdefinition. Um die Attributdeklaration mit einer Elementdeklaration zu verbinden, muss die Elementdeklaration auf einen komplexe Typdefinition weisen. Diese Typdefinition verweist dann auf die Attributdeklaration. Weiterhin gibt es die Möglichkeit Attributdeklaration in eine Gruppe zusammenzufassen. In diesem Fall können alle Attribute mit Hilfe einer Referenz zum komplexen Typen hinzugefügt werden.

In Abbildung 5.1 wird der Zusammenhang zwischen den Element- und

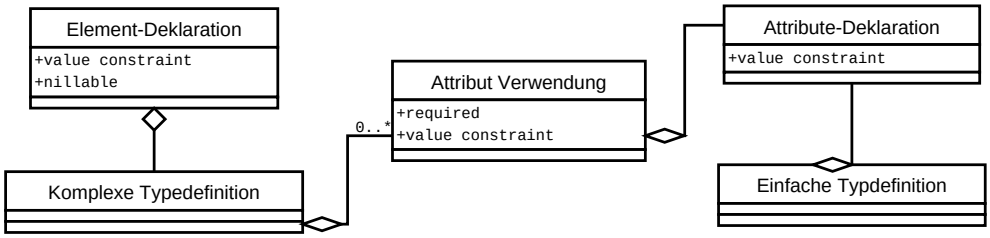


Abbildung 5.1: Beziehung von Element-Deklaration und Attribut-Deklaration

Attribut-Deklarationen nochmals verdeutlicht.

5.1.1 Insert Attribute

Mit Hilfe der Operation *Insert Attribute* wird ein neues Attribut zu einem Element oder eine Menge von Elementen hinzugefügt.

Wohlgeformtheit

Bevor die Anweisung ausgeführt und gegebenenfalls ein Schemaevolutionsschritt generiert werden kann, wird geprüft ob die Anweisung nicht die Wohlgeformtheit der XML Instanz verletzt. Dazu muss sichergestellt werden, dass das Attribut nur zu Elementen hinzugefügt werden kann. Weiterhin muss geprüft werden, dass das Element nicht schon ein Attribut mit dem Namen besitzt.

Validität und Evolutionsschritt

Bevor einem Element ein neues Attribut hinzugefügt wird muss überprüft werden, ob die Element-Deklaration eine passende Attribut-Deklaration besitzt. Dazu muss der Element-Deklaration eine komplexe Typdefinition zugeordnet sein. Dieser Typdefinition ist weiterhin eine Menge von Attribut-Verwendungen zugewiesen, wovon jede auf eine Attribut-Deklaration weist. Passende Attribut-Deklaration bedeutet, dass eine dieser Attribut-Deklaration den Namen des neuen Attributes besitzt und der neue Attributwert der Typdefinition der Attribut-Deklaration entspricht.

Im weiteren Verlauf werden die Fälle untersucht die auftreten können, wenn keine passende Attribut-Deklaration existiert. Weiterhin werden Evolutionsschritte abgeleitet, die das Schema anpassen.

Wenn eine der folgenden Situationen auftritt, existiert keine passende Attribut-Deklaration.

(**I**) Element-Deklaration besitzt einfachen Inhaltstyp

(**II**) Attribut-Verwendung ist als verboten definiert

(**III**) Attributwert entspricht nicht der Typdefinition

(**IV**) es existiert keine passende Attribut-Deklaration und Attribut-Wildcard

(**V**) Attribut ist vom Typ ID und der Attributwert entspricht einer bereits existierenden ID

(**VI**) Attribut ist vom Typ IDREF und der Attributwert entspricht keiner existierenden ID

(**VII**) Attribut ist Teil eines optionalen Schlüssels (unique), wobei der Schlüsselwert bereits existiert

(**VIII**) Attribut ist Teil einer Schlüsselreferenz (keyref), wobei kein passender Schlüsselwert existiert

(I) Wenn wir eine einfache Typdefinition vorfinden, ist eine Schemaevolution notwendig. Da innerhalb einfacher Typdefinitionen keine Attribute deklariert werden können, wird diese zu einer komplexen Typdefinition mit einfachem Inhaltstyp umgewandelt. Zusätzlich wird die umgeformte Typdefinition um eine Attributdeklaration erweitert. Diese wird als optional definiert und trägt den Namen des neuen Attributes. Ein Beispiel dafür zeigt die Abbildung 5.2.

Wenn wir eine komplexe Typdefinition vorfinden, muss geprüft werden, ob eine Attribut-Deklaration mit dem Namen unseres neuen Attributes bereits vorhanden ist. Als erstes schauen wir, ob lokal definierte Attribut-Deklarationen innerhalb unserer Typdefinition existieren. Weiterhin kann auf globale Attribut-Deklarationen oder Attribut-Gruppen referenziert worden sein. Und zu guter Letzt müssen alle komplexen Basistypen unserer Typedefinition geprüft werden, da durch die Vererbung auch die Attribut-Deklarationen der Elterntypen gültig sind.

Wenn wir eine zu unserem Attribut passende Deklaration gefunden haben, muss folgendes geprüft werden.

(II) Falls die Benutzung des Attributes als verboten definiert ist, muss diese Eigenschaft durch eine Schemaevolution auf optional geändert werden. Außerdem wird geprüft ob ein fester Attributwert definiert wurde. Falls

```
vorher:
<xs:element name="Element" type="xs:byte"/>

nachher:
<xs:element name="Element">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:byte">
        <xs:attribute name="newAttribute" use="optional">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Abbildung 5.2: Einfache Typdefinition wird um ein Attribut erweitert

```
vorher:
<xs:attribute name="Name" use="prohibited" fixed="fester_Wert"/>
nachher:
<xs:attribute name="Name" use="optional"/>
```

Abbildung 5.3: Attribut-Verwendung wird optional und fester Attributwert entfernt

dies zutrifft, muss die Definition des festen Attributwertes aus der Attribut-Deklaration entfernt werden. Dies gilt auch, wenn der neue Attributwert mit dem festen Attributwert übereinstimmt. Ein Beispiel hierfür ist in Abbildung 5.3 ersichtlich. Hierbei muss darauf hingewiesen werden, dass die Definition des festen Attributwertes nur entfernt werden kann, weil die Benutzung des Attributes vorher verboten war und deshalb auch keine Attribute des Typen in der XML-Instanz vorkamen.

(III) Wenn der Attribut-Deklaration eine Typdefinition zugewiesen wurde, muss der neue Attributwert dieser entsprechen. Falls dies nicht der Fall ist wird durch eine Schemaevolution die Zuweisung der Typdefinition verändert. Dabei kann der Verweis entweder entfernt werden oder wir verweisen auf den ersten Basistypen, welcher unserem Attributwert entspricht. Systematisch werden alle Basistypen der Typdefinition getestet. Dabei ist der Attributwert mindestens zu dem Typ aller einfachen Typen (AnySimpleType) gültig. Die Abbildung 5.4 zeigt ein passendes Beispiel.

(IV) Wenn keine passende Attribut-Deklaration gefunden wurde und auch keine Attribut-Wildcard definiert wurde, wird die Typdefinition durch eine Attribut-Deklaration erweitert. Die Attribut-Verwendung wird dabei als optional definiert. Ein Beispiel hierfür ist in Abbildung 5.5 dargestellt.

Sonderfälle

Typdefinition *anyType* Wenn der Element-Deklaration keine Typdefinition zugewiesen wurde, erhält sie implizit den Typ *anyType*. *AnyType* ist der Basistyp aller einfachen und komplexen Typen. Da der Typ uneingeschränkten Element-Inhalt erlaubt, ist in diesem Fall keine Schemaevolution nötig.

(V,VI) ID/IDREF-Mechanismus Eine besondere Betrachtung benötigen die Typen ID, IDREF und IDREFS, da sie eine globale Bedeutung haben. ID-Attribute müssen über die gesamte Instanz eindeutig sein. Jedes IDREF-Attribut muss auf ein existierendes ID-Attribut verweisen. Bei einem IDREFS-Attribut ist der Attributwert eine Liste von ID-Werten. Falls das hinzuzufügende Attribut vom Typ ID ist, wobei der Attributwert bereits als ID existiert, wird durch folgende Schemaevolution die Gültigkeit des Schemas erhalten. Es wird der komplette ID/IDREF-Mechanismus, durch das Löschen aller ID, IDREF und IDREFS-Zuweisungen, entfernt. Falls das Attribut vom Type IDREF oder IDREFS ist, genügt es die Zuweisung der Typedefinition IDREF des selektieren Attributes zu entfernen. In beiden Fällen wird der ID/IDREF-Mechanismus dadurch unbrauchbar und deshalb sollten diese Update-Operationen untersagt werden.

(VII,VIII) Identitätsbeschränkungs-Definitionen Ähnlich ist die Situation mit den Identitätsbeschränkungs-Definitionen. Diese sind Erweite-

```
update insert attribute abc{"1000"} into //ELEMENT

vorher:
<xs:attribute name="Anzahl" use="optional" type="xs:byte"/>

nachher:
<xs:attribute name="Anzahl" use="optional" type="xs:int"/>
```

Abbildung 5.4: Typedefinition wird angepasst

```
vorher:
<xs:complexType>
  <xs:sequence>
    <xs:element name="ELEMENT"/>
  <xs:sequence>
</xs:complexType>

nachher:
<xs:complexType>
  <xs:sequence>
    <xs:element name="ELEMENT"/>
  <xs:sequence>
  <xs:attribute name="newAttribut" use="optional"/>
</xs:complexType>
```

Abbildung 5.5: Typdefinition mit neuer Attribut-Deklaration erweitern

rungen von Typen und nicht wie beim ID/IDREF-Mechanismus selbst Typen. Außerdem können nicht nur Attribute, sondern auch Elemente und Kombinationen dieser selektiert werden. Im Gegensatz zum ID/IDREF-Mechanismus können Identitätsbeschränkungen innerhalb eines bestimmten Bereiches von Elementen definiert werden. Die selektierten Felder werden nicht wie beim ID/IDREF-Mechnismus als Zeichenketten, sondern als Werte verglichen.

Das hinzuzufügende Attribute kann Teil einer Schlüsselreferenz (keyref) oder eines optionalen Schlüssels (unique) sein. Falls das neue Attribut durch ein Feld eines optionalen Schlüssels selektiert wird, muss die Eindeutigkeit, der durch den Selektor identifizierten Tupel, sichergestellt werden. Wenn dies nicht zutrifft, sind ähnlich wie beim ID/IDREF-Mechanismus die Identitätsbeschränkungs-Definition des optionalen Schlüssels und falls existiert, die zugehörige Definitionen der Schlüsselreferenz zu entfernen. Wie beim ID/IDREF-Mechanismus sollte auch in diesem Fall die Ausführung der Update-Operationen untersagt werden.

5.1.2 Delete

Mit Hilfe der Operation *Delete* wird ein Attribut entfernt.

Wohlgeformtheit

Beim Löschen kann die Wohlgeformtheit einer XML Instanz nicht verletzt werden. Wenn das zu löschende Attribut nicht existieren sollte, kann die Updateoperation nur abgebrochen werden. Eine Schemaevolution ist nicht möglich.

Validität und Evolutionsschritt

Beim Löschen eines Attributes wird die Gültigkeit des Schemas verletzt, wenn mindestens eine der folgenden Situationen auftritt.

(**I**) Attribut-Verwendung fordert eine entsprechende Attribut-Informationseinheit

(**II**) Attribut ist vom Typ ID und es existiert mindestens eine Referenz auf diese

(**III**) Attribut-Informationseinheit ist Teil eines zwingenden Schlüssels (key) einer Identitätsbeschränkungs-Definition

(**IV**) Attribut-Informationseinheit ist Teil eines optionalen Schlüssels (unique) einer Identitätsbeschränkungs-Definition und es existiert mindestens eine Schlüsselreferenz auf diese

(**V**) Attribut-Verwendung besitzt eine Wertebereich-Beschränkung (fixed oder default)

(I) In dem Fall, dass die Attribut-Verwendung die Existenz der Attribut-Informationseinheit in der Instanz als erforderlich definiert, ist es notwendig die Attribut-Verwendung durch eine Schemaevolution von erforderlich auf optional zu ändern.

(II) Falls die Typ-Definition des Attributes ID ist, muss sichergestellt werden, dass keine fehlerhaften Referenzen entstehen. Falls beim Durchsuchen der Instanz existierende Referenzen gefunden werden, gibt es nur eine Möglichkeit dies mit einer Schemaevolution zu behandeln. Dabei wird allen Attribut-Deklarationen die Zuweisung der Typ-Definition IDREF oder IDREFS entfernt. Eine bessere Möglichkeit ist es, nicht das Schema sondern die Instanz zu verändern. Dabei werden entweder die Referenzen oder die Elemente, welche die diese beinhalten, entfernt. Die Auswirkungen durch das Ändern der Instanz sind sicherlich nicht so groß wie bei der Schemaevolution, da nicht der ganze ID/IDREF-Mechanismus entfernt wird. Eine eindeutige

Lösung gibt es jedoch nicht und deshalb sollte die Update-Operation in diesem Fall untersagt werden. Der Benutzer sollte vorher die Referenz entfernen bzw. anpassen, bevor die ID gelöscht werden kann.

(III,IV) Ähnlich ist die Situation bei den Identitätsbeschränkungs-Definitionen. Hierbei müssen wir jedoch zwischen optionalen und zwingenden Schlüsseln unterscheiden. Falls das zu löschende Attribut Teil eines zwingenden Schlüssels ist, wird durch eine Schemaevolution die Schlüsseldefinition von zwingend auf optional geändert. Weiterhin muss sichergestellt werden, dass keine fehlerhaften Referenzen entstehen. Wenn eine Referenz auf unseren Schlüssel in der Instanz existieren sollte, kann dies durch folgende Schemaevolutionen gelöst werden. Eine Möglichkeit ist das komplette Entfernen der Identitätsbeschränkungs-Definitionen. Eine zweite Möglichkeit ist die Anpassung der Identitätsbeschränkungs-Definition. Falls die Schlüssel auch ohne das Feld, welches unser Attribut selektiert, eindeutig bleiben, kann dieses aus der Identitätsbeschränkungs-Definition entfernt werden. Die Update-Operation sollte jedoch wie bei dem ID/IDREF-Mechanismus verboten werden, solange noch existierende Referenzen existieren.

(V) Falls die Attribut-Verwendung eine Wertebereich-Beschränkung besitzt, kommt es zu folgender Situation. In [SCH04c] wird beschrieben, dass der durch eine Wertebereich-Beschränkung definierter Wert nach der Schema-Validierung im Post-Schema-Information-Set enthalten sein muss. Der XML-Prozessor muss deshalb nach dem Löschen eine neue Attribut-Informationseinheit mit dem in der Wertebereich-Beschränkung angegeben Wert erzeugen. Mit dem Löschen eines Attributes wird erwartet, dass die Attribut-Informationseinheit danach nicht mehr existiert. Das Löschen eines Attributes mit festem Attributwert hat jedoch keine Auswirkungen, da die Attribut-Informationseinheit in dieser Situation automatisch wiederhergestellt wird. Das Löschen eines Attributes mit vorgegeben Attributwert hätte nicht das Entfernen der Attribut-Informationseinheit zur Folge, sondern das Ersetzen mit dem vorgegeben Attributwert. Eine Schemaevolution in diesem Fall ist nicht möglich, da dass Entfernen der Wertebereich-Beschränkung auch Einfluss auf die ungeparsten Instanzen haben kann. Aus diesen Gründen sollte die Ausführung der Update-Operation untersagt werden.

5.1.3 Replace

Mit der Replace-Anweisung wird ein existierendes Attribut inklusive Attributewert gegen ein neues Attribut mit gewünschtem Wert ersetzt.

Dies kann immer ersetzt werden, durch eine Delete gefolgt von einer Insert-Operation. Wichtig ist aber, dass die Validitätsbedingungen erst nach der Ausführung beider Operationen erfolgt, da sonst das gelöschte Attribut

immer durch eine Schemaevolution als optional definiert wird.

Wohlgeformtheit

Es muss sichergestellt sein, dass das Element nicht zwei Attribute mit gleichen Namen besitzt.

5.1.4 Rename

Das Umbenennen eines Attributes kann auch als Kombination einer Delete und Insert-Operationen gesehen werden. Die Reihenfolge spielt bei der Ausführung der Operationen keine Rolle. Auch die Gültigkeitsbedingungen können direkt nach der Ausführung jeder Operation überprüft werden.

Wohlgeformtheit

Es muss sichergestellt sein, dass das Element nicht zwei Attribute mit gleichen Namen besitzt.

5.2 Updates auf Elementebene

5.2.1 Insert Into | Preceding | Following

Mit Hilfe der Operation *Insert* wird ein neues Element hinzugefügt. Der einzige Unterschied zwischen *Insert Into*, *Insert Preceding* und *Insert Following* ist die Position, wo das neue Element eingefügt wird.

Beim Einfügen eines Elementes wird die Gültigkeit des Schemas verletzt, wenn mindestens eine der folgenden Situationen auftritt.

(**I**) Element-Deklaration besitzt eine einfache Typdefinition

(**II**) Komplexe Typdefinition hat einfachen oder leeren Inhaltstyp

(**III**) Elementgruppe besitzt keine passende Element-Deklaration

(I) Falls die Element-Deklaration eine einfache Typdefinition besitzt, wird sie durch eine komplexe Typedefinition ersetzt. Die neue Typdefinition ist vom Inhaltstyp gemischt (mixed) und erlaubt damit die Existenz von Kindelementen und Zeichen-Informationseinheiten. Zusätzlich enthält sie eine Elementgruppe mit der neuen Element-Deklaration.

Ein Beispiel dafür ist in der Abbildung 5.6 dargestellt.

```
vorher:
<xs:element name="Element" type="xs:byte"/>

nachher:
<xs:element name="Element">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Neues_Element" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Abbildung 5.6: Umwandelung der einfachen Typedefinition

Außerdem ist hierbei noch anzumerken, dass es nach [SCH04c] nur möglich ist die Zeichen-Informationseinheiten zu validieren, wenn die Element-Deklaration den einfachen Inhaltstyp besitzt. Deshalb geht durch die Evolution die Information der einfachen Typdefinition verloren.

(II) Eine Komplexe Typdefinition kann Einfachen, Leeren, Gemischten oder Element als Inhaltstyp haben.

Wenn die Typdefinition einen Einfachen oder Leeren Inhaltstyp hat, darf das Element keine Unterelemente besitzen. Deshalb wird der Inhaltstyp zum Gemischten Inhaltstyp umgewandelt. Weiterhin wird die Typdefinition um eine Elementgruppe mit einer Liste und einer optionalen Element-Deklaration erweitert.

(III) Die Komplexe Type-Definition der Element-Deklaration besitzt eine Elementgruppe, die entweder eine Liste, Auswahl oder Menge aller Element ist.

Falls die Elementgruppe vom Type *all* ist, wird sie um eine Element-Deklaration erweitert. Die neue Element-Deklaration ist optional und bekommt den Namen des hinzuzufügenden Elements.

Falls die Elementgruppe vom Type *choice* sein sollte, wird die Anzahl der maximal auswählbarer Element (maxOccurs) um eins erhöht. Außerdem wird eine Element-Deklaration mit dem Namen des hinzuzufügenden Elements hinzugefügt.

Die am häufigsten verwendete Form der Elementgruppe ist die Liste (sequenz). Die Liste wird um eine neue optionale Element-Deklaration erweitert. Die Position der Element-Deklaration wird durch die gewählte Insert-Operation bestimmt.

```
vorher:
<xs:element name="Editor" minOccurs="1"/>

nachher:
<xs:element name="Editor" minOccurs="0"/>
```

Abbildung 5.7: Anpassen der Partikeleigenschaft

5.2.2 Delete

Beim Löschen eines Elementes wird die Gültigkeit des Schemas verletzt, wenn mindestens eine der folgenden Situationen auftritt.

(**I**) Das zu löschende Element ist das ROOT-Element

(**II**) Die Länge der Folge gleichnamiger Elemente entspricht dem durch das Partikel definierten minimalen Vorkommen

(**III**) Element ist vom Typ ID und es existiert mindestens eine Referenz auf dieses

(**IV**) Element ist Teil eines zwingenden Schlüssels (key) einer Identitätsbeschränkungs-Definition

(**V**) Element ist Teil eines optionalen Schlüssels (unique) einer Identitätsbeschränkungs-Definition und es existiert mindestens eine Referenz auf dieses

(I) Falls durch die Delete-Operation das ROOT-Element selektiert wird, muss die Ausführung der Operation untersagt werden. Ein gültiges XML-Dokument muss ein ROOT-Element besitzen, deshalb gibt es auch keine mögliche Schemaevolution.

(II) Falls die Länge der Folge gleichnamiger Elemente nach dem Löschen geringer ist als das durch das Partikel der Element-Deklaration definierte minimale Vorkommen, wird durch eine Schemaevolution die Partikeleigenschaft angepasst. Hierbei wird der Wert des Attributes *minOccurs* der zugehörigen Element-Deklaration um eins verringert. Ein Beispiel dafür ist in Abbildung 5.7 dargestellt.

(III) Falls das Element eine einfache Typdefinition besitzt und diese der Typ ID ist, kann die Updateanweisung nur ausgeführt werden, wenn keine Referenz auf die alte ID existiert. Um dies zu prüfen müssen alle IDREFs der

Instanz geprüft werden. Es gibt nur eine Möglichkeit dies mit einer Schemaevolution zu beheben. Dabei wird allen Attributen und Elementen, welche die Typdefinition IDREF oder IDREFs besitzen, die Typdefinition entfernt. Da dies aber den gesamten ID/IDREF-Mechanismus zerstört, sollte unter diesen Umständen keine Schemaevolution erlaubt sein.

(IV,V) Die Situation mit den Identitätsbeschränkungs-Definitionen für Elemente wird gleich der für Attribute behandelt. Unterschieden wird zwischen optionalen und zwingenden Schlüsseln. Falls das zu löschende Element Teil eines zwingenden Schlüssels ist, wird durch eine Schemaevolution die Schlüsseldefinition von zwingend auf optional geändert. Weiterhin muss sichergestellt werden, dass keine fehlerhaften Referenzen entstehen. Wenn eine Referenz auf unseren Schlüssel in der Instanz existieren sollte, kann dies durch folgende Schemaevolutionen gelöst werden. Eine Möglichkeit ist das komplette Entfernen der Identitätsbeschränkungs-Definitionen. Eine zweite Möglichkeit ist die Anpassung der Identitätsbeschränkungs-Definition. Falls die Schlüssel auch ohne das Feld, welches unser Element selektiert, eindeutig bleiben, kann dieses aus der Identitätsbeschränkungs-Definition entfernt werden. Die Update-Operation sollte jedoch wie bei dem ID/IDREF-Mechanismus verboten werden, solange noch existierende Referenzen existieren.

5.2.3 Replace

Mit Hilfe der Replace-Anweisung wird ein existierendes Element durch ein neues ersetzt.

Dies kann immer ersetzt werden, durch eine Delete gefolgt von einer Insert-Operation. Dabei ist zu beachten, dass die Validitätsbedingungen erst nach der Ausführung beider Operationen erfolgt, da sonst das gelöschte Element immer durch eine Schemaevolution als optional definiert wird.

5.2.4 Rename

Das Umbenennen eines Elementes kann auch als Kombination einer Delete und Insert-Operationen gesehen werden. Die Reihenfolge spielt bei der Ausführung der Operationen keine Rolle. Auch die Gültigkeitsbedingungen können direkt nach der Ausführung jeder Operation überprüft werden.

5.3 Besonderheiten

5.3.1 Updates von Kommentaren und PIs

Neben den Attributknoten, Elementknoten und Textknoten kann ein XML-Dokument auch Kommentare und PIs (engl. Processing Instructions) enthalten. Bei der Validierung werden Kommentare und PIs jedoch ignoriert und spielen deshalb bei der Schemaevolution keine Rolle.

Kapitel 6

Implementierungsdetails und Ergebnisse

In diesem Kapitel wird die im Rahmen der Studienarbeit entstandene prototypische Implementierung beschrieben. Es wird ein Überblick über den Aufbau, die Funktionsweise und den dabei aufgetretenen Problemen gegeben. Abschließend wird anhand eines konkreten Beispiels der Ablauf dargestellt.

6.1 Aufgabe

Mit Hilfe des Prototyps soll gezeigt werden, wie die im Kapitel 4 aufgestellten theoretischen Grundlagen zur Ableitung der Schemaevolutionsschritte umgesetzt werden können.

Dazu erhält der Prototyp als Eingabe die Updateanweisung und ein XML-Dokument mit dazugehörigem XML-Schema. Anhand dieser Informationen muss der Prototyp entscheiden, ob eine Schemaevolution nötig ist. Wenn ja generiert der Prototyp anhand definierter Regeln einen Schemaevolutionsschritt in Form einer Updateanweisung.

Die aktuelle Version des Prototyps akzeptiert als Eingabe ausschließlich XPath-Updateanweisungen. XQuery-Anweisungen werden zum jetzigen Zeitpunkt nicht unterstützt.

6.2 Verwendete Technologien

Für die Umsetzung der Implementierung wurde eine Menge von existierenden Softwaremodulen verwendet. Bei der Auswahl wurde vor allem Wert auf Plattformunabhängigkeit, freie Verfügbar- und Erweiterbarkeit gelegt.

Die Implementierung des Prototyps erfolgte in der Programmiersprache Java unter der Verwendung der JDK 1.5.0 von Sun. Zur Entwicklung wurde die IDE (engl. Integrated Development Environment) *Eclipse* verwendet.

Zur Verarbeitung der XML-Dokumente und XML-Schemata wurden die Apache-Projekte Xerces und Xalan ausgewählt. Xerces ist ein sehr weit verbreiteter XML-Parser, der unter anderem die Schnittstellen DOM als auch SAX2 unterstützt. Außerdem ist Xerces derzeit der einzige Parser der eine Implementierung der Schnittstelle PSVI (Post Schema Validation Infoset) besitzt. PSVI ist ein Bestandteil der W3C-Empfehlung XML-Schema und stellt eine Abbildung des Datenmodels von XML-Schema auf eine Baumstruktur dar. Ähnlich wie beim DOM kann über die Baumstruktur navigiert werden und mit Hilfe von Methoden auf die Inhalte zugegriffen werden.

Um die Updateanweisungen auswerten zu können ist ein XPath-Prozessor nötigt. Aus diesem Grund wurde die Möglichkeit genutzt, Xerces in Xalan zu integrieren. Xalan ist ein XSLT-Prozessor und besitzt eine vollständige Implementierung eines XPath-Prozessors.

Programmierumgebung	Sun JDK 1.5.0
IDE	Eclipse 3.2
XML Parser (Modul)	Apache Xerces 2.7.0
XSLT Prozessor (Modul)	Apache Xalan 2.7.0

6.3 Architektur

Dieser Abschnitt beschreibt den Aufbau des Systems und die Aufgaben der verschiedenen Module. Das System besteht aus einer GUI (engl. Graphical User Interface) und einem Prozessor mit dem Namen XEUpdate (XML Evolution Update).

Die Aufgabe der *GUI* ist die Interaktion mit dem Benutzer und dient der Ein- und Ausgabe der Daten.

Der *XEUpdate-Prozessor* ist der eigentliche Kern des Systems. Er besteht aus einem Parser-, Anfrage-, Validierungs- und Evolutionsmodul. In diesem Abschnitt wird später auf die einzelnen Module genauer eingegangen.

Der XEUpdate-Prozessor bekommt als Eingabe die Updateanweisung, das zugehörige XML-Dokument und XML-Schema. Als ersten wird überprüft, ob die Updateanweisung die Gültigkeit des Schemas verletzt. Wenn dies zutrifft generiert der Prozessor anhand definierter Regel einen Schemaevolutionsschritt. Die Schemaevolution wird in Form einer Updateanweisung ausgegeben. Bei der Abarbeitung können verschiedene Fehlern auftreten, die

beim Auftreten vom Prozessor ausgegeben werden und den Abbruch der Abarbeitung zur Folge haben.

Die Abbildung 6.1 verdeutlicht den Aufbau des Prozessors und zeigt dessen Module.

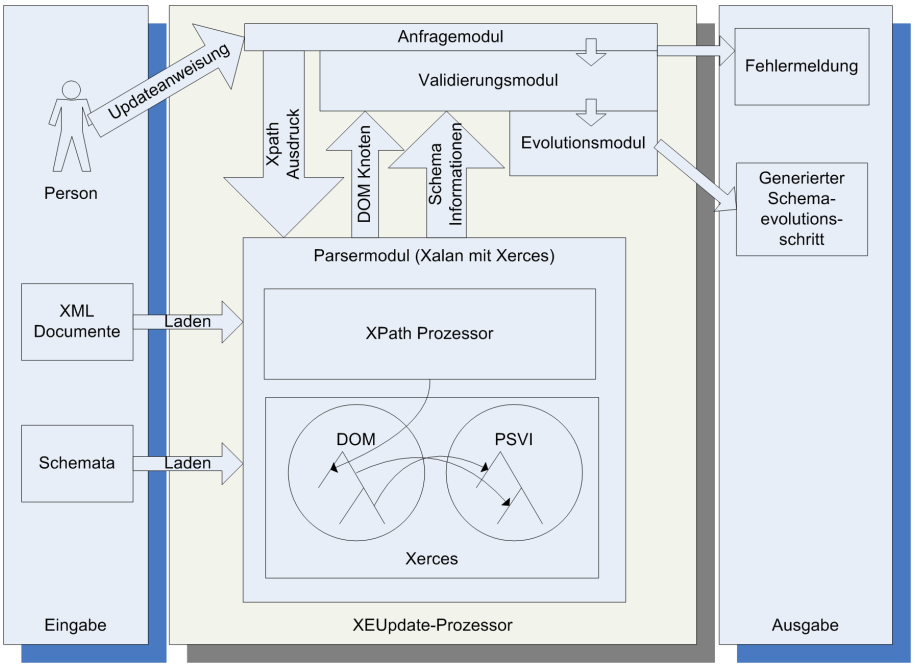


Abbildung 6.1: Aufbau des XEUpdate-Prozessors

Das *Parsermodul* hat die Aufgabe die XML-Dokumente und Schemata zu laden und mit Hilfe von Schnittstellen alle nötigen Informationen zur Validierung und Erstellung der Evolutionsschritte den anderen Modulen zu Verfügung zu stellen.

Dazu lädt und parst das Parsermodul zuerst das XML-Schema und erstellt eine Abbildung in Form eines PSVI-Baums. Danach wird das XML-Dokument geparst und in einem DOM-Baum überführt. Um die Gültigkeit der Dokumente zu gewährleisten, wird das Dokument während des Parsens gegenüber dem Schema überprüft. Gleichzeitig bekommt jedes Element und Attribut ein Verweis auf die zugehörige Deklaration im PSVI-Baum. Das Ergebnis sind zwei Bäume (DOM und PSVI), die miteinander verknüpft sind.

Nachdem der Prozess abgeschlossen ist, können die anderen Module mit Hilfe der Schnittstellen DOM und PSVI auf den Dokument- und Schemainhalt zugreifen. Zur Identifizierung der betroffenen DOM-Knoten ist es möglich, einen XPath-Ausdruck zu übergeben. Dieser wird vom Parsermodul evaluiert und das Ergebnis als eine Menge von DOM-Knoten zurückgegeben.

Das *Anfragemodul* hat die Aufgabe, die Updateanweisung zu parsen und in ihre Bestandteile (engl. Token) zu zerlegen. Wenn kein Syntaxfehler aufgetreten ist, wird der XPath-Ausdruck an das Parsermodul gesendet und die Information über die Updateoperation an das Validierungsmodul.

Das *Validierungsmodul* überprüft, ob durch die Ausführung der Updateanweisung die Gültigkeit des XML-Dokumentes erhalten bleibt. Wenn die Updateanweisung die Gültigkeit verletzt, ist eine Schemaevolution nötig. Die Informationen, die während des Überprüfens gesammelt wurden und die Updateoperation werden an das Evolutionsmodul übergeben.

Das *Evolutionsmodul* hat die Aufgabe, die nötigen Schemaevolutionsschritte zu generieren, um die Gültigkeit der Dokumente zu erhalten. Die nötigen Informationen bekommt das Modul von dem Validierungsmodul. Wenn es möglich ist, einen oder mehrere Schemaevolutionsschritte zu generieren, werden diese in Form von XPath-Updateanweisungen ausgegeben.

6.4 Beispiel

Dieser Abschnitt zeigt den Ablauf des Programms und die Ableitung des Schemaevolutionsschrittes anhand eines Beispiels.

Szenario

Das ausgewählte Beispiel ist ein Ausschnitt einer Literatur-Datenbank. Es wurde ein einfaches Schema erstellt, das der Speicherung von Buchinformationen dient. Das Beispieldokument enthält bereits ein Buch, welches mit der Information des Herausgebers erweitert werden soll. Beim Entwurf des Schemas wurde jedoch nicht vorgesehen, den Herausgeber des Buches zu speichern.

GUI

Wird der Prototyp gestartet, erscheint die GUI (engl. Graphical user interface), wie sie in Abbildung 6.2 ersichtlich ist. Das im Hintergrund liegende Fenster ist das Hauptprogramm. Es besitzt ein Menü zur Steuerung des Programms und einen Bereich zur Ausgabe von Debuginformationen. Das Fenster im Vordergrund ist ein Dialogfenster, welches der Interaktion mit dem Benutzer dient.

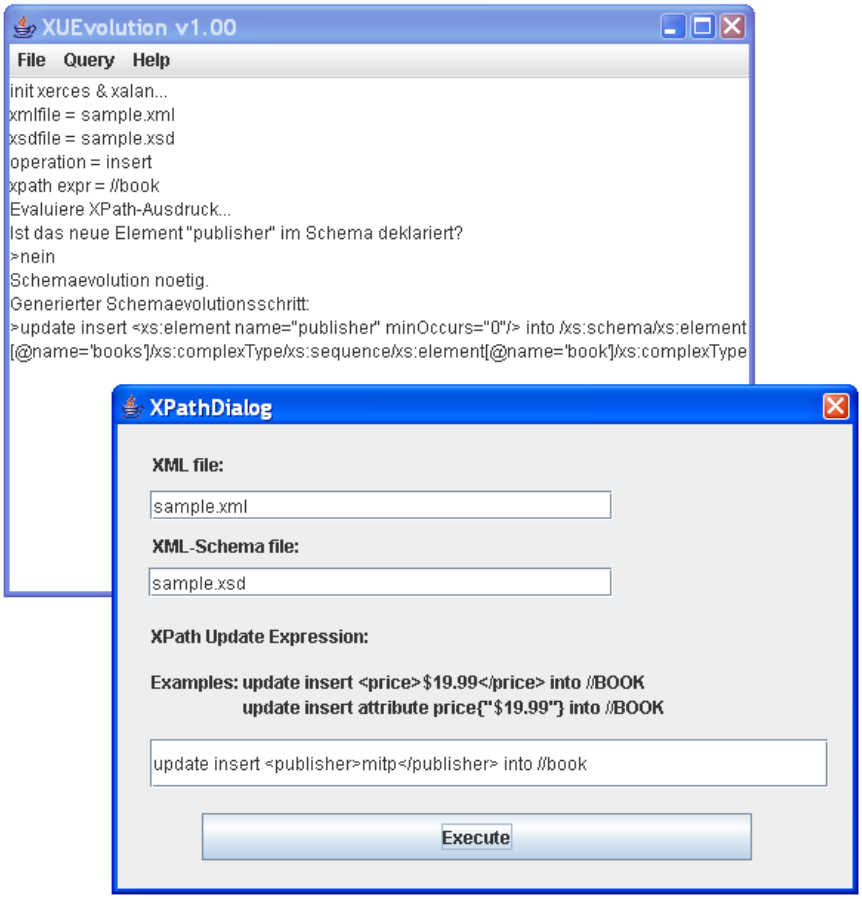


Abbildung 6.2: Beispiel der GUI

Testdaten

Das für das Beispielszenario erstellte XML-Schema ist in der Abbildung 6.3 ersichtlich. Das Wurzelelement trägt den Namen *books* und besteht aus einen lokal definierten Komplexen Typen. Der Komplexe Typ besteht aus einer Sequenz mit dem Element *book*, das beliebig oft verkommen kann. Weiterhin besteht das Element *book* aus einer Sequenz der Elemente *title*, *author* und *year*, die mit Ausnahme von *author* genau einmal vorkommen müssen.

Neben dem XML-Schema bekommt der Prototyp noch als Eingabe die XML-Instanz und die gewünschte Updateanweisung. Ein Beispiel für ein Instanzdokument ist in Abbildung 6.4 dargestellt.

Mit Hilfe der Updateoperation in Abbildung 6.5 wird jedes Buch mit der Information des Herausgebers *mitp* erweitert.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
    <xs:element name="books">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="book" minOccurs="0"
                            maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="title"/>
                        <xs:element name="author"
                                    maxOccurs="unbounded"/>
                        <xs:element name="publisher"/>
                        <xs:element name="year"/>
                    </xs:sequence>
                </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Abbildung 6.3: XML-Schema für das Beispiel

```
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="sample.xsd">
    <book>
        <title>Datenbanken: Konzepte und Sprachen</title>
        <author>Andreas Heuer</author>
        <author>Gunter Saake</author>
        <year>2000</year>
    </book>
</books>
```

Abbildung 6.4: XML-Instanz für das Beispiel

```
update insert <publisher>mitp</publisher> into //book
```

Abbildung 6.5: Updateanweisung für das Beispiel

Ablauf

Dieser Abschnitt beschreibt den Ablauf des Prozessors. Über die GUI wurden drei Parameter übergeben, die Updateanweisung, das XML-Dokument und das zugehörige Schema.

1. Parsermodul Als erstes werden dem Parsermodul die URLs für das XML-Dokument und dem XML-Schema übergeben. Das Parsermodul überführt danach das Schema in einen PSVI-Baum und XML-Dokument in einen DOM-Baum.

2. Anfragemodul Als nächstes wird die Updateanweisung durch das Anfragemodul in seine Teile (engl. Token) zerlegt. Für das Beispiel ist es die Updateoperation *insert (element) into*, das einzufügende Element *publisher* und der XPath-Ausdruck *//book*.

Der XPath-Ausdruck wird an das Parsermodul übermittelt. Das Parsermodul gibt in diesem Fall den DOM-Knoten des Elementes *book* zurück.

3. Validierungsmodul Das Validierungsmodul erfragt die Informationen über die Updateoperation und holt sich den selektierten DOM-Knoten vom Parsermodul. Das Validierungsmodul prüft, ob das XML-Dokument nach der Ausführung der Updateanweisung gültig ist. In diesem Beispiel sieht das Parsermodul, dass das Element *publisher* in das Element *book* eingefügt werden soll. Anhand der Updateoperation schlussfolgert das Parsermodul, dass zuerst die Typdefinition des selektierten Knoten überprüft werden muss. Durch die Verknüpfung des DOM- und PSVI-Baumes erfragt das Parsermodul die Element-Deklaration des Schemas ab. Von der Element-Deklaration navigiert das Parsermodul mit Hilfe der PSVI-Schnittstelle zur Typdefinition des selektierten Elementes. Es handelt sich in diesem Fall um eine Komplexe Typdefinition, die eine Sequenz enthält mit einer Liste von Unterelementen. Das Parsermodul findet in der Sequenz keine passende Element-Deklaration mit dem Namen *publisher*. Die Validierung ist fehlgeschlagen und damit werden die gesammelten Informationen an das Evolutionsmodul übergeben.

4. Evolutionsmodul Das Evolutionsmodul leitet anhand bekannter Regeln die Schemaevolutionsschritte ab. Für das gewählte Beispiel wird die

Typdefinition um das Unterelement *publisher* erweitert. Da die Typdefinition vom komplexen Typ ist und eine Sequenz enthält, wird eine Udpateanweisung abgeleitet, die diese Sequenz um das Element *publisher* erweitert. Das neue Unterelement wird als optional deklariert und bekommt implizit den Typen *anyType* zugewiesen. Anhand der Position der Sequenz im PSVI-Baum wird ein XPath-Ausdruck abgeleitet.

Der abgeleitete Schemaevolutionsschritt ist in Abbildung 6.6 ersichtlicht.

```
update insert <xs:element name="publisher" minOccurs="0"/> into
       /xs:schema/xs:element[@name='books']/xs:complexType
       /xs:sequence/xs:element[@name='book']/xs:complexType
```

Abbildung 6.6: Abgeleiteter Evolutionsschritt

6.5 Ergebnisse

Das Ergebnis ist eine prototypische Implementation, welche aus atomaren Updateanweisungen Schemaevolutionsschritte generiert. Das Grundgerüst ist das Parsermodul, das über verschiedene Schnittstellen alle nötigen Informationen über die Schema- und Instanzdokumente bereitstellt. Die Aufgaben der Anfrageverarbeitung, Validierung und der Evolution wurde auf separate Module verteilt. Jedes der Module ist erweiterbar. Das Anfragemodul könnte mit einem XQuery-Parser erweitert werden, ohne die anderen Module dabei zu beeinflussen. Das Validierungs- und Evoltionsmodul können mit zusätzlichen Regeln erweitert werden.

Der aktuelle Prototyp akzeptiert als Eingabe XPath-Updateanweisungen, mit deren Hilfe Elemente und Attribute eingefügt und gelöscht werden können.

Aufgetretene Probleme

Während der Entwicklung traten unter anderem folgende Probleme auf.

Schemainformation Das PSVI-Modul des XML-Parsers bildet ein Schema in einen Baum ab und stellt die Informationen über eine Schnittstelle bereit. Die Abbildung ist nicht eineindeutig und deshalb ist es nicht immer möglich, anhand der Position im PSVI-Baum die Position in der Schema-Instanz zu finden. Da der XML-Parser eine Implementation eines Opensource Projektes, ist konnte das Problem durch eine Erweiterung der Implementation gelöst werden.

Unvollständige Implementation Die Implementation der Schnittstelle PSVI ist noch nicht vollständig. Z.B. gibt es die Möglichkeit, von einem Attributknoten eines DOM-Baumes, die Attributdeklaration zu erfragen. Diese Funktion liefert derzeit aber keinen Wert zurück. Dieses Problem konnte nur teilweise umgangen werden, indem über die Elementdeklaration zu der Attributdeklaration navigiert wurde.

XQuery-Unterstützung Die Komplexität von XQuery ist gleich eine Programmiersprache. Eine freie XQuery-Implementation mit Updateerweiterung wurde nicht gefunden. Aus diesen Gründen wurde der Prototyp auf XPath reduziert.

DOM3 Validation Es gibt eine Empfehlung vom W3C für eine Schnittstelle zur Validierung eines DOM-Baumes. Damit ist es möglich, vor der Ausführung einer Änderung zu erfragen, ob diese Anweisung die Gültigkeit verletzt. Leider war zu diesem Zeitpunkt nur eine unvollständige Implementation dieser Schnittstellte verfügbar. Weiterhin ist das Ergebnis einer Validierung nur gültig, ungültig oder unbekannt. Diese Information reicht nicht aus, um einen Schemaevolutionsschritt abzuleiten.

Kapitel 7

Verwandte Arbeiten

Dieses Kapitel gibt einen Überblick über Arbeiten, die mit dem in dieser Studienarbeit behandelten Themen verwandt sind. Dazu gehören XML-Anfragesprachen, XML-Schemasprachen und der Prozess der Schemaevolution.

XML-Anfragesprachen

Relationale Anfragesprachen wurden bereits durch die Entwicklung der Relationalen Datenbanksysteme weit erforscht. Die Anfragesprache SQL entwickelte sich zum Standard für relationale Datenbanksysteme.

Da XML zur Verarbeitung Semistrukturierter Daten entworfen wurde, sind relationale Anfragesprachen in ihrer Form für XML ungeeignet. Im Gegensatz zu SQL wird für eine XML-Anfragesprache die Eigenschaft der Rekursivität gefordert, um mit Pfaden beliebiger Länge umgehen zu können. Weiterhin ermöglicht XML, durch den ID/IDREF-Mechanis, die Bildung von Zyklen, was auch eine besondere Beachtung durch die Anfragesprache erfordert.

Lorel [AQM+97] ist eine Anfragesprache, die speziell für die Verarbeitung Semistrukturierter Daten entwickelt wurde. Durch die Entwicklung von XML entstand die Arbeit [LOR99], die beschreibt, wie das Semistrukturierte Datenmodell von Lorel an das von XML angepasst werden kann.

XPath [XPA05], XSLT [Cla99] und XQuery [XQU05b] sind Entwicklungen vom W3C. Genauer ist XPath eine Sprache zur Achsennavigation und Basis weiterer W3C Entwicklungen, wie z.B. XQuery und XSLT. XQuery ist eine XML Anfragesprache mit SQL-ähnlichen Konstrukten. In der ersten Version von XQuery wird keine Updatefunktionalität enthalten sein. Es gibt aber Arbeiten wie z.B. die von Tantarinov [TIHW01], Lehti [Leh01] und Hänsel [Hän02], welche die Möglichkeiten zur Erweiterung von XQuery mit

Updatefunktionalität beschreiben.

XSLT ist eine Sprache zur Transformation von XML-Dokumenten, wobei anhand von Schablonen (engl. Templates) der Transformationsvorgang beschrieben wird.

Der Sprachvorschlag XUpdate [LM00] basiert auf einer XSLT-ähnliche Syntax, die mit für Updates, notwendigen Methoden wie z.B. *rename* und *insert* erweitert wurde. Entwickelt wurde der Vorschlag von der *XML:DB*, welche eine Kooperation mittelgroßer Firmen ist.

Ein Vorschlag für eine graphische Anfragesprache ist XML-GL (XML Graphical Language) [XML99]. Dabei werden XML-Dokument und DTD auf das in XML-GL definierte graphische Datenmodell XML-GDM abgebildet. Durch die visuelle Darstellung eignet es sich gut als Programmoberfläche.

Der Vollständigkeit halber werden die XML-Anfragesprachen XQL [XQL98] und XML-QL [XML98] hier noch genannt.

Trotz der Menge der existierenden Sprachvorschläge, sind die in der Praxis vorwiegend eingesetzten Sprachen die Programmierschnittstellen DOM [DOM04] und SAX2 [Meg98].

In der Arbeit von Bonifati [BL01] werden fünf Sprachvorschläge auf Basis ihrer Anfrageeigenschaften verglichen.

Schemasprachen

Die Struktur eines XML-Dokumentes ist selbstbeschreibend und wird durch das Markup (Auszeichnung) vorgegeben. Es gibt eine Reihe von Vorschlägen für XML Schemasprachen, die es ermöglichen, die Struktur eines XML-Dokumentes einzuschränken. DTD und XML-Schema [SCH04c] sind Entwicklungen vom W3C. DTD ist ein Erbe der Sprache SGML [Gol96] und ist in dem Dokument der Syntaxdefinition von XML [XML04] enthalten. Im Vergleich zu DTD unterscheidet sich XML-Schema vor allem durch die in XML gehaltene Syntax, dem erweiterten Typsystem, der Trennung von Typ- und Elementdeklaration und der Möglichkeit von Vererbungen.

Relax NG [REL01] und Schematron [SCH04a] sind Schemasprachen, die nicht vom W3C entwickelt wurden. Relax NG ist ähnlich zu XML-Schema, bietet aber Features wie z.B. die Struktur vom Inhalt der Daten abhängig zu machen. Schematron stellt eine Erweiterung zu existierenden Schemasprachen dar. Ein Schema in Schematron besteht aus einer Menge von Behauptungen (engl. Asserts), die mit Hilfe von XPath-Ausdrücken definiert werden.

DSD [KMS02] ist eine Schemasprache, die es erlaubt, alle existierenden Definitionen zu überschreiben. Somit ist es bei einer Schemaevolution nicht notwendig, das existierende Schema zu ändern. Es wird ein neues Schema erstellt, welches das alte Schema importiert und die veränderten Definitionen

enthält.

Schemaevolution

Die Struktur eines XML-Dokumentes unterliegt einem Evolutionsprozess, da sich auch die Anforderungen an den Inhalt der Daten ändern. Bei der Verwendung eines XML Schemas ist es deshalb notwendig die Struktur und gegebenenfalls existierende XML-Dokumente anzupassen.

In der Arbeit von Tiedt [Tie05] wird eine Änderungssprache für XML-Schemata beschrieben. Weiterhin wird gezeigt, wie die existierenden Instanzdokumente nach einer Schemaevolution angepasst werden.

In den Arbeiten von Zeitz [Zei01] und von Su [SKR02] wird untersucht, welche Auswirkungen Umformungen der DTD auf die XML-Dokumente haben und wie diese angepasst werden können. Zeitz beschreibt in seiner Arbeit, wie mittels XSLT die DTDs und dazugehörigen XML-Dokumente angepasst werden. In der Arbeit von Su werden die DTD und das XML-Dokument in ein Objektorientierten Datenmodell umgewandelt. Auf diesen Objektorientierten Daten wird dann mittel SERF [CJR98], ein System zur Schemaevolution für objektorientierte Systeme, die Anpassung der Dokumente durchgeführt.

In der Arbeit von Klettke, Meyer und Hänsel [MEI05] wird untersucht, welche Auswirkungen ein XML Update auf das durch eine DTD gegebenes XML Schema hat. Updates auf XML-Dokumente können die Struktur verändern und somit das Dokument ungültig gegenüber dem gegebenen Schema machen. Es werden verschiedene Wege zur Lösung dieses Problems geschildert. Eine Möglichkeit, die in der Arbeit detailliert beschrieben wird, ist es die DTD nach einem XML Update anzupassen.

Die Arbeit [BDH+04] beschreibt, wie eine DTD bedingt durch eine Updateopertation angepasst werden kann. Dabei wird das XML-Dokument zunächst in einen Baum und das Schema in einen Bottom-Up Automaten überführt. Der Automat akzeptiert nur Bäume, die dem Schema entsprechen. Der Graph wird durch einen Algorithmus umgeformt und liefert eine Menge von regulären Ausdrücken als Ergebnis. Jeder Ausdruck stellt eine Möglichkeit für ein Schema dar, das die alten und neuen XML-Dokumente akzeptiert.

In der Arbeit [GMR05] werden die Auswirkungen der Schemaevolutionen auf die Gültigkeit der XML-Dokumente untersucht. Dabei wird zuerst untersucht, welche Arten von Änderungen an einem Schema notwendig sind. Weiterhin wird untersucht, wie der Aufwand der Revalidierung minimiert werden kann.

Systeme

Die Softwarefirmen wie z.B. IBM, Microsoft und Oracle haben ihre Datenbanksysteme um XML Support erweitert. Dabei gibt es meist die Möglichkeit, XML auf relationale Tabellen abzubilden oder sie einem nativen Format abzuspeichern. Als Anfragesprachen werden SQL/XML und XQuery unterstützt.

Wie sieht es aus mit der Unterstützung von XML Schema und Schema Evolution?

Die Arbeit [BOSdL05] beschreibt, auf welche Weise das von IBM entwickelte Datenbanksystem DB2 die Schemaevolution unterstützt. Um eine Versionisierung von XML Schemata zu ermöglichen, werden XML-Dokumente und Schemata unabhängig voneinander verwaltet. Auf statische Typüberprüfung und die Unterstützung des Imports von Schemata in einem XML-Dokument wird verzichtet. Die XML-Dokumente werden in einer Spalte einer Tabelle mit dem Datentyp XML abgelegt. Die Schemata werden einem Schemarepository registriert. Jeder XML-Instanz kann nun ein beliebiges XML-Schema aus dem Repository zugewiesen werden, ohne die Instanz dabei zu verändern.

Weiterhin wird ein Protokoll vorgestellt, welches das System auf eine neue Version eines Schemas hinweist. Die neuen Dokumente bekommen automatisch das neue Schema zugewiesen. Für die bereits existierenden Dokumente wird gezeigt, dass bei abwärtskompatiblen Schemata die Zuweisungen automatisch vorgenommen werden kann. Wenn keine automatische Anpassung möglich ist, bleibt die Zuordnung der älteren Schemata erhalten. Weiterhin wird gezeigt, wie die Schemaänderungen auch Auswirkungen auf die Anfragen haben können, selbst wenn das Schema abwärtskompatibel ist.

Somit bleibt die Aufgabe der Schemaevolution bei dem Anwender. Die Möglichkeit, gleichzeitig verschiedene Schemata zu verwenden, bringt vor allem bei der Verarbeitung von Anfragen und Änderungen einen deutlichen Mehraufwand. Unter Umständen müssen verschiedene Indizes für die verschiedenen Schemata erstellt werden.

Kapitel 8

Schlussbetrachtung

Die Rolle von XML für die Verarbeitung Semistrukturierter Daten wächst rasant. Seit der Veröffentlichung von XML ist eine unüberschaubare Menge von Technologien entstanden. XML hat sich zum universellen Austauschformat entwickelt. Umso wichtiger ist es Technologien zu erforschen, die den Evolutionsprozess unterstützen.

8.1 Zusammenfassung

Diese Arbeit gibt einen detaillierten Überblick, wie Schemaevolutionsschritte anhand von Updateoperationen abgeleitet werden können. Es wird ein Überblick über vorhandene Update-, Schema- und Schemaevolutionssprachen gegeben. Die prototypische Implementierung zeigt wie das theoretische Wissen in einem System umgesetzt werden kann.

8.2 Ausblick und weitere Ideen

Um die Erforschung der 'XML Schemaevolution' weiterzuführen, sollten folgende Punkte betrachtet werden:

- Eine Schemaevolutionssprache sollte entwickelt werden, die dem abstrakten Datenmodell von XML-Schema zugrunde liegt. Derzeitige Schemaevolutionssprachen verwenden die Repräsentationsform in XML, um die Evolutionsschritte zu beschreiben.

- Neben der Schemaevolutionssprache stellt auch das Problem des Auffinden der gesuchten Typdefinition und Deklaration ein Problem dar. In allen bekannten Vorschlägen wird XPath benutzt. Deshalb sollte eine

Navigationssprache entwickelt werden, die speziell für das Datenmodell von XML-Schema ist.

- Die Entwicklung einer formalen Beschreibung für alle Umformungen.

Verzeichnis der Abkürzungen

ACID	**A**tomicity **C**onsistency **I**solation **D**urability
CSS	**C**ascading **S**tyle **S**heet
DOM	**D**ocument **O**bject **M**odel
DTD	**D**ocument **T**ype **D**efinition
FLWOR	**F**or **L**et **W**here **O**rder by **R**eturn
GUI	**G**raphical **U**ser **I**nterface
IDF	**I**nterface **D**efinition **L**anguage
ISO	**I**nternational **O**rganization for **S**tandardization
JDK	**J**ava **D**evelopment **K**it
ODMG	**O**bject **D**ata **M**anagement **G**roup
OEM	**O**bject **E**xchange **M**odel
OMG	**O**bject **M**anagment **G**roup
OQL	**O**bject **Q**uery **L**anguage
PI	**P**rocessing **I**nstruction
RELAX	**Re**gular **La**nguage Description for **X**ML
SAX	**S**imple **A**PI for **X**ML
SGML	**S**tandard **G**eneralized **M**arkup **L**anguage
SQL	**S**tructured **Q**uery for **L**anguage
TREX	**T**ree **R**egular **E**xpressions for **X**ML
UML	**U**nified **M**odelling **L**anguage
W3C	**W**orld **W**ide **W**eb **C**onsortium
XDM	**X**Query 1.0 and XPath 2.0 **D**ata **M**odel
XEM	**X**ML **E**volution **M**anagement
XML	e**X**tensible **M**arkup **L**anguage
XML-GL	**XML G**raphical **L**anguage
XPath	**X**ML **Path** Language
XQuery	**X**ML **Query** Language
XSD	**X**ML **S**chema **D**efinition
XSL	e**X**tensible **S**tylesheet **L**anguage
XSL-FO	e**X**tensible **S**tylesheet **L**anguage **F**ormatting **O**bjects
XSLT	e**X**tensible **S**tylesheet **L**anguage **T**ransformation

XUL	**X**ML **U**pdate **L**anguage
XUpdate	**X**ML **Update** Language

Abbildungsverzeichnis

1.1 Aufbau der XML Arbeitsgruppen 2
1.2 Beziehungen der Sprachen 3

2.1 Beispiel eine FLWOR-Anweisung 17
2.2 Beispiel eines XSLT-Stylesheets 20

4.1 Darstellung der Primären Komponenten 35
4.2 Darstellung der Hilfskomponenten 36

5.1 Beziehung von Element-Deklaration und Attribut-Deklaration 40
5.2 Einfache Typdefinition wird um ein Attribut erweitert 42
5.3 Attribut-Verwendung wird optional und fester Attributwert entfernt 42
5.4 Typedefinition wird angepasst 43
5.5 Typdefinition mit neuer Attribut-Deklaration erweitern 44
5.6 Umwandelung der einfachen Typedefinition 48
5.7 Anpassen der Partikeleigenschaft 49

6.1 Aufbau des XEUpdate-Prozessors 55
6.2 Beispiel der GUI 57
6.3 XML-Schema für das Beispiel 58
6.4 XML-Instanz für das Beispiel 58
6.5 Updateanweisung für das Beispiel 59
6.6 Abgeleiteter Evolutionsschritt 60

Literaturverzeichnis

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. *The Lorel Query Language for Semistructured Data*. Int. Journal on Digital Libraries, 1997.

[BDH+04] Beatrice Bouchou, Denio Duarte, Mirian Halfeld, Ferrari Alves, Dominique Laurent, and Martin A. Musicante. *Schema Evolution for XML: A Consistency-Preserving Approach*. Mathematical Foundations of Computer Science, 2004.

[BL01] A. Bonifati and D. Lee. *Technical Survey of XML Schema and Query Languages*. Technical Report, UCLA Computer Science Dept., citeseer.ist.psu.edu/bonifati01technical.html, 2001.

[BOSdL05] Kevin Beyer, Fatma Oezcan, Sundar Saiprasad, and Bert Van der Linden. *DB2/XML: Designing for Evolution*. IBM, 2005.

[CJR98] Kajal T. Claypool, Jing Jin, and Elke A. Rundensteiner. SERF: Schema evolution through an extensible re-usable and flexible framework. pages 314–321, 1998.

[Cla99] James Clark. *XSL Transformations (XSLT)*. W3C, http://www.w3.org/TR/1999/REC-xslt-19991116, 1999.

[CR05] Don Chamberlin and Jonathan Robie. *XQuery Update Facility Requirements (Working Draft)*. W3C, http://www.w3.org/TR/2005/WD-xquery-update-requirements-20050603, 2005.

[CT04] John Cowan and Richard Tobin. *XML Information Set (Second Edition)*. W3C, http://www.w3.org/TR/xml-infoset, 2004.

[DOM98] *Document Object Model (DOM) Level 1 Specification*. W3C, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, 1998.

[DOM04] *Document Object Model (DOM) Level 3 Core Specification.* W3C, http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407, 2004.

[GMR05] Giovanna Guerrini, Marco Mesiti, and Daniele Rossi. *Impact of XML Schema Evolution on Valid Documents.* WIDM'05, 2005.

[Gol96] Charles F. Goldfarb. *The Roots of SGML – A Personal Recollection.* http://www.sgmlsource.com/history/roots.htm, 1996.

[HS00] Andreas Heuer and Gunter Saake. *Datenbanken: Konzepte und Sprachen.* mitp, 2000.

[Hän02] Birger Hänsel. *Änderungsoperationen in XML-Anfragesprachen.* Universität Rostock, Diplomarbeit, 2002.

[Kep04] Stephan Kepser. *A Simple Proof for the Turing-Completeness of XSLT and XQuery.* Extreme Markup Languages 2004, 2004.

[KMS02] Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. The DSD schema language. *Automated Software Engineering*, 9(3):285–319, 2002. Kluwer. Earlier version in Proc. 3rd ACM SIGPLAN-SIGSOFT Workshop on Formal Methods in Software Practice, FMSP '00.

[Leh01] Patrick Lehti. *Design and Implementation of a Data Manipulation Processor for an XML Query Language.* Technische Universität Darmstadt, Diplomarbeit, 2001.

[LM00] Andreas Laux and Lars Martin. *XUpdate - XML Update Language.* XML:DB Initiative for XML Databases, 2000.

[LOR99] *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language.* Stanford University, http://www-db.stanford.edu/lore/pubs/xml.pdf, 1999.

[Mar00] Lars Martin. *Requirements for XML Update Language.* XML:DB Initiative for XML Databases, 2000.

[Meg98] David Megginson. *Simple API for XML.* XML-DEV mailing list, http://www.saxproject.org, 1998.

[MEI05] *Evolution - The Reverse Side of the XML Update Coin.* Database Research Group University of Rostock, 2005.

[REL01] *RELAX NG Specification.* OASIS Open, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, 2001.

[SCH04a] *Final Committee Draft of ISO Schematron.* www.schematron.com, http://www.schematron.com/iso/dsdl-3-fdis.pdf, 2004.

[SCH04b] *XML Schema Part 0: Primer Second Edition.* W3C, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028, 2004.

[SCH04c] *XML Schema Part 1: Structures Second Edition.* W3C, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028, 2004.

[SKR02] Hong Su, Diane K. Kramer, and Elke A. Rundensteiner. *XEM: XML Evolution Management.* Worcester Polytechnic Institute, 2002.

[Tie05] Tobias Tiedt. *Schemaevolution und Adaption von XML-Dokumenten und XQuery-Anfragen.* Universität Rostock, Diplomarbeit, 2005.

[TIHW01] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Updating XML. In *SIGMOD Conference*, citeseer.ist.psu.edu/tatarinov01updating.html, 2001.

[W3C98] *The World Wide Web Consortium Issues XML 1.0 as a W3C Recommendation.* W3C, http://www.w3.org/Press/1998/XML10-REC-fact, 1998.

[W3C00] *A Little History of the World Wide Web.* W3C, http://www.w3.org/History.html, 2000.

[WIK05] *Evolution.* Wikipedia, http://en.wikipedia.org/wiki/Evolution, 2005.

[XML98] *XML-QL: A Query Language for XML.* W3C, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, 1998.

[XML99] *XML-GL: a Graphical Language for Querying and Restructuring XML Documents.* Università di Milano, http://www8.org/w8-papers/1c-xml/xml-gl/xml-gl.html, 1999.

[XML04] *Extensible Markup Language (XML) 1.0 (Third Edition).* W3C, http://www.w3.org/TR/2004/REC-xml-20040204, 2004.

[XPA99] *XML Path Language (XPATH) Version 1.0.* W3C, http://www.w3.org/TR/1999/REC-xpath-19991116, 1999.

[XPA05] *XML Path Language (XPATH) 2.0.* W3C, http://www.w3.org/TR/2005/CR-xpath20-20051103, 2005.

[XQL98] *XML Query Language (XQL).* W3C, http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[XQU05a] *XML Syntax for XQuery 1.0 (XQueryX).* W3C, http://www.w3.org/TR/2005/CR-xqueryx-20051103, 2005.

[XQU05b] *XQuery 1.0: An XML Query Language.* W3C, http://www.w3.org/TR/2005/CR-xquery-20051103, 2005.

[XQU05c] *XQuery 1.0 and XPath 2.0 Data Model (XDM).* W3C, http://www.w3.org/TR/2005/CR-xpath-datamodel-20051103, 2005.

[XQU05d] *XQuery 1.0 and XPath 2.0 Formal Semantics.* W3C, http://www.w3.org/TR/2005/CR-xquery-semantics-20051103, 2005.

[XQU05e] *XQuery 1.0 and XPath 2.0 Full-Text.* W3C, http://www.w3.org/TR/2005/WD-xquery-full-text-20051103, 2005.

[XQU05f] *XQuery 1.0 and XPath 2.0 Functions and Operators.* W3C, http://www.w3.org/TR/2005/CR-xpath-functions-20051103, 2005.

[XQU05g] *XSLT 2.0 and XQuery 1.0 Serialization.* W3C, http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20051103, 2005.

[XSL01] *Extensible Stylesheet Language (XSL) Version 1.0.* W3C, http://www.w3.org/TR/2001/REC-xsl-20011015, 2001.

[Zei01] Andre Zeitz. *Evolution von XML-Dokumenten.* Universität Rostock, Studienarbeit, 2001.