Evolution von XML-Dokumenten

Studienarbeit

Andre Zeitz
Universität Rostock
Fachbereich Informatik, LS DBIS
Albert-Einstein-Straße 21
D-18051 Rostock

Zusammenfassung

Eine typische Form semistrukturierter Daten sind XML-Dokumente, deren Verbreitung seit der Verabschiedung des XML-Standards 1998 stark zugenommen hat. Die Beschreibung der Struktur von XML-Dokumenten wird durch Verwendung einer Dokumententyp Definition (DTD) ermöglicht. Dokumente, deren Struktur einer DTD entspricht, heißen gültige Dokumente. Ändert sich die Struktur einer DTD, so ist es wünschenswert, diese Änderungen auf den XML-Dokumenten nachzuvollziehen, die gültig bezüglich dieser DTD sind. Derartige Umformungen werden als Evolution bezeichnet.

Diese Arbeit zeigt, welche Umformungen einer DTD welche Auswirkungen auf bestehende XML-Dokumente haben. Im Mittelpunkt steht dabei die Frage, welche Umformungen generell möglich und an welche Randbedingungen diese gekoppelt sind, insbesondere um die Gültigkeit der Dokumente nicht zu verletzen und den Datenverlust minimal zu halten. Weiterhin wird der Entwurf einer Architektur präsentiert, mit der Umformungen von DTDs und dazugehörigen Dokumenten automatisiert und unter Verwendung von XSLT durchgeführt werden können. Abschließend werden einzelne Umformungsoperationen an einem geeigneten Datenmodell beschrieben.

Abstract

XML documents are a typical kind of semistructured data. The usage of XML has increased significantly since 1998, when the XML recommendation was released. The structure of XML documents can be described using a Document Type Definition (DTD). Documents are called valid documents if their structure corresponds to a DTD. If the structure of a DTD changes, it is desirable to apply these changes to documentes with respect to this DTD. These transformations are called evolution.

This paper shows how different changes to the DTD effect XML documents. The focus will be on the question what kinds of transformations are possible and to which conditions they are bound especially to preserve the validity of the documents and to minimize data loss. Furthermore, this paper presents a draft of a system architecture for transforming DTDs and XML documents valid to these DTDs using XSLT. Finally, some transformations are described using an appropriate data model.

Inhalt

1 Einleitung

1.1 Motivation

Die effiziente Verwaltung von großen Datenmengen erfordert die Strukturierung dieser Daten in einem Schema, so daß eine bestimmte Menge von Daten einer gewissen Struktur unterliegt. Diese Struktur ist bisher in den meisten Fällen starr. Soll sie verändert werden, so sind große Anstrengungen nötig.

XML-Dokumente können als Vertreter der semistruturierten Daten ebenfalls einem Schema unterliegen, dessen Strukturvorgaben sie entsprechen. Aufgrund der Tatsache, daß für ähnliche oder gar identische Anwendungsgebiete unterschiedliche XML-Schemata erstellt und benutzt werden können, entsteht der Wunsch, Dokumente, die jeweils einem Schema entsprechen so umzuformen, daß sie einem anderen Schema unterliegen. Ziel dieser Arbeit soll es deshalb sein, mögliche Umformungen eines Schemas daraufhin zu untersuchen, unter welchen Bedingungen sie möglich und in welcher Weise sie durchführbar sind. Zur Beschreibung der umzuformenden Schemata soll die im XML-Standard verankerte Document Type Definition (DTD) dienen, da sie einfach ist und ihr ein abgeschlossener Standardisierungsprozeß zugrunde liegt.

Gegenstand der Betrachtungen sind also Umformungen einer DTD und die damit verbundenen Änderungen der XML-Dokumente, die gültig bezüglich dieser DTDs sind. Nach Durchführung der Umformungsoperationen sollen die angepaßten Dokumente, wie in Abbildung 1.1 dargestellt, gültig bezüglich der veränderten DTD sein.

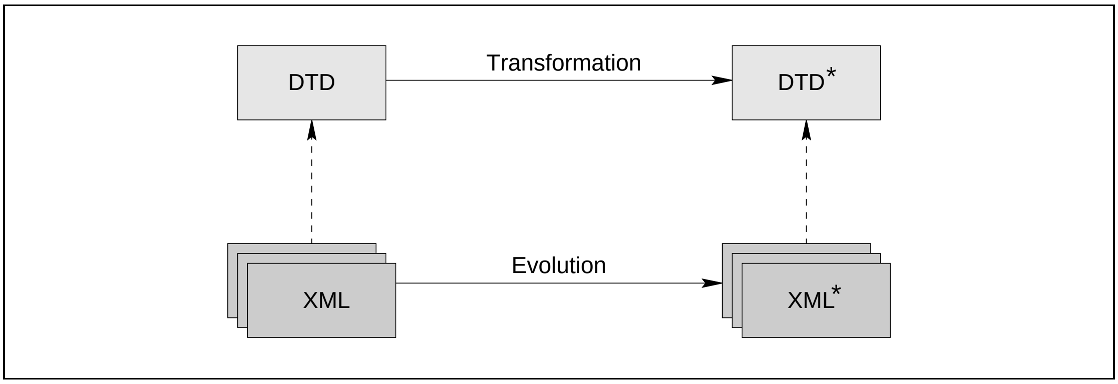

Abbildung 1.1: Szenario eines Evolutionsschrittes

1.2 Aufbau der Arbeit

Ziel dieser Arbeit ist es, Umformungen von XML-Schemata in Form von DTDs und von XML-Dokumenten zu betrachten, deren Struktur einer solchen DTD entspricht, so daß nach Ausführung aller Änderungsoperationen die veränderten XML-Dokumente gültig bezüglich der tranformierten DTD sind.

Die theoretischen Grundlagen, die für das Verständnis dieser Arbeit notwendig sind, werden in **Kapitel 2 (Theoretische Grundlagen)** erläutert. Dort werden zunächst semistrukturierte Daten und XML eingeführt und miteinander in Verbindung gebracht. Insbesondere wird der Begriff der Informationskapazität von XML-Schemata eingeführt. Anschließend erfolgt eine Kurzeinführung in XSLT, womit sich beispielsweise die oben erwähnten Transformationen auf XML-Dokumenten durchführen lassen. Zur Beschreibung von XML-Dokumenten und von Operationen oder besser Änderungsoperationen auf diesen Dokumenten bietet sich die Verwendung eines geeigneten Modells an. Deshalb wird zuerst das *Object Exchange Model* (OEM) charakterisiert, das zur Darstellung semistrukturierter Daten geschaffen wurde. Schließlich wird ein weiteres Modell eingeführt, welches durch Erweiterung von OEM hervorgeht und an die Besonderheiten von XML angepaßt ist.

In **Kapitel 3 (Evolution)**, dem Schwerpunkt der Arbeit, wird nach einer Erläuterung des Evolutionsbegriffs untersucht, welche Änderungen von DTDs welche Auswirkungen auf XML-Dokumente haben, deren Struktur einer solchen DTD entspricht. Dazu werden die betrachteten Operationen in atomare und komplexe Operationen unterteilt und danach klassifiziert, auf welchen Bestandteilen von XML sie ausgeführt werden bzw. ob sie eine Änderung des Datenbestandes oder der Informationskapazität der betrachteten Dokumente nach sich ziehen.

Ein Konzept für den Aufbau eines Systems, das die in Kapitel 3 beschriebenen Operationen auf DTDs und XML-Dokumenten durchführt, wird in **Kapitel 4 (Systementwurf)** erstellt. Dabei liegt der Schwerpunkt auf einer allgemeinen Konzeption, Implementierungsdetails werden nicht betrachtet.

In **Kapitel 5 (Transformationsbeschreibung)** wird der Versuch unternommen, einige der in Kapitel 3 beschriebenen Operationen an einem erweiterten OE-Modell zu erläutern, das in Kapitel 2 eingeführt wurde. Es ist deshalb nur ein Versuch, weil dieses Modell einerseits nur eine vereinfachte Darstellung von XML zuläßt und sich damit andererseits keine Bedingungen überprüfen lassen, die zur Durchführung der Umformungen erfüllt sein müssen. Trotzdem wird dadurch ein möglicher Anfang zur formalen Beschreibung der Änderungsoperationen gemacht.

Schließlich zeigt **Kapitel 6 (Ausblick)** einen kurzen Ausblick auf Themen auf, die sich auf der Basis dieser Arbeit realisieren lassen bzw. die eine Weiterführung dieser Arbeit darstellen.

2 Theoretische Grundlagen

Dieses Kapitel bietet eine Einführung in die theoretischen Grundlagen, die zum Verständnis der Arbeit notwendig sind. Dazu werden zunächst semistrukturierte Daten und danach XML als ein Vetreter der semistrukturierten Daten eingeführt. Im Anschluß daran wird mit XSLT eine Möglichkeit zur Transformation von XML-Dokumenten erläutert, auf die später zurückgegriffen wird. Abschließend werden ein Modell zur Darstellung semistrukturierter Daten und eine Erweiterung davon eingeführt, die speziell zur Darstellung von XML dient.

2.1 Semistrukturierte Daten

Semistrukturierte Daten sind in den letzten Jahren immer mehr in das Interesse der Forschung, insbesondere der Datenbankforschung gerückt. Gründe dafür sind vor allem die fast dramatische Ausdehnung des Internet und die damit verbundene Herausforderung, derartige Daten in Datenbanken zu integrieren. Die überraschende Verbreitung von XML im letzten Jahr unterstützte die Fokusierung der Datenbankforscher auf semistrukturierten Daten zusätzlich. Nachfolgend sollen nun semistrukturierte Daten (weitestgehend nach [Abi97]) genauer charakteriesiert werden.

Semistrukturierte Daten sind durch folgende oder einen Teil der folgenden Merkmale gekennzeichnet:

Die Datenstruktur ist unregelmäßig: Semistrukturierte Daten sind oft heterogen. Dadurch können Bestandteile solcher Daten unvollständig sein, andere enthalten wiederum Zusatzinformationen. Weiterhin werden ähnliche oder gleiche Informationen unterschiedlich strukturiert bzw. mit verschiedenen Datentypen modelliert.

Die Struktur ist implizit: Die Datenstruktur kann aus den Daten abgeleitet werden. Dabei ist jedoch weder garantiert, daß wirklich das komplette Schema abgeleitet noch daß das abgeleitete Schema eindeutig ist, was durch die Heterogenität der Daten bedingt ist. Das bedeutet, daß u.U. aus einem Datensatz, der semistrukturiert ist, verschiedene Schemata abgeleitet werden können.

Die Struktur ist unvollständig: Aufgrund der Vielfalt semistrukturierter Daten ist es kaum oder sogar überhaupt nicht möglich, die Struktur solcher Daten zu beschreiben.

Das Schema ist schwach typisiert: Die Menge der Datentypen, mit denen semistrukturierte Daten modelliert werden, ist nur schwach ausgeprägt. Dies bedeutet zwar einerseits den

Verlust an Korrektheit, bietet aber andererseits den Autoren derartiger Daten ein großes Maß an Freiheit.

Das Schema kann a-posteriori definiert werden: Im Gegensatz zu traditionellen Datenbanken muß für semistrukturierte Daten das Schema nicht feststehen, bevor die Daten angelegt werden. Während bei Datenbanken ein Schema fest ist, ändert es sich bei semistrukturierten Daten häufig.

Das Schema ist sehr groß: Als Konsequenz der Heterogenität ist das Schema sehr groß. Es kann teilweise so groß wie die Datenmenge selbst und unter Umständen sogar größer sein. Schemata traditioneller Datenbanken sind dagegen relativ klein.

Das Schema wird ignoriert: Für bestimmte Anfragen auf semistrukturierten Daten wird deren Schema ignoriert. Diese Anfragen dienen meist dem einfachen "Durchstöbern" der Daten, wozu ein Schema nicht benötigt wird.

Das Schema unterliegt häufigen Änderungen: Während in Standarddatenbanksystemen das Schema fix ist, kann es bei semistrukturierten Daten geändert werden. Diese Schemänderungen werden darüber hinaus sehr oft durchgeführt.

Das Schema ist vielseitig: Je nach Sicht oder je nach Zeitpunkt der Datenakquisition kann die Struktur der Daten variieren.

Der Unterschied zwischen Schema und Daten ist unscharf: Der Grundsatz aus Standarddatenbanksystemen, daß Schema und Daten klar voneinander getrennt sind, gilt nicht bei semistrukturierten Daten. Gründe dafür sind beispielsweise, daß sich das Schema genau wie die Daten ändern kann und geändert wird oder das Schema sehr groß sein oder gar ignoriert werden kann.

2.2 Extensible Markup Language (XML)

XML [BPSM98, BPSMM00] ist eine Meta-Sprache, die einen Standard zur Dokumentenverarbeitung darstellt und 1998 vom World Wide Web Consortium (W3C) verabschiedet wurde. Aufgrund ihrer Beschaffenheit qualifiziert sich diese Sprache insbesondere zum Austausch und zur Speicherung von semistrukturierten Daten. Nachfolgend soll ihr Aufbau kurz dargestellt werden.

2.2.1 Bestandteile der Sprache

Den wichtigsten Teil von XML stellt das XML-Dokument (Instanz) dar. Ein Dokument besteht aus Elementen. Elemente können selber wiederum Elemente und darüberhinaus auch Attribute und Werte enthalten. Die genaue Struktur eines konkreten Dokumentes wird durch ein Schema beschrieben. Die Analyse eines XML-Dokumentes wird von einem XML-Prozessor durchgeführt, wobei dieser die Struktur des Dokumentes überprüft, um dieses dann letztendlich zu verarbeiten.

Elemente

Elemente stellen den Grundbestandteil eines XML-Dokumentes dar. Ein Element besteht immer aus einem Paar von Tags, einem öffnenden und einem schließenden Tag. Alles, was sich zwischen dem öffnenden und dem schließenden Tag befindet, ist Bestandteil des Elementes. Ein Ausnahme stellt das leere Element dar, da dieses aus nur einem Tag bestehen kann[1] und keine weiteren Elemente enthält. Die Bestandteile eines Elementes können Attribute, Werte oder auch wieder Elemente sein.

Enthält ein Element Kind-Elemente, dann können die Struktur des Elementes und die Quantifizierung der Kind-Elemente mit folgenden Hilfsmitteln beschrieben werden:

Sequenz: Ein Element besteht aus mehreren Elementen,[2] deren Reihenfolge fest vorgeschrieben ist. Die Häufigkeit des Auftretens dieser Kind-Elemente in der Seqeuenz ist von deren Quantor abhängig. Die Häufigkeit des Auftretens der Sequenz selbst kann ebenfalls mit einem Quantor festgelegt werden, der an diese Sequenz gebunden wird.

Alternative: Ein Element besteht aus einem oder mehreren Subelementen. Mit Hilfe von Quantoren kann bestimmt werden, wie oft welche Kind-Elemente bzw. wie oft die Alternative/Sequenz im Vater-Element vorkommen darf.

Mixed content Type: Ein Element kann aus Elementen und Werten gleichermaßen bestehen. Ein Element dieses Typs kann aber auch nur aus Werten oder nur aus Subelementen bestehen.

Quantoren können entweder einzelne Elemente oder ganze Elementgruppen (Sequenz, Alternative) an sich binden. Mögliche Qantoren sind:

$+$ n-maliges Auftreten, $n \geq 1$

$*$ n-maliges Auftreten, $n \geq 0$

? optionales Auftreten

Besitzt ein Element keinen Quantor, so muß das Element, die Alternative oder die Sequenz genau einmal innerhalb des Vater-Elementes vorkommen. Diese Einschränkung kann auch als Quantor 1 bezeichnet werden.

Jedes Element darf nur genau einmal definiert werden, mehrmalige Definitionen sind unzulässig.

[1] Ein leeres Element kann auch aus einem öffnendem und einem schließendem Tag bestehen

[2] Eine Sequenz kann neben Elementen auch aus Alterntiven und weiteren Sequenzen bestehen, wobei die Tiefe solcher Schachtelungen beliebig ist. Gleiches gilt für die Alternative.

Reservierte Elemente

Alle Elemente, deren Name mit “xml:” beginnt, sind laut [BPSM98, BPSMM00] reserviert, und sollten nicht neu definiert werden. Sie gehören dem sogenannten XML-Namensraum (XML Namespace, [BHL99]) an.

Attribute

Zur genaueren Beschreibung eines Elementes dienen Attribute. Jedes Element kann eine Liste von Attributen enthalten, wobei an diese Liste besondere Anforderungen gestellt werden. Jedes Attribut einer Attributliste bestitzt einen Namen, einen Typ, einen Wert dieses Typs und eine *Default Declaration*. Ein Attribut wird über seinen Namen identifiziert. Mit der *Default Declaration* wird festgelegt, ob und wie ein Attribut im Dokument als Bestandteil eines Elementes vorkommen muß. Dabei wird zwischen den folgenden vier Varianten unterschieden:

Attributwert: Wird bei der Definition eines Attributes für die *Default Declaration* nur ein Attributwert angegeben, dann ist die Angabe dieses Attributes im Element eines Dokumentes nicht zwingend erforderlich. Ist im Dokument kein Attribut angegeben, so besitzt das entsprechende Element trotzdem dieses Attribut, wobei sich dessen Wert aus dem Attributwert der *Default Declaration* ergibt. Dieser Wert stellt dann einen voreingestellten Wert (Default-Wert) dar. Wird dagegen im Dokument ein Attribut angegeben, dann gilt dessen Wert.

FIXED: Wird ein Attribut als FIXED definiert, dann ist dieses Attribut in allen Elementen anzugeben, zu denen es gehört. Der Wert, der für alle diese Attribute im Dokument anzugeben ist, wird durch die Attributdefinition festgelegt. D.h. alle diese Attribute haben denselben Wert im Dokument.

IMPLIED: Attribute, die als IMPLIED definiert wurden, können optional im Dokument auftreten. Treten sie im Dokument auf, so ist der Wert unter Einhaltung der lexikalischen Anforderungen an Attributwerte bezüglich des Attributtyps frei wählbar.

REQUIRED: Ein als REQUIRED definiertes Attribut muß zwingend in allen zu diesem Attribut gehörenden Elementen angegeben werden. Deren Wert ist unter Einhaltung der lexikalischen Anforderungen an Attributwerte bezüglich des Attributtyps frei wählbar.

Wird eine Attributliste für ein Element mehrfach definiert, so werden diese Listen intern gemischt, d.h. es existiert nur eine Atributliste, welche alle Attribute enthält, die in den einzelnen Attributlisten eines Elementes definiert wurden. Wird in solchen Listen ein Attribut mehrmals definiert, so wird nur das Attribut beachtet, das zuerst definiert wurde.

Der Wert eines Attributes besteht aus einer Zeichenkette und unterliegt gewissen Einschränkungen, die vom Typ des Attributes abhängen. Die möglichen Attributtypen sind:

CDATA: Ein Attribut vom Typ CDATA darf nahezu alle Zeichen enthalten, inklusive des Leerzeichens. Dabei zählen auch Leerzeichen vor und hinter der Zeichenkette mit zum Wert. CDATA stellt den allgemeinsten aller Attributtypen dar. In einem XML-Dokument, das an keine DTD gebunden ist, dürfen alle Attribute nur von diesem Typ sein.

ENTITY/ENTITIES: Der Wert eines Attributes vom Typ ENTITY enthält den Namen einer *unparsed Entity*[3], die in der DTD des Dokumentes definiert sein muß. Innerhalb des Attributwertes darf kein Leerzeichen vorhanden sein. Der Attributtyp ENTITIES stellt eine Erweiterung zum Typ ENTITY dar. Ein Wert dieses erweiterten Typs enthält mehrere Namen von unparsed Entities, die in der DTD des Dokumentes definiert sein müssen. Dabei werden die einzelnen Namen durch Leerzeichen voneinander getrennt.

Enumeration: Der Aufzählungstyp ermöglicht die Einschränkung des Attributwertes auf einen bestimmten Wertebereich, wobei die einzelnen Werte dieses Wertebereichs aufgezählt werden. Ein Wert dieses Typs darf keine Leerzeichen enthalten.

ID: Mit einem ID-Attribut kann ein Element eindeutig innerhalb eines Dokumentes bezeichnet werden, d.h. der Wert einer ID darf in einem Dokument nur genau einmal vergeben werden. Außerdem darf jedes Element höchstens ein ID-Attribut besitzen. In einem ID-Attribut dürfen keine Leerzeichen vorkommen.

IDREF/IDREFS: Der Wert eines IDREF-Attributes ist immer identisch zum Wert einer ID innerhalb des Dokumentes. Mit diesem Typ lassen sich Referenzen bzw. einfache Fremdschlüsselbeziehungen[4] nachbilden. Im Attributwert eines solchen Typs dürfen keine Leerzeichen vorkommen. Der Typ IDREFS erweitert den Attributtyp IDREF. Ein IDREFS-Attribut enthält mehrere, durch Leerzeichen voneinander getrennte ID-Werte, die allesamt im Dokument vorkommen müssen.

Notation Type: Der Wert eines Attributes diesen Typs beinhaltet den Namen einer Notation (siehe unten), die in der DTD des Dokumentes definiert sein muß. Er darf keine Leerzeichen enthalten.

NMTOKEN/NMTOKENS: Der Wert eines Named-Token-Attributes ist eine einfache Zeichenkette ohne Leerzeichen. Der Wert eines NMTOKENS-Attribut darf dagegen mehrere aus einfachen Zeichenketten bestehende Werte enthalten, die mit Leerzeichen voneinander getrennt sein müssen.

Für alle Attributtypen, mit Ausnahme von CDATA gilt, daß Leerzeichen vor und nach dem Attributwert vom XML-Prozessor ignoriert werden.

Werte

Werte dienen in XML zur Beschreibung von Elementen. Sie sind Bestandteil von Elementen und können aus nahezu beliebigen Zeichenketten bestehen. Werte dürfen kein Markup, dafür jedoch Entity-Referenzen enthalten.

[3] vom XML-Prozessor nicht zu verarbeitende Entität

[4] Es lassen sich nur einfache Fremdschlüsselbedingungen nachbilden, da mit diesem Typ immer nur auf einen Attributwert verwiesen werden kann

Entitäten

Entitäten dienen zur vereinfachten Beschreibung von eigenständigen Dingen in XML. Es besteht die Möglichkeit, via Referenz auf Entitäten zu verweisen. XML kennt die folgenden verschiedenen Entitätstypen:

Allgemeine Entitäten oder auch Text-Entitäten dienen zur einfachen Substitution von Zeichen innerhalb eines Dokumentes. Sie werden vor allem eingesetzt, um immer wiederkehrende Textpassagen bzw. Werte abzukürzen. Weiterhin können mit allgemeinen Entitäten einzelne Zeichen einer Zeichentabelle referenziert werden.

Parameter-Entitäten werden, ähnlich wie bei allgemeinen Entitäten, dazu benutzt, Texte zu substituieren. Der Unterschied besteht darin, daß Parameter-Entitäten ausschließlich innerhalb von DTDs referenziert werden dürfen. Sie werden somit entweder dazu benutzt, häufig auftretende Definitionen in einer DTD abzukürzen oder externe DTDs in eine DTD einzufügen.

Externe Entitäten ermöglichen es, Inhalte von externen Dateien in ein XML-Dokument zu kopieren. Dabei wird, wie auch bei den anderen Entitäten, eine Substitution vorgenommen, wobei eine Referenz auf ein externes Dokument beim Verarbeiten des XML-Dokumentes durch den Inhalt dieses Dokumentes ersetzt wird.

Nicht analysierte Entitäten bzw. *Unparsed Entities* dienen zur Referenzierung von externen Dateien, deren Inhalt nicht vom XML-Prozessor verarbeitet wird. Verwendung findet dieser Typ von Entities bei der Referenzierung von Dateien im Binärformat.

Allgemeine, Externe und Parameter-Entitäten werden vom XML-Prozessor verarbeitet. Enthalten diese Entitäten Markup, so wird dieses beim Parsen durch den Prozessor analysiert und ausgewertet. Dabei müssen öffnendendes und schließendes Tag innerhalb derselben Entität vorkommen, was ausgeschließt, daß das öffndende Tag einer Eintität in einer anderen Entität geschlossen wird.

Für alle Typen von Entitäten gilt gleichermaßen, daß Rekursionen ausgeschlossen sind und somit keine Zyklen zwischen Entitäten entstehen können. Außerdem gilt, daß Entitäten mehrfach definiert werden dürfen. Dabei wird jedoch nur die erste Defnition einer Entität betrachtet, alle weiteren sind nicht relevant.

Weitere Sprachbestandteile

Ein weiterer Bestandteil von XML sind Notationen, die als Typbezeichner von nicht zu parsenden Entitäten verwendet werden, d.h. der Typ einer unparsed Entity wird mit Hilfe einer Notation beschrieben. Darüberhinaus können XML-Dokumente Kommentare (*Comments*) und Prozessoranweisungen (*Processing Instructions*) enthalten, mit denen sich spezielle Anweisungen für den XML-Prozessor formulieren lassen.

2.2.2 Beschreibung von Schemata

Die Struktur eines konkreten XML-Dokumentes kann auf verschiedenen Wegen beschrieben werden, beispielsweise mit Hilfe einer Dokumententyp Definition (Document Type Definition, DTD). In einer DTD wird festgelegt, wie die einzelnen Elemente eines Dokumentes aufgebaut sind, d.h. ob sie weitere Elemente und in welcher Kardinalität enthalten, welche Attribute von welchem Typ und mit welcher *Default Declaration* vorkommen. Außerdem können in einer DTD Entitäten und Notationen definiert werden, auf die aus dem XML-Dokument heraus zugegriffen werden kann. DTDs werden in einem eigenen Format beschrieben. Sie können direkt im XML-Dokument beschrieben werden, in einer separaten Datei oder beides. Außerdem können DTDs andere DTDs inkludieren. Werden DTDs aus verschiedenen anderen DTDs zusammengesetzt oder wird eine DTD sowohl im Dokument als auch extern definiert, so werden diese einzelnen DTDs gemischt. Alle Attributtypen, Default Declarations usw., die in diesem Abschnitt beschrieben wurden, entsprechen den Vorgaben für DTDs.

Eine weitere Möglichkeit zur Beschreibung ist *XML Schema*, was im Unterschied zu DTDs eine bessere Typisierung und eine genauere Angabe von Kardinalitäten zwischen Elementen und Subelementen ermöglicht. Außerdem erfolgt die Beschreibung eines *XML Schemas* in Form eines XML-Dokumentes.

Eine letzte Variante zur Schemabeschreibung von XML-Dokumenten ist das *Resource Description Framework*, dessen Ziel die Beschreibung der Semantik von Ressourcen ist. Ressourcen sind dabei beliebige Objekte, die eindeutig identifizierbar sind und gewisse Eigenschaften (Properties) besitzen. Properties wiederum haben typisierte Werte. Mit Hilfe von sogenannten Statements werden letztlich Ressourcen mit Properties und deren Werten bzw. verschiedene Ressourcen miteinander verknüpft.

Die am weitesten verbreitete Variante ist eine DTD, da diese einerseits fest im XML-Standard verankert ist und sich damit in einem abgschlossenen Standardisierungsstadium befindet. Andererseits sind DTDs sehr einfach gehalten. Aufgrund der Tatsache, daß XML-Schema sehr mächtig und auch relativ einfach ist, wird diese Form der Schemabeschreibung in Zukunft sicher an Bedeutung gewinnen.

2.2.3 Eigenschaften von XML-Dokumenten

Wohlgeformtheit

Ein Dokument wird als *wohlgeformt* bezeichnet, wenn zu jedem Start- ein End-Tag exisitiert (Ausnahme leeres Element), verschiedene Unterelemente korrekt inneinander verschachtelt sind, jeder Attributname in einer Attributliste nur einmal vergeben wird und in einem Dokument, zu dem keine DTD gehört, alle Attribute vom Typ CDATA sind.

Gültigkeit

Ein Dokument, das den Wohlgeformtheitsanforderungen genügt und dessen Struktur der Vorgabe einer DTD entspricht, heißt *gültig*.

2.2.4 Informationskapazität von XML-Schemata

Ein wichtiges Kriterium von XML-Schemata ist die Informationskapazität. Sie ist ein Maß dafür, welche Daten in welcher Quantität in einem XML-Dokument gespeichert werden können, das gültig bezüglich des betrachteten Schemas ist.

In [Hul84] wird der Begriff Informationskapazität für Schemata relationaler Datenbanken formal eingeführt. Nachfolgend soll dieser Begriff auf XML-Dokumente bzw. auf XML-Schemata in Form von DTDs abgewandelt werden.

Definition: Seien $\mathbf{P}$ und $\mathbf{Q}$ DTDs und $\mathcal{I}$ die Menge der Instanzen einer DTD, d.h. die Menge der XML-Dokumente, die gültig bezüglich dieser DTD sind. Seien außerdem ω und σ Abbildungen mit $\omega : \mathcal{I}(\mathbf{P}) \rightarrow \mathcal{I}(\mathbf{Q})$ (kurz: $\omega : \mathbf{P} \rightarrow \mathbf{Q}$) und $\sigma : \mathcal{I}(\mathbf{Q}) \rightarrow \mathcal{I}(\mathbf{P})$ (kurz: $\sigma : \mathbf{Q} \rightarrow \mathbf{P}$). Dann bestimmt $\mathbf{P}$ die DTD $\mathbf{Q}$ (Bezeichnung: $\mathbf{P} \preceq \mathbf{Q}$), wenn es Abbildungen ω und σ gibt, so daß jede Instanz von $\mathbf{P}$ durch ω auf $\mathbf{Q}$ und durch Anwendung von σ wieder auf $\mathbf{P}$ abgebildet werden kann. Das heißt, daß die Hintereinanderausführung von ω und σ auf einer beliebigen Instanz von $\mathbf{P}$ ergibt wieder die Originalinstanz, kurz: $(\omega \circ \sigma)(\mathcal{I}(\mathbf{P})) = \mathcal{I}(\mathbf{P})$.

Gilt $\mathbf{P} \preceq \mathbf{Q}$, so bedeutet dies, daß alle XML-Dokumente, die gültig bezüglich der DTD $\mathbf{P}$ sind, so transformiert werden können, daß sie gültig bezüglich $\mathbf{Q}$ sind, und daß sie zurücktransformiert werden können, so daß das wieder das Originaldokument entsteht. Insbesondere bedeutet dies auch, daß die Informationskapazität von $\mathbf{Q}$ mindestens so groß wie die von $\mathbf{P}$ ist.

Bemerkung: Gilt $\mathbf{P} \preceq \mathbf{Q}$ und $\mathbf{Q} \preceq \mathbf{P}$, so sind $\mathbf{P}$ und $\mathbf{Q}$ äquivalent (Bezeichnung: $\mathbf{P} \sim \mathbf{Q}$) und die Informationskapazitäten von $\mathbf{P}$ und $\mathbf{Q}$ sind identisch.

2.2.5 Semistrukturiertheit von XML

Nachfolgend soll gezeigt werden, warum XML-Dokumente zu den semistrukturierten Daten gehören. Dazu wird erläutert, welche der Eigenschaften semistrukturierter Daten aus Abschnitt 2.1 auf Seite 8 in welcher Weise auf XML zutreffen. Zur Beschreibung von Schemata werden hierbei DTDs verwendet.

Die Datenstruktur ist unregelmäßig: Durch Verwendung der Alternative und der Quantoren $*$, $+$ und ? ist es möglich, daß Elemente in einem XML-Dokument nur optional auftreten müssen. Auf Attributebene kann dies in der DTD durch Definition von IMPLIED-Attributen oder die Vergabe eines Default-Attributwertes geschehen. Die Optionalität von Elementen und Attributen bewirkt, daß sich die Datenstruktur mehrerer Dokumente, die allesamt gültig bezüglich ein und derselben DTD sind, stark voneinander unterscheidet.

Die Struktur ist implizit: Aus einem XML-Dokument kann eine DTD abgeleitet werden, so daß die Struktur des Dokumentes dieser DTD entspricht, das Dokument also gültig bezüglich dieser DTD ist. Aus einem Dokument können jedoch verschiedene DTDs abgeleitet werden, so daß das Dokument zu jeder dieser DTDs gültig ist. Die Ableitung einer DTD aus einem XML-Dokument ist nicht eindeutig.

Das Schema ist schwach typisiert: Der XML-Standard sieht für die Verwendung von DTDs nur sehr einfache Typen vor, deren Aufbau sich nur wenig voneinander unterscheidet. Eine Unterteilung in numerische und alphanumerische Daten ist beispielsweise mit diesen Datentypen nicht möglich.

Das Schema kann a-posteriori definiert werden: Aufgrund der Tatsache, daß DTD und XML-Dokument getrennt voneinander sein können (externe Definition der DTD) und die DTD eines Dokumentes nur dann benötigt wird, wenn das Dokument analysiert werden soll, kann die DTD unabhängig vom Dokument und vor allem später als das Dokument erstellt werden.

2.3 XSL Transformations (XSLT)

XSLT [Cla99] ist eine Standardisierungsempfehlung des W3C, die im November 1999 als Teil des XSL-Standards[5] verabschiedet wurde. Mit XSLT ist es möglich, XML-Dokumente zu transformieren. Das Ergebnis einer solchen Transformation kann wieder ein XML-Dokument sein, aber auch andere Formate können damit erzeugt werden. Dokumente werden für die Umwandlung als Baum interpretiert, Bestandteile von XML (Elemente, Attribute) werden als Knoten in diesem Baum betrachtet.

2.3.1 Transformationsablauf

Für die Durchführung einer Transformation mit Hilfe von XSLT werden neben dem umzuwandelnden XML-Dokument ein XSL-Stylesheet und ein XSLT-Prozessor benötigt. Der XSLT-Prozessor liest das zu wandelnde Dokument ein, überführt es in eine Baumstruktur und führt die Transformation nach den Regeln durch, die im Stylesheet enthalten sind (siehe Abbildung 2.1). Mit diesen Regeln wird ein Zielbaum aufgebaut, der nach der kompletten Bearbeitung des Dokumentes durch den XSLT-Prozessor serialisiert und schließlich ausgegeben wird, beispielsweise in eine Datei.

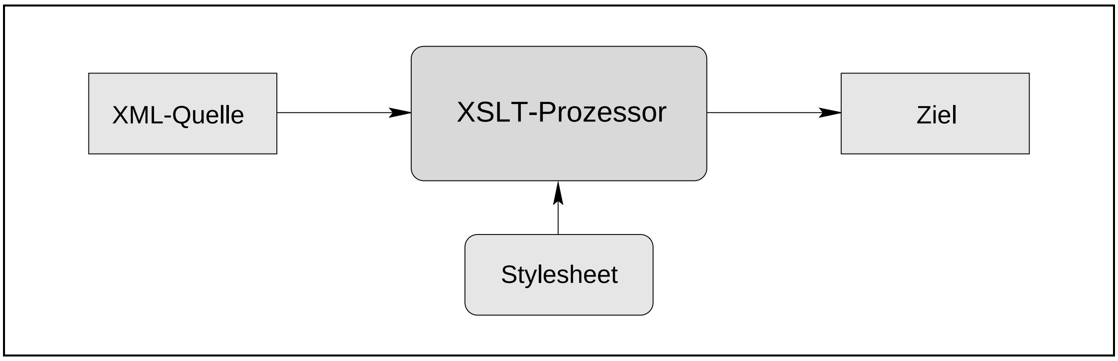

Abbildung 2.1: Transformation eines XML-Dokumentes

[5]Der XSL-Standard besteht aus XSLT, XPath und den *Formatting Objects*.

2.3.2 Aufbau eines Stylesheets

Ein XSL-Stylesheet ist ein XML-Dokument, das einzelne Schablonen (Templates) enthält. Ein Template enthält wiederum eine Menge von Anweisungen (Aktionen). Trifft der XSLT-Prozessor beim Einlesen des Dokumentes auf einen Abschnitt (einen Knoten), der auf ein Template paßt (Pattern Matching), dann wird dieses Template verarbeitet, d.h. es werden die Aktionen des Templates ausgeführt.

```
<xsl:template match="*[name()='abc']">
  <xsl:for-each select="@*">
    <xsl:copy/>
  </xsl:for-each>
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Abbildung 2.2: XSLT-Template: In diesem Template werden zuerst alle Attributknoten des aktuellen Knotens (Name *abc*) kopiert. Danach werden alle Knoten, die Subelemente des aktuellen Knotens sind, kopiert und rekursiv verarbeitet.

Die Knoten, auf die ein Template angewendet werden soll, werden mit Hilfe von XPath-Ausdrücken [CD99] adressiert (siehe Beispiel in Abbildung 2.2). Mit XPath wird der Pfad eines Knotens von einem anderen Knoten aus innerhalb des XML-Baumes beschrieben. XPath enthält darüberhinaus mit der sogenannten *Core Function Library* Routinen, die u.a. folgende Bestandteile hat:

- Routinen zur String-Verarbeitung
- Mathematische Routinen
- Funktionen, die Auskunft über Anzahl, Name oder Lage von Knoten geben

Zum Aufbau des Zielbaumes können Knoten jeglicher Art (Elemente, Attribute, Kommentare usw.) erzeugt werden. Dabei bietet XSLT Konstrukte in Form von vordefinierten Elementen und Attributen, die aus Programmiersprachen bekannt sind, wie:

- bedingte Ausführungen
- Definition und Benutzung von Variablen und Konstanten
- Iteration über Knotenmengen

Darüber hinaus können Elementknoten sortiert oder Kind-Elemente rekursiv verarbeitet werden. Alle Textknoten, deren zugehöriges Element nicht in einem Template verarbeitet wird, werden in den Ergebnisbaum übernommen. Im Anhang sind im Abschnitt B zur Veranschaulichung zwei vollständige XSL-Dateien abgebildet, Hinweise zu diesen Dateien sind dort ebenfalls zu finden.

2.3.3 Konfliktauflösung

Tritt die Situation auf, daß meherere Templates auf einen verabeiteten Knoten eines Dokumentes passen, dann muß eines dieser Templates zur Verarbeitung ausgewählt werden. Welches ausgewählt wird, hängt von den Prioritäten ab, die für jedes Template vergeben werden können. Alle Templates, für die keine Priorität vergeben wurde, erhalten eine Default-Priorität. Diese liegt je nach Aufbau des Template-Musters (Pattern) zwischen -0,5 und 0,5. Dabei wird immer das Template zur Verarbeitung ausgewählt, das die höchste Priorität besitzt. Es muß gewährleistet sein, daß niemals zwei verschiedene Templates mit demselben Muster auch dieselbe Priorität besitzen.

2.3.4 Sicherheit

Der XSLT-Standard schließt Endlosschleifen nicht aus, d.h. solche Endlosschleifen können bei der Abarbeitung von Templates auftreten. Es ist somit Aufgabe des Template-Autors, diese zu vermeiden.

2.3.5 Updatestrategie

Bei der Umformung von XML-Dokumenten mit XSLT werden stets neue Dokumente erzeugt. Dies bedeutet, daß die Originaldaten, also die umzuformenden Dokumente unverändert bleiben.

2.3.6 Einschränkungen

Bei der Benutzung von XSLT treten einige Einschränkungen auf, die es zu beachten gilt:

1. Der Zugriff auf die DTD eines Dokumentes ist nicht möglich. Trotzdem wird der Zugriff auf bestimmte Daten aus der DTD ermöglicht. Beispielsweise kann im Dokument explizit nach ID-Attributen gesucht werden.

2. Ein Zugriff auf geparste Entitäten ist nicht möglich, da diese vor ihrer Verarbeitung vollständig aufgelöst werden. Ungeparste Entitäten können dagegen referenziert werden, da sie nur innerhalb von Attributen vom Typ ENTITY oder ENTITIES auftreten dürfen.

2.4 Object Exchange Model (OEM)

Das *Object Exchange Model* wurde in [PGMW94] zur Beschreibung und Darstellung von semistrukturierten Daten eingeführt und soll nachfolgend erläutert werden.

2.4.1 Aufbau des Modells

Grundlage des OEM ist die OEM-Datenbank. Eine OEM-Datenbank O besteht aus (N, A, v, r), mit

N Menge von Objektidentifikatoren (Knoten)

A Menge von benannten, gerichteten Kanten, bestehend aus (p, l, c), mit $p, c \in N$ und l String

v Funktion, die jeden Knoten $n \in N$ entweder auf einen atomaren Wert (Integer, String, etc.) oder den komplexen Wert $\mathcal{C}$ abbildet

r Wurzelknoten (Root), $r \in N$.

Jeder Knoten, von dem keine Kanten ausgehen, heißt Blatt und besitzt einen atomaren Wert. Ein Knoten, von dem nur Kanten ausgehen, heißt Wurzelknoten. Alle Knoten, die keine Blätter sind, besitzen komplexe Werte.

Jeder Knoten, der vom Wurzelknoten aus über Kanten erreichbar ist, ist persistent (Persistenz durch Erreichbarkeit). Der Wurzelknoten selber ist immer persistent. In jeden Knoten dürfen mehrere Kante münden. Zyklen innerhalb der OEM-Datenbank sind zulässig.

2.4.2 Operatoren

Zum Aufbau und zur Manipulierung einer OEM-Datenbank sind Änderungsoperationen notwendig, die hier eingeführt werden sollen.

createNode(Identifier i, Value v) Erzeugen eines neuen Knotens. Dabei darf der Identifikator i noch nicht vergeben worden sein, der Wert v muß entweder atomar (Integer, String, etc.) oder der komplexe Wert $\mathcal{C}$ sein.

updateNode(Identifier i, Value v) Update eines Knotenwertes. Der neue Wert v muß entweder atomar (Integer, String, etc.) oder der komplexe Wert $\mathcal{C}$ sein.

createArc(Identifier i, Identifier j, Label l) Erzeugen einer neuen Kante vom Knoten i zum Knoten j. Mit l wird der Bezeichner (Label) der zu erzeugenden Kante angegeben.

deleteArc(Identifier i, Identifier j, Label l) Löschen der Kante mit dem Label l zwischen den Knoten mit den Identifikatoren i und j.

Knoten einer OEM-Datenbank sind genau dann persistent, wenn es zu ihnen einen Weg vom Wurzelknoten über vorhandene Kanten gibt (Persistenz durch Erreichbarkeit). Um einen neuen Knoten in die Datenbank aufzunehmen, muß eine Kante erzeugt werden, die von einem bereits persistenten Knoten zu diesem neuen Knoten führt. Um den Knoten zu löschen, muß lediglich diese Kante wieder entfernt werden. Im allgemeinen wird ein Knoten gelöscht, indem alle Kanten entfernt werden, die in diesen Knoten münden. Eine Operation zum Löschen eines Knotens wird dementsprechend nicht benötigt. In Abbildung 2.3 auf der nächsten Seite ist die graphische Darstellung einer OEM-Datenbank (OEM-Graph) abgedruckt.

2.4.3 Migration von OEM zu XML

Um XML-Daten als Spezialfall semistrukturierter Daten darzustellen, bietet sich die Modifikation des *Object Exchange Model* an. Eine Möglichkeit wird dazu in [GMW99] beschrieben. Dabei

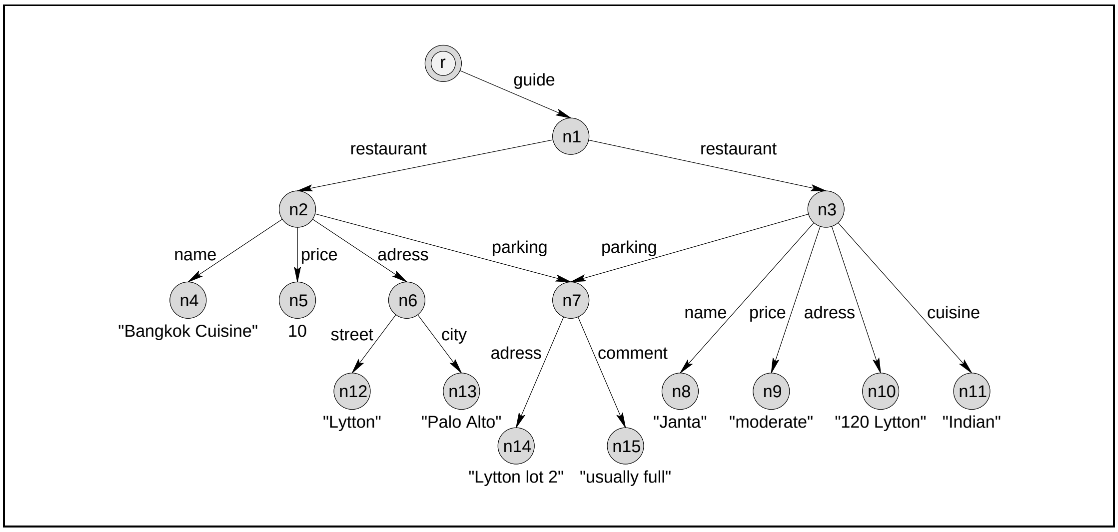

Abbildung 2.3: Beispiel eines OEM-Graphen

werden die Werte der Knoten um mehrere Strings erweitert, die zur Modellierung von Attributen dienen. Zum Zugriff auf diese Attributstruktur sind deshalb Operationen auf Zeichenketten notwendig. Graphenoperationen wie die oben vorgestellten können hierzu nicht benutzt werden, was im Grunde einen Stilbruch darstellt, da auf Elemente mit Operationen auf Graphen zugegriffen wird, der Zugriff auf Attribute aber den Einsatz von Strinoperationen verlangt. Nachfolgend soll deshalb eine andere Erweiterung von OEM vorgestellt werden, die diesen Nachteil beseitigt.

Zunächst wird OEM bezüglich der Knoten abgeändert, d.h. es wird zwischen zwei verschiedenen Knotentypen unterschieden – einer zur Darstellung von Attributen und einer zur Darstellung von Elementen. Diese Knoten bestehen wie in OEM aus einem Identifier und einem Wert, der entweder atomar oder komplex sein kann. Zur Darstellung von sogenanntem *mixed Content* kann ein Elementknoten sowohl einen atomaren als auch den komplexen Wert $\mathcal{C}$ besitzen. Das Label einer Kante entspricht dem Namen des Attributes bzw. Elementes, in dessen Knoten sie mündet. Dabei gilt, daß in jeden Knoten nur maximal eine Kante führen darf. In Abbildung 2.4 ist ein Beispielgraph dieses Modells dargestellt. Dabei werden Elementknoten durch Kreise und Attributknoten durch Quadrate symbolisiert.

Die vorhandenen Änderungsoperationen des OE-Modells, die in Abschnitt 2.4.2 erläutert wurden, werden in diesem erweiterten Modell durch die folgenden Operationen ersetzt:

Erzeugen eines Attributknotens: Mit der Operation *createAttribute(Identifier i, Value v)* wird ein neuer Attributknoten mit dem Identifikator i und dem Wert v erzeugt. Dabei darf der Identifikator noch nicht vergeben worden sein, der Wert v muß atomar sein (CDATA, ID, IDREF, etc.).

Update eines Attributwertes: Die Operation *updateAttribute(Identifier i, Value v)* ändert den Wert des Attributknotens i auf den Wert v, wobei v atomar (CDATA, ID, IDREF, etc.) sein muß.

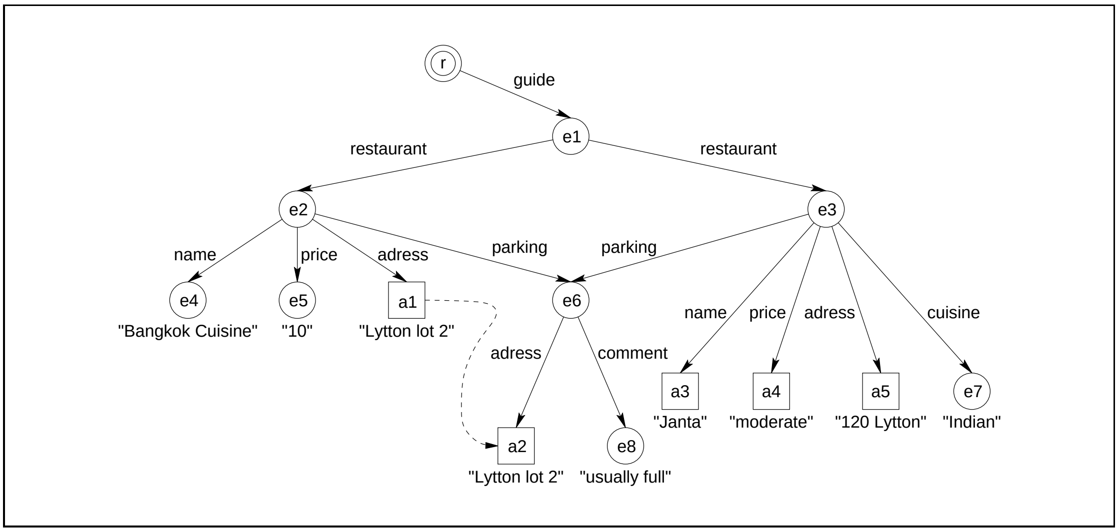

Abbildung 2.4: Beispiel eines erweiterten OEM-Graphen (Der gestrichelte Pfeil symbolisiert eine ID-IDREF-Beziehung zweier Attribute)

Erzeugen eines Elementknotens: Mittels *createElement(Identifier i, Value v)* wird ein neuer Elementknoten mit dem Identifier i konstruiert. Dabei darf i noch nicht vergeben worden sein, der Wert v muß entweder atomar (PCDATA), der komplexe Wert $\mathcal{C}$ oder, zur Darstellung des mixed Content Type, beides sein.

Update eines Elementwertes: Mit der Operation *updateElement(Identifier i, Value v)* wird der Wert des Elementes i auf den Wert v gesetzt, wobei dieser atomar, komplex oder beides sein kann.

Erzeugen einer Kante: Die Operation *createArc(Identifier i, Identifier j, Label l)* erzeugt eine Kante mit dem Label l vom Knoten i zum Knoten j. Dabei kann sowohl eine Kante zwischen zwei Elementknoten als auch eine Kante von einem Element- zu einem Attributknoten konstruiert werden.

Löschen einer Kante: Mit *deleteArc(Identifier i, Identifier j, Label l)* wird die Kante mit dem Label l zwischen vom Knoten i zum Knoten j gelöscht.

Der Vorteil dieses Modells besteht zum einen darin, daß sich XML-Daten einfach damit darstellen lassen, da zwischen Elementen und Attributen samt Werten unterschieden wird. Zum anderen sind die Änderungsoperationen auf diesem Modell konsistent, denn sie arbeiten nur auf den Knoten und Kanten eines Graphs, komplizierte Stringoperationen sind nicht notwendig.

Aufgrund des einfachen Aufbaus des Modells ist es jedoch nicht möglich, Schemainformationen (DTD) von XML-Daten darzustellen, wie beispielsweise Notationen, Entitäten, Attributtypen oder Quantoren. Entity-Referenzen können ebenfalls nicht dargestellt werden.

3 Evolution

In diesem Kapitel werden Evolutionsschritte an XML-Daten, als Vertreter der semistrukturierten Daten, daraufhin untersucht, ob sie realisierbar sind und wie sie sich umsetzen lassen. Das Ziel ist es herauszufinden, welche Änderung an einem Schema, in diesem Fall einer DTD, welche Konsequenzen für die DTD selbst und schließlich für die Daten eines XML-Dokumentes hat, dessen Struktur durch diese DTD bestimmt wird. Da die Menge der denkbaren Umformungen sehr groß ist, werden hier nur solche Umformungen betrachtet, die sinnvoll erscheinen.

3.1 Einführung

Mit Evolution wird im allgemeinen ein Entwicklungsprozess bezeichnet. Im Zusammenhang mit strukturierten und auch semistrukturierten Daten bezeichnet Evolution die Änderung eines Datenmodells und infolge dessen die Änderung der Daten, die diesem Modell unterliegen. Die Auswirkungen, die diese Umformungen auf die Daten haben, können auf vielfältige Weise klassifiziert werden. Eine mögliche Klassifikation unterscheidet danach, ob die Modifizierung des Datenmodells einen Verlust, einen Erhalt oder einen Zuwachs der Informationskapazität bedeutet. Eine weitere Variante unterscheidet nach Informationserhalt, -verlust oder -änderung. Denkbar ist auch, nach der Schwierigkeit der Umwandlung oder deren Durchführbarkeit überhaupt zu unterscheiden, und eine andere Variante differenziert danach, ob eine Änderung des Schemas überhaupt Auswirkungen auf die Ebene der Daten hat und letztlich auch auf den Daten nachgeführt werden muß.

Dieses Kapitel befaßt sich mit der Evolution von XML-Schemata in Form von Dokument Typ Definitionen (DTDs) und deren Auswirkung auf XML-Dokumente, deren Struktur diesen DTDs entspricht. Dabei bleibt die Semantik der einzelnen Dokumente unbetrachtet – es bleibt die Aufgabe der Autoren von Dokumenten und DTDs, die Semantik der Daten zu bestimmen. Ausgangspunkt aller Umformungen sind stets gültige, wohlgeformte Dokumente, wobei gefordert wird, daß im Falle des Datenverlusts dieser minimal sein soll. Ergebnis der Umformungen sind ebenfalls wieder gültige, wohlgeformte Dokumente. Wird eine Umformung als nicht realisierbar erachtet, dann bedeutet dies, daß eine solche Umformung den Verlust der Gültigkeit der Dokumente zur Folge hätte.

Die Umformungen der DTD werden in den folgenden Abschnitten danach unterschieden, auf welcher Ebene sie durchgeführt werden, beispielsweise auf Ebene der Attribute oder auf Ebene der Elemente. Weiterhin wird danach unterschieden, wie sich die Informationskapazität und die Informationen von Dokumenten bei der Umformung verhalten. Da alle Bestandteile von XML ihre eigene Bedeutung haben und auf verschiedene Weise benutzt werden, erscheinen nicht alle möglichen Umformungen sinnvoll. Darum werden hier weitestgehend nur solche

Umformungsoperationen betrachtet, die als praktisch relevant angesehen werden können.

3.2 Umformung auf Attributebene

In diesem Abschnitt werden Umformungen betrachtet, bei denen Attribute hinsichtlich ihrer Bestandteile geändert werden. D.h. es wird untersucht, welche Änderung der Attributdefinition in einer DTD welche Auswirkungen auf die konkreten Attribute im XML-Dokument haben. Dabei muß insbesondere darauf geachtet werden, daß die Attributwerte immer den lexikalischen Anforderungen des jeweiligen Attributtyps entsprechen.

3.2.1 Änderung des Attributnamens

Soll der Name eines Attributes in einer DTD umbenannt werden, so müssen im dazugehörigen XML-Dokument die entsprechenden Attribute ebenfalls umbenannt werden. Dabei darf jeder Attributname in einer Attributliste nur genau einmal vorkommen, d.h. ist der neue Name des Attributes bereits vergeben, dann muß das Attribut, das diesen Namen bereits trägt, ebenfalls umbenannt werden, oder die Umbenennung ist nicht durchführbar.

3.2.2 Änderung des Attributwertes

Soll dagegen der Wert eines Attributes geändert werden, so muß bei einem als FIXED definiertes Attribut diese Änderung im Dokument nachvollzogen werden, d.h. jeder Wert dieses Attributes muß angepaßt werden. Die Änderung des Default-Wertes eines Attributes hat auf das Dokument keinerlei Auswirkungen.

3.2.3 Änderung der Attributreihenfolge

Die Änderung der Attributreihenfolge in einer DTD hat keinerlei Auswirkungen auf ein XML-Dokument, da diese Reihenfolge syntaktisch ohne Bedeutung ist. Die Reihenfolge von Attributen eines Elementes ist in einem Dokument frei wählbar.

3.2.4 Hinzufügen neuer Attribute

Beim Hinzufügen von neuen Attributen zu einer vorhanden Attributliste in einer DTD muß beachtet werden, daß XML-Attribute, die laut [BPSMM00] reserviert sind, nicht neu definiert werden, jeder Attributname pro Attributliste nur einmal vergeben werden darf und Attributwerte den lexikalischen Anforderungen des entsprechenden Attributtyps entsprechen müssen. Im Abschnitt A auf Seite 60 befinden sich Hinweise zu diesen lexikalischen Anforderungen. Bezüglich des Typs eines hinzuzufügenden Attributes gilt es zu beachten:

- In jeder Attributliste darf höchstens ein ID-Attribut vorkommen.
- Jeder Wert eines ID-Attributes muß innerhalb eines Dokumentes eindeutig sein, d.h. er darf pro Dokument nur einmal vergeben werden.

- Ist das neue Attribut vom Typ IDREF oder IDREFS, dann müssen die verwendeten Werte identisch zu einer bzw. zu mehreren IDs sein, die im XML-Dokument vorkommen. Ist dies nicht der Fall, dann müssen die fehlenden ID-Attribute im Dokument hinzugefügt werden.

- Ist das neue Attribut vom Typ ENTITY oder ENTITIES, so müssen die Werte dieses Attributes Namen von nicht zu parsenden Entitäten sein, die in der zum Dokument gehörigen DTD definiert wurden.

- Wenn der Typ des neuen Attributes der Notationstyp ist, dann muß jeder Attributwert dieses Attributes identisch zum Namen einer Notation sein, die in der DTD des Dokumentes definiert wurde.

- Ist das neu hinzugefügte Attribut vom Aufzählungstyp, dann müssen die Attributwerte Elemente des Wertebereiches sein, der bei der Definition des Attributes festgelegt wurde.

Wurde das neue Attribut als FIXED oder REQUIRED definiert, so müssen alle Elemente im Dokument, zu denen das Attribut gehört, um dieses erweitert werden. Dabei ist bei einem als FIXED defnierten Attribut der zu vergebene Wert durch die Definition der Attributliste vorgeschrieben, d.h. alle Attribute müssen diesen Wert besitzen. Andernfalls müssen alle Attributwerte, die diesen Anforderungen nicht entsprechen, angepaßt werden. Wurde das Attribut dagegen als REQUIRED definiert, dann ist der Wert des Attributes frei wählbar.

3.2.5 Entfernen vorhandener Attribute

Das Entfernen eines Attributes aus einer Attributliste bedeutet immer, daß dieses Attribut immer aus allen Elementen, in denen es im Dokument vorkommt, zu entfernen ist. Ist das zu löschende Attribut vom Typ ID, so darf kein anderes Attribut mittels IDREF oder IDREFS darauf verweisen. Wird diese Bedingung nicht erfüllt, d.h. gibt es Referenzen auf das zu entfernende ID-Attribut, können diese Referenzen beipielsweise umgesetzt oder auch entfernt werden. Eine andere Möglichkeit besteht darin, innerhalb eines beliebigen Elementes im Dokument ein neues ID-Attribut einzufügen, dessen Wert identisch zu der ID ist, die entfernt werden soll. Eine letzte Variante ändert den Wert eines anderen ID-Attributes auf den der zu löschenden ID, was wiederum bedingt, daß darauf nicht via IDREF/IDREFS verwiesen wird.

3.2.6 Änderung der *Default Declaration*

Die *Default Declaration* eines Attributes legt fest, ob und in welcher Form dieses Attribut in Elementen des XML-Dokumentes vorkommen müssen. Dabei wird zwischen den vier verschiedenen *Default Declarations* FIXED, IMPLIED, REQUIRED und einem Wert, der als Default-Wert fungiert, unterschieden (siehe Abschnitt 2.2.1 auf Seite 11). Bei der Umwandlung einer *Default Declaration* in eine andere müssen bestimmte Grundregeln beachtet werden, um die Gültigkeit des Dokumentes zu erhalten. Diese Regeln werden im folgenden hinsichtlich der verschiedenen *Default Declarations* erläutert. Die Informationskapazität wird durch diese Umformungen nicht verändert.

Die Auswirkungen der einzelnen Umformungen auf die Informationen eines XML-Dokumentes werden auf Tafel 3.1 auf Seite 27 zusammengefaßt. Dabei gilt für diese wie auch für alle anderen Tafeln in diesem Kapitel, daß bei einer konkreten Operation mehrere Fälle auftreten können. Bei der Umwandlung der Default Declaration eines Attributes beispielsweise von IMPLIED nach REQUIRED kann zum einen der Fall auftreten, daß das betroffene Attribut im Dokument vorkommt, was die Erhaltung der Information bzw. des Attributwertes bedeutet. Kommt dieses Attribut jedoch nicht im Dokument auf, dann muß es samt Attributwert eingefügt werden – es erfolgt also eine Informationserweiterung.

Attributwert

Die Umwandlung dieser *Default Declaration* in eine IMPLIED-Deklaration kann problemlos erfolgen. Soll diese jedoch in eine REQUIRED- oder FIXED-Deklaration umgeformt werden, dann bedingt dies, daß dieses Attribut in jedem Element vorkommt, zu dem es gehört. Bei der Umwandlung in eine FIXED-Deklaration muß außerdem darauf geachtet werden, daß alle Werte dieses Attributes innerhalb des Dokumentes den durch die Attributdefinition vorgeschriebenen Wert besitzen. Gilt dies nicht, so müssen die Werte entsprechend angepaßt werden.

FIXED

Die *Default Declaration* von Attributen, die als FIXED definiert wurden, kann problemlos geändert werden. Es kann dabei zu keinem Konflikt kommen.

IMPLIED

Ein IMPLIED-Attribut läßt sich ohne weiteres in ein Attribut mit einem Attributwert umwandeln. Bei der Mutation in ein REQUIRED-Attribut (siehe dazu das Beispiel[1] in Abbildung 3.1 auf der nächsten Seite) muß gewährleistet sein, daß dieses Attribut in allen dazugehörigen Elementen im Dokument vorkommt. Wird diese Bedingung nicht erfüllt, so müssen die Elemente, in denen das Attribut nicht vorkommt, um dieses erweitert werden. Der Attributwert ist dabei frei wählbar. Bei der Umwandlung in ein FIXED-Attribut ist der Wert dagegen vorgeschrieben. Existieren Attribute, deren Wert nicht identisch zum vorgeschriebenen ist, dann müssen diese angepaßt werden.

REQUIRED

REQUIRED-Attribute können problemlos in IMPLIED-Attribute oder “einfache” Attribute[2] umgeformt werden. Bei der Umwandlung eines REQUIRED-Attributes in ein FIXED-Attribut

[1] In diesem und den folgenden Beispielen werden im oberen Teil der Ausschnitt einer DTD und ein XML-Fragment, daß dieser DTD genügt, abgebildet. Der DTD-Ausschnitt im unteren Teil geht aus dem oberen durch Transformation hervor, das untere XML-Fragment ist das Ergebnis der DTD-Änderung, die auf dem oberen XML-Teil nachgeführt wurde.

[2] Attribute, deren *Default Declaration* ein Attributwert ist

```
<!ELEMENT person (name)>
<!ATTLIST person age    CDATA         #REQUIRED>
<!ATTLIST person gender (male|female) #IMPLIED >

<!ELEMENT name (#PCDATA)>
```

```
<person age="42">
   <name>Meier</name>
</person>
```

```
<!ELEMENT person (name)>
<!ATTLIST person age    CDATA         #REQUIRED>
<!ATTLIST person gender (male|female) #REQUIRED>

<!ELEMENT name (#PCDATA)>
```

```
<person age="42" gender="male">
   <name>Meier</name>
</person>
```

Abbildung 3.1: Änderung der *Default Declaration* von IMPLIED auf REQUIRED

muß die Bedingung erfüllt sein, daß alle Attributwerte identisch zu dem Wert sind, der in der Definition vorgeschriebenen wurde. Andernfalls müssen diese Werte entsprechend angepaßt werden.

3.2.7 Änderung des Attributtyps

Bei der Umformung eines Attributtyps sind die im Abschitt 3.2.4 aufgeführten Bedingungen zum Hinzufügen neuer Attribute einzuhalten. Kommt es bei der Umwandlung zum Verlust von Attributwerten, dann besteht die Möglichkeit, die Attributwerte, die nach der Wandlung nicht mehr im Dokument existieren, separat zu speichern, beispielsweise in einer separaten Datei. Diese Attributwerte können aber auch im Dokument gespeichert bleiben, jedoch entweder in einem speziellen Element, das als eine Art Container für gelöschte Attribute fungiert, oder in Kommentaren bzw. Prozessoranweisungen (Processing Instructions).

Es gilt grundsätzlich, daß die Umwandlung immer dann möglich ist, wenn der Wert des zu wandelnden Attributes der Leerstring ist. Außerdem kommt es bei der Änderung des Attributes von einem Plural- in einen Singulartypen[3] zum Informationsverlust, wenn das Attribut vom

[3] Singulartypen sind alle Attributtypen, deren Name im Singular vorkommt (NMTOKEN, IDREF etc.). Pluraltypen sind dementsprechend alle Attributtpyen, deren Name im Plural vorkommt (NMTOKENS, IDREFS etc).

	Attributwert	FIXED	IMPLIED	REQUIRED
Attributwert		✓ ✂ ✐	✓	✓ ✐
FIXED	✓		✓	✓
IMPLIED	✓	✓ ✂ ✐		✓ ✐
REQUIRED	✓	✓ ✂	✓	

Tafel 3.1: Informationsverhalten bei Änderung der *Default Declaration*

✓ : Informationserhaltung, ✂ : -änderung, ✐ : -erweiterung, ✎ : -reduktion

Pluraltyp mehrere Werte enthält. Bei diesem Spezialfall wird einer der Attributwerte ausgewählt und übernommen, die anderen Werte gehen entweder verloren oder werden, wie oben beschrieben, separat gesichert. Die Besonderheiten einzelner Attributtypen, die darüberhinaus bei der Umformung zu beachten sind, werden nachfolgend erläutert. Daüberhinaus wird in Tafel 3.2.7 auf Seite 30 zusammengefaßt, wie sich die Informationskapazität und die Informationen bei Umformung des Attributtyps ändern.

CDATA

Ein Wert vom Attribut dieses Typs darf Leerzeichen enthalten. Bei der Umwandlung in einen Singulartypen kann es deshalb zum Konflikt kommen, da dieser keine Leerzeichen enthalten darf. Um diese Änderung des Attributtyps trotzdem durchzuführen, muß der Attributwert dahingehend geändert werden, daß er keine Leerzeichen mehr enthält, was eine Änderung der Information des Attributes gleichkommt. Bei der Umwandlung in einen Pluraltypen wird der Attributwert falsch interpretiert, wenn er Leerzeichen enthält, da er dann so betrachtet wird, als wären es mehrere Attributwerte. Dies bedeutet eine Änderung der Informationen des Attributwertes. Der gleiche Fall tritt analog bei der Mutation eines Attributes von einem Pluraltypen mit mehreren Werten in den Typ CDATA auf, da dann mehrere Werte als ein einziger Wert betrachtet werden. Enthält der Wert des zu wandelnden Attributes am Anfang und/oder am Ende Leerzeichen, so wird dieser Wert nach der Wandlung ebenso falsch interpretiert, da diese Leerzeichen bei einem CDATA-Attribut mit zum Wert gehören, bei allen anderen Typen werden dagegen solche Leerzeichen vom XML-Prozessor ignoriert.

ENTITY/ENTITIES

Bei der Änderung eines Attributtypen auf den Typ ENTITY muß darauf geachtet werden, daß der Wert des zu wandelnden Attributes gleich dem Namen einer ungeparsten Entität ist, die in der DTD des Dokumentes definiert wurde. Analog gilt dies für die Umwandlung in den Typ IDREFS, d.h. die Werte des umzuformenden Attributes müssen allesamt Namen von ungeparsten Entitäten sein, die in der DTD definiert wurden.

Enumeration

Die Umwandlung des Attributtyps in den Aufzählungstypen erfordert es, auf Dokumentebene zu überprüfen, ob alle Attributwerte Elemente des Wertebereiches sind, der bei der Definition des Attributes festgelegt wurde. Wird diese Bedingung erfüllt, so ist die Umwandlung durchführbar. Ist sie dagegen nicht erfüllt, dann müssen entweder die Attributwerte geändert werden, die nicht in der Menge der möglichen Werte vorkommen, oder der Wertebereich des Attributes muß entsprechend angepaßt werden.

Die reine Änderung des Wertebereiches eines Enumeration-Attributes in der DTD muß genau dann auf dem Dokument nachgeführt werden, wenn aus dem Wertebereich ein Wert entfernt wird, der im Dokument vorkommt. In diesem Fall ist der betroffene Attributwert so zu ändern, daß er Element des Wertebereiches ist, der für das Attribut in der DTD vereinbart wurde.

ID

Eine Umwandlung vom ID-Typ in andere Typen ist zum einen daran gebunden, daß keine anderen Attribute im Dokument via IDREF bzw. IDREFS auf dieses Attribut verweisen. Um in diesem Fall die Umformung dennoch durchführen zu können, müssen aller Referenzen auf dieses Attribut "umgesetzt" werden. Ein Sonderfall tritt dann auf, wenn ein Attribut vom Typ ID in ein IDREF- bzw. in ein IDREFS-Attribut gecastet werden soll. Das bedeutet, daß ein Attribut, auf das bisher verwiesen wurde, nun selbst zu einer Referenz wird und auf sich selbst verweist, was eine Verletzung der Gültigkeit des Dokumentes nach sich ziehen würde. Eine solche Umwandlung des Attributtyps ist deshalb an eine der folgende Umformungen gekoppelt:

- Es wird einem Element des Dokumentes, daß noch keine ID enthält, ein ID-Attribut hinzugefügt, dessen Wert identisch zum Wert des zu wandelnden Attributes ist.
- Ein vorhandener ID-Wert wird auf den Wert des zu wandelnden Attributes gesetzt. Dabei muß darauf geachtet werden, daß nicht auf diese ID per IDREF bzw. IDREFS verwiesen wird. Wird doch darauf verwiesen, dann sind diese Referenzen umzusetzen oder es ist von vorn nach einem der hier angegebenen Punkte zu verfahren.
- Der Wert des zu ändernden Attributes wird auf den Wert einer ID innerhalb des Dokumentes gesetzt. Alle Attribute, die via IDREF oder IDREFS auf das zu ändernde Attribut verweisen, müssen ebenfalls auf den Wert einer ID aus dem Dokument gesetzt werden.

IDREF

Wird der Typ eines Attributes auf IDREF abgeändert, dann muß der Wert dieses Attributes gleich dem Wert einer ID des Dokumentes sein. Ein Sonderfall stellt die Modifizierung des Attributtyps von IDREF auf ID dar, da dies bedeutet, daß aus einer Referenz auf ein anderes Attribut nun ein Attribut wird, das selber von anderen referenziert werden kann. Ein einfaches Wandeln des Typs ist nicht ausreichend, da dann im Dokument zwei IDs mit demselben Wert vorkommen würden.[4] Die Umwandlung ist darum an eine der folgendenden Umformungen gebunden:

[4] Es ist außerdem darauf zu achten, daß das Element, zu dem das zu wandelnde Attribut gehört, noch keine ID besitzt (siehe Abschnitt 3.2.4).

- Die ID, auf die das zu ändernde Attribut verweist, wird auf einen neuen Wert gesetzt. Dabei können Werte von IDREF- und IDREFS-Attributen, die darauf verweisen, entsprechend abgeändert werden. Sie können jedoch auch ihren alten Wert behalten, wobei sie dann das neue ID-Attribut referenzieren.

- Der Wert des zu wandelnden Attributes wird auf einen anderen Wert gesetzt. Dieser Wert darf nicht der Wert einer anderen ID des Dokumentes sein. Falls dies jedoch beabsichtigt ist, so müssen die “andere” ID und alle darauf verweisenden IDREF- und IDREFS-Attribute einen neuen Wert bekommen.

IDREFS

Die Umwandlung eines Attributtyps von oder in den Typ IDREFS verhält sich ähnlich wie die Umwandlung von oder in den Typ IDREF. Der Unterschied besteht darin, daß IDREFS ein Pluraltyp ist und bei der Änderung in einen Singulartypen Informationen im Attributwert verloren werden. Diese Informationen können aber, wie oben beschrieben, separat gesichert werden.

Notation Type

Die Änderung eines Attributtyps in den Notationstypen erfordert, daß der Attributwert identisch zum Namen einer Notation ist, die in der DTD des Dokumentes definiert wurde.

3.3 Umformung auf Elementebene

In diesem Abschnitt werden Umformungen von Elementen und deren Bestandteilen betrachtet, d.h. Subelemente, die in Alternativen oder Sequenzen gruppiert, oder Quantoren, die an einzelne Elemente bzw. an ganze Gruppen (Sequenz, Alternative) gebunden sein können.

3.3.1 Änderung des Elementnamens

Soll der Name eines Elementes geändert werden, so muß zunächst zugesichert werden, daß der neue Name nicht bereits an ein anderes Element vergeben wurde. Danach muß der Name dieses Elementes selbst, in allen “Vaterelementen” der DTD und schließlich im Dokument geändert werden.

3.3.2 Änderung des Quantors

Die Änderung eines Element-Quantors kann unterschiedliche Auswirkungen auf das XML-Dokument haben. Je nach Umformung kann es zur Änderung der Informationskapazität und unter Umständen auch zum Verlust von Informationen kommen. Abhängig von der Änderung des Quantors muß überprüft werden, ob diese Änderungen Auswirkungen auf das Dokument haben. Tafel 3.3 auf Seite 31 gibt Auskunft darüber, wie sich konkrete Umformungen eines Quantors auf die Informationen und die Informationskapazität des Dokumentes auswirken.

	CDATA	ENTITY	ENTITIES	Enumeration	ID	IDREF	IDREFS	Notation Type	NMTOKEN	NMTOKENS
CDATA		$\rightarrow$ ✓ ✂	$\nearrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\nearrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\nearrow$ ✓ ✂
ENTITY	$\rightarrow$ ✓		$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓
ENTITITES	$\searrow$ ✓ ✂	$\searrow$ ✓ ✎		$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\searrow$ ✓	$\rightarrow$ ✓	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\rightarrow$ ✓
Enumeration	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓		$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓
ID	$\rightarrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\nearrow$ ✓ ✂	$\rightarrow$ ✓ ✂		$\rightarrow$ ✂	$\nearrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\rightarrow$ ✓ ✂	$\nearrow$ ✓ ✂
IDREF	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✂		$\rightarrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓
IDREFS	$\searrow$ ✓ ✂	$\searrow$ ✓ ✎	$\rightarrow$ ✓	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎		$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\rightarrow$ ✓
Notation Type	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓		$\rightarrow$ ✓	$\nearrow$ ✓
NMTOKEN	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\rightarrow$ ✓	$\nearrow$ ✓	$\rightarrow$ ✓		$\nearrow$ ✓
NMTOKENS	$\searrow$ ✓ ✂	$\searrow$ ✓ ✎	$\rightarrow$ ✓	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\rightarrow$ ✓	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	

Tafel 3.2: Durchführbarkeit von Änderungen des Attributtyps

$\rightarrow$: Kapazitätserhaltung, $\nearrow$: -erweiterung, $\searrow$: -reduktion

✓ : Informationserhaltung, ✂ : -änderung, ✐ : -erweiterung, ✎ : -reduktion

Die Änderung von Quantoren, die eine ganze Sequenz oder Alternative an sich binden, verhält sich genau so, wie die Änderung von Element-Quantoren. Tafel 3.3 gilt somit auch für die Quantorenänderung von Sequenzen und Alternativen.

Die Änderung des Quantors kann mit der Änderung der *Default Declaration* bei Attributen (siehe Abschnitt 3.2.6 auf Seite 24) verglichen werden. Der Unterschied dazu besteht darin, daß die Modifizierung der *Default Declaration* keine Auswirkungen auf die Informationskapazität hat und insbesondere keine Daten verloren werden.

	1	?	$+$	$*$
1		$\rightarrow$ ✓	$\nearrow$ ✓	$\nearrow$ ✓
?	$\rightarrow$ ✓ ✐		$\nearrow$ ✓ ✐	$\searrow$ ✓
$+$	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎		$\rightarrow$ ✓
$*$	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎	$\rightarrow$ ✓ ✐	

Tafel 3.3: Informations- und Kapazitätsverhalten bei Änderung des Quantors

$\rightarrow$: Kapazitätserhaltung, $\nearrow$: -erweiterung, $\searrow$: -reduktion

✓ : Informationserhaltung, ✄ : -änderung, ✐ : -erweiterung, ✎ : -reduktion

3.3.3 Änderung der Elementreihenfolge

Die Modifizierung der Reihenfolge von Elementen ist nur innerhalb einer Sequenz von Bedeutung. Innerhalb einer Alternative und innerhalb von mixed Content ist die Reihenfolge der Elemente unwichtig, ebenso verhält es sich mit der Reihenfolge, in der Elemente in einer DTD definiert werden. Wird die Reihenfolge von Elementen innerhalb einer Sequenz geändert, so muß dies im Dokument nachvollzogen werden, indem alle Elemente, inklusive ihrer Subelemente und Werte, in die neue Reihenfolge gebracht werden.

3.3.4 Hinzufügen eines Elementes

Beim Hinzufügen eines Elementes zu einer DTD wird danach unterschieden, ob ein Element neu definiert oder ein bereits definiertes Element zu einer vorhandenen Alternative, Sequenz bzw. einem mixed Content hinzugefügt wird.

Neudefinition eines Elementes

Ist der Name des neuen Elementes innerhalb der DTD noch nicht vergeben, dann kann diese Änderung der DTD ohne weiteres durchgeführt werden. Der Grund dafür ist, daß ein Element, das lediglich definiert aber nicht Unterelement des Wurzelelementes ist, nicht im Dokument auftreten kann und darf. Es können also beliebig viele Elemente in der DTD zusätzlich definiert werden, ohne daß dies Auswirkungen auf das Dokument hat. Werden der DTD neue Elemente hinzugefügt, deren Name bereits vergeben ist, dann muß entweder der Name des neuen Elementes oder der Name des bereits existierenden Elementes, wie in Abschnitt 3.3.1 beschrieben, geändert werden.

Hinzufügen eines Elementes zu Alternative, Sequenz oder mixed Content

Voraussetzung für das Hinzufügen eines Elementes ist, daß dieses Element in der DTD definiert wurde. Gilt dies nicht, dann muß diese Definition wie oben beschrieben durchgeführt werden. Das Einfügen eines Elementes in eine Alternative wirkt sich nicht auf das Dokument aus, da dieses Element innerhalb der Alternative nur optional vorkommen muß. Das Einfügen eines Elementes zu einem mixed Content wirkt sich ebenfalls nicht auf das Dokument aus. Dagegen kann das Einfügen eines Elementes in eine Sequenz Auswirkungen auf das Dokument haben, abhängig davon, welcher Quantor an die Sequenz bzw. das einzufügende Element gebunden ist. Immer dann, wenn an die Sequenz der Quantor $*$ bzw. ? gebunden ist und die Seuqenz nicht im Dokument vorkommt, dann hat eine solche Umformung auch keine Auswirkungen auf das Dokument.

Ansonsten gilt: Wird an das einzufügende Element der Quantor $*$ oder ? gebunden, dann muß das neue Element nicht in das Dokument eingefügt werden. Wird an das einzufügende Element dagegen der Quantor 1 bzw. $+$ gebunden, so muß das Element im Dokument hinzugefügt werden. Ist zum Zeitpunkt des Einfügens nicht klar, welchen Wert und welche Attributwerte das Element annehmen soll, dann kann ein fester Wert vergeben werden, der als Nullwert interpretiert wird. Da in XML keine Nullwerte exisiteren, ist ein solcher Wert vom Autor des Dokumentes festzulegen.

3.3.5 Löschen eines Elementes

Beim Entfernen eines Elementes aus einer DTD muß danach unterschieden werden, ob die Definition eines Elementes oder lediglich ein Element als Bestandteil eines anderen entfernt werden soll. Soll ein Element als Bestandteil eines anderen gelöscht werden, dann muß zum einen das Element aus der Definition des Vaterelementes in der DTD gelöscht und zum anderen muß jedes Vorkommen des Elementes im Dokument entfernt werden. Soll ein Element dagegen komplett gelöscht werden, so muß zusätzlich die Definition dieses Elementes in der DTD entfernt werden.

Um gelöschte Elemente nicht vollkommen zu verlieren, besteht wie beim Entfernen von Attributen die Möglichkeit, diese Elemente in einer separaten Datei, in speziellen Elementen, in Kommentaren oder in Prozessoranweisungen abzulegen. In Abbildung 3.2 auf der nächsten Seite befindet sich ein Beispiel, an dem das Löschen eines Elementes vorgeführt wird. Dabei wird das zu löschende Element in einem XML-Kommentar gespeichert.

3.3.6 Änderung zwischen Alternative und Sequenz

Die Änderung zwischen Alternative und Sequenz hängt hauptsächlich vom Quantor ab, an den die zu wandelnde Sequenz bzw. Alternative gebunden ist. Dabei gilt für die Quantoren $*$ und $+$ gleichermaßen:

```
<!ELEMENT Person      (Name, Alter, Geburtstag)>
<!ELEMENT Name        (#PCDATA)>
<!ELEMENT Alter       (#PCDATA)>
<!ELEMENT Geburtstag> (#PCDATA)>
```

```
<Person>
   <Name>Meier</Name>
   <Alter>48</Alter>
   <Geburtstag>29.02.1952</Geburtstag>
</Person>
```

```
<!ELEMENT Person      (Name, Alter)>
<!ELEMENT Name        (#PCDATA)>
<!ELEMENT Alter       (#PCDATA)>
```

```
<Person>
   <Name>Meier</Name>
   <Alter>48</Alter>
</Person>
<!--   <Geburtstag>29.02.1952</Geburtstag>   -->
```

Abbildung 3.2: Löschen eines Elementes

1. Bei der Umformung einer Alternative in eine Sequenz muß am Dokument überprüft werden, ob die betroffenen Elemente in ihrer Struktur noch der DTD entsprechen. Der Grund dafür ist, daß die Struktur der Elemente, die auf eine Alternative mit dem Quantor $*$ oder $+$ passen, weniger eingeschränkt ist als die einer Sequenz mit dem Quantor $*$ oder $+$. Entspricht die Dokumentstruktur nach der Wandlung nicht mehr der DTD, so müssen entweder Elemente in der neu entstandenen Sequenz entfernt oder neue Elemente hinzugefügt werden. Letzteres wird im Beispiel in Abbildung 3.3 auf der nächsten Seite vorgeführt.

2. Die Umwandlung einer Sequenz in eine Alternative kann dagegen immer ohne Probleme durchgeführt werden, da alle Konstrukte von Elementfolgen, die mit einer Sequenz gebildet werden können, auch immer auf eine Alternative mit dem Quantor $*$ oder $+$ passen.

Für den Quantor ? gilt bei der Umwandlung einer Alternative in eine Sequenz: Kommt die Alternative im entsprechenden Element des Dokumentes vor, dann muß dieses Element entweder um die Subelemente unter Einhaltung der vorgeschriebenen Reihenfolge bereichert werden, die in der “neuen” Sequenz fehlen, oder die bereits vorhanden Subelemente werden aus dem Dokument gelöscht. Bei der Umformung von der Sequenz zur Alternative müssen bei Vorhanden-sein der Sequenz[5] alle Kind-Elemente bis auf eines entfernt werden. Welches der Kind-Elemente dabei erhalten bleibt, ist frei wählbar. Eine andere Möglichkeit ist das Löschen der kompletten Sequenz, da diese durch den Quantor ? nur optional vorkommen muß, wobei jedoch die Forderung nach minimalem Datenverlust bei Umformung von Dokumenten von Abschnit 3.1 auf Seite 22 verletzt wird. Kommt die umzuwandelnde Sequenz oder Alternative im Dokument nicht vor, so ist diese Umformung ohne weiteres durchführbar.

Bei der Mutation einer Sequenz in eine Alternative und umgekehrt verhält sich der Quantor 1 ähnlich wie der Quantor ?, wenn diese Sequenz bzw. diese Alternative im Dokument vorkommt. D.h. bei der Umwandlung einer Sequenz in eine Alternative wird ein Kind-Element übernommen, alle anderen werden entfernt, im umgekehrten Fall müssen neue Subelemente hinzugefügt werden.

Unter Verwendung der Quantoren $*$, $+$ und ? gilt zusammenfassend, daß die Umwandlung einer Alternative in eine Sequenz und umgekehrt immer dann ohne weiteres möglich ist, wenn die Elemente der zu ändernden Alternative bzw. Sequenz nicht im Dokument vorkommen.

```
<!ELEMENT A (a|b)*>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

```
<A>
   <a/><a/><b/>
</A>
```

```
<!ELEMENT A (a, b)*>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

```
<A>
   <a/><b/><a/><b/>
</A>
```

Abbildung 3.3: Umformung einer Alternative in eine Sequenz

3.3.7 Änderung des *Content Type*

Die verschiedenen Inhalts-Typen eines Elementes sind ANY, EMPTY, MIXED und Children (Kind-Elemente), wobei Kind-Elemente immer innerhalb einer Sequenz oder Alternative auftreten. Nachfolgend sollen Umformung zwischen den verschiedenen Content Types diskutiert werden.

[5]Durch den Quantor ? muß die Sequenz nur optional vorkommen

ANY

Die Umwandlung des Content Type eines Elementes von ANY nach EMPTY bewirkt, daß alle Subelemente gelöscht werden. Bei der Umwandlung in den gemischten Typ müssen alle Subelemente entfernt werden, die dort nicht mehr vorkommen dürfen. Bei der Umformung in den Children-Typ müssen ebenfalls die Subelemente gelöscht werden, die laut Definition des neuen Elementes nicht mehr enthalten sein dürfen. Darüberhinaus müssen hier Umformungsschritte durchgeführt werden, die davon abhängig sind, ob das neue Element aus einer Alternative oder einer Sequenz aufgebaut ist. Besteht der Children-Typ aus einer Sequenz, dann muß die Reihenfolge beachtet werden. Stimmt die Reihenfolge, in der die Elemente im Dokument auftreten, nicht mit der überein, die durch die Definition des neuen Elementes festgelegt wurde, dann müssen die Elemente umgeordnet und in diese Reihenfolge gebracht werden. Soll ein Element vom Typ ANY dagegen in eine Alternative (Child-Typ) umgeformt werden, dann ist diese Umformung vom Quantor der Alternative und von den Subelementen des neuen Elementes abhängig. Ist die Alternative an den Quantor $*$ oder $+$ gebunden und kommen darin Elemente, Sequenzen oder Alternativen vor, die nicht zulässig sind, dann sind diese zu entfernen. Ist die Alternative aber an den Quantor 1 oder ? gebunden, dann darf das neue Element jeweils nur ein Subelement enthalten, alle anderen müssen entfernt werden. Welches Subelement dabei übernommen wird, ist frei wählbar.

EMPTY

Elemente, die als EMPTY definiert wurden, können ohne weiteres in den gemischten oder den Typ ANY umgformt werden. Die Mutation in den Children-Typ hängt jedoch vom Quantor ab. Sie läßt sich genau dann ohne Zusatzaufwand realisieren, wenn die Alternative bzw. die Sequenz des zu mutierenden Elementes an den Quantor $*$ oder ? gebunden ist. Ist diese aber an den Quantor $+$ oder 1 gebunden, dann müssen neue Subelemente in das umzuwandelnde Element eingefügt werden. Welche Subelemente in welcher Anzahl und in welcher Reihenfolge eingefügt werden müssen, wird durch die Definition der Alternative bzw. Sequenz des neuen Elementes in der DTD festgelegt. Die Belegung von evtl. vorkommenden Attribut- und Elementwerten ist dabei durch den Autor des XML-Dokumentes bzw. der dazugehörigen DTD vorzunehmen. Soll ein EMPTY-Element, das im Dokument durch Leerelement-Tag repräsentiert wird, in einen anderen Inhaltstypen umgwandelt werden, dann muß immer dieses Leerelement-Tag durch ein öffnendes und ein schließendes Tag ersetzt werden.

MIXED

Eine Umformung von MIXED in einen beliebigen anderen Content Type bewirkt immer, daß eventuell auftretende Elementwerte entfernt werden müssen. Subelemente und Attribute bleiben dagegen erhalten, wie im Beispiel auf Abbildung 3.4 auf der nächsten Seite gezeigt wird. Bei der Umwandlung nach EMPTY müssen sogar alle Subelemente gelöscht werden. Dagegen müssen beim Umformen in den Children-Typ nur Subelemente gelöscht werden, die laut Definition dort nicht mehr vorkommen dürfen. Besteht das neue Element aus einer Sequenz, dann muß ggf. noch die Reihenfolge der Subelemente angepaßt werden.

Children

Die Modifizierung von Children nach ANY oder MIXED ist ohne weiteres möglich. Die Umformung nach EMPTY bewirkt letztlich, daß alle Kind-Elemente zu entfernen sind.

Die Auswirkungen der Umformung des Content Type auf die Informationen und die Informationskapazität eines Dokumentes sind auf Tafel 3.4 zusammengefaßt.

	ANY	EMPTY	MIXED	Children
ANY		$\searrow$ ✓ ✎	$\rightarrow$ ✓	$\rightarrow$ ✓ ✎
EMPTY	$\nearrow$ ✓		$\nearrow$ ✓	$\nearrow$ ✐
MIXED	$\searrow$ ✓ ✎	$\searrow$ ✓ ✎		$\searrow$ ✓ ✎
Children	$\nearrow$ ✓	$\searrow$ ✎	$\nearrow$ ✓	

Tafel 3.4: Informations- und Kapazitätsverhalten bei Änderung des *Content Type*

$\rightarrow$: Kapazitätserhaltung, $\nearrow$: -erweiterung, $\searrow$: -reduktion

✓ : Informationserhaltung, ✄ : -änderung, ✐ : -erweiterung, ✎ : -reduktion

```
<!ELEMENT Person (#PCDATA|Name)*>
<!ELEMENT Name   (#PCDATA)>
```

```
<Person>
   Bemerkung: ...
   <Name>Meyer</Name>
</Person>
```

```
<!ELEMENT Person (Name)>
<!ELEMENT Person (#PCDATA)>
```

```
<Person>
   <Name>Meyer</Name>
</Person>
```

Abbildung 3.4: Umformung vom *mixed Content Type* nach ANY

3.4 Umformung auf Entitätsebene

In diesem Abschnitt sollen Umformungen innerhalb von Entity-Deklarationen und Änderungen von Entitätstypen betrachtet werden. Da Entitäten ausschließlich in DTDs definiert jedoch aufgrund der verschiedenen Typen sowohl im XML-Dokument als auch in der DTD referenziert werden können, kann die Umformung einer Entität innerhalb der DTD neben einer Umformung des Dokumentes auch eine weitere Umformung der DTD zur Folge haben.

3.4.1 Allgemeine Entitäten

Die Umwandlung einer allgemeinen Entität in eine externe Entität ist möglich, indem der Wert der Entität in der Entity-Definition durch eine System- oder eine Public-ID ersetzt wird. Die Umformung in eine Parameter-Entität ist zwar grundsätzlich möglich, jedoch darf diese Entität dann nicht mehr aus einem Dokument heraus referenziert werden. Das bedeutet, alle Referenzen, die auf die zu ändernde Entität verweisen, müssen nach der Änderung entweder auf eine andere Entity verweisen oder aus dem Dokument entfernt werden. Wird lediglich der Name einer allgemeinen Entität geändert, dann müssen alle betroffenen Entity-Referenzen im Dokument entsprechend angepaßt werden.

3.4.2 Parameter-Entitäten

Parameter-Entitäten dürfen ausschließlich in DTDs verwendet werden. Soll eine Entität in eine Parameter-Entität umgewandelt werden, dann müssen entweder alle Referenzen auf diese Entität innerhalb des Dokumentes umgesetzt oder gelöscht werden. Soll dagegen eine Parameter-Entität in eine andere Entität umgeformt werden, dann muß zuerst die DTD an den Stellen angepaßt werden, an denen diese Entity referenziert wird. D.h. es müssen entweder die Entity-Referenzen dahingehend geändert werden, daß sie auf andere Parameter-Entitäten verweisen, oder diese Referenzen müssen entfernt werden. Alternativ dazu können auch alle Referenzen per Hand aufgelöst werden.

Die Änderung des Wertes einer Parameter-Entität dagegen wirkt sich zunächst auf die DTD aus, in der sie definiert und verwendet wird, und schließlich auf das Dokument, in dem die Bestandteile benutzt werden, die unter Verwendung dieser Entität in der DTD definiert wurden. Ändert sich durch diese Umformung die Struktur der DTD, so muß diese Strukturänderung auf Dokumentebene nachvollzogen werden, um die Gültigkeit des betroffenen Elementes zu wahren.

Die Änderung des Entity-Namen hat wie bei allgemeinen Entitäten zur Folge, daß die Referenzen auf diese Entität entsprechend auf den neuen Namen der Entität gesetzt werden müssen. Im Unterschied zu allgemeinen Entitäten werden diese Umbenennungen in der DTD durchgeführt, da sie nur dort referenziert werden können.

3.4.3 Nicht analysierte Entitäten

Eine nicht analysierte Entität darf ausschließlich durch Attribute vom Typ ENTITY bzw. ENTITIES referenziert werden. Gleichzeitig darf ein Attribut der o.g. Typen nur auf ungeparste Entitäten verweisen. Soll eine unparsed Entity in eine andere Entität umgewandelt werden, so müssen deshalb alle Attribute, die darauf verweisen, angepaßt werden, d.h. sie müssen entweder auf den Namen einer anderen unparsed Entity gesetzt oder gelöscht werden, sofern dies möglich ist. Ein weitere Möglichkeit besteht darin, den Typ der betroffenen Attribute zu ändern, so daß die Attributwerte nicht mehr identisch zum Namen einer unparsed Entity sein müssen. Eine letzte Möglichkeit zur Umformung einer nicht analysierten Entität besteht in der Änderung des Namens. Dabei müssen jedoch lediglich die Werte der ENTITY-Attribute an den neuen Namen angepaßt werden, die auf diese Entität verweisen.

3.4.4 Allgemeine Schlußfolgerungen

Enthalten allgemeine und externe, analysierte Entitäten Markup, so gilt für sie gleichermaßen, daß zu allen öffnenden Tags in der Entität auch schließende Tags vorkommen müssen (siehe Abschnitt 2.2.1 auf Seite 13). Grundsätzlich ist die Umformung des Entity-Typs wie oben beschrieben zwar möglich, aber aufgrund der Verschiedenheit dieser Typen bezüglich ihrer Verwendung scheint eine solche Umwandlung bei einem guten Entwurf der DTD praktisch nicht relevant.

3.5 Änderung von Notationen

Eine Notation besteht aus zwei Teilen: einem Namen und einer ID. Die Änderung der ID hat keine Auswirkungen auf die DTD. Die Änderung des Namens der Notation wirkt sich dagegen auf alle nicht analysierten Entitäten einer DTD aus, deren Typ durch diese Notation beschrieben wird. Diese Änderung wirkt sich ebenfalls auf die Attribute aus, die vom Notations-Typ sind und deren Wert identisch zum Namen der zu ändernden Notation ist. Bei der Modifizierung des Namens einer Notation müssen deshalb lediglich die Typbezeichner der Entitäten in der DTD und die betroffenen Attributwerte im Dokument auf den neuen Namen der Notation gesetzt werden.

3.6 Komplexe Umformungen

Alle bisher diskutierten Umformungen von DTD und XML-Dokumenten werden als atomar bezeichnet. Nun sollen Umformungen betrachtet werden, die sich auf einfache, atomare Umformungen zurückführen lassen, die miteinander verknüpft werden (Baukastenprinzip), und darum als komplex bezeichnet werden. Durch das Zurückführen komplexer auf atomare Umformungsoperationen wird es einfacher, die Menge der insgesamt möglichen Umformungen zu beschreiben und umzusetzen. Die Möglichkeiten, die sich bei der Zusammensetzung einfacher Regeln ergeben, sollen in den folgenden Abschnitten erläutert werden. Ein wichtiger Punkt ist dabei die Reihenfolge, in der die einzelnen atomaren Operationen ausgeführt werden, da sie die Informationskapazität des Dokumentes stark beeinflußt.

3.6.1 Umbewegen von Elementen, Attributen und Werten

Das Umbewegen eines Elementes von einem Vater-Element zu einem anderen inklusive vorhandener Attribute, Werte und Subelemente bedeutet, daß das zu bewegende Element an einem Ort gelöscht und an einem anderen wieder eingefügt werden muß. Werden diese beiden Schritte in der angebenen Reihenfolge ausgeführt, also erst Löschen, dann Einfügen, so wird es unter hoher Wahrscheinlichkeit zum Verlust von Informationen innerhalb des Dokumentes kommen, denn wenn das Element zuerst aus einem Vater-Element gelöscht wird, dann sind die Informationen verloren. Wird dagegen das Element zuerst in einem anderen Element eingefügt, also kopiert, und dann im anderen gelöscht, so bleiben die Informationen erhalten. Dieses Umbewegen kann mit Attributen und Werten auf die gleiche Weise umgesetzt werden. Inkonsistente Zustände nach dem Kopieren des Elementes/Attributes sind nicht von Bedeutung, entscheidend ist hierbei, daß die Gültigkeit des Dokumentes nach dem Umbewegen, d.h. nach Kopieren und Löschen, wieder garantiert ist.

Ein Problemfall tritt dann auf, wenn ein Element/Attribut zu einem Element umbewegt werden soll, das im Dokument nicht auftritt, da es an den Quantor $*$ oder ? gebunden ist. In diesem Fall muß entweder das "fehlende" Element hinzugefügt werden, oder die Umformung ist nicht durchführbar.

3.6.2 Mehrfache Änderung von Quantoren

Die Änderung von mehreren Quantoren einer Gruppe ist, ähnlich dem Umbewegen von Elementen, an eine bestimmte Reihenfolge gebunden, um den Verlust an Informationen zu begrenzen oder gar auszuschließen. Dabei gilt, daß immer zuerst die Quantorenänderungen ausgeführt werden, die kapazitätserhaltend oder kapazitätserweiternd sind. Kapazitätsverringernde Änderungen von Quantoren sind zum Schluß auszuführen. Der Grund dafür liegt in der Tatsache begründet, daß die Änderung eines Quantors, die die Informationskapazität eines Dokumentes verringert, zu einem Informationsverlust führen kann. Im Anschluß daran kann aber die Ausführung einer kapazitätserhaltenden oder -erweiternden Umformung bedeuten, daß Informationen, die bei der ersten Umformung verloren wurden, gar nicht hätten gelöscht werden müssen.

In Abbildung 3.5 auf der nächsten Seite ist ein Beispiel für die mehrfache Änderung eines Quantors angegeben. Dabei wird zum einen der Quantor der Sequenz von 1 auf $*$ und zum anderen der Quantor des Subelementes a in dieser Sequenz von $*$ auf 1 gesetzt. Würde zuerst der Quantor des Subelementes a modifiziert werden, so müßte ein a-Element gelöscht werden, obwohl die Änderung des Sequenzquantors von 1 auf $*$ das Entfernen dieses Elementes im Dokument überflüssig macht. Wird also zuerst der Quantor der Sequenz und dann der Quantor des Subelementes geändert, dann bleiben wie im Beispiel alle Daten erhalten.

Ein weiteres Beispiel befindet sich in Abbildung 3.6 auf Seite 41. Dort werden nur Quantoren innerhalb einer Sequenz geändert. Wird dabei zuerst der Quantor 1 in $+$ und dann der Quantor $*$ auf 1 modifiziert, dann bewirkt die Umformung insgesamt keinen Datenverlust. In beiden Beispielen wurden oben die Ausgangs-DTD und unten die DTD abgebildet, die durch Transformation aus der ersten hervorgehen. In der Mitte befindet sich der Ausschnitt eines XML-Dokumentes, das von dieser Transformation unberührt bleibt und dessen Struktur deshalb beiden DTDs entspricht.

```
<!ELEMENT A (a*, b?)>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

```
<A>
  <a/><a/><b/>
</A>
```

```
<!ELEMENT A (a, b?)*>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

Abbildung 3.5: Mehrfache Änderung von Quantoren

3.6.3 Änderung der Elementgruppierung

Unter dem Ändern einer Elementgruppierung wird zum einen das Löschen und Einfügen der Bestandteile einer Gruppe (Sequenz oder Alternative, siehe Abschnitte 3.3.4 und 3.3.5) verstanden. Diese Bestandteile können Elemente oder wieder Sequenzen bzw. Alternativen sein. Zum anderen zählen die Operationen dazu, die Bestandteile von einer Gruppe in eine andere umbewegen (Abschnitt 3.6.1), das mehrfache Ändern von Quantoren einer Gruppe oder deren Bestandteile (Abschnitt 3.6.2), sowie die Überführung einer Sequenz in eine Alternative (Abschnitt 3.3.6).

3.6.4 Verfeinern von Sequenzen und Alternativen

Um den Grad der Strukturierungstiefe eines Dokumentes zu erhöhen, besteht die Möglichkeit, Sequenzen und Alternativen eines Elementes zu verfeinern (Refinement, Schachteln, Nesten). Dazu wird in der DTD ein neues Element definiert, dessen Bestandteile die Elemente, Alternativen bzw. Sequenzen sind, die separat in einem Element gehalten werden sollen. Diese zu separierenden Subelemente werden dann inklusive ihrer Quantoren in ihrem Vaterelement durch das neu definierte Element ersetzt. Abschließend muß diese Änderung im Dokument nachgeführt werden, indem das neu definierte Element eingefügt und die zu separierenden Elemente in dieses umbewegt werden. In Abbildung 3.7 auf Seite 42 wird die Verfeinerung an einem Beispiel vorgeführt.

3.6.5 Auflösen von Elementen

Im Gegensatz zur Verfeinerung von Sequenzen und Alternativen bietet das Auflösen von Elementen (Entnesten) die Möglichkeit, den Strukturierungsgrad von Dokumenten zu verringern. Beim Auflösen eines Elementes wird dieses Element in der DTD durch die Definition seiner

```
<!ELEMENT A (a*, b?, a)>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

```
<A>
  <a/> <a/> <a/>
</A>
```

```
<!ELEMENT A (a, b?, a+)>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
```

Abbildung 3.6: Mehrfache Änderung von Quantoren (2)

Subelemente ersetzt, also durch eine Alternative bzw. eine Sequenz. Im Dokument werden alle öffnenden und schließenden Tags des aufzulösenden Elementes entfernt.

3.6.6 Zusammenfassen und Aufspalten von Elementen

Die Tiefe der Strukturierung eines Elementes kann auch dadurch geändert werden, indem alle Elemente einer Sequenz oder Alternative zu einem Element verschmolzen oder alle Elementwerte zu einem Wert zusammengefaßt werden, beispielsweise durch Konkatenation. Analog dazu ist ein Aufspalten eines Elementes in mehrere Elemente möglich, wobei die Aufspaltung des Elementwertes beliebig erfolgen kann, ebenso die Aufteilung der Subelemente unter den neuen Elementen.

3.6.7 Änderungen zwischen Elementen, Attributen und Werten

Um den Aufbau eines Dokumentes grundlegend zu ändern, besteht die Möglichkeit, Umwandlungen zwischen Elementen, Attributen und Elementwerten vorzunehmen, welche nachfolgend kurz erläutert werden sollen.

Die Umwandlung eines Attributes in ein Element kann realisiert werden, indem ein neues Element definert wird, das Subelement des Elementes ist, zu dem das Attribut gehört. Dieses neue Element bekommt den Wert des umzuformenden Attributes zugewiesen, das Attribut selbst wird aus seinem Element entfernt. Wurde das umzuwandelnde Attribut als IMPLIED definiert, dann wird das neue Element mit dem Quantor ? definiert, in allen anderen Fällen ist kein Quantor notwendig.

Die Umwandlung eines Elementes in ein Attribut geschieht analog, d.h. es wird ein neues Attribut im Vater-Element definiert, dessen Wert ergibt sich aus dem zu wandelnden Element. Das Element selbst wird schließlich gelöscht. Der Typ des neuen Attributes muß so gewählt

```
<!ELEMENT Dokument (Titel, Autor+)>
<!ELEMENT Titel    #PCDATA>
<!ELEMENT Autor    #PCDATA>
```

```
<Dokument>
   <Titel>XML in der Praxis</Titel>
   <Autor>Henning Behme</Autor>
   <Autor>Stefan Mintert</Autor>
</Dokument>
```

```
<!ELEMENT Dokument (Titel, Autoren)>
<!ELEMENT Titel    #PCDATA>
<!ELEMENT Autoren  (Autor+)>
<!ELEMENT Autor    #PCDATA>
```

```
<Dokument>
   <Titel>XML in der Praxis</Titel>
   <Autoren>
      <Autor>Henning Behme</Autor>
      <Autor>Stefan Mintert</Autor>
   </Autoren>
</Dokument>
```

Abbildung 3.7: Verfeinern einer Sequenz

werden, daß alle Bedingungen erfüllt sind, die beim Hinzufügen eines neuen Attributes (siehe Abschnitt 3.2.4) gefordert werden. Die Default Declaration kann dabei im einfachsten Fall IMPLIED sein. Wenn das zu wandelnde Element immer einen Wert besitzt, dann kann auch REQUIRED gewählt werden. Besitzt das zu wandelnde Element immer ein und denselben Wert, dann kann das neue Attribut auch als FIXED definiert werden.

Eine weitere Möglichkeit der Umwandlung besteht darin, den Wert eines Attributes dem Wert eines Elementes hinzuzufügen. An welcher Stelle der Attributwert dem Elementwert hinzugefügt wird, ist frei wählbar. Das betroffene Attribut ist nach der Umwandlung zu löschen. Im Gegensatz dazu kann ein Teil eines Elementwertes “herausgeschnitten” und einem neudefinierten Attribut eines Elementes zugewiesen werden. Welcher Teil des Elementwertes dazu benutzt wird, liegt dabei im Ermessen des Nutzers. Oft benutzte Teile von Elementwerten können entweder in allgemeine Entitäten oder in Elemente, insbesondere leere Elemente umgewandelt werden. Bei der Umwandlung in ein Element (siehe dazu das Beispiel in Abbildung 3.8 auf der nächsten Seite) muß die DTD dahingehend angepaßt werden, daß einerseits das Element definiert wird, wenn es noch nicht definiert wurde. Andererseits muß unter Umständen die Defi-

```
<!ELEMENT Satz #PCDATA>
```

```
<Satz>
   Dies ist ein Test.
</Satz>
```

```
<!ELEMENT Satz (#PCDATA|TEST)*>
<!ELEMENT Test EMPTY>
```

```
<Satz>
   Dies ist ein <Test/>.
</Satz>
```

Abbildung 3.8: Umwandlung eines Teilwertes in ein Element

nition des Elementes, dessen Wert geändert werden soll, angepaßt werden, so daß das Elememt vom mixed Content Type ist.

3.7 Probleme

Ein wichtiges Kriterium zur Umformung von Dokumenten ist die Erhaltung deren Gültigkeit bezüglich einer gegebenen DTD. Kann die Gültigkeit bei der Umformung eines Dokumentes nicht erhalten werden, so darf diese Umwandlung nicht durchgeführt werden – das Dokument bleibt unverändert.

Sollen nun mehrere Dokumente, die alle ein und derselben DTD entsprechen, umgeformt werden, so besteht die Möglichkeit, daß nur ein Teil dieser Dokumente erfolgreich umgeformt werden kann. Die anderen Dokumente bleiben dagegen unberührt, da eine Umwandlung ihre Gültigkeit verletzen würde. Das Ergebnis der Umformung dieser Dokumente sind nun zwei Dokumentenkollektionen, wobei eine Kollektion XML-Dokumente enthält, die der “alten” DTD entsprechen. In der anderen Kollektion befinden sich Dokumente, die einer neuen DTD unterliegen. Da das Ziel der Umwandlung einer Menge von Dokumenten aber eine Menge von Dokumenten ist, die allesamt einer einzigen DTD unterliegen, muß nach Wegen gesucht werden, die verhindern, daß wie oben beschrieben verschiedene Dokumentenkollektionen entstehen.

Eine Möglichkeit besteht nun darin, die beiden Kollektionen samt ihrer DTD zu generalisieren, d.h. erneut umzuformen, so daß aus zwei verschiedenen Kollektionen eine neue, allgemeine entsteht (siehe Abb. 3.9 auf der nächsten Seite). Jedoch kann auch hier wieder der Fall eintreten, daß die Umformung einzelner Dokumente fehlschlägt. Eine andere Variante besteht deshalb darin, nur die bestehenden DTDs zu einer neuen DTD zu generalisieren. Dabei bleiben die Dokumente zwar unberührt, sie entsprechen nach der Generalisierung der DTDs aber alle einer einzigen DTD.

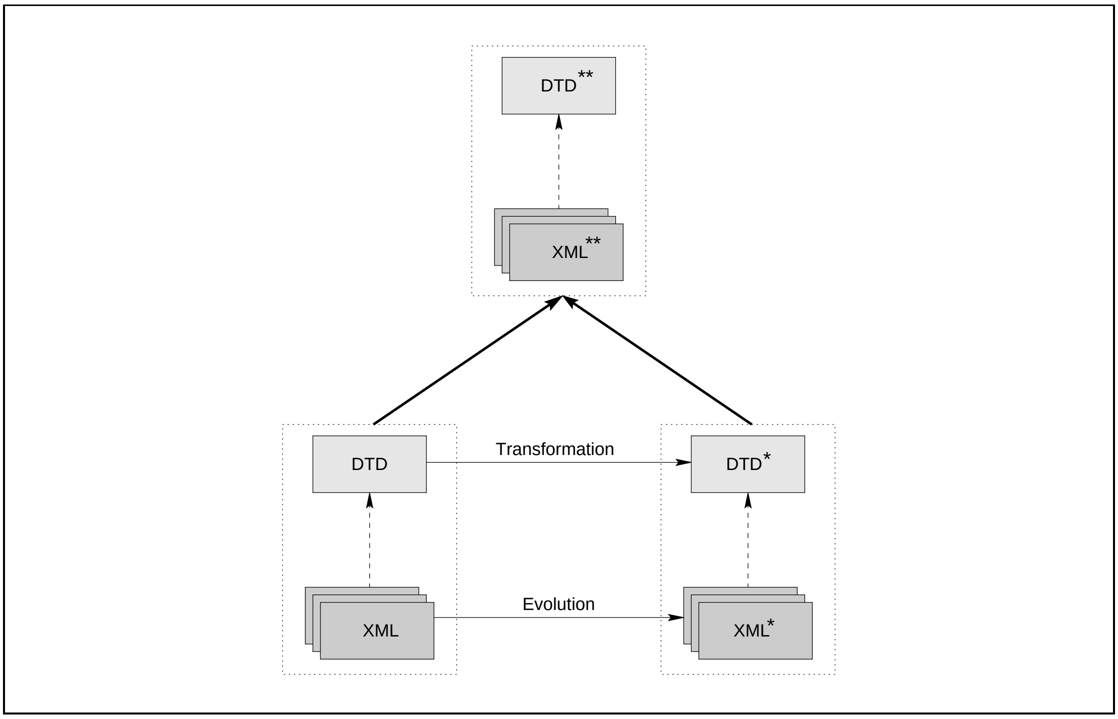

Abbildung 3.9: Generalisierung verschiedener Dokumentenkollektionen

4 Systementwurf

Das folgende Kapitel zeigt eine Möglichkeit auf, mit der die Umformungen von XML-Dokumenten aus Kapitel 3 realisiert werden können, wobei die Konzeption eines Systems im Vordergrund stehen soll. Darüberhinaus wird bewertet, ob und in welcher Weise XSLT für den Einsatz in einem solchen System geeignet ist.

4.1 Grundlegendes, Transformationsablauf

Um die Anforderungen an ein Systen zur Umformung von gültigen XML-Dokumenten spezifizieren zu können, müssen zunächst drei Sachverhalte erläutert werden:

1. Jedes umzuformende Dokument entspricht in seiner Struktur einer DTD. Diese DTD muß ebenfalls einer Umformung unterzogen werden.
2. Die Umformung eines Dokumentes und der DTD dieses Dokumentes wird durch eine Transformationssprache formuliert.
3. Die Transformation einer DTD ist ein einmaliger Vorgang. Dagegen wird die Umformung der Dokumente, die dieser DTD entsprechen, mehrmals durchgeführt – pro Dokument eine Umwandlung.

Die Umformung gestaltet sich dann wie folgt: Im ersten Schritt wird die DTD umgeformt. Dadurch wird aus einer bestehenden DTD eine "neue" DTD generiert, die die Struktur der XML-Dokumente nach ihrer Umwandlung beschreibt. Im zweiten Schritt werden sequentiell alle Dokumente umgeformt, die gültig bezüglich der "alten" DTD sind. Nach der Transformation sind sie dann gültig bezüglich der "neuen" DTD.

4.2 Beschreibung von DTD-Änderungen

Um die Umformungen von XML-Dokumenten durchführen zu können, ist zunächst die Beschreibung der Änderungen des Schemas nötig, also die Änderung der DTD. Die Beschreibung solcher DTD-Änderungen kann auf verschiedene Art und Weise geschehen. Die erste Möglichkeit besteht darin, die Änderung an einer bestehenden DTD zu beschreiben, beispielsweise durch Angabe eines Regelwerkes, wobei die einzelnen Regeln beschreiben, welche Teile eines Dokumentes in welcher Weise umgeformt werden. Mit dieser Methode können alle DTD-Umformungen eindeutig beschrieben werden und alle damit verknüpften Änderungen des Dokumentes lassen sich daraus ableiten.

Die zweite Möglichkeit beinhaltet den Vergleich zweier DTDs und die Ableitung der Unterschiede zwischen beiden. Diese Variante ist praktisch nicht benutzbar, da keine ausreichenden semantischen Informationen gegeben sind und deshalb ein Vergleich nur in einer beschränkten Anzahl von Fällen funktionieren würde. Die dritte Variante zur Beschreibung der Änderung einer DTD besteht darin, zwei XML-Dokumente zu vergleichen, die jeweils gültig bezüglich einer DTD sind, um schließlich die Unterschiede zwischen den jeweiligen DTDs abzuleiten. Dazu werden die Dokumente zunächst mit dem in [Boy00] vorgestellten Algorithmus auf ihre kanonische Form reduziert und schließlich verglichen. Das Problem besteht jedoch auch hier wieder darin, daß semantische Informationen fehlen und die Ableitung einer DTD aus einem Dokument nicht eindeutig ist. Bei zwei Dokumenten, deren Struktur vollkommen identisch ist und deren Unterschied nur in den Namen der Elemente besteht, würde nach dieser Methode nicht erkannt werden, daß sie bis auf die Benennung der Elemente äquivalent sind. Diese Variante scheidet damit ebenfalls aus. Somit ist der einzig praktikable Weg, Änderungen von DTDs zu beschreiben, die direkte Beschreibung selbst.

4.3 Verwendbarkeit von XSLT

Da es sich bei der Durchführung von Evolutionsschritten um die Transformation von XML-Dokumenten handelt, bietet sich der Einsatz der in Abschnitt 2.3 vorgestellte Transformationssprache XSLT an. Mit XSLT lassen sich alle Umformungen eines XML-Dokumentes auf einfache Weise beschreiben und mit Hilfe eines XSLT-Prozessors realisieren. Darüberhinaus ist XSLT in der Lage, bestimmte Bedingungen zu überprüfen, die für die Durchführung von Transformationen aus Kapitel 3 erfüllt sein müssen, um die Gültigkeit des Dokumentes zu erhalten. Die Bedingungen, die auf diese Weise überprüft werden können, sind allesamt am Dokument selber überprüfbar, beispielsweise ob in jeder Attributliste der Name eines Attributes schon vergeben ist. Dagegen ist XSLT nicht in der Lage, Bedingungen eines Dokumentes zu überprüfen, wenn dafür Daten aus der DTD benötigt werden, beispielsweise um den Typ von Attributen oder den Quantor von Elementen zu testen.

XSLT ist also in der Lage, beliebige Umformungen an einem XML-Dokument vorzunehmen und Bedingungen zu testen, für deren Test keine Daten aus der DTD des Dokumentes benötigt werden. Es können jedoch keine Bedingungen überprüft werden, wenn dafür Informationen aus der DTD notwendig sind. Einen Ausweg stellen in diesem Fall sogenannte DTD-Parser dar, mit denen es möglich ist, DTDs zu analysieren.

4.3.1 Schwierigkeiten

Die Schwierigkeit bei der Verwendung von XSLT liegt einerseits für den Benutzer darin, daß die Beschreibung von Transformationen sehr aufgebläht wirkt und so schnell dazu führt, daß er die Übersicht verliert. XPath als Addressierungssprache für Knoten innerhalb eines Dokumentes ist zwar sehr mächtig, könnte aber noch um einige Adressierungsmöglichkeiten erweitert werden. Beispielsweise ist es derzeit nicht möglich, Nachbarschaftsbeziehungen (das Element hinter einem anderen Element in einer Sequenz) oder Bereichsprädikate (alle Elemente mit einem Wert zwichen 3 und 13) auszudrücken.

Weil es bei der Evolution von XML-Dokumenten erforderlich ist, sowohl Teile der Dokumente zu übernehmen, die einem Prädikat genügen als auch Teile, die diesem Prädikat nicht genügen, ist es häufig erforderlich, zwei Templates zur Durchführung einer einzigen Umformungsoperation anzufertigen. Dabei wird im ersten beschrieben, welche Operationen mit Dokumentteilen auszuführen sind, die dem Prädikat entsprechen, während im zweiten Template alle anderen Teile behandelt werden. Soll beispielsweise ein Element umbenannt werden, so muß mit Hilfe eines Templates die Umbenennung aller umzubenennenden Elemente erfolgen. Ein zweites Template muß alle anderen Elemente einfach nur kopieren. Im Anhang ab Seite 61 wird dies anhand zweier Beispiele demonstriert.

Aufgrund der Tatsache, daß XSLT aus einem physischen XML-Dokument immer ein neues Dokument erzeugt, ist die Performance bei der Umformung insbesondere bei massenhaften vorkommenden XML-Dokumenten als gering einzuschätzen. Eine Steigerung wäre durch sogenannte In-Place-Updates möglich, bei denen die Umformungen auf den Originaldaten ausgeführt würden.

4.4 Systemanforderungen

An das System, mit dem Umformungen von gültigen XML-Dokumenten durchgeführt werden können, werden die folgenden Anforderungen gestellt:

Alle Umformungen, die an Dokumenten vorgenommen werden, müssen die Gültigkeit dieser Dokumente erhalten. Für die Transformationen werden eine Transformationsbeschreibung, eine DTD und eine Menge von Dokumenten benötigt, die dieser DTD entsprechen. Das System muß anhand der Transformationsbeschreibung zum einen die Umformung der gegebenen DTD in eine neue und zum anderen die Umformungen der einzelnen Dokumente vornehmen.

Sollen während der Transformation Daten aus einzelnen Dokumenten gelöscht werden, so ist dafür eine Interaktion mit dem Nutzer notwendig. Dieser entscheidet, ob die Daten tatsächlich entfernt werden sollen bzw. auf welche Weise sie gesichert werden, z.B. in einer separaten Datei oder in Kommentaren (siehe dazu Abschnitt 3.2.7 auf Seite 26). Optional kann auch dann eine Interaktion erfolgen, wenn bedingt durch die Transformationen neue Daten in das bearbeitete Dokument eingefügt werden müssen. Der Nutzer kann dabei die neuen Daten dem System direkt mitteilen. Erfolgt beim Einfügen neuer Daten keine Nutzerinteraktion, dann werden vom Systen automatisch voreingestellte Daten in das Dokument eingefügt.

4.5 Aufbau und Funktionsweise des Systems

Aus den Anforderungen an das System, das die Umformungen eines XML-Dokumentes durchführt, ergibt sich der in Abbildung 4.1 auf der nächsten Seite dargestellte Systemaufbau. Dieses System besteht im Kern aus einem XSLT-Prozessor und zwei Modulen, einem XML- und einem DTD-Modul. Das DTD-Modul formt eine gegebene DTD anhand einer Transformationsbeschreibung in eine neue DTD um.

Die Aufgabe des XML-Moduls ist es zunächst zu überprüfen, ob die Umformung des Dokumentes überhaupt durchführbar ist.[1] Müssen aufgrund der Umformungen Daten aus dem Dokument gelöscht werden, dann erfolgt eine Interaktion mit dem Nutzer, der darüber entscheidet,

[1] Die Transformation eines Dokumentes kann dann nicht durchgeführt werden, wenn dadurch die Gültigkeit dieses Dokumentes verloren geht. In diesem Fall erfolgt ein Hinweis an den Nutzer.

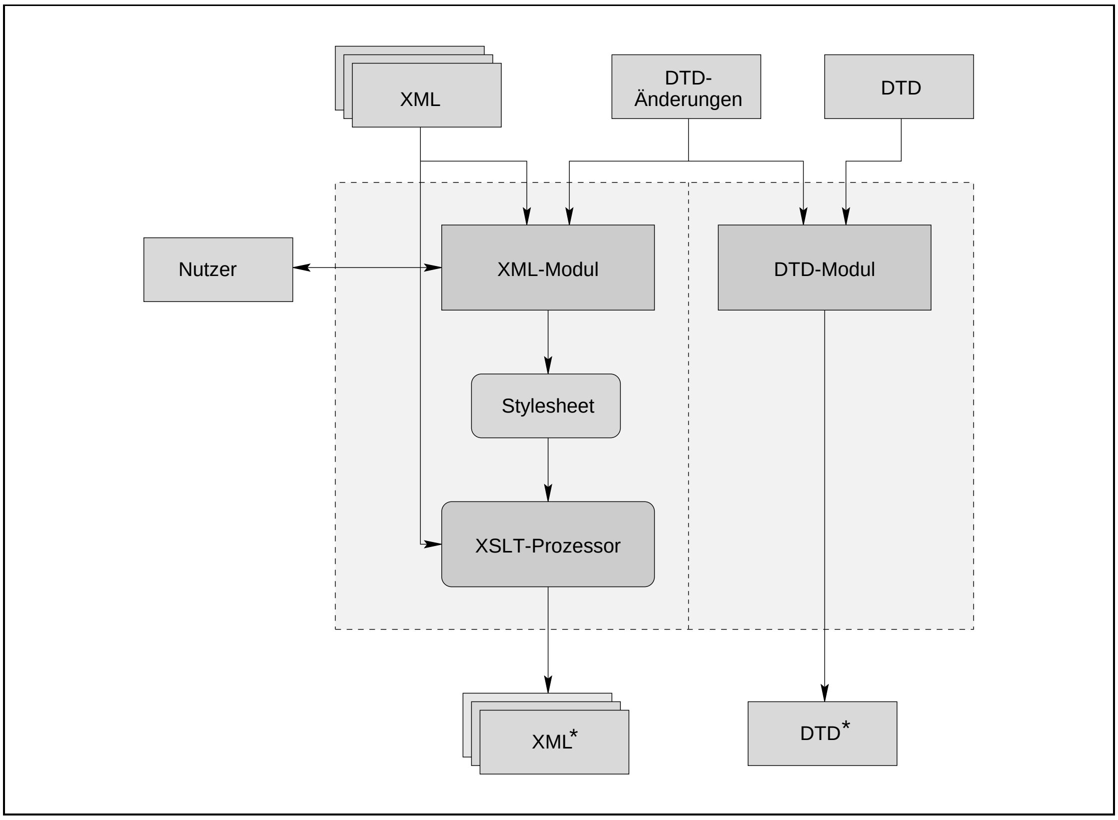

Abbildung 4.1: Systemaufbau

ob und wie diese Daten entfernt werden sollen. Zur Nutzerinteraktion kommt es optional auch dann, wenn neue Daten in das Dokument eingefügt werden sollen. Dabei werden die einzufügenden Daten vom Nutzer direkt an das System übergeben. Als nächstes generiert das XML-Modul aus dem umzuwandelnden XML-Dokument und der Transformationsbeschreibung ein XSL-Stylesheet und leitet dieses an den XSLT-Prozessor weiter. Dieser führt dann letzlich mit diesem Stylesheet die Umformung des Dokumentes durch.

Wichtig ist zum einen, daß die Umformungen von DTD und XML-Dokument zwar auf derselben Transformationsbeschreibung basieren aber unabhängig voneinander in verschiedenen Prozessen ablaufen. Zum anderen muß für jedes umzuformende Dokument ein eigenes Stylesheet generiert werden, da trotz gleicher DTD und gleicher Transformationsbeschreibung verschiedene Dokumente meist unterschiedlich umgeformt werden müssen.

4.6 Erweiterung, Ausblick

Es gibt mehrere Möglichkeiten, das im vorigen Abschnitt konstruierte System zu verbessern bzw. zu vereinfachen.

Beispielsweise ist XSLT bzw. XPath in der derzeitigen Version des Standards nicht dazu in der Lage, auf die DTD eines Dokumentes zuzugreifen. Werden derartige Zugirffe in einer neuen Version möglich, dann kann die Aufgabe der Überprüfung von Bedingungen zur Transformation der Dokumente vollständig vom DTD-Modul auf die Stylesheets und damit auch auf den XSLT-Prozessor übertragen werden. Die Aufgaben des XML-Moduls würden damit deutlich reduziert werden.

Wird XML-Schema zur Beschreibung des Schemas verwendet, dann sind schon heute XSLT-Prozessoren in der Lage, alle Bedingungen zur Dokumentumwandlung vollständig selber zu überprüfen. Der Grund dafür ist zum einen, daß ein solches Schema ein gültiges XML-Dokument ist, zum anderen kann XSLT auf mehreren Dateien arbeiten, in diesem Fall also auf dem umzuwandelnden Dokument und auf dem Schema. In diesem Fall würde das DTD-Moduls ebenfalls wegfallen.

5 Transformationsbeschreibung

Die Beschreibung von Transformationen auf XML-Dokumenten läßt sich durch mehrere Möglichkeiten realisieren. Eine Variante ist die reine Beschreibung der Umformungen, eine andere die Beschreibung der Daten jeweils vor und nach Ausführung der Änderungsoperationen, so daß aus der Differenz der Daten die Änderungen abgeleitet werden können. Für beide dieser Wege wird eine Beschreibung der Daten benötigt, für die erste Variante ist außerdem eine Beschreibung der Operationen notwendig.

Hier soll die Möglichkeit der Beschreibung der Änderungsoperationen ansatzweise[1] und informal[2] realisiert werden. Die Beschreibung der Daten wird dabei anhand des erweiterten OE-Modells aus Abschnitt 2.4.3 vorgenommen, die Änderungsoperationen werden auf die atomaren Operationen dieses Datenmodells zurückgeführt. Randbedingungen, die zur Durchführung der Änderungsoperationen erfüllt sein müssen, um die Gültigkeit des umzuformenden Dokumentes zu wahren, werden hier jedoch nicht detailliert erläutert, da diese in Kapitel 3 ausführlich diskutiert wurden.

5.1 Operationen auf Attributebene

Zuerst werden einige Operationen modelliert, mit denen Umformungen auf Attributebene von XML-Dokumenten durchgeführt werden können.

5.1.1 Umbenennung von Attributen

Zur Umbenennung eines Attributes muß zunächst überprüft werden, ob der neue Name des umzubenennenden Attributes in der Attributliste noch nicht vergeben wurde. Handelt es sich um ein ID- oder IDREF-Attribut, so muß außerdem noch überprüft werden, ob die ID-IDREF-Abhängigkeiten auch nach der Umformung erfüllt sind. Analog ist bei Attributen vom Typ NOTATION bzw. ENTITY zu verfahren. Sind alle Bedingungen erfüllt, dann wird die Umformung durchgeführt, indem zuerst die Kante gelöscht wird, die in dem Attribut mündet, das umbenannt werden soll. Danach wird eine neue Kante erzeugt, die sich an derselben Stelle befindet, wie die zuvor gelöschte und dessen Label dem Namen des Attributes entspricht (Beispiel in Abbildung 5.1 auf der nächsten Seite).

[1] Ziel dieses Kapitels ist es lediglich, eine Variante zur Beschreibung von Umformungen zu skizzieren, eine komplette Beschreibung würde den Rahmen sprengen.

[2] Eine formale Beschreibung würde ebenfalls den Rahmen einer solchen Arbeit übersteigen. Außerdem scheint in diesem Fall eine informale Beschreibung leichter lesbar zu sein, da u.a. keine neuen Operatoren oder sonstige Sprachelemente eingeführt werden müssen.

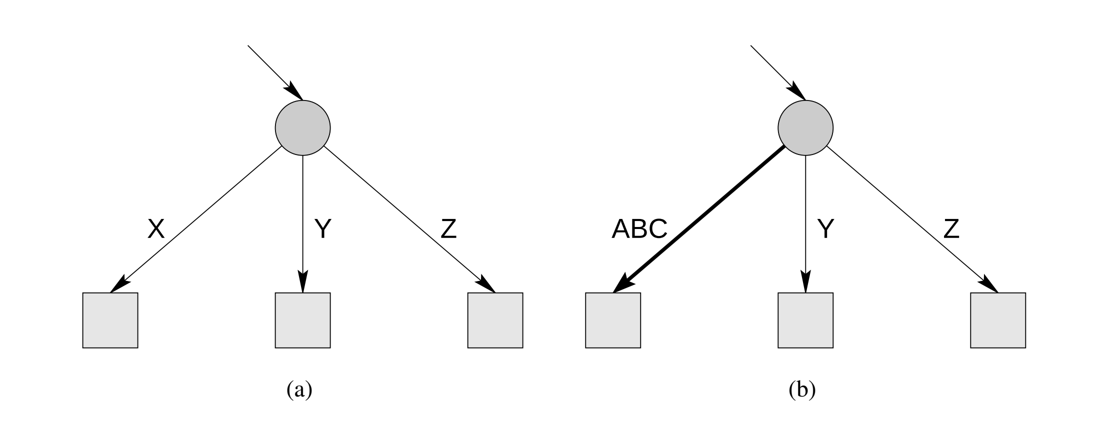

Abbildung 5.1: Umbenennung eines Attributes: die Kante mit dem Label *X* wird durch die neue Kante mit dem Label *ABC* ersetzt.

5.1.2 Ändern von Attributwerten

Soll der Wert eines Attributes geändert werden, so muß nach Überprüfung der Randbedingungen lediglich ein Update der betroffenen Attributknoten durchgeführt werden. Handelt es sich dabei um ein ID-Attribut, dann müssen außerdem alle Knoten von IDREF-Attributen angepaßt werden, die auf dieses Attribut verweisen. (Beispiel in Abbildung 5.2)

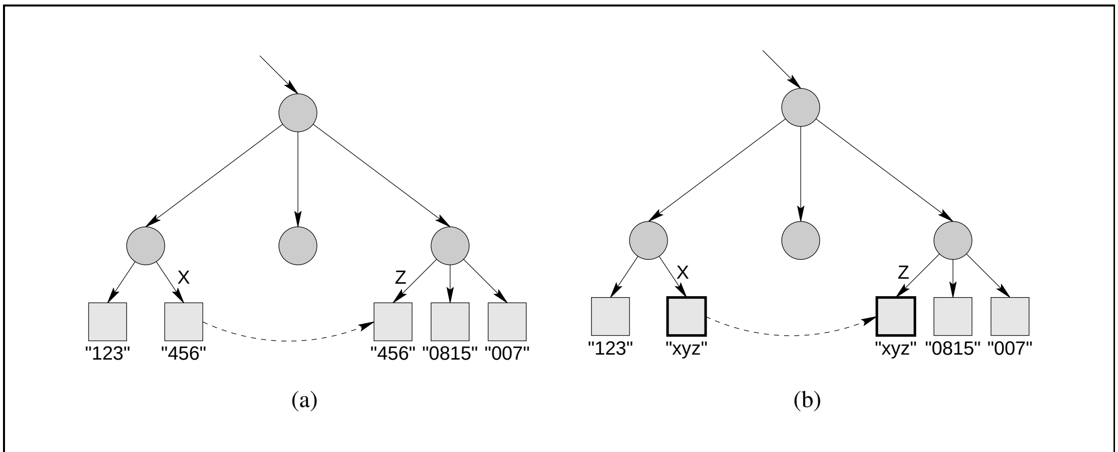

Abbildung 5.2: Änderung eines Attributwertes: Das Attribut *X* wird auf einen neuen Wert gesetzt. Da dieses Attribut vom Typ ID ist, wird das Attribut *Z* (Typ IDREF) ebenfalls auf diesen neuen Wert gesetzt.

5.1.3 Einfügen neuer Attribute

Das Einfügen eines neuen Attributes bedingt, daß der Name dieses Attributes in der Attributliste noch nicht vergeben wurde. Bei einem Attribut vom Typ ID, IDREF, NOTATION und ENTITY sind weitere Bedingungen zu Überprüfen (siehe dazu Abschnitt 3.2.4 auf Seite 23). Werden alle Bedingungen erfüllt, dann wird ein Attributknoten erzeugt und eine Kante, die vom Elementknoten, dem das Attribut hinzugefügt werden soll, zu diesem Attributknoten führt. Der Wert des Attributknotens entspricht dabei dem Attributwert, das Label dem Attributnamen. (Beispiel in Abbildung 5.3)

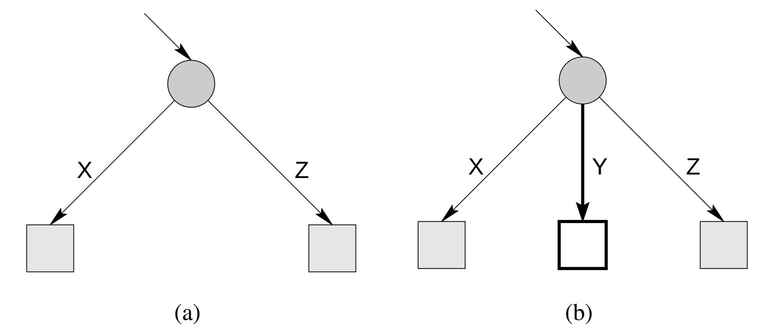

Abbildung 5.3: Einfügen eines neuen Attributes: dazu werden ein Attributknoten und eine Kante erzeugt.

5.1.4 Löschen von Attributen

Zum Löschen eines Attributes muß, nach Überprüfung der Bedingungen, lediglich die Kante zum entsprechenden Attributknoten entfernt werden. Da dieser Knoten dann von keinem anderen mehr über eine Kante erreicht werden kann, ist er nicht mehr persistent und gilt damit als gelöscht. (Beispiel in Abbildung 5.4 auf der nächsten Seite)

5.1.5 Änderung der *Default Declaration*

Die Änderung der *Default Declaration* eines Attributes von IMPLIED auf FIXED[3] erfolgt in zwei Schritten: Zuerst wird der Wert aller betroffenen Attributknoten geändert, deren Wert nicht der Vorgabe des FIXED-Attributes entspricht. Exisitieren Elemente, in denen das Attribut nicht vorkommt, nach der Umformung aber vorkommen muß, so werden dort neue Attributknoten erzeugt und mit einer Kante von diesen Elementen aus verbunden. (Beispiel in Abbildung 5.5 auf der nächsten Seite)

[3] Die anderen Umformungen der *Default Declaration* werden hier nicht beschrieben

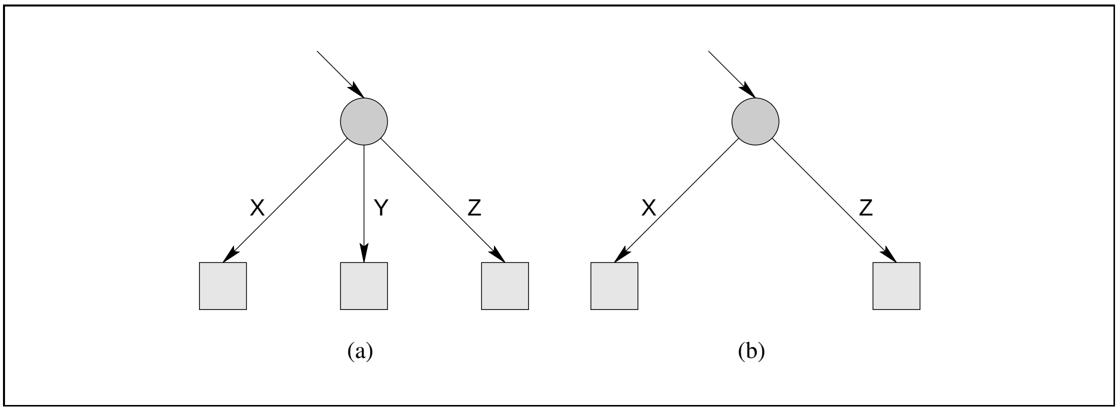

Abbildung 5.4: Entfernen eines Attributes: es muß nur die Kante entfernt werden, der Attributknoten wird dadurch automatisch gelöscht.

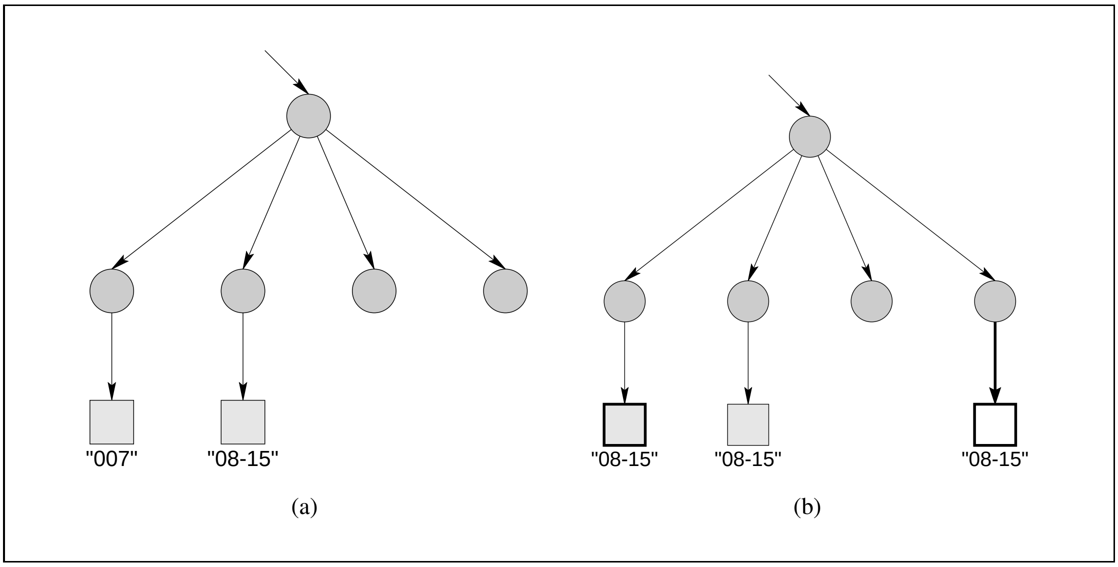

Abbildung 5.5: Änderung der *Default Declaration* von IMPLIED auf FIXED: Zu jedem Elementknoten, der noch kein Attributknoten besitzt, muß ein Attributknoten und eine Kante hinzugefügt werden.

5.2 Operationen auf Elementebene

In diesem Abschnitt werden einige ausgewählte Operationen am erweiterten OE-Modell dargestellt, mit denen Änderungen von Elementen umgesetzt werden können.

5.2.1 Einfügen und Löschen von Elementen

Um ein neues Element einzufügen, muß zuerst ein Elementknoten und dann eine Kante vom Knoten des Vaterelementes erzeugt werden (siehe Abbildung 5.6). Zum Löschen eines Elementes muß dagegen nur die Kante zum entsprechenden Elementknoten gelöscht werden. Der Elementknoten ist danach nicht mehr erreichbar und daher auch nicht mehr persistent.

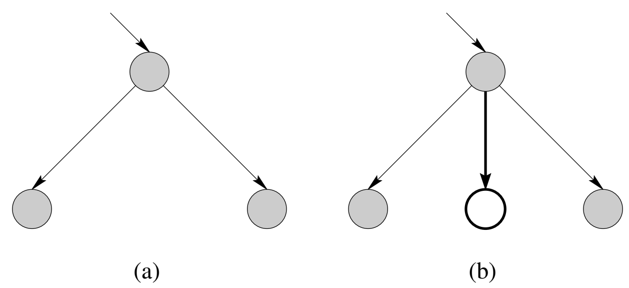

Abbildung 5.6: Einfügen eines neuen Elementes: Es müssen pro neuem Element ein Elementknoten und eine Kante zu diesem Knoten aus erzeugt werden.

5.2.2 Änderung des Quantors

Die Änderung des Elementquantors von $*$ auf $+$ [4] wirkt sich nur auf Elemente aus, die nicht in ihren Vaterelementen auftauchen.[5] In diesem Fall muß für jedes dieser Vaterelemente ein Elementknoten und eine Kante vom Vaterknoten zum neu erzeugten Knoten erstellt werden. (Beispiel in Abbildung 5.7 auf der nächsten Seite)

5.2.3 Änderung des *Content Type*

Als Beispiel zur Modifizierung des *Content Type* soll hier die Umformung von MIXED nach ANY modelliert werden. Ein Element, das vom mixed Content Type ist, darf sowohl Subelemente als auch Werte besitzen. Laut dem erweiterten OE-Modell darf solch ein Element jedoch nur Subelemente und maximal *einen* Wert haben. Elemente vom Typ ANY dürfen dagegen nur Subelemente besitzen. Deshalb muß auf allen Knoten von umzuwandelnden Elementen, die

[4] Alle anderen Umformungen zwischen den Quantoren $*$, $+$, ? und 1 werden hier nicht modelliert

[5] Der Quantor $*$ bedeutet, daß das Element, über das quantifiziert wird, n-mal innerhalb des Vaterelementes vorkommen darf, mit $n \geq 0$.

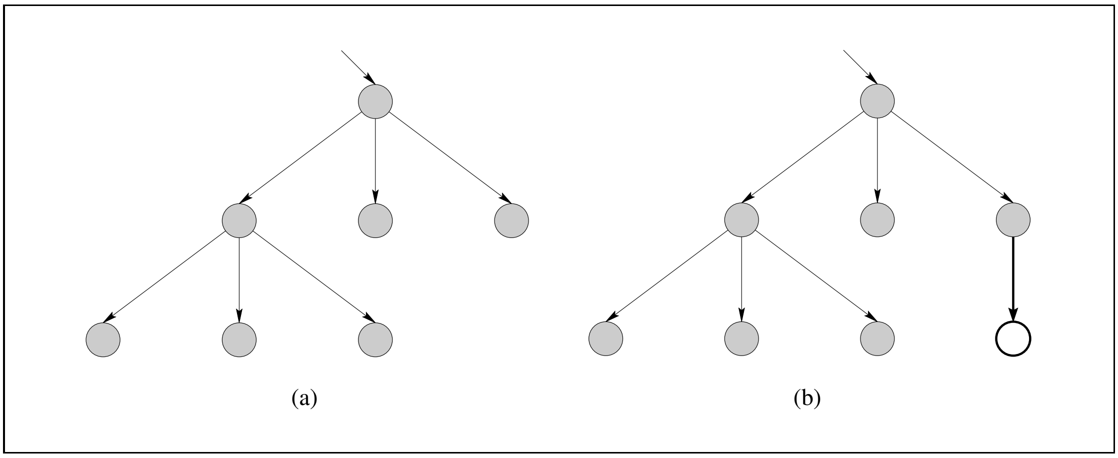

Abbildung 5.7: Änderung des Quantors von $*$ auf $+$: Ggf. müssen ein neuer Elementknoten und eine neue Kante zu diesem Knoten hinzugefügt werden.

sowohl einen atomaren als auch den komplexen Wert $\mathcal{C}$ besitzen, eine Update-Operation ausgeführt werden, wobei der atomare Wert des jeweiligen Knotens gelöscht wird. (Beispiel in Abbildung 5.8)

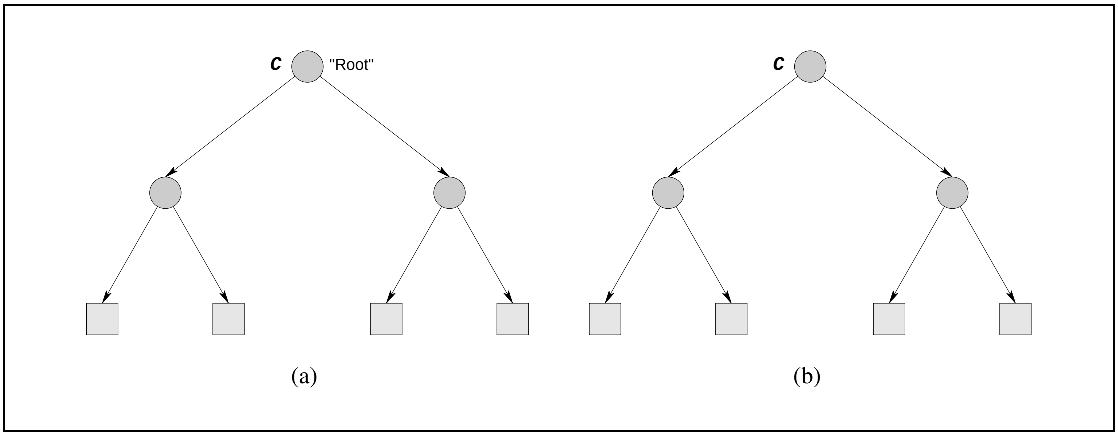

Abbildung 5.8: Änderung des *Content Type* von MIXED auf ANY: Der umzuformende Knoten darf nur noch den komplexen Wert $\mathcal{C}$ besitzen, der atomare Wert muß durch ein Update entfernt werden.

5.3 Komplexe Operationen

Abschließend werden drei komplexe Umformungsoperationen mit Hilfe des erweiterten OEM modelliert und erläutert.

5.3.1 Umbewegen von Elementen

Soll ein Element innerhalb des XML-Baumes umbewegt werden, dann müssen alle Subelemente, Attribute und Werte dieses Elementes mit umbewegt werden. Am erweiterten OE-Modell muß dafür lediglich das betroffene Element selber umbewegt werden, seine Bestandteile bleiben sozusagen daran “kleben”. Zum Umbewegen muß die Kante zum entsprechenden Elementknoten entfernt und eine neue generiert werden, die zum Elementknoten führt. Diese Kante muß außerdem von dem Elementknoten ausgehen, der als neuer Vater des Elementes fungieren soll. (Beispiel in Abbildung 5.9)

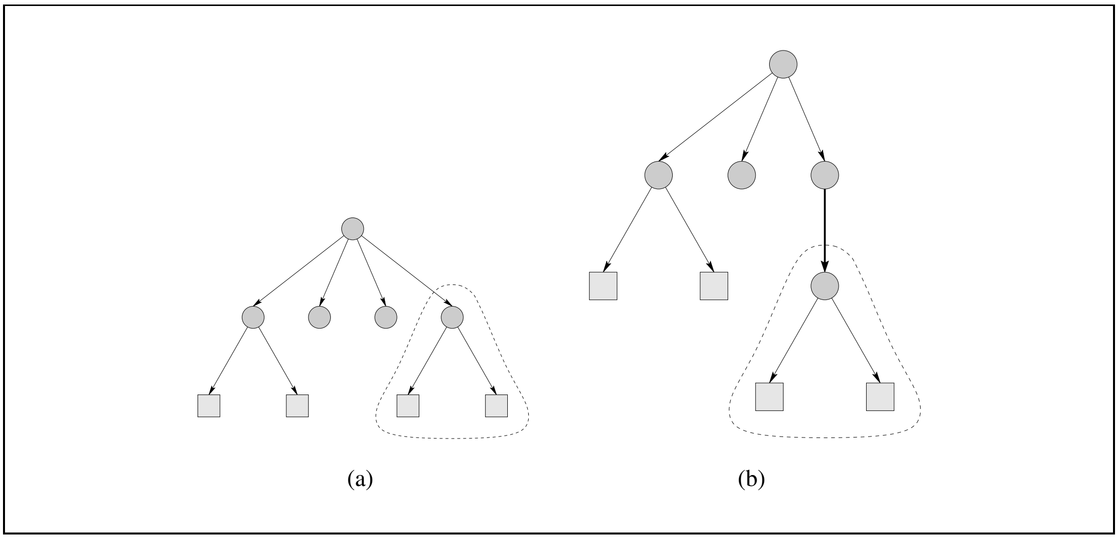

Abbildung 5.9: Umbewegen eines Elementes innerhalb des XML-Baumes

5.3.2 Verschachteln von Elementen

Zum Verschachteln (Nesten) von Elementen muß zunächst einer neuer Elementknoten samt Kante erzeugt werden, der die zu verschachtelnden Elemente als Subelemente aufnimmt. Danach werden die zu schachtelnden Elemente, wie im vorigen Abschitt beschrieben, umbewegt. D.h. die Kanten zu den betroffenen Knoten werden durch Kanten ersetzt, die vom neuen Vaterknoten ausgehen. (Beispiel in Abbildung 5.10 auf der nächsten Seite)

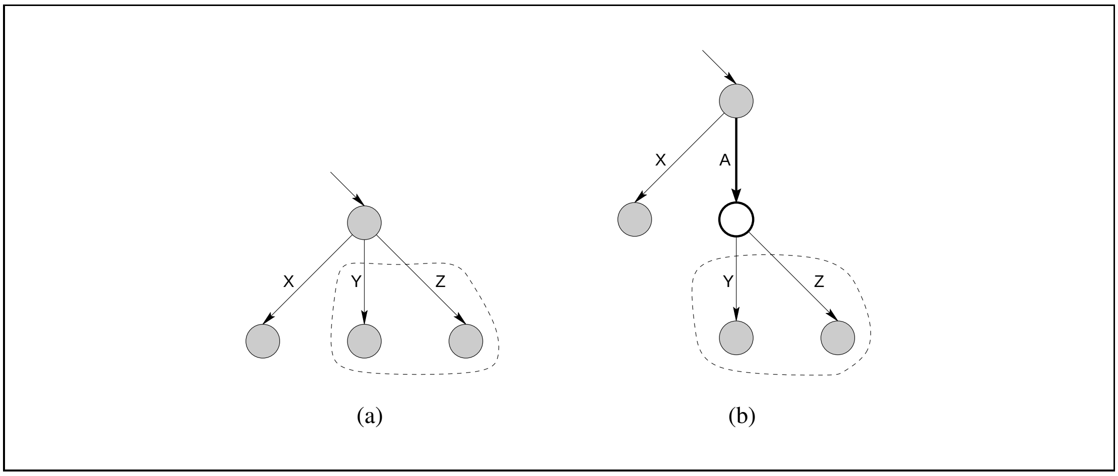

Abbildung 5.10: Verschachteln von Elementen

5.3.3 Auflösen von Elementen

Das Auflösen von Elementen (Entnesten) ist die Umkehroperation des Verschachtelns von Elementen (Nesten). Dabei werden alle Subelemente samt Werten und Attributen von einem Element in ein anderes umbewegt. Die Umsetzung dieser Operation gleicht dementsprechend dem Umbewegen von Elementen innerhalb des XML-Baumes. Der Unterschied besteht zum einen darin, daß beim Entnesten die betroffenen Subelemente um genau eine Ebene im Baum aufsteigen, beim “einfachen” Umbewegen kann die diese Ebene beliebig geändert werden. Zum anderen wird das zu “entnestende” Element am Ende der Umformung gelöscht. (Beispiel in Abbildung 5.11)

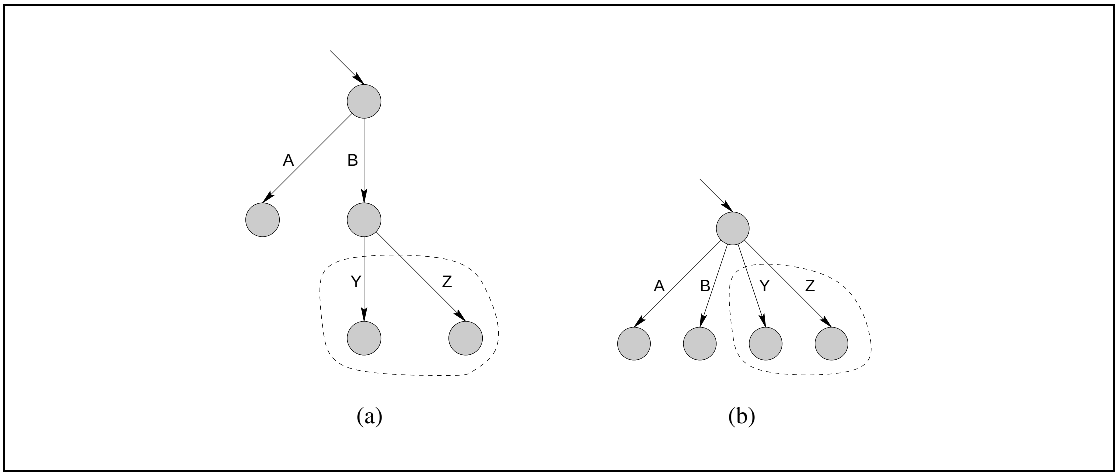

Abbildung 5.11: Entnesten eines Elementes

6 Schlußbetrachtung

Die Verwendung von semistrukturierter Daten hat in letzter Zeit, auch durch die rasante Entwicklung des Internet, stark zugenommen. Eine sehr gewichtige Rolle nimmt dabei XML als Vetreter der semistrukturierten Daten ein. Ziel dieser Arbeit war es zu untersuchen, welche Umformungen eines XML-Schemas in Form einer DTD welche Konsequenzen auf die Dokumente haben, die gültig bezüglich der umzuformenden DTD sind. Außerdem war grundsätzlich zu klären, welche Umformungen überhaupt und unter welchen Bedingungen möglich sind.

6.1 Zusammenfassung

Diese Arbeit gibt einen detaillierten Überblick darüber, welche sinnvoll erscheinenden Umformungen von DTDs und dazugehörigen XML-Dokumenten möglich und an welche Bedingungen diese geknüpft sind. Dabei stellte sich heraus, daß nahezu jede Umformung unter gewissen Bedingungen möglich ist. Jedoch läßt sich nicht verhindern, daß gewisse Umformungen einen Verlust von Daten nach sich ziehen.

Weiterhin wurde ansatzweise erläutert, in welcher Weise sich die Transformationssprache XSLT eignet, um die Umformungen auf den Dokumenten durchzuführen. Dabei wurde eine Grobarchitektur vorgeschlagen, die die Umformung von DTD und dazugehörigen Dokumenten realisiert. Die Basis dieser Architektur ist XSLT. Abschließend wurde anhand einer eigenen, einfachen Beschreibung von XML-Dokumenten eine Methode vorgestellt, mit der sich Umformungen solcher XML-Dokumente darstellen lassen.

6.2 Ausblick

Um die Thematik der "Evolution von XML-Dokumenten" weiter zu betrachten, ist die Bearbeitung folgender Punkte denkbar und wünschenswert:

- Die Umformungen beziehen sich nicht ausschließlich auf die Schemabeschreibung mit Hilfe von DTDs, sondern auch mit Hilfe von XML-Schema. Ein Grund dafür ist die größere Mächtigkeit von XML-Schema verglichen mit DTDs. Aufgrund der Tatsache, daß die Beschreibung eines Schemas durch XML-Schema ein XML-Dokument ist, wird es möglich, bei der Umformung von XML-Dokumenten mit XSLT auch auf Schemainformationen zuzugreifen, was bei DTDs im Moment nicht möglich ist.

- XSLT wird in Zukunft[1] die Möglickeit besitzen, zumindest auf Teile der DTD zuzugrei-

[1] Da im Moment noch an einer neuen Version des XSLT-Standards gearbeitet wird, ist dies nicht sicher.

fen. Damit ließe sich die in Kapitel 4 vorgestellte Architektur dahingehend verändern, daß ein Großteil der Aufgaben aus dem XML-Modul auf XSLT übertragen wird.

- Wichtige Grundlage für alle Umformungen ist eine formale Beschreibung aller Umformungsoperationen, sowohl für DTDs als auch für XML-Dokumente. Dafür ist ein Modell notwendig ist, mit dem sich XML-Dokumente und DTDs vollständig darstellen lassen.

- Ein weiterer und entscheidender Punkt ist die Implementierung eines Systems, daß alle Umformungen von DTDs und dazugehörigen Dokumenten weitestehend automatisiert durchführen kann.

- XML-Dokumente werden häufig auch in Datenbanken gespeichert. Es ist deshalb wichtig herauszufinden, wie sich die Änderung von XML-Dokumenten durchführen läßt, die in Datenbanksystemen gespeichert sind, ob dies überhaupt möglich ist und wie effizient.

A Lexikalische Anforderungen an XML-Attribute

Für die lexikalische Beschreibung von Attributen in XML exisitieren in [BPSM98, BPSMM00] zwei verschiedene EBNF-Produktionen. Dabei gibt es eine allgemeine und eine spezielle Produktion, so daß ein Attribut, daß den Anforderungen einer speziellen EBNF-Produktion gerecht wird, in ein anderes Attribut umgewandelt werden kann, dessen Aufbau einer einfacheren Produktion entspricht. Im umgekehrten Fall, also bei der Umwandlung eines Attributes von einer allgemeinen Produktion in ein Attribut, daß einer speziellen Produktion entspricht, kann es dagegen zu Problemen kommen, da im Attribut des allgemeinen lexikalischen Typs Zeichen enthalten sein können, die in einem Attribut eines speziellen lexikalischen Typs nicht vorkommen dürfen.

Tafel A.1 gibt eine Übersicht darüber, welche Attributtypen welchen EBNF-Produktionen entsprechen müssen und ob diese Produktionen allgemein oder speziell sind. Für Attribute vom Typ CDATA wurde keine Produktion festgelegt, Werte dieses Attributtyps dürfen beliebige Strings sein. Grundsätzlich gilt, daß Attribute von speziellen lexikalischen Anforderungen in solche mit allgemeineren ohne weiteres umgewandelt werden können[1], im umgekehrten Fall ist eine Umwandlung nicht immer möglich.

Grad	EBNF-Produktion	Attributtyp
Allgemein	—	CDATA
Speziell	Nmtoken	NMTOKEN, NMTOKENS, Enumeration
Sehr speziell	Name	ID, IDREF, IDREFS, ENTITY, ENTITIES, Notation

Tafel A.1: Lexikalische Ordnung von XML-Attributen

[1] Dies bezieht sich lediglich auf lexikalischen Anforderungen

B XSLT-Beispiele

Nachfolgend sind zwei vollständige XSLT-Files abgedruckt, die die Benutzung von XSLT verdeutlichen sollen. Das erste Stylesheet in Abbildung B.1 dient dazu, alle Elemente mit dem Namen a in x umzubenennen, alle anderen Elemente und alle Attribute sollen übernommen werden. Das erste Template im nachfolgenden Stylesheet übernimmt die Aufgabe des Umbenennens, indem für alle Elemente mit dem Namen a ein neues Element x generiert wird. Die Attribute des a-Elementes werden in das neue Element kopiert und eine rekursive Verarbeitung der Subelemente angestoßen. Im zweiten Template werden alle Elemente samt Attributen in den Ergebnisbaum kopiert, deren Name verschieden von a ist. Auch hier muß eine Verarbeitung der Subelemente iniziiert werden.

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

<xsl:template match="*[name()='a']">
  <xsl:element name="x">
    <xsl:for-each select="@*">
      <xsl:copy/>
    </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<xsl:template match="*[name()!='a']">
  <xsl:copy>
    <xsl:for-each select="@*">
      <xsl:copy/>
    </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Abbildung B.1: Umbennennung von Elementen mit XSLT

Mit Hilfe des zweiten Stylesheets aus Abbildung B.2 werden alle Attribute *Attr1* in *Attr2* umbenannt, sofern in der Attributliste noch kein solches Attribut vorkommt. Andernfalls wird keine Umbennenung vorgenommen. Attribute, die nicht *Attr1* heißen, werden wie alle Elemente einfach in den Ergebnisbaum kopiert.

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">


<xsl:template match="*[not(@Attr1) or @Attr2]">
  <xsl:copy>
    <xsl:for-each select="@*">
      <xsl:copy/>
    </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="*[@Attr1 and (not(@Attr2))]">
  <xsl:copy>
    <xsl:for-each select="@*">
      <xsl:choose>
        <xsl:when test="name()='Attr1'">
          <xsl:attribute name="Attr2">
            <xsl:value-of select="current()"/>
          </xsl:attribute>
        </xsl:when>
        <xsl:otherwise>
          <xsl:copy/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Abbildung B.2: Umbenennung von Attributen mit XSLT

Tafeln

Abbildungen

Literatur

[Abi97] ABITEBOUL, SERGE: *Querying Semi-Structured Data*. In: AFRATI, FOTO N. und PHOKION KOLAITIS (Herausgeber): *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece*, Band 1186, Seiten 1–18. Springer, 1997.

[AQM+97] ABITEBOUL, SERGE, DALLAN QUASS, JASON MCHUGH, JENNIFER WIDOM und JANET L. WIENER: *The Lorel Query Language for Semistructured Data*. International Journal on Digital Libraries, Volume 1:68–88, 1997.

[BHL99] BRAY, TIM, DAVE HOLLANDER und ANDREW LAYMAN (Herausgeber): *Namespaces in XML*. World Wide Web Consortium, Januar 1999. W3C Recommendation, verfügbar unter: *http://www.w3.org/TR/1999/REC-xml-names-19990114.html*.

[Boy00] BOYER, JOHN (Herausgeber): *Canonical XML, Version 1.0*. World Wide Web Consortium, Oktober 2000. W3C Candidate Recommendation, verfügbar unter: *http://www.w3.org/TR/REC-xml-c14n-20001026.html*.

[BPSM98] BRAY, TIM, JEAN PAOLI und C. M. SPERBERG-MCQUEEN (Herausgeber): *Extensible Markup Language (XML), Version 1.0*. World Wide Web Consortium, Februar 1998. W3C Recommendation, verfügbar unter: *http://www.w3.org/TR/1998/REC-xml-19980210.pdf*.

[BPSMM00] BRAY, TIM, JEAN PAOLI, C. M. SPERBERG-MCQUEEN und EVE MALER (Herausgeber): *Extensible Markup Language (XML), Version 1.0 (Second Edition)*. World Wide Web Consortium, Oktober 2000. W3C Recommendation, verfügbar unter: *http://www.w3.org/TR/2000/REC-xml-20001006.pdf*.

[Bra98] BRADLEY, NEIL: *The XML Companion*. Addison Wesley Longman, 1. Auflage, 1998.

[BT99] BEERI, CATRIEL und YARIV TZABAN: *SAL: An Algebra for Semistructured Data*. In: CLUET, SOPHIE und TOVA MILO (Herausgeber): *ACM SIGMOD Workshop on The Web and Databases (WebDB'99), Philadelphia, Pennsylvania, USA, Informal Proceedings, INRIA*, Juni 1999.

[Bun97] BUNEMAN, PETER: *Semistructured Data*. In: *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona*, Seiten 117–121. ACM Press, 1997.

[CAW98] CHAWATHE, SUDARSHAN S., SERGE ABITEBOUL und JENNIFER WIDOM: *Representing and Querying Changes in Semistructured Data*. In: *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA*, Seiten 4–13. IEEE Computer Society, Februar 1998.

[CD99] CLARK, JAMES und STEVE DEROSE (Herausgeber): *XML Path Language (XPath), Version 1.0*. World Wide Web Consortium, November 1999. W3C Recommendation, verfügbar unter: *http://www.w3.org/TR/1999/REC-xpath-19991116*.

[Cla99] CLARK, JAMES (Herausgeber): *XSL Transformations (XSLT), Version 1.0*. World Wide Web Consortium, November 1999. W3C Recommendation, verfügbar unter: *http://www.w3.org/TR/1999/REC-xslt-19991116.html*.

[GMW99] GOLDMAN, ROY, JASON MCHUGH und JENNIFER WIDOM: *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language*. In: CLUET, SOPHIE und TOVA MILO (Herausgeber): *ACM SIGMOD Workshop on The Web and Databases (WebDB'99), Philadelphia, Pennsylvania, USA, Informal Proceedings, INRIA*, Seiten 25–50, Juni 1999.

[Hul84] HULL, RICHARD: *Relative Information Capacity of Simple Relational Database Schemata*. In: *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Waterloo, Ontario, Canada*, Seiten 97–109. ACM, April 1984.

[PGMW94] PAPAKONSTANTINOU, YANNIS, HECTOR GARCIA-MOLINA und JENNIFER WIDOM: *Object Exchange Across Heterogeneous Information Sources*. Technischer Bericht, Stanford University Computer Science Department, 1994. Verfügbar unter: *ftp://db.stanford.edu/pub/papakonstantinou/1994*.