Gütekriterien für XML-Anwendungen -Studienarbeit-

M. Übner

23. August 2005

Inhaltsverzeichnis

1 Einleitung und Motivation

XML ist ein Standard zur Dokumentenauszeichnung, der in den letzten Jahren immer mehr an Bedeutung gewonnen hat. Viele Unternehmen gehen heute dazu über, ihre Informationen im XML-Format abzulegen, anstelle der Nutzung proprietärer Datenformate. Dies hat den Vorteil, dass die Informationen, die im XML-Format abgelegt wurden, plattformunabhängig, über Systeme hinweg ausgetauscht werden können und überdies hinaus auch noch vom Menschen lesbar sind. Informationen im XML-Format, können leicht von unterschiedlichen Applikationen mit Hilfe von Parsern, die in großer Anzahl zur Verfügung stehen, gelesen und verarbeitet werden.

XML ist als Standardformat für Computerdokumente sehr flexibel, denn es handelt sich genauer um eine Metaauszeichnungssprache. Das heißt, die Tag- bzw. Elementmenge ist nicht fest vorgegeben, so dass der Nutzer sie an seine persönlichen oder geschäftlichen Bedürfnisse anpassen kann. Dennoch gibt es eine Art Grammatik, basierend auf den Entwicklungen des W3C, für XML-Dokumente die vorschreibt, wie XML-Dokumente im gewissen Rahmen auszusehen haben. Dieser Umstand führte zur Entwicklung der vorhin schon angesprochenen Parser. Mit ihnen ist es möglich, die Wohlgeformtheit eines Dokumentes zu testen, was eine notwendige Bedingung für korrekte XML-Dokumente darstellt. Außerdem ermöglichen Parser für Anwendungen einen bequemen Zugang zum Inhalt von XML-Dokumenten, und es kann die Gültigkeit von Dokumenten bezüglich eines Schemas überprüft werden.

XML bietet als Strukturierungssprache eine ganze Reihe von Vorteilen zur Auszeichnung von Dokumenten. Elementnamen können bis auf einige wenige Restriktionen, was die verwendeten Zeichen angeht, frei gewählt werden. Die Elemente werden dann für eine nahezu beliebige Strukturierung und Auszeichnung der Dokumente eingesetzt. Nahezu beliebig heisst hier, dass die Regeln der Wohlgeformtheit beachtet werden müssen. Desweiteren ist es möglich, Attribute für Elemente zu formulieren. Es gibt zwar auch hier einige Einschränkungen, die zur Wohlgeformtheit der Dokumente verlangt werden, doch ist die Anzahl der Attribute, die Wahl der Attributnamen, bis auf die Zeichenrestriktionen für Attributnamen, und der Inhalt der Attribute sehr variabel, frei nach den Anforderungen der Anwendung gestaltbar.

Weitere große Vorteile von XML sind die Portabilität und die Ausrichtung auf lange Anwendbarkeit des Formates. Das heißt, XML kann man gute Chancen einräumen, auch noch in Jahrzehnten verwendbar zu sein. Spezielle Datenformate und Speichertechniken führten schon häufig zum Verlust von Informationen, wie beispielsweise im Fall der Mondlandung, wo große Mengen an Information durch proprietäre Datenformate und veraltete Hardware verloren gegangen sind. Weitere Vorteile sind die extreme Einfachheit, die gute Dokumentiertheit und die Eigenschaft als selbstbeschreibendes Datenformat. XML-Dokumente sind einfacher Text und können in der Regel von jedem Editor gelesen und modifiziert werden. Der vielleicht größte Vorteil von XML liegt in der explizit gegebenen Struktur der Dokumente, d.h. Zeilenendezeichen, Tabulatoren und Elementgrenzen sind direkt vermittelt. In Anbetracht all dieser Vorteile scheint die Zeit für viele der herstellergebundenen Datenformate abgelaufen. Die vielen Freiheiten in XML führen auch einige Probleme mit sich. Eine ganze Reihe von Erweiterungen sind

erst nach Verabschiedung des Standards XML 1.0 im Februar 1998 hinzugekommen. Zuerst wurden Namensräume hinzugefügt, dann kam die Extensible Stylesheet Language (XSL) gefolgt von weiteren und es gibt aktuell Entwicklungen zusätzlicher Erweiterungen, wie zum Beispiel der XML Query Language, um nur eine von vielen zu nennen. XML bildet die Grundlage für weitere Entwicklungen.

Das Hinzufügen solch neuer Module führt zum Veralten vorhandener Techniken. Parser sind teilweise nicht zu bestimmten Reaktionen auf Strukturen verpflichtet, so dass es schwierig ist, sich an Richtlinien zu orientieren, die schon bald nicht mehr relevant sein könnten. Damit ist zum Beispiel gemeint, dass ein validierender Parser ein externes Entity in ein Dokument substituieren muss, ein nicht validierender Parser hat hier noch Entscheidungsspielraum und muss nicht unbedingt das Entity einfügen [HaM03] Seite 53. Viele Erweiterungen von XML sind in diesem Sinne noch großen Wandlungen unterworfen, das macht exakte Aussagen zu einem guten Design recht schwierig.

Ein gutes Design ist von vielerlei Faktoren abhängig, auf die im Inhalt dieser Studienarbeit weiter eingegangen werden sollen, und somit sollen Empfehlungen gegeben werden können, wie man gute XML-Anwendungen erstellt.

Teils durch die in XML realisierte Liberalität, die die Freiheit zur Adaption an unterschiedliche Problemstellungen ermöglicht, und teils durch den Status von Erweiterungen, wie zum Beispiel XQuery, welches sich zur Zeit im Beta-Test befindet und auf Feedback von der XML-Community wartet, um dann mit der weiteren Arbeit fortzufahren, gibt es eine Reihe von Möglichkeiten, Fehlentscheidungen beim Design der Dokumente zu treffen. Ein konkretes Problem ist, dass unterschiedliche Personen unterschiedliche Tagmengen für semantisch gleich gemeinte Begriffe erstellen. Um solche Arten von Redundanzen und Uneindeutigkeiten zu vermeiden und um den Austausch und die Verarbeitungsfähigkeit von XML-Dokumenten zu verbessern, einigen sich Personen oder Organisationen manchmal auf bestimmte Tag-Mengen, welche man dann auch als Applikationen bezeichnet (eine XML-Applikation ist hier kein Programm, sondern eine Anwendung von XML in einem bestimmten Problemfeld).

Die so eingesetzte Methode ist ein möglicher Ansatz, das Problem der Redundanzen und Inkonsistenzen von XML in den Griff zu bekommen. Wie aber sollen sich Entwickler mit anderen schwierigen Designentscheidungen verhalten? Zu offen gehaltene Empfehlungen für nur teilweise umgesetzte Techniken in den aktuellen Browsern stellen ein großes Problem für XML dar. Die Qualität der Ausführung von Empfehlungen für Erweiterungen und Standards schwankt stark in den aktuell eingesetzten Browsern.

Es gibt eine Menge an Möglichkeiten, wie zum Beispiel die Verwendung von XLinks, die mit sinnvollen Konzepten Dokumente aufwerten können, die aber von der Masse der heute verwendeten Browsern nicht oder nur unvollständig unterstützt werden. Außerdem sind eine Vielzahl der Konzepte zu den Erweiterungen nur optional. Um beim Beispiel XLink zu bleiben, die Semantik für eine solche Verbindung ist nur eine nützliche Empfehlung und keineswegs vorgeschrieben. Viele der begangenen Fehlentscheidungen beim Konstruieren von XML-Dokumenten, wie zum Beispiel im Bereich der Lesbarkeit von Dokumenten oder bei der Wiederverwendung von Elementen, Dokumentteilen oder ganzen Dokumenten, könnte man mit Hilfe eines geeignet guten Designs in den Griff bekommen. Doch leider ist es zur Zeit so, dass es wenig Empfehlungen oder Hinweise

für ein gutes Design von XML-Dokumenten gibt. Es existiert kaum Fachliteratur und Publikationen im Internet sind wenig verlässlich bezüglich ihrer Verfügbarkeit. Außerdem stellt man bei genauerer Betrachtung fest, dass Veröffentlichungen konkret zu dem Thema Gütekriterien von XML oder gutes XML-Dokumentendesign nicht existieren. Einzig und allein Teilbetrachtungen guten Designs werden in einigen Veröffentlichungen vorgenommen, diese werden hier an entsprechender Stelle auch Beachtung finden.

Die Aufgabenstellung für diese Studienarbeit soll nun sein, Qualitäts- bzw. Gütekriterien von XML-Anwendungen zu identifizieren und solche auch neu zu formulieren. Aus diesen Kriterien sollen dann Eigenschaften abgeleitet werden, und es sollen Empfehlungen gegeben werden können, wie ein gutes Design von XML-Dokumenten unterstützt wird.

An dieser Stelle soll eine kleine Einführung in die unterschiedlichen Kontexte vorstellen, wie in den entsprechenden Applikationen Einfluss auf die Güte der XML-Inhalte genommen werden kann. Der erste Teil der Arbeit beschäftigt sich mit den XML-Grundlagen. Das beginnt mit der grundlegenden Konstruktion von Dokumenten und ihrem elementaren Aufbau. Es wird versucht, die durch XML bereitgestellten Basiskonzepte zu einer einheitlichen und sinnvollen Richtlinie zu formulieren. Ferner wird versucht, die Verwendung von Elementen und Attributen in den jeweiligen Situationen zu erklären. Es wird sich weiter mit den zur Verfügung stehenden Attributtypen und ihrem Einsatz in speziellen Situationen beschäftigt. Dokument-Type-Definitions (DTD's) sind auch ein essentieller Teil im Bereich der XML-Grundlagen, genau wie die erweiterten Möglichkeiten durch andere Schemata, wie zum Beispiel XML-Schema. Im weiteren Verlauf sind dann Erweiterungen wie XPath und XLink von Interesse, deren vollständige oder teilweise Unterstützung in Browsern und zukünftigen Erweiterungen in XML, wie zum Beispiel die Erweiterung der XML Query Language. Unabhängig von den angewandten XML-Techniken, muss man sich entweder für ein gutes Design für dokumentenzentriertes XML oder für datenzentriertes XML entscheiden. Aus diesem Grund wird bei der Formulierung der Kriterien im Bereich der Grundlagen eine Unterteilung in dokumenten- und datenzentriertes XML vorgenommen.

Im zweiten Teil der Arbeit rückt der Fokus auf allgemein wünschenswerte Eigenschaften von Softwareprodukten. Zum Erreichen der Gütekriterien wird unter anderem eine Analogiebetrachtung durchgeführt, die in anderen Bereichen der Informatik, wie zum Beispiele der Softwaretechnik und im Bereich der Datenbanken nach erstrebenswerten Eigenschaften der Produkte sucht, um diese dann auf XML-Dokumente übertragen zu können. Gerade im Bereich der Softwaretechnik gibt es schon einige bekannte Kriterien von Software, die auch im XML-Bereich hilfreich sein könnten. Als ein Beispiel soll hier die Wiederverwendbarkeit von Software genannt werden. Diese Eigenschaft kann man, in einem gewissen Rahmen, durch den Einsatz von Entities nachbilden. Die Modularisierung von XML-Inhalt und eine unterstützte Teamarbeit, sowie Wartbarkeit sind andere wünschenswerte Eigenschaften. Weitere interessante Kriterien sind der Grad der Verteilung von XML-Dokumenten und auch die vertretbare Größe von Dokumenten hinsichtlich der Verarbeitungsgeschwindigkeit.

Es lohnt sich auch, einen Blick auf den Sektor der Datenbanken, speziell der rela-

tionalen Datenbanken, zu werfen. Dort sind solche Eigenschaften wie Konsistenz und Redundanzfreiheit zur Übertragung auf XML-Dokumente von Interesse.

Der nächste Abschnitt beschäftigt sich mit dem Einsatz von Pattern im Bereich XML. Der Grundgedanke bei der Verwendung von Pattern ist, dass eine gute, allgemeine Lösung für ein ähnliches Problem bereits zur Verfügung steht. Anstatt eine neue und möglicherweise schlechte Lösung selbst zu entwickeln, sollte vorher der Einsatz von erprobten Pattern überprüft werden. Hierbei kommt dem Entwickler das Wissen und die Erfahrung vieler anderer Entwickler zugute, zusätzlich zu einer teils enormen Zeitersparnis. Beim Einsatz von strukturellen Pattern zielen die Vorteile auf das erleichterte Erstellen von XML-Dokumenten und der Steigerung der Qualität solcher Dokumente. Dabei kann man sich das Ziel des Erstellungsprozesses als ein Ausbalancieren von gegensätzlichen Kräften vorstellen. In nicht-trivialen Problemen ist das Ausblenden dieser Kräfte oft nicht vollständig möglich, d.h. man muss gewisse Kompromisse eingehen können [PAT]. Faktoren, die die Balance beeinflussen, sind zum Beispiel auch hier die Größe der Dokumente, die Einfachheit bei der Erstellung und die Leichtigkeit der Verarbeitung durch Programme.

Im darauf folgenden Kapitel soll dann ein Blick auf einige Ansätze aus der aktuellen Forschung geworfen werden. Diese Arbeiten beschäftigen sich häufig mit den Möglichkeiten, die durch die Document-Type-Definition und anderen Schemasprachen zu Verfügung gestellt werden. Ein anderer vielversprechender Ansatz ist, die Anzahl der Hierarchieebenen zu limitieren. Das visuelle menschliche Wahrnehmungsvermögen ist sehr schnell in der Lage, Strukturen zu erkennen. Dabei ist einleuchtend, dass einfachere Strukturen schneller erkannt werden, im Vergleich zu komplexeren. Die Hierarchisierung, sprich die Schachtelung von Elementen, entspricht einer Strukturierung. Wenn man die Hierarchisierung unbegrenzt zulassen würde, dann würde für den menschlichen Leser der Vorteil der Struktur, ab einem gewissen Grad der Hierarchie, verloren gehen. Auch für Maschinen, die XML-Dokumente verarbeiten müssen und dafür den Typ der DOM-Parser nutzen, wäre ein hoher Grad an Struktur in gewissem Sinne nicht von Vorteil. Die Verarbeitung eines Dokumentes mit Hilfe eines DOM-Parsers geschieht für das ganze Dokument vollständig, mit einem Mal. Ein DOM-Parser erstellt entsprechend des Dokuments einen Baum im Arbeitsspeicher. Sämtliche Anfragen und Änderungen beziehen sich auf diesen Baum. Ein hoher Grad an Hierarchisierung bedeutet eine hohe Knotenanzahl und würde den Dokumentenbaum stark wachsen lassen. Weil der erstellte Baum permanent im Arbeitsspreicher gehalten wird, ist eine Bearbeitung von großen Dokumenten mit einem DOM-Parser oftmals problematisch.

Der Abschnitt über die eigenen Ansätze beinhaltet dann Empfehlungen, die den Erkenntnissen der vorangegangenen Kapitel entliehen wurden und zu einem homogenen Konzept vereint wurden. Es wird eine Modellierung von XML-Anwendungen ähnlich der Modellierung relationaler Datenbanken vorgestellt.

Im Schlussabschnitt wird noch einmal das Problem zusammengefasst, es wird über das Erreichte resümiert und auf das noch Offenstehende verwiesen, und außerdem soll noch einmal versucht werden, auf Trends, aktuelle Tendenzen und Entwicklung aus der Forschung und der Praxis hinzuweisen, so dass eine Aussage bezüglich des Designs von XML-Dokumenten für die Zukunft ermöglicht wird.

2 Grundlagen guter XML-Dokumente

Im Bereich der Grundlagen guter XML-Dokumente sind viele Faktoren zu beachten. Bei XML-Dokumenten unterscheidet man, nicht ganz scharf, zwei Typen von Dokumenten. Die Rede ist hier einerseits von dokumentenzentrierten und andererseits von datenzentrierten Dokumenten. Die Charakteristik von datenzentrierten Anwendungen ist eine klar strukturierte, ähnlich der von relationalen Datenbanken. Für diesem Bereich ist es typisch, dass kein bis sehr wenig gemischter Inhalt auftritt, dieser lässt sich aufgrund seiner Art ohnehin schwer von Computersystemen behandeln bzw. weiterverarbeiten. Der übliche Anwendungsbereich von solch datenzentrierten Anwendungen konzentriert sich auf die Interaktionen zwischen Maschinen und auf den Bereich der Kommunikation zwischen Mensch und Maschinen.

Die Charakteristik von dokumentenzentrierten Anwendungen ist eine völlig andere. Dokumentenzentrierte Dokumente enthalten in der Regel sehr viel gemischten Inhalt. Ziel der Anwendung von XML ist es, einem Dokument eine Struktur zu verleihen, dies gilt auch für dokumentenzentrierte Anwendungen. Bei diesen Dokumenten handelt es sich vom Absatz eines Artikels, mit entsprechenden Auszeichnungen, über Bücher bis hin zu kompletten Buchbänden. Die Hauptzielgruppe solcher Anwendung ist der menschliche Konsument. Natürlich gibt es noch andere Verbraucher, wie Roboter und andere Suchmaschinen, die die Struktur solcher Dokumente schnell erfassen müssen.

Diese Unterscheidung nach Konsumenten führt uns gleich zur nächsten Unterscheidung bei den Gütemerkmalen von XML-Dokumenten. Es ist klar, dass man unterschiedliche Gütemerkmale bei unterschiedlichen Zielgruppen veranschlagen muss. Es gibt Anwendungen von XML, wo die erzeugten Dokumente zur Interaktion zwischen Maschinen nur wenige Minuten auf einem Server existieren. Ein Mensch wird wohl, außer aus Debugging-Gründen, ein solches Dokument nicht zu lesen bekommen. Aus diesem Grund ist ein Design, die menschliche Wahrnehmung unterstützend, kaum priorisiert. Hauptaufgabe ist es hier, dass die Maschinen die Struktur schnell erfassen und verarbeiten können.

Wenn man diese Einteilungen zusammenfasst, dann erhält man eine Tabelle mit zwei Spalten und zwei Zeilen, die die grundlegenden Zielbereiche für gute designte XML-Dokumente widerspiegeln.

Ausrichtung	Dokumententyp: datenzentriert	Dokumententyp: dokumentenzentriert
menschlich	datenzentriert mit menschliche Ausrichtung	dokumentenzentriert mit menschliche Ausrichtung
maschinell	datenzentriert mit maschineller Ausrichtung	datenzentriert mit maschineller Ausrichtung

2.1 Vier unterschiedliche Zielbereiche

Als Designer ist es wichtig, den Einsatzbereich der XML-Applikation zu kennen, um das Design der Dokumente entsprechend der Situation optimieren zu können. In den folgenden Abschnitten soll gezeigt werden, auf welche Feinheiten in den einzelnen Bereichen gesondert geachtet werden muss und auch welche allgemeinen Anforderungen an XML-Dokumente gestellt sein sollten. Bei der Einteilung in die vier Bereiche gibt es unvermeidbare Überlappungen bezüglich der Designrichtlinien, dennoch ist oftmals eine unterschiedliche Akzentuierung hinsichtlich dieser Anforderungen nötig. Es kann auch sein, dass bestimmte Anforderungen in einigen Bereichen keine Rolle spielen.

2.1.1 Datenzentriert / menschliche Orientierung

Datenzentrierte Dokumente mit dem Fokus auf der Verwendung durch den Menschen trifft man in der Praxis im Bereich von Formularen und tabellarischen Übersichten, bzw. Notationen häufig an . Außerdem sind sie klassisch bei der Interaktion und Kommunikation zwischen Systemen anzutreffen. Zum Beispiel werden heutzutage Konfigurationsdateien immer häufiger im XML-Format abgelegt oder Informationen werden in Tabellenform dargestellt. In den oben genannten Anwendungsfällen kann es die Situation erfordern, dass Menschen die Inhalte solcher Dateien lesen müssen. Entweder, um Konfigurationseinstellungen zu ändern oder um tabellarische Informationen auszuwerten. In einem solchen Kontext ist es dann sinnvoll, semantische Informationen gleich mit in den Element- und Attributbezeichnern zu kodieren. Kryptische Bezeichner machen Rechnern nichts aus, aber der Mensch würde dadurch in seinen Aufnahmefähigkeiten nicht unterstützt werden. Die Frage, wann man Elemente und wann man Attribute einsetzt, soll im allgemeinen Teil dieses Kapitels beantwortet werden, wenn auch eine allgemeingültige Antwort auf diese Frage nicht möglich ist, da sie immer vom Kontext der Anwendung abhängt.

2.1.2 Datenzentriert / maschinelle Orientierung

Dieser Zielbereich entspricht einer klassischen Anwendung von XML als rein datenzentrierte Applikation. Ein Beispiel können XML-Dokumente sein, die für die recht kurze Dauer einer Transaktionen auf einem Server erstellt werden und nachdem die Transaktion beendet ist, werden auch die XML-Dokumente wieder gelöscht. In einem solchen Kontext bräuchte man an semantiktragenden Tagnamen kein Interesse zu haben, es sei denn, die von Maschinen für Maschinen erzeugten Dokumente müssten von anderen Maschinen für Menschen durchsucht werden. Oder es müssen gelegentlich Debugging-Arbeiten an den Inhalten durchgeführt werden. In solchen Fällen wäre eine semantische Unterstützung hilfreich.

Außerdem gibt es in diesem Bereich weniger Probleme bezüglich der Komplexität von Dokumenten, da Maschinen in dieser Hinsicht sehr leistungsfähig sind. Schwächen bei verwendeten Parsern (wie z.B. Dom-Parser) müssen auch hier beachtet werden.

Der Hauptfokus liegt in diesem Bereich eindeutig auf der schnellen und komfortablen Verarbeitung von XML-Dokumenten. Dies wird durch die Wahl eines geeigneten

Parsers und der dazu passenden Dokumentgröße unterstützt. Desweiteren ist in diesem Zielbereich ein Auftreten von gemischter Inhalt nach Voraussetzung eher die Ausnahme. Gemischter Inhalt wird von der jeweiligen Anwendung in aufwendigen Analysen ausgewertet.

2.1.3 Dokumentenzentriert / menschliche Orientierung

Auch dieser Unterbereich entspricht einer klassischen Anwendung von XML. Die Informationen sind in einer narrativen Form dargestellt. Das heißt, Instanzen solcher Dokumente beinhalten Absätze, Artikel bis hin zu kompletten Büchern. Es ist üblich, dass solche Dokumente viel gemischten Inhalt enthalten, trotzdem existiert ein gewisser Grad an Strukturierung. Die Auszeichnung von Strings geschieht in solchen Dokumenten häufig aus dem Grund, um dem Anwender der Dokumente das Auffinden bestimmter Inhalte zu erleichtern. Die häufigste Anwendung dieser Art von Dokumenten ist auf den Menschen ausgerichtet. Deshalb ist es auch notwendig, die menschliche Wahrnehmung so gut wie möglich zu unterstützen. Das heißt, die Semantik von Elementen und Attributen muss mit in den Strukturbezeichnern integriert werden. Als Beispiel soll die Auszeichnung des oder der Autoren eines Buches dienen. Bei der Auszeichnung sollten Abkürzungen oder kryptischen Kodierungen vermieden werden.

2.1.4 Dokumentenzentriert / maschinelle Orientierung

Die Kombination ein dokumentenzentriertes XML-Dokument mit einer finalen Ausrichtung auf Maschinen zu konstruieren, dürfte in der Praxis höchst selten auftreten. Der vielleicht wichtigste Anwendungsfall von großer Bedeutung wäre, Anfragen von Suchmaschinen innerhalb solcher Dokumente. Letztendlich sind alle Fälle, wo Maschinen Informationen aus solchen Dokumenten herausfiltern, auf eine menschliche Ausrichtung zurück zu führen. Die Art der Strukturierung solcher Dokumente ist eine andere, wie sie der Verarbeitung durch Rechner entgegenkommt. Zusammenfassend muss gesagt werden, dass dieser Bereich eher theoretischer Natur ist und in der Praxis nicht auftreten wird.

2.1.5 Bemerkungen

Die empirische Erkenntnis ist, dass die Gütekriterien von XML-Dokumenten immer durch den zu modellierenden Problemkontext dominiert werden. Sowohl Nutzerausrichtung als auch Dokumententyp sind wesentliche Bestandteile dieses Kontextes. In der Praxis dürfte es kaum sinnvolle Anwendung für eine Ausrichtung dokumentenzentriert / datenorientiert geben.

In den praktischen Anwendungsfällen ist es wichtig, gerade die Ausrichtung des Nutzers mit dem Dokumententyp in Einklang zu bringen. Ist das Ziel ein menschlicher Konsument, dann sollten die Wahrnehmungsfähigkeiten des Menschen unterstützt werden. Auf der anderen Seite, wenn die maschinelle Weiterverarbeitung das Ziel ist, dann sollten die maschinellen Fähigkeiten maximal unterstützt werden. Der Problemkontext führt zu

einer nicht ganz scharfen, unexakten Einteilung in daten- bzw. dokumentenzentrierte XML-Instanzen.

2.2 Allgemeine Richtlinien

Wenn man sich mit dem Design von XML-Dokumenten beschäftigt, gibt es eine ganze Reihe von Regel, bei deren Beachtung man die Qualität der erstellten Dokumente steigern kann. Eine mögliche Fehlerquelle kann für den ungeübten User die *Casesensitivität in XML-Dokumenten* sein. Wenn ein Nutzer wenig Erfahrung im Umgang mit XML hat, oder sich nicht des Umstandes bewusst ist, dass Case-Sensitivität in XML eingesetzt wird, dann werden Entscheidungen zumeist zufällig und innerhalb des Dokumentes inkonsitent gefällt, ohne Rücksicht auf die zur Modellierung zur Verfügung stehenden Mittel, was insgesamt für ein schlechtes Design spricht. Daher ist es sinnvoll, innerhalb von Dokumenten nicht den Schreibstil zu wechseln. Das soll heißen, Elemente und Attribute nicht abwechselnd groß und dann wieder klein zu schreiben, oder gar Mischformen zu nutzen. Eine gute Idee ist es, im preamble des Dokuments in Form eines Kommentars, die Richtlinien für die Verwendung von Elementen und Attributen zu notieren.

Resultierende Probleme der Case-Sensitivität sind eher von temporär kurzer Dauer, doch behindern sie ein schnelles und damit effektives Design von Dokumenten.

Im Allgemeinen zeichnet es ein gutes Vorgehen aus, XML-Dokumente zur Unterstützung anderer Autoren stark zu kommentieren und die Entscheidungen, die beim Design getroffen wurden auch noch kurz zu erläutern. Beim Kommentieren kann ein Vorgehen aus der Programmierung adaptiert werden. Es stehen zwei unterschiedliche Arten von Kommentaren zur Verfügung.

Strategischer Kommentar	beschreibt, wie Strukturen verwendet werden (was mit Elementen gemacht wird). Steht deshalb vor der zu beschreibenden Struktur
Taktischer Kommentar	beschreibt eine Entscheidung innerhalb einer einzelnen Zeile. Steht deshalb auch auf der gleichen Höhe.

Ein vorhin schon erwähntes und immer wieder auftretendes Problem ist, die *Entscheidung* zu treffen, *wann Elemente modelliert und wann man auf Attribute* von Elementen zurückgreifen sollte. Es gibt zwei grundsätzlich vorherrschende Meinungen. Eine Meinung sagt, dass Elemente nur für Informationen verwendet werden sollten und Attribute sollen Metainformationen über die Elemente beinhalten. Zeichnet man in einer Anwendung ein Element mit `<Student>` aus, so müsste man die Metainformation der Matrikelnummer nicht als Kindelement, sondern in einer Form wie `<Student matrikelnummer=“012345“>` notieren. Die andere Meinung ist, dass die Trennung von Daten und Metadaten nicht immer so exakt ist und es sei von der Situation abhängig, wofür die Daten verwendet werden.

Eine Tatsache ist aber, dass die Struktur von Attributen stark eingeschränkt ist, da es sich einfach nur um einen einfachen String handelt. Eine elementbasierte Struktur ist da viel flexibler und zudem noch einfach erweiterbar.

2.2.1 Verwendung von DTD's

Der Zweck von Document Type Definitions, kurz DTD's, ist eine formale Syntax zu beschreiben, die den Aufbau von Elementen, einschließlich derer Attribute und Entities, sowie deren Kontext und Inhalt in Dokumenten beschreibt. Die für ein Dokument zutreffende DTD wird durch eine im Vorspann des Dokuments deklarierte Document-Typ-Deklaration vorgenommen. DTD's erlauben fundamentale Restriktionen an die Form von Dokumenten, dabei beschreiben sie eine allgemeine Form und keine konkrete Instanz eines Dokuments.

Einschränkungen, die bei der Verwendung von DTD's hingenommen werden müssen, sind, dass nichts über die Instanzen eines Elements, noch über das Aussehen der Zeichendaten ausgesagt werden kann. Es können keine Datentypen wie `int` oder `boolean` zu Elementen zugeordnet werden, dies liegt an der Entstehung und an der ursprünglich angedachten Verwendung von XML. Abhilfe kann da die später noch vorgestellte Schemasprache XML-Schema bringen.

Es gibt zwei, bzw. drei *Möglichkeiten DTD's einzusetzen*. Eine Möglichkeit ist, dass man die DTD in einer externen Datei speichert. Dabei ist empfohlen, die Endung der Datei als `.dtd` zu speichern, dies ist nur eine Empfehlung und nicht in der Spezifikation vorgeschrieben. Eine andere Variante ist, die DTD intern, im Vorspann des XML-Dokuments zu deklarieren. Dies ist vor allem bei einfachen DTD's vorteilhaft, da der Leser sie somit gleich zur Hand hat und nicht mit mehreren Dokumenten umgehen muss. Die dritte Möglichkeit ist eine Mischform der ersten beiden. Das heißt, es kommt sowohl eine interne als auch eine externe DTD zum Einsatz. Wenn man *interne und externe DTD's gemeinsam verwendet, dann kann es zu ungewollten Effekten kommen*, hervorgerufen durch Umdefinitionen bzw. durch Überdeckungen zwischen den beiden DTD's. Ein Parser ließt zuerst die interne DTD, dann die externe DTD. Deshalb haben die Definitionen der internen DTD Vorrang vor denen der externen. Auf der anderen Seite, kann durch ein Mischen und dem zusätzlichen Einsatz von Parameterentities ein nützlicher Schaltermechanismus realisiert werden.

```
<?xml version=''1.0''?>
<DOCTYPE student [
   <!ELEMENT vorname     (#PCDATA)>
   <!ELEMENT nachname    (#PCDATA)>
   <!ELEMENT matrikel    (#PCDATA)>
   <!ELEMENT name        (vorname, nachname)>
   <!ENTITY % mitarbeiter ''INCLUDE''>
   <!ENTITY % zusatz SYSTEM ''zusatz.dtd''>
   %zusatz;
]>
```

Die externe DTD `zusatz.dtd` enthält eine Deklaration der Form:

```
<![%mitarbeiter; [
   <!ELEMENT mitarbeiternummer (#PCDATA)>
]]>
```

Somit wird es möglich, ein Parameterentity der externer DTD in der internen DTD zu redefinieren, um Informationen modular ein- bzw. auszublenden.

Um ungewollten Effekte zu verhindern, ist der Einsatz eines Algorithmus denkbar, welcher für ein XML-Dokument mit interner DTD und eine externe DTD überprüft, ob es Überschneidungen gibt und wenn dies der Fall ist, könnten entsprechende Warnhinweise ausgegeben werden.

Wenn sich für den Einsatz von DTD's entschieden wurde, ist es ein guter Stil, wenn die Reihenfolge der Deklarationen für Elemente und Attribute in allen verwendeten DTD's nach dem gleichen Prinzip gestaltet würde. Für den Parser ist die Deklarationsreihenfolge der Elemente nicht relevant. Eine gute Variante wäre ein Postorder-Aufbau. Ein Postorder-Aufbau bzw. -Durchlauf verarbeitet ausgehend von einem Baum zuerst den linken Teilbaum in Postorder, dann den rechten Teilbaum in Postorder und zum Schluss die Wurzel des Baumes. Auf diese Weise würden Deklarationen mit einem niedrigen Abstraktionslevel unmittelbar bei ihren zugehörigen Elementdeklarationen mit einem höheren Abstraktionslevel stehen. Zugehörigkeit meint hier, dass Elemente in anderen Deklarationen weiterverwendete werden. Ein Beispiel ist die Deklaration einer Adresse. Die Subelemente von Adresse, nämlich `Straße, PLZ,` und `Wohnort` würden direkt über der Deklaration von `Adresse` erscheinen. Als Problemlösungstrategie entspricht dies der bottom-up-Methode.

Eine andere Vorgehensweise wäre ein Preorder-Aufbau bzw. -Durchlauf. Dieses Vorgehen ist top-down. Ausgehend von einem Baum wird zuerst die Wurzel, dann der linke Teilbaum in Preorder und zum Ende der rechte Teilbaum in Preorder durchlaufen. Bei diesem Vorgehen würden zuerst die Elemente der höchsten Abstraktionsebene deklariert.

Bei der Deklaration von Elementen bestimmt man den Inhalt der Elemente, bestimmt die Reihenfolge der Inhaltselemente und auch die Häufigkeit ihres Auftretens. Dabei ist möglich, den Inhalt eines Elementes ausschließlich durch ein einzelnes Element zu definieren. Wenn ein solcher Fall auftritt, sollte überprüft werden, ob diese Art der Deklaration sinnvoll ist oder ob es möglich ist, die Deklaration der Elements auf die Deklaration des Inhaltselementes zurückzuführen. Auf diese Weise würden Ketten von einelementigen Elementdeklarationen vermieden.

An dieser Stelle sollen bezüglich der Verwendung von *Entities* einige Hinweise gegeben werden. Mit Entities ist es möglich, einmal einen Inhalt zu definieren und diesen dann beliebig oft im Dokument durch den Einsatz des Entities zu substituieren. Dabei muss der Ersetzungstext des Entities in jedem Fall wohlgeformt sein. Zyklische Referenzen sind nicht zulässig bei der Deklaration von Entities, das schließt auch den Fall der Eigenreferenzierung ein. Wie schon weiter oben erwähnt, kann es beim Einsatz von *extern geparsten Entities* zu Problemen kommen. Bei diesen Entities wird der Inhalt in einer externen Datei abgelegt. Im eigentlichen XML-Dokument wird die Referenz auf das Entity durch eine Entity-Deklaration in der DTD deklariert. Das möglicherweise auftretende Problem ist, dass der Parser die Referenz im Dokument nicht durch den Ersetzungstext ersetzt. Ein nicht-validierender Parser ist nicht zur Ersetzung externer Entity-Referenzen verpflichtet, ein validierender Parser schon, [HaM03] Seite 53.

Probleme können auch beim Gebrauch von ungeparsten Entities auftreten. Der Zweck ungeparster Entities ist der, dass nicht-XML-Inhalt in XML-Dokumente integriert wer-

den soll. Dabei wird ein ungeparstes Entity als Wert eines Attributes eines Elements in das XML-Dokument eingefügt. Gegenwärtige Probleme sind dabei, dass XML kein Verhalten von Anwendungen beim Entdecken solcher Entities garantieren kann, [HaM03] Seite 55. Als Beispiel soll das Einfügen eines Bildes in ein XML-Dokument dienen. Der Parser kann beim Entdecken des Entities nicht wissen, um was für einen Typ von Inhalt es sich handelt. Somit ist es wahrscheinlich, dass das Bild nicht angezeigt, nicht einmal geladen wird. Zwar sind Hinweise auf den Inhalt durch die Nutzung von `<!NOTATION ... >`-Elementen möglich, aber in der Praxis wird dieses Konstrukt selten angewendet, außerdem gibt es weder einen Standard noch einen Vorschlag wie die Identifikatoren der Inhalte für Notation-Elemente auszusehen haben.

Laut Expertenmeinung ist der Gebrauch von ungeparsten Entities kompliziert und fehlerträchtig und hätte nicht in XML aufgenommen werden sollen, zumal es Alternativen, durch XLink und URL's, für die Integration von nicht XML-Inhalt gibt, [HaM03] Seite 55.

Werden DTD's zur Strukturierung von XML-Dokumenten eingesetzt, sollte vorher der Einsatz von Standard-DTD's überprüft werden. Dies hat den Vorteil des reibungslosen Austausches von Dokumenten in konkreten Anwendungsszenarien. Bis heute gibt es keine allumfassende Quelle für DTD's, obwohl es Versuche in diese Richtung gegeben hat.

- http://www.schema.net/ von James Tauber
- http://www.xml.org/xml/registry.jsp von OASIS
- http://www.biztalk.org von Microsoft

Außerdem ist das W3C eine große Quelle für standardisierte DTD's. Da der bei weitem größere Anwendungsbereich von XML außerhalb des Internets liegt, hat wahrscheinlich jede Industrie die den Einsatz von Informationstechnik forciert, bereits ihre eigenen DTD's generiert. Somit obliegt es dem Dokumentdesigner, wenn vorhanden, geeignete DTD's aus entsprechenden Domänen zu recherchieren.

2.2.2 Einsatz von Namensräumen

Bei großen XML-Dokumenten mit Inhalt aus verschiedenen Applikationsbereichen sind Namensräume notwendig und hilfreich. Namensräume werden einerseits zur Unterscheidung von gleichbenannten Elementen und Attributen einer Applikation in einem Dokument genutzt. Dabei ist die Applikation hier gleichzusetzen mit einem Vokabular, bzw. einem anwendungsspezifischen Kontext. Im Umkehrschluss werden Namensräume auch dafür verwendet, Elemente und deren Attribute in einer Applikation zusammenzufassen.

Die Definition von Namensräumen geschieht durch die Bindung eines Präfixes an einen URI (Unified Resscource Identifier) in einem xmlns-Attribut eines Elements. Es ist darauf hinzuweisen, dass beim Einsatz von Namensräumen die Attribute von mit Präfixen versehenen Elementen nicht automatisch dem gleichen Namensraum zugeordnet sind. Solche Attribute ohne Präfix befinden sich weiterhin in keinem Namensraum, auch dann,

wenn ein default-Namensraum zugeordnet wurde. Eine weitere wichtige Information ist, dass die Parser die URI's kontrollieren, da diese standardisiert sind. Die Namen der Präfixe sind nebensächlich, es sind die URI's die Zeichenweise vom Parser zum Vergleich benutzt werden. Abweichungen bei Zeichen bedeuten ungleiche Namensräume.

Die Namensraum-Deklaration sollte in Abhängigkeit von der Verteilung der jeweiligen Elemente vorgenommen werden. Bei Elementen eines Vokabulars die stark verteilt sind, sollte die Deklaration des Namensraumes im Wurzelelement vorgenommen werden. Eine starke Verteilung bedeutet, dass die Elemente eines Applikationsbereiches über das gesamte Dokument verteilt sind. Sind die Elemente konzentriert, mit wenig anderem Vokabular, so ist die Deklaration des Namensraumes im Elternelement zu realisieren. Man kann sicherlich die Namensraumangaben vollständig im Wurzeldokument des Dokuments unterbringen, dass ist bei den oben geschilderten Situationen wenig nützlich oder hilfreich.

Ein schlechter Stil wäre es, Präfixe innerhalb eines Dokuments umzudefinieren. Es ist möglich, ein Präfix in einem Element zu deklarieren und es dann in einem anderem Element mit einem neuen URI zu kombinieren. In einem solchen Fall hat das Präfix des nächst gelegenen Vorfahren oberste Priorität. Da es jedoch keine sinnvolle Anwendung für ein solches Vorgehen gibt, sollte ein Umdefinieren in allen Fällen vermieden werden, [HaM03] Seite 69.

Falls es nötig sein sollte, die Präfixe in Dokumenten zu ändern, wäre der Einsatz von Parameter-Entity-Referenzen sehr sinnvoll. Die Idee ist die, dass Namensraum-Präfix und den Doppelpunkt jeweils durch ein Parameter-Entity zu ersetzen. Bei notwendigen Änderungen muss nur das Entity geändert werden. Dieses Vorgehen kann bei großen Dokumenten eine enorme Aufwandsersparnis bedeuten.

2.2.3 Internationalisierung

XML-Dokumente bestehen aus Unicode-Text. Unicode ist ein Zeichensatz der in verschiedenen Kodierungen auftreten kann. Ein XML-Dokument kann auch in einem anderen als dem Unicode-Zeichensatz erstellt werden, beim Parsen wird es dann in Unicode transformiert. Es ist immer ratsam Zeichensatz-Metadaten, sprich die Encoding-Deklaration, in allen Dokumenten mit anzugeben. Zwar ignorieren Parser häufig solche Informationen, Nachteile können aus so einer Deklaration jedoch nicht erwachsen.

Es sollten immer die Zeichensatz-Informationen für extern geparste Entities angegeben werden, da sich diese vom Hauptdokument unterscheiden können bzw. sie von mehreren Dokumenten unterschiedlicher Kodierung verwendet werden können. Man sollte nur die Standardnamen für Standardkodierungen verwenden, denn Programme erkennen manchmal Aliasbezeichnungen für Zeichensätze, dieses wiederum schränkt die Portabilität ein. Bei Verwendung von Kodierungen die nicht in der XML 1.0 Spezifikation vorgenommen wurden, sollte man die Namen, die bei der IANA (Internet Assigned Number Authority) unter ftp://ftp.isi.edu/in-notes/iana/assignments/charakter-sets registriert sind, anwenden.

Der Einsatz von plattformabhängigen Zeichensätzen sollte vermieden werden, da eine weitreichende Unterstützung nicht gewährleistet werden kann. Bei häufigem Einsatz

von Zeichenreferenzen sollten Entities zum Einsatz kommen. Bei einem geeignet gewählten Bezeichner steigert dies auch die Lesbarkeit von Dokumenten. Als Beispiel wird die Deklaration von Griechischen Buchstaben angegeben. Die Zeichensatzreferenz für den griechischen Buchstaben ϵ lautet `''#x3B5''`. Es ist ein deutlicher Vorteil, wenn im Preamble des Dokumentes eine Entity-Deklaration der Form `<!ENTITY epsilon ''#x3B5;''>` vorgenommen wird und diese dann anstelle der unverständlichen Zeichensatzreferenz mittels `&epsilon;` im Dokument substituiert werden kann.

2.2.4 XML als Dokumentenformat

Wenn man XML als Dokumentenformat einsetzt, ist klar, dass der Anwender solcher Dokumente menschlich ist. XML ist ausgezeichnet für große Texte geeignet, da die Wurzeln in SGML zu finden sind. SGML ist wiederum ein allgemeiner Standard, wenn auch ein komplexer, zur Dokumentenauszeichnung. Ein Merkmal von textorientierten Texten ist, dass sie eine kaum erkennbare Struktur besitzen. Es ist ein gebräuchliches Muster, Metainformationen über solche Dokumente in einem Kind des Wurzelelementes zu kapseln, [HaM03] Seite 95. Solche Dokumente sind im Allgemeinen in Unterabschnitte, bei Büchern sind es Kapitel, Abschnitte und Absätze, unterteilt. Auszeichnungen sind hier kein fundamentaler Bestandteil.

Eine wichtige Anforderung ist, dass solche Dokumente für eine lange Zeit beständig sein müssen. Dabei bringen Schemasprachen keine Vorteile gegenüber DTD's. Auch aus dem Grund der Dauerhaftigkeit der Dokumente sollte das Format, z.B die DTD, sehr gut dokumentiert sein, obwohl generell gilt, dass eine gute Dokumentierung immer von Vorteil ist. Wenn man XML-Dokumente dokumentenzentriert verwenden möchte, sollten Standardformate wie TEI, die Text Encoding Initative (*http://www.tei-c.org/*, oder DocBook (*http://www.docbook.org/* als Mittel der Wahl eigenen Applikationen vorgezogen werden. TEI ist ein sehr gutes Beispiel für eine narrative DTD, mit Stärken in der wissenschaftlichen Analyse von Texten, [HaM03] Seite 96. DocBook ist für neuere Dokumente, speziell bei Computerdokumentationen gebräuchlich. Dabei handelt es sich eigentlich um ein Format zum Erstellen von Dokumenten, d.h. es muss zuvor in ein repräsentatives Format, zum Beispiel HTML oder Rich Text Format (RTF) konvertiert werden, [HaM03] Seite 102.

Bei der Verwendung von DTD's für dokumentenzentriertes XML sollten freie, nicht kommerzielle DTD's proprietären vorgezogen werden. Es sollte auch vermieden werden Standard-DTD's zu modifizieren. Für Elemente und Attribute sollten vollständige Namen verwendet werden und die Struktur von Dokumenten sollte nicht durch Leer- oder Trennzeichen gestaltet werden, sondern allein durch Auszeichnungen. In XML sollte generell (und deshalb auch in dokumentenzentrierten Dokumenten) kein Inhalt durch Stylesheets dargestellt werden.

2.2.5 XPath

Bei der Nutzung von XPath-Ausdrücken gibt es auch die Möglichkeit, nicht abgekürzte Lokalisierungspfade zu verwenden. Diese nicht abgekürzten Lokalisierungspfade bieten

weitere Möglichkeiten an, da sie sehr ausführlich sind und Zugriff auf die meisten Achsen in XML gewährleisten. Trotzdem ist zu beachten, dass die nicht abgekürzten Lokalisierungspfade im XSLT-Mustervergleich nicht zulässig sind. Das heisst, wenn man mit XSLT-Stylesheets arbeiten möchte, ist zu berücksichtigen, das die in XSLT formulierten Template-Regeln nicht mit den vollständig angegebenen Pfaden zusammenarbeiten werden.

2.2.6 XLink

Die Verwendung von XLink's in XML-Dokumenten kann die Qualität der Dokumente erheblich steigern. XLink ist sehr flexibel, die Unterstützung reicht bis hin zur Multidirektionalität. Ein aktuelles Problem ist, dass Browser, wenn überhaupt, nur einfache XLinks unterstützen. XLink-Elemente können durch Browser interpretiert werden, wie diese es wollen. In anderen Fällen hängt das Verhalten von den Applikationen ab. Ein Hinweis zur Verbesserung von Dokumenten mit XLink's ist, dass die Link-Semantik immer mit integriert werden sollte. XLink-Elemente können Attribute wie `xlink:title` und `xlink:role` besitzen, mit diesen können dann Informationen über die Verbindung beschrieben werden. Mit `xlink:title` werden Informationen in Textform, ähnlich eines Tooltips angefügt und mit `xlink:role` wird auf eine URI verwiesen, die eine entfernte Ressource kennzeichnet.

2.2.7 XML-Schema

Im Unterschied zu DTD's, die zur fundamentalen Strukturierung von Dokumenten eingesetzt werden, nutzt man Schemasprachen zur Formulierung komplexerer Restriktionen an Dokumente. Die drei am weitesten verbreitetsten Schemasprachen sind XML-Schema, Relax-NG und Schematron. Als entsprechender Vertreter soll XML-Schema im Folgenden näher betrachtet werden. Als Hinweis sei noch kurz zu Schematron gesagt, dass hier ein völlig neuer Ansatz für Schemasprachen gewählt wurde. Schematron basiert nicht auf einer Art Grammatik, sondern vielmehr auf dem Identifizieren von Mustern innerhalb der Instanzbäume. Durch diesen Ansatz ist es möglich, Strukturen zu repräsentieren, die in anderen grammatikbasierten Schemasprachen schwerer zu beschreiben sind [Sch05].

Eine Besonderheit von Relax NG ist, dass es neben Eigenschaften, wie Einfachheit und der Unterstützung von Namensräumen, die es mit anderen Schemasprachen teilt, sowohl eine XML-Syntax als auch eine kompakte Nicht-XML-Syntax unterstützt [Rel05].

Ein Vorteil von XML-Schema ist, dass ein Schemadokument selbst auch ein gültiges XML-Dokument ist. Das heißt, die Verarbeitung und Manipulationen erfolgen auf die gleiche Weise wie bei Instanzdokumenten.

Deklarationen in "höheren" Schemasprachen bieten viele Vorteile und ermöglichen ein höheres Niveau, dennoch ist die Anwendung von DTD's nicht obsolet wie ein Vergleich zeigen wird. DTD's erlauben gerade im Bereich der

- Verschachtelungen von Elementen
- Häufigkeitsbeschränkungen von Elementen
- Zulässigkeit von Attributen
- Attributtypen und Defaultwerte

vergleichsweise einfache Konstruktionen [HaM03]. Die dargebotenen Konzepte scheinen speziell im Bereich der narrativen Formate ausreichend. Außerdem sind DTD's auf dem Gebiet der Deklaration von Entities weiterhin äußerst wertvoll.

Wie oben angedeutet, sind die Möglichkeiten die durch XML-Schema geboten werden von größerer Mächtigkeit als die der DTD's. Dies ist auch zur Formulierung komplexerer Strukturinhalte notwendig. Die wichtigsten Beiträge von XML-Schema sind Möglichkeiten zur Formulierung von

- einfachen und komplexen Datentypen
- Ableitungen und Vererbung von Typen
- Häufigkeitsbeschränkungen für Elemente
- namensraumsensitiven Element- und Attributdeklarationen

Der größte Fortschritt sind die Ergänzung von Datentypen für geparste Zeichendaten und Attributwerte. Außerdem ist es möglich, spezifische Regeln für Elemente und Attribute zu formulieren. Zum Beispiel kann die Anzahl und Reihenfolge von Kindelementen innerhalb eines Dokumentes an konkreten Stellen beschrieben werden.

Ein weiterer Vorteil ist, dass Namensräume in XML-Schema berücksichtigt werden. In DTD's ist das nicht der Fall, dies aus Gründen der Chronologie der Entstehung. Es ist angeraten, Namensräume möglichst früh bei Beschreibungen von Dokumenten mit XML-Schema einzuführen, da andernfalls der Aufwand für Änderungen von Namensräumen beträchtlich werden kann.

Da an dieser Stelle nicht die Syntax der Strukturbeschreibungssprache XML-Schema vorgestellt werden soll, wird auf detailliertere Darstellungen verzichtet. Deklarationen in XML-Schema sind als Konstrukt möglicherweise mächtiger, aus diesem Grund aber auch komplexer und aufwändiger als in der Sprache der DTD. Die Darstellung einer Schachtelung von Elementen inklusive ihrer Attribute erfordert in XML-Schema eine vergleichsweise umfassende Konstruktion mit "komplexen Typen" und Ableitungen von Standard-Datentypen. Für Informationen zu diesem Thema sei zum Beispiel auf [Vli03] oder [SkW04] verwiesen.

An dieser Stelle seien aber noch zwei Hinweise, einen guten Stil unterstützend, gegeben. Es ist üblich, auch das Schema-Dokument vom Instanz-Dokument aus zu referenzieren. Das erreicht man einfach durch das Verwenden des Attributes `schemaLocation`, falls man einen Namensraum nutzt, bzw. durch `noNamespaceSchemaLocation`, falls man keinen Namensraum verwendet.

Der zweite Hinweis betrifft die Einbindung von Metainformationen. Wird ein XML-Schema definiert, ist es guter Stil, Metainformationen über das Dokument, zum Beipiel über den Autor, den Verwendungszweck oder die Sprache in das Dokument zu integrieren. XML-Schema bietet zu diesem Zweck die Möglichkeit, zusätzliche Informationen durch ein `<xs:annotation>-Element` an ein Schema-Element anzufügen. Ein annotation-Element wird optional als erstes Kindelement dem zugeordneten Schema-Element angefügt. In dem <xs:annotation>-Element kann wiederum eine beliebige Anordnung der Elemente `<xs:documentation>` und `<xs:appinfo>` auftreten. Zwischen beiden Elementen gibt es keinen Unterschied. Ihre Existenz beruht auf einer unterschiedlichen Intention der Entwickler bei der Verwendung. Beider Elemente sind in der Lage beliebige Anordnungen, sowohl von Zeichendaten, als auch von Markup aufzunehmen. Folgendes Beispiel veranschaulicht obige Erklärungen.

```
<xs:schema xmlns:xs=''http://www.w3.org/2001/XMLSchema''>
   <xs:annotation>
      <xs:documentation xml:lang=''de''>
         Einfaches Beispiel aus O'Reilly's
         <a href=''http://www.oreilly.com/catalog/xmlnut''>XML
         in a Nutshell.</a>
         Copyright 2002 O'Reilly &amp;amp; Associates
      </xs:documentation>
   </xs:annotation>
   <xs:element name=''vollstaendigerName'' type=''xs:String''/>
</xs:schema>
```

3 Analogien aus dem Bereich der Praktischen Informatik

Zu betrachtende Analogien kommen einerseits aus dem Bereich der Softwaretechnik und zum anderen aus dem Bereich der Datenbanken.

Bereich Softwaretechnik Im Bereich der Softwaretechnik wurden eine Reihe von Qualitätseigenschaften für Softwareprodukte formuliert. Allgemein angestrebte Eigenschaften, die sich auf XML-Inhalt übertragen lassen, sind

- Wiederverwendbarkeit von Software
- gute Wartbarkeit von Software
- Modularisierung von Software
- Portabilität von Software
- Effizienz von Software
- Verständlichkeit von Software
- Flexibilität von Software

Aus dem Bereich der Datenbanken zu kopierende Eigenschaften sind

- Redundanzfreiheit
- Normalformen

Einige der Eigenschaften können und werden bereits gut für XML-Inhalt angewendet. Die Modularisierung wird durch Multi-File-Strukturen unterstützt. Andere Eigenschaften überlappen sich bei entsprechender Anwendung. Effizienz und Redundanzfreiheit in XML-Dokumenten können sowohl in enger Verbindung stehen, im Kontext der Speicherauslastung, sie können sich jedoch auch konträr verhalten, falls Daten für eine entsprechende Anwendungsperformance redundant gespeichert werden müssen.

3.1 Wiederverwendbarkeit, Modularisierung von XML-Inhalt

Im allgemeinen Kontext betrachtet sind Eigenschaften von Software wie *Wiederverwendbarkeit, Modularisierung, Flexibilität* und *Wartbarkeit* mehr oder weniger stark korreliert.

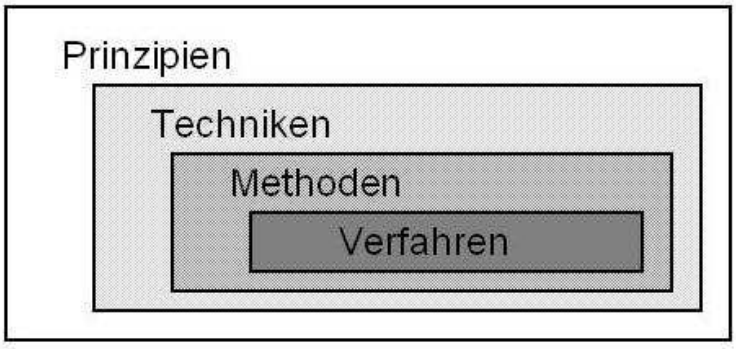
Wiederverwendbarkeit von Software bedeutet, Software-Lösungen im Ganzen oder nur Teile, für gleiche oder ähnlich strukturierte Probleme einsetzen zu können. Die Realisierung der Eigenschaft bedeutet eine Abstraktion von der konkreten Problemstellung hin

zu einem verallgemeinerten Lösungsansatz. Der verallgemeinerte Ansatz ist in der Lage, Klassen von Problemen zu verarbeiten.

Im Bereich von XML-Anwendungen wird versucht, Klassen von Problemen durch Standardisierungen zu abstrahieren. Als Lösung bieten sich die Möglichkeiten unterschiedlicher Schema-Sprachen, wie zum Beispiel DTD's oder XML-Schema, und in ihrer Folge die Erstellung von Namensräumen für Problemklassen. Äußerungen solcher Lösungsstrategien sind Subsprachen wie MathML, der Dublin-Core-Standard oder das Ressource-Description-Framework, RDF.

Ein Problem bei Namensräumen ist, dass Lösungen in unterschiedlich komplexen Graden erstellt werden. Das führt zu Schwierigkeiten im Bereich der Anwendung solcher Standards. Unterschiedliche Anwendungen stellen unterschiedliche Ansprüche an den Grad der Abstraktion des Anwendungsbereiches. Somit kann es für den Anwender schwer werden, die Übersicht über alle zur Verfügung stehenden Konzepte eines Applikationsbereiches zu behalten, um eine optimale Lösung zu formulieren.

In der Softwaretechnik unterscheidet man verschiedene Ebenen zur Lösung von Problemen. Die untenstehende Grafik verdeutlicht die Begriff. Während man die Wiederverwendbarkeit von Software in die höchste Ebene, in die Ebene der zugrundeliegenden Prinzipien von Lösungsansätze einordnet, muss man die Modularisierung als eine Ausprägung des Prinzips Wiederverwendung in eine niedrigere Ebene einordnen. Modularisierung ist eine Technik zur Realisierung der Wiederverwendung.



Neben Namensräumen und Standards zur Modularisierung von Applikationen sind Entities ein weiteres Mittel zur Unterstützung der Wiederverwendung von Dokumentteilen. Der Grundgedanke von Entities ist der, Inhalt einmal zu deklarieren, um ihn dann beliebig oft im Dokument zu substituieren. Dabei unterscheidet man verschiedene Typen von Entities.

Entities können sowohl intern als auch extern auftreten. Sie können geparste und ungeparste Daten enthalten und für den Einsatz in DTD's wird sich des speziellen Typs des Parameterentities bedient. Entities wurden im Grundlagenabschnitt ausführlicher behandelt, deshalb sei für detailliertere Informationen an diese Stelle verwiesen.

Wie schon die Wiederverwendbarkeit, so sind auch Wartbarkeit, Flexibilität und Verständlichkeit von Software in den Bereich der Softwareprinzipien einzuordnen. Die Wartbarkeit und gleichzeitig auch die Verständlichkeit von Software wird maßgeblich durch

die Form der Kommentierung und der Dokumentierung, sowie durch den Grad der Modularisierung bestimmt. XML und seine Erweiterungen sind durch die Entwicklungsprozesse beim W3C sehr gut dokumentiert. Die Dokumentierung von Anwendungen, also auch von XML-Anwendungen im speziellen, ist ein separates Arbeitsfeld und würde den Rahmen an dieser Stelle sprengen. Als Hinweis seid gesagt, dass XML-Dokumente in Verbindung mit XSLT-Anwendung die Erzeugung von Dokumenten hilfreich unterstützen können.

Durch Schemasprachen entwickelte Standards repräsentieren auch eine Form der Kommentierung. Bei detaillierter Betrachtung ist an den Deklarationen abzulesen, welche Elemente und Attribute durch das Schema unterstützt werden. Wie in einem vorigen Abschnitt vorgestellt, sollte die Technik der Kommentierung aus dem Bereich der Softwareentwicklung übernommen werden. Zum Einsatz kommen strategische und taktische Kommentare. Die Anwendung wurde im Grundlagenabschnitt erklärt. Ein weiterer Vorteil der aus den grundlegenden Eigenschaften von XML entspringt ist, dass sich einer entsprechend kodierten Syntax sich XML-Dokumente selber kommentieren.

3.1.1 Portabilität von XML-Inhalt

Portabilität von Software zwischen Systemen ist ein wichtiger Punkt bei der Entwicklung und stellt immer wieder ein großes Problem dar. Software, die für Unix-Systeme entwickelt wurde, ist meist nur mit viel Aufwand auf Windows-Systeme zu übertragen. Außerdem zeigt das Beispiel von Java-Anwendungen, dass die verwendete Laufzeitumgebung die Funktionsfähigkeit negativ beeinflussen kann. In dieser Beziehung stellt sich XML als sehr fortschrittlich heraus, denn die Portabilität ist hier schon in der Grundidee der Entwicklung verankert. Insofern sind XML-Applikationen nahezu grenzenlos portabel, wenn man die entsprechende Kodierung der Dokumente im Auge behält und die Verwendung proprietärer Zeichensätze meidet. Auf der anderen Seite handelt es sich bei XML ja auch um ein Dokumentformat und weniger um eine Anwendung, die auf Systemaufrufe von Betriebsystemen aufbaut.

3.2 Effizienz und Flexibilität von XML-Inhalt

Mit der Effizienz und Flexibilität von Software verhält es sich anders. Wenn Software möglichst effizient arbeitet, dann liegt dem häufig eine Art der Optimierung zugrunde. Eine solche Optimierung baut wiederum oftmals auf der Struktur der Daten auf. Das wiederum impliziert, dass eine solche Software nicht sehr flexibel auf geänderte Bedingungen reagieren kann. Dieses Verhalten kann man auch auf XML-Anwendungen übertragen. Wenn man Anwendungen stark durchstrukturiert, sich also im Bereich von datenzentriertem XML bewegt, dann ist eine solche Anwendung von XML-Parsern bedeuteten effizienter auszuwerten, im Vergleich zu weniger strukturiertem, gemischtem Inhalt. Eine schwache Strukturierung ermöglicht wiederum ein flexibleres Darstellen von Informationen. Als Beispiel soll einmal die Information eines Datums dargestellt werden. Entscheidend ist der Abstraktionsgrad bzw. der Grad der Strukturierung der Information. Wir haben die Möglichkeit ein Datum als ein Element darzustellen, näm-

lich <Datum>21.02.2005</Datum> . An dieser Stelle ist natürlich auch eine Darstellung in der Form <Datum>21. Februar 05</Datum> möglich. Es wird deutlich, dass für eine solche Anwendung mit einem hohen Abstraktionsgrad Informationen wesentlich flexibler notiert werden können. Auf der anderen Seite obliegt es der Anwendung, die im Element enthaltenen Informationen auszuwerten. Entweder kennt die Anwendung die Darstellung oder es muss eine Analyse über mehrere Elemente durchgeführt werden, um die Informationen zu erhalten. Ein solches Vorgehen reduziert der Grad der Effizienz der Anwendung teilweise massiv.

Strukturiert man die Darstellung der Datumsinformation stärker, dass heisst senkt man das Abstraktionsniveau, wird die Information für Parser respektive Anwendungen effizienter auswertbar. Eine mögliche Darstellung könnte wie folgt aussehen, <Datum><Tag>21</Tag><Monat>Feb.</Monat><Jahr>2005</Jahr></Datum>. Ein anderer Faktor, der die Effizienz von Applikationen beeinflusst, ist die Größe der Dokumente. Die zwei unterschiedlichen Typen von Parsern, DOM- und SAX-Parser, haben unterschiedliche Stärken und Schwächen. SAX-Parser werden gewählt, wenn Dokumente groß und nur wenig Elemente von Interesse sind. DOM-Parser werden hauptsächlich dann verwendet, wenn Dokumente kleiner sind und in dem Dokument mit vielen Strukturen wechselnd und häufig gearbeitet wird.

Die zu modellierenden Eigenschaften aus dem Datenbankenbereich wurden in dem nachfolgenden Paper [ArL04] umfassend bearbeitet. Es wird eine Normalform für XML-Dokumente beschrieben, die unter anderem auf die Vermeidung von Redundanz innerhalb von Dokumenten ausgelegt ist.

3.3 Eine Normalform für XML-Dokumente

In dem Artikel [ArL04] von Arenas und Libkin wird eine Normalisierung von XML-Dokumenten beschrieben. Die Notwendigkeit und Motivation ist auch hier, redundante Informationen zu entdecken, da sie für Update-Anomalien anfällig sind. In dieser Arbeit werden redundante Informationen als Ursache von bestimmten funktionalen Abhängigkeiten unter Pfaden eines Dokumentenbaumes aufgefasst.

Ziel ist es, eine beliebige DTD in eine "gut"-designte DTD zu konvertieren. Dafür gilt es, Prinzipien für ein gutes Design zu identifizieren und einen Algorithmus zur Konstruktion zu entwerfen.

Zum Erreichen dieses Zieles, müssen erst einmal Grundlagen für wichtige verwendete Begriffe und Notationen geschaffen werden. Es gilt Fragen zu klären wie:

- was sind Redundanz und Update-Anomalien in XML
- wichtige Definitionen und Basiseigenschaften von funktionalen Abhängigkeiten
- eine Definition was "unerwünschte" funktionale Abhängigkeiten sind

und dergleichen mehr. Auf dieser Theorie aufbauend kann dann ein Algorithmus zur Konvertierung von DTD's beschrieben werden. Dieser Algorithmus ist so konstruiert,

dass er keine "schlechten " funktionalen Abhängigkeiten zulässt.

Redundanz
Redundanz entsteht durch funktionale Abhängigkeiten, kurz FD's. Die Autoren haben im wesentlichen zwei unterschiedliche FD's identifiziert, die Redundanz hervorrufen.

1. FD durch (*,*)Beziehungen. Der später hier vorgestellte Lösungsansatz behebt die Redundanz durch das Aufsplitten von Informationen.

2. Relative FD sind analog zu mehrwertigen Abhängigkeiten im Bereich Datenbanken. Die Lösung zur Vermeidung von Redundanz ist hier eine Restrukturierung des Dokuments.

In dem unten folgenden Beispiel sind die Informationen in Abbildung a) redundant bezüglich der ersten Form dargestellt. Es ist möglich, dass Studenten an mehreren Kursen teilnehmen können und umgekehrt nehmen an Kursen mehrere Studenten teil, somit werden die Namen der Studenten mehrfach gespeichert. Update-Anomalien treten zum Beispiel dann auf, wenn der Name eines Studenten geändert werden soll, er aber nicht konsistent in allen Kursen geändert wird, an denen er teilnimmt. Nach obiger Lösungsstrategie werden die Informationen ähnlich separiert, wie es auch dem Aufteilen in Tabellen in RDBS'en entspricht. In der Abbildung b) sieht man eine normalisierte DTD, in der die Namen von Studenten in einem Subelement von Kurs gespeichert werden.

```
<!DOCTYPE courses [
    <!ELEMENT courses (course*)>
    <!ELEMENT course (title, taken_by)>
        <!ATTLIST course
            cno CDATA #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT taken_by (student*)>
    <!ELEMENT student (name, grade)>
        <!ATTLIST student
            sno CDATA #REQUIRED>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT grade (#PCDATA)>
]>
```

a)redundante DTD

```
<!DOCTYPE courses [
    <!ELEMENT courses (course*, info*)>
    <!ELEMENT course (title, taken_by)>
        <!ATTLIST course
            cno CDATA #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT taken_by (student*)>
    <!ELEMENT student (grade)>
        <!ATTLIST student
            sno CDATA #REQUIRED>
    <!ELEMENT grade (#PCDATA)>
    <!ELEMENT info (number*, name)>
    <!ELEMENT number EMPTY>
        <!ATTLIST number
            sno CDATA #REQUIRED>
    <!ELEMENT name (#PCDATA)>
]>
```

b)normalisierte DTD

Um weiter fortfahren zu können, werden jetzt wichtige Grundlagen und Notationen eingeführt. Die Begriffe gelten für alle kommenden Definitionen

El	-	Eine Menge von Elementname
Att	-	Eine Menge von Attributnamen
Str	-	mögliche Werte für Attribute vom Type String
Vert	-	identifizieren Knoten

Attribute starten immer mit @.
S und $\perp$ sind reservierte Symbole, in keiner der Mengen enthalten.

Formale Definition einer DTD D=(E, A, P, R, r)

- E $\subseteq$ El
- A $\subseteq$ Att
- P ist eine Abb. von E auf eine Element-Typ-Definition.
  Geg. $\tau\epsilon$ E: P(τ) = S oder ein regulärer Ausdruck der Form $\alpha ::= \epsilon \mid \tau' \mid \alpha|\alpha \mid \alpha,\alpha \mid \alpha*$
  Abb. auf ϵ ist das leere Element, $\tau'\epsilon E$.

Das dritte Element des regulären Ausdrucks beschreibt die Alternative, vier und fünf beschreiben wie üblich die Sequenz und die Wiederholung von Elementen.

- R ist eine Abb. von E auf Att, wenn $@A\epsilon R(\tau)$ dann ist @A für τ definiert.
- r ist das Rootelement, $r \notin P(\tau), \forall\tau\epsilon E$.
- ϵ ist das leere Element
- S ist eine Deklaration als #PCDATA.

Pfad in D
ist eine Folge w = w_1,...,w_n wobei

- w_1 = r
- $w_i\epsilon P(w_{i-1})$, für i = 2,...,n-1
- $w_n\epsilon P(w_{n-1}$ oder w_n =@l für ein @l*epsilon*R($_{n-1}$)
- Länge von w ist gleich length(w) = n
- Das letzte Element last(w) = w_n

Ein Pfad(D) steht für einen beliebigen Pfad in D und EPfad(D) für einen Pfad, der mit einer Elementdeklaration endet.

Formale Definition eines XML Baumes T=(V, lab, ele, att, root)

- V $\subseteq$ Vert
- lab: V $\rightarrow$ El
- ele: V $\rightarrow$ Str $\cup$ V*
- att: ist eine partielle Funktion V $\times A \rightarrow$ Str
- root ϵ V ist die Wurzel von T

XML-Baum t ist konform zu D, $T \models D$ wenn

- lab ist eine Abb. von V auf E
- für jeden Knoten vϵV: wenn P(lab(v)) = S, dann ist ele(v) = [s], wobei s ϵ Str.
  ansonsten ele(v) = [v_1,...,v_n], wobei lab(v_1) ... lab(v_n)
  muss Element des regulären Ausdrucks def. durch
  P(lab(v)) sein.
- att: ist eine partielle Funktion von V $\times A \rightarrow$ Str, so dass für jedes vϵV und @lϵA att(v,@l) definiert ist, wenn @lϵR(lab(v)) ist.
- lab(root) = r

Man sagt, T ist kompatibel mit D, T $\diamond$ D, wenn gilt Pfad(T) $\subseteq$ Pfad(D)

In dem hier vorgestellten Ansatz werden XML-Bäume durch Mengen von Tupeln dargestellt. Da der Begriff der FD, wie er hier vorgestellt wird, eine Ordnung innerhalb eines Knotens außer acht lässt, muss zuvor der Begriff der Subsummierung unter Bäumen eingeführt werden.

Ein Baum T_1 wird durch einen Baum T_2 subsummiert, $T_1 \preceq T_2$, wenn

- $V_1 \subseteq V_2$
- $root_1 = root_2$
- lab_2 muss $\forall v \epsilon V_1$ lab_1 entsprechen
- ebenso muss att_2 für alle Knoten und Attribute att_1 entsprechen
- $\forall v \epsilon V_1$, ele_1(v) ist eine Unterliste einer Permutation von ele_2

Die Relation $\preceq$ erzeugt eine Äquivalenzrelation. Zwei Bäume T_1 und T_2 sind gleich, $T_1 \equiv T_2$, wenn $T_1 \preceq T_2$ und $T_2 \preceq T_1$. [T] ist die Äquivalenzklasse von T, man schreibt [T]$\models$D, wenn $T_1 \models$D für ein $T_1 \epsilon$[T]. Für $T_1 \equiv T_2$ gilt auch Pfad(T_1)=Pfad(T_2), daraus folgt $T_1 \diamond$D wenn $T_2 \diamond$D.

Im nächsten Abschnitt sollen die entscheidenden FD's erklärt werden, dafür werden jedoch noch ein paar kleinere, hier eher informal beschriebene Definitionen benötigt. Für detailliertere Informationen wird auf den Originalartikel verwiesen.

Da wir XML-Bäume durch Tupel beschreiben wollen, ist es notwendig, möglichst solche Tupel zu selektieren, die einen maximalen Informationsgehalt haben. Dies wird durch eine Ordnung auf Tupeln erreicht.

Hat man zwei Tupel t_1 und t_2, dann schreibt man $t_1 \sqsubseteq t_2$, wenn t_1.p definiert ist, so ist es auch t_2.p.
Außerdem gilt, $t_1 \neq \perp$ bedeutet t_1.p = t_2.p.

Die Notation t.p bedeutet, dass wir einen konkreten Tupel eines XML-Baumes haben und über diesen und dessen Pfade auf die Elemente zugreifen.

Informal ausgedrückt bedeutet $t_1 \sqsubseteq t_2$, dass t_2 vom Informationsgehalt her mächtiger ist als t_1. Die Definition verhält sich analog für Mengen von Tupeln über Bäume. Zwischen zwei Mengen X, Y von Tupeln besteht die Relation $X \sqsubseteq^b Y$, wenn $\forall t_1 \epsilon X : \exists t_2 \epsilon Y$,

so dass gilt $t_1 \sqsubseteq t_2$.

tree-tupels
sind Funktionen eines Pfades über eine DTD. Das heisst, ein Pfad bildet eine DTD auf Knoten, Strings und $\perp$ ab. Die Menge aller tree-tupels für eine DTD D wird mit T(D) bezeichnet.

$tree_D$
ist ein XML-Baum, der aus einem tree-tupel mit dazugehöriger DTD konstruiert wird.

$tupels_D(T)$
ist eine Menge von tree-tupels t für einen XML-Baum T, wobei für alle t gilt: $tree_D(t) \preceq T$ und diese Menge muss maximal bezüglich $\sqsubseteq$ sein.

$trees_D$
ist ein XML-Baum T für eine DTD D und eine Menge von tree-tupels $X \subseteq T(D)$ für den gilt: T subsummiert alle tree-tupel-Bäume, $tree_D(t) \preceq T$ und er ist außerdem minimal bezüglich $\preceq$.

Funktionale Abhängigkeiten, *engl. functional dependency, kurz FD's*
werden auf Basis von tree-tupels definiert. Eine FD ist für eine DTD D ein Ausdruck der Form

$$S_1 \rightarrow S_2$$

wobei S_1 und S_2 nichtleere Teilmengen von Pfad(D) sind. Das heisst, Pfade über D bestimmen andere Pfade von D funktional. Die Menge aller FD's wird mit FD(D) bezeichnet. Für eine Teilmenge S von Pfad(D), $S \subseteq$ Pfad(D) und tree-tupels t_1 und $t_2 \epsilon$ T(D) bedeutet

$t_1.S = t_2.S$, dass für alle $p \epsilon S$: $t_1.p = t_2.p$ ist

Wenn nun $S_1 \rightarrow S_2 \epsilon$ FD(D), und $S_1 \cup S_2 \subseteq$ Pfad(D) und T ist ein XML-Baum für den gilt $T \diamond D$, dann erfüllt T die FD: $S_1 \rightarrow S_2$, wenn für alle t_1, $t_2 \epsilon \, tupels_D(T)$ gilt:

$t_1.S_1 = t_2.S_2$ und
$t_1.S_1 \neq \perp \Rightarrow t_1.S_2 = t_2.S_2$

T ist konform zu Σ, $T \models \Sigma$ für $\Sigma \subseteq FD(D)$, wenn $T \models \varphi : \forall \varphi \epsilon \Sigma$.
Man schreibt $T \models (D, \Sigma)$, wenn $T \models \Sigma$ und $T \models D$. Für eine gegebene DTD D und $\Sigma \subseteq$ FD(D) und $\varphi \epsilon$ FD(D) sagt man, FD(D) impliziert φ, $FD(D) \vdash$, wenn für jeden Baum T mit $T \models D$ und $T \models \Sigma$ es der Fall ist, dass $T \models \varphi$. Die Menge aller durch (D,Σ) implizierten Abhängigkeiten wird als $(D, \Sigma)^+$ bezeichnet.

Aufbauend auf diese Theorie und Grundlagen kann nun die Definition einer Normalform für XML-Dokumente vorgenommen werden.

Für gegebene DTD D und $\Sigma \subseteq$FD(D), (D,Σ) ist in Normalform, XNF, wenn jede nichttriviale FD $\varphi \epsilon (D, \Sigma)^+$ der Form S $\rightarrow$ p.@l oder S $\rightarrow$ p.S, es der Fall ist das S $\rightarrow$ p in (D,Σ)$^+$ ist.

Nichttriviale Abhängigkeiten $\varphi \epsilon$FD sind der Form: (D,$\emptyset$) $\vdash \varphi$.

Auf den Algorithmus und weitere Ausführungen soll an dieser Stelle verzichtet werden, da das den Umpfang dieser Arbeit sprengen würde. Für weiterführende und detailliertere Informationen sei nochmals an den Originalartikel verwiesen.

Der wichtigste Beitrag dieses Ansatzes ist die intensive Betrachtung und Formalisierung von theoretischen Grundlagen innerhalb von XML-Bäumen. Besonders gründlich ist die Darstellung von XML-Bäumen, deren Instanz-Bäumen und den besonderen funktionalen Abhängigkeiten innerhalb der Bäume. Außerdem wurden mit dem Informations-Splitten und dem Restrukturieren von Strukturen zwei gut applizierbare Strategien für unterschiedliche Arten von Redundanzen in dem Algorithmus angewendet.

4 Verwendung von Pattern

Die Verwendung von Pattern führte in anderen Bereichen der Informatik, speziell im Bereich der objektorientierten Softwareentwicklung zur Verbesserung der Qualität von Software. Zum einen profitiert man von der Zeitersparnis bei der Erstellung von Lösungen und zum anderen sind die angewandten Pattern von hoher softwaretechnischer Güte.

Diese Vorteil möchte man auch bei der Erstellung von XML-Inhalten nutzen. Das Ziel des Designprozesses ist die adäquate Darstellung von Informationen. Wie schon in der Einleitung erwähnt, kann man den Prozess des Dokumentendesigns als ein Ausbalancieren von gegensätzlich aufeinander wirkenden Kräften beschreiben. Die wichtigsten Faktoren die bei der Erstellung von XML-Dokumenten zu beachten sind

- Größe der Dokumente
- Simplizität bei der Erstellung
- Simplizität bei der Verarbeitung
- Flexibilität der Dokumente
- Konsistenz des Inhalts
- Grad der Abstraktion

Die *Größe der Dokumente* ist wichtig für die Weiterverarbeitung, sowohl für Menschen als auch für Maschinen. Kleinere Dokumente sind für den Menschen besser handhabbar, die Übertragung von kleineren Dokumenten über Netze funktioniert i.A. auch besser als mit großen, aus diesen Gründen werden kleiner Dokumente bevorzugt.

Eine *einfache Erstellung* von Dokumenten ist für das Verständnis der Autoren wichtig und senkt die Fehlerquote beim Entwickeln der Designs.

Die *Simplizität der Verarbeitung* beeinflusst direkt die Komplexität der verarbeitenden Software.

Die *Flexibilität* ist ein Mass dafür, wie Dokumentinstanzen für einen Dokumenttyp variieren können. HTML als Dokumenttyp bietet eine breite Palette für HTML-nutzende Dokumente. Ein Abrechnungssystem hingegen arbeitet sehr restriktiv und sollte keine Abweichungen erlauben.

Konsistenz ist ein ganz wesentlicher Punkt, ohne Berücksichtigung der Konsistenz würden Informationen verfälscht und die Semantik von Dokumentinhalten verloren gehen.

Der *Abstraktionslevel* bestimmt die Granularität der Darstellung von Informationen und ist abhängig von den Anforderungen der Applikation.

Im Folgenden sollen Pattern vorgestellt werden, bei der Betrachtung ist es hilfreich, unterschiedliche Abstraktionslevel zu unterscheiden. Im ersten von zwei Abschnitten werden einige **Grundlegende Pattern** vorgestellt, denen dann im zweiten Abschnitt Pattern mit konkretem Bezug folgen.

4.1 dynamische Dokumente

Das erste grundlegende Pattern beschäftigt sich mit *Dynamischen Dokumenten*. Wenn Formate schnell entwickelt werden sollen oder wenn mehrere Entwickler, zu unterschiedlichen Zeitpunkten Beiträge leisten, ist die Verwendung eines festes Dokumentenformates nicht möglich. Die Lösung ist, dass wenn sich Datenstrukturen im Programm ändern, müssen sich auch die XML-Strukturen ändern. In aktuellen Sprachen, wie Java nutzt man die Fähigkeit zur Reflektion von Klassen und Metadaten aus, um die Accessor- und Mutator-Methoden zu identifizieren. Ein Beispiel soll das Vorgehen illustrieren.

```
public person {
   public String getName() { ... }
   public void setName(String name) { ... }
   public Address getAddress() { ... }
   public void setAddress(Address address) { ... }
}

public Address {
   public String getCity() { ... }
   public void setCity(String city) { ... }
   public String getState() { ... }
   public void setState(String state) { ... }
}
```

Die Entwicklungsumgebungen und Sprachen, in die XML bereits integriert ist, verfügen meist über mindestens eine Bibliothek, um Objekte in XML-Strukturen zu konvertieren. So wird ein Objektdesign um nur wenige Zeilen ergänzt, und man hat das XML-Dokumentendesign mit abgedeckt. Die obigen beiden Strukturen würden dann in etwa folgende Form konvertiert werden. Der dynamische Aspekt ist der, wenn sich Klassen oder spezieller Signaturen von Methoden ändern, dann ändert sich auch die Struktur und das markup der korrespondierenden XML-Dokumente.

```
<person>
   <name>Kyle Downey</name>
   <address>
      <city>Forest Hills</city>
      <state>Queens</state>
   </address>
</person>
```

4.2 Komposition und Wiederverwendung

Ein weiteres grundlegendes Pattern ist das der *Komposition und Wiederverwendung*. Es basiert darauf, dass, wo immer es möglich ist, man auf vorhandene Strukturen und Standards zurückgreifen soll, um daraus die entsprechenden Zieldokumente zu formulieren.

Eine typisches Beispiel sind Namensräume. Wie auch schon im Grundlagenabschnitt postuliert, sollten Elemente eher durch Namensräume unterstützt werden, anstelle eigene Elemente zu entwerfen. Dieses Pattern würde beinahe überall Anwendung finden, da die Standardisierung sich nahezu über alle Bereiche erstreckt, wo Bedarf besteht oder bald bestehen könnte. Es ist eine Frage des Informierens über vorhandene Standards, wenn man neue Applikationen entwickeln möchte.

Durch Namensräume ist es sehr leicht, komplette Elemente aus vorhandenen Spezifikationen in die eigenen zu importieren. Ein ausgezeichnetes Beispiel ist die Nutzung von RDF und DublinCore-Standards.

```
<?xml version=''1.0'' encoding=''UTF-8''standalone=''yes''>
   <rdf:RDF xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
   xmlns:dc=''http://www.purl.org/dc/elements/1.1''>
   <rdf:Description about=''urn:isbn:0596002920''>
   <dc:Title>XML in a Nutshell</dc:Title>
   <dc:Creator>W. Scott Means</dc:Creator>
   ...
   </rdf:Description>
```

Wie man an diesem Beispiel gut sieht, ist es leicht Elemente zu importieren. Ein wichtiger Punkt bei der Nutzung von Standards ist, dass man von den Errungenschaften und der Professionalität der Experten auf jedem einzelnen Gebiet profitieren kann. Standards bieten Fachwissen auf jedem Gebiet, welches ein einzelner Entwickler niemals erreichen kann.

4.3 Selbsterklärende Dokumente

Selbsterklärende Dokumente sind ein Pattern, das Elemente als Teil des Dokumentes beinhalten, die den eigentlichen Inhalt der Dokumente beschreiben. Dieses Pattern findet Anwendung, wenn die Informationen im Dokumente nicht ausreichend sind oder wenn komplexer XML-Inhalt für Menschen verständlich gemacht werden muss. Außerdem gilt auch hier wieder, die Semantik von Elementen in die Bezeichner zu integrieren. Im Allgemeinen möchte man Kommentare auch an bestimmte Elemente binden, deshalb wird folgend ein grundlegender Typ für Annotationen vorgeschlagen.

```
<complexType name=''documentableType''>
   <sequence>
      <element name=''annotation'' type=''string''/>
   </sequence>
</complexType>
```

Falls die Annotationen in Dokumenten zur Pflicht werden, dann sollte man in Betracht ziehen, die Dokumentierung als eigenen Teil zu entwickeln. Dafür sollten die durch XML-Toolkit zur Verfügung stehenden Werkzeuge eingesetzt werden.

4.4 Multipart-Dateien

Als letzter grundlegender Patterntyp soll der *Multipart-Dateien*-Typ erklärt werden. Multipart-Dateien sind ein Mechanismus, um einen Dokumenteninhalt in mehrere Dateien aufzuspalten. Dieses Pattern findet immer dann Anwendung, wenn Dokumente zu groß werden, wenn sich Bereiche von Dokumenten schneller ändern als andere oder wenn man Inhalt mit anderen Hauptdokumenten teilen möchte.

Realisiert wird dieses Pattern, indem man zum Dokumentenschema ein <import> oder <include> Element hinzufügt und dem Attribut `href` dieses Elementes eine gültige relative oder absolute URI übergibt.

Die grundlegenden Pattern stammen im Wesentlichen von [ARC05], dort können auch nähere Informationen und weiterführende Quellen nachgeschlagen werden.

Im diesem zweiten Abschnitt soll jetzt Bezug zu **konkreten Pattern** genommen werden.

4.5 Universelles Root-Pattern

Es existiert ein *Universelles Root Pattern*, welches für unterschiedliche Transaktionen (Transaktionen soll hier im weitesten Sinne verstanden werden) ein einzelnes Rootelement beschreibt, dieses Rootelement beinhaltet dann als Optionen mehrere Elemente. Ein solches Rootelement ist immer dann sinnvoll, wenn man mehrere unterschiedliche, aber dennoch semantisch verwandte Dokumenttypen verarbeiten muss. Ein Beispiel für ein solches Vorgehen könnte wie folgt aussehen.

```
<!ELEMENT Transaction (AddAddress | RemoveAddress | UpdateAddress)>
<!ELEMENT AddAddress (AddressBookEntry)>
<!ELEMENT RemoveAddress (AddressID)>
<!ELEMENT UpdateAddress (AddressID, AddressBookEntry)>

    <Transaction>
       <AddAddress>
          <AddressBookEntry>
          ...
          </AddressBookEntry>
       </AddAddress>
    </Transaction>
```

4.6 Envelope Pattern

Das *Envelope Pattern* bietet einen Dokumenttyp, der als Hülle für andere, zum Beispiel willkürliche XML-Daten dient. Denkbar wäre eine Trennung beim Transport von Inhalten, so dass die Transportdaten vom eigentlichen Inhalt getrennt werden. Das Vorgehen bei der Nutzung dieses Pattern ist, dass man die Zieldaten in die "Metadaten" einhüllt. Dabei kann man gleich sinnvoll mit Namensräumen arbeiten. Folgendes Beispiele veranschaulicht die letzten Ausführungen noch einmal.

```
<e:Envelope xmlns:e=''http://xmlpatterns.com/envelope.dtd''>
    <e:sender name=''bob''/>
    <e:receiver name=''http://xmlpatterns.com/message-receiver.cgi''/>
    <myStuff:message xmlns:myStuff=''http://xmlpatterns.com/my-stuff.dtd''>
       This is my message
    </myStuff:message>
</e:Envelope>
```

4.7 Domain-, Container-Pattern

Einfache, wieder recht grundlegende Pattern sind *kurze, verständlichen Namen* und das *Domain-Pattern*. Namen für Elemente und Attribute sollten zumindest nach zwei Regeln formuliert werden:

1. Namen sollten lang genug sein, um die Bedeutung im Dokument für das entsprechende Zielpublikum zu tragen

2. Namen sollten kurz genug sein, um leicht geschrieben werden zu können

Die Domain- bzw. Bereichs-Pattern setzen direkt auf Namen auf. Jedes Dokument gehört in einen bestimmten Arbeits- bzw. Lebensbereich und jeder dieser Bereiche hat seine eigenen, mehr oder weniger spezifischen Konzepte und Termini. Das Domain-Pattern beschreibt, dass Konzepte und Begriffe aus den entsprechenden Bereichen auch Elemente und Attribute in den korrespondierenden Dokumenten werden müssen.

Um mehr Struktur in ein Dokument durchzusetzen, kann man das *Container-Pattern* nutzen. Das Container-Pattern beschreibt ein Elternelement, das mehrere, verwandten Elemente, auf einer Strukturebene, zusammen gruppiert. Es ist ein sehr allgemeines Pattern und bildet häufig die Basis für andere, spezifischere Pattern. Aufgrund der Einfachheit wird auf ein Beispiel verzichtet.

Das *Collection-Element-Pattern* ist eng verwandt mit dem Container-Pattern. Es wird angewendet, falls Elemente desselben Typs wiederholt nacheinander, auf der gleichen Hierarchie auftreten. Solche Wiederholungen benötigen oft einen Kontext und Metadaten die mit der Gruppe verbunden werden können. Ein Beispiel verdeutlicht dieses und das vorhergehende Container-Pattern.

```
<KitchenInventory>
   <GlassList location=''cupboard3''>
      <Glass type=''shot'' number=''5''/>
      <Glass type=''highball'' number=''8''/>
   </GlassList>
   <MugList>
      ...
   </MugList>
</KitchenInventory>
```

4.8 Choice-Reducing Pattern

Um den Aufbau der Dokumente für Autoren zu erleichtern, kann man ein *Choice-Reducing-Pattern* einsetzen. Komplexe DTD's mit vielen Elementen und Attributen erschweren das Erstellen von Dokumenten für Autoren. Durch den Einsatz dieses Patterns kann der Lernaufwand für Autoren reduziert werden. Eine Voraussetzung ist, dass die DTD's in logische Einheiten separierbar sind und dass die Dokumente durch Menschen erstellt werden. Der Schlüssel ist, die Anzahl der Alternativen für die Autoren an jedem Punkt im Dokument zu reglementieren. Wenn man Elemente in verwandte Mengen gruppiert, kann man die Anzahl der Alternativen an jedem Punkt im Dokument einschränken. Der Autor hat dann die Auswahl in Gruppen höherer Ebenen zu treffen. Das folgende Beipiel verdeutlicht das Vorgehen.

```
<!ELEMENT Doc ( Para | OrderedList | UnorderedList | Figure | Artwork )+>

<!ELEMENT Doc (Para | List | Illustration )+)>
<!ELEMENT List (OrderedList | UnorderedList )>
<!ELEMENT Illustration (Figure | Artwork )>
```

4.9 Separate Metadata-Data Pattern

Wenn man solche Abstraktionen auf höheren Ebenen nutzt, kann es notwendig sein, dass Metadaten in das Dokument aufgenommen werden müssen. Mit dem *Separate-Metadaten-Data-Pattern* unterstützt man die notwendige Trennung von Metadaten und Daten. Metadaten beschreiben hier Daten über Daten. Ohne eine klare Trennung ist eine Differenzierung zwischen Typen von Daten nicht immer möglich.

```
<ArticleSummaries>
   <Author>Phred Smith</Author>
   <Name>Patterns of Stereo Design</Name>
   <Author>J.R. Dolby</Author>
   <Summary>
```

```
      Use of patterns to arrange stereo components.
   </Summary>
</ArticleSummary>
```

Hier ist eine Unterscheidung der beiden Autorenelemente hinsichtlich der Semantik im Dokument nicht machbar. Die Trennung von Metadaten und Daten beinflusst die Erstellung und Verarbeitung von XML-Inhalt wesentlich. Bei der Trennung von Daten sollten Metadaten vor den Daten die sie beschreiben erscheinen. Zum einen verdeutlicht das die Metadaten in ihrer Verwendung und zum anderen wird die Verarbeitungssoftware unterstützt, indem Informationen über die Daten geliefert werden noch bevor die eigentlichen Daten übernommen wurden. Da die Trennung von Metadaten wieder ein grundlegendes Pattern ist, wird an dieser Stelle auf ein Beispiel verzichtet und in den spezifischeren, auf dieses Pattern aufsetzenden Pattern nachgeliefert.

Direkt auf das Separierungspattern setzen die Pattern *Metadata in separate Dokument* und *Head-Body* auf. Metadaten in einem separaten Dokument zu speichern macht Sinn, wenn sehr viele Daten vorhanden sind und diese die Komplexität des Dokumentes erhöhen. Außerdem ist es möglich, dass Programme nicht auf Metadaten angewiesen sind, durch Aussparung würde die Verarbeitungszeit gesenkt werden. Für den Fall, dass verschiedene Dokumente die gleichen Metadaten verwenden, ist dieses Pattern wieder von enormen Vorteil. Ein Beispiel mit zwei Dateien untermauert dieses Pattern. Die erste Datei, author-info.xml, enthält Autoreninformationen.

```
<Author>
   <Name>Robert Smith</Name>
   <BirthDate>21-04-1960</BirthDate>
   <Address>123 Maple Street</Address>
   </Author>
```

In der zweiten Datei wird die erste eingebunden, die Informationen auf diese Weise verwendet.

```
<!DOCTYPE DOCUMENT[ <!ENTITY author-info SYSTEM ”author-info.xml”> ]>
<Document>
   <Head>
      &author-info;
   </Head>
   <Body>
      This is the document.
   </Body>
</Document>
```

An diesem Beispiel wird gleich ein weiteres, nämlich das *Head-Body-Pattern* deutlich. Bei großen Menge von Metadaten kann man eine Separierung auch durch die Verwendung von zwei Kindelementen erreichen. Im ersten Element, dem Head-Element werden

die Metadaten untergebracht, im zweiten Element, dem Body-Element folgen die eigentlichen Informationen. Diese eher spezifischen Pattern wurden [PAT] entnommen. Dort können auch noch weitere Pattern gefunden werden.

Als Hinweis sei gesagt, dass es viele Analogien zwischen dem Grundlagenabschnitt und dem über Pattern gibt, trotz der unabhängigen Behandlung der Bereiche. Das Erscheinung ist darauf zurückzuführen, dass Pattern auch die Manifestierung eines intuitiven Verständnis von Strukturierung als auch Ergebnisse empirischer Erfahrungen sind.

5 aktuelle Ansätze aus der Forschung

Die meisten Ansätze wie die von [DLL03] und [EmM] beinhalten ähnliche Ideen, wie zum Beispiel die Reglementierung der Anzahl von Hierarchieebenen in Dokumenten oder die Realisierung von Normalformen, zur Vermeidung von Redundanzen.

5.1 XML-Dokumente mit garantiert "guten" Eigenschaften

In dem Artikel von D.W.Embley und W.Y.Mok [EmM] wird die Entwicklung von XML-Dokumenten beschrieben, die garantiert gute Eigenschaften besitzen. Es wird eine formale Definition für den Vorschlag einer Normalform für XML-Dokumente vorgestellt. Diese Normalform mit dem Namen XNF garantiert im Wesentlichen zwei wichtige Eigenschaften:

- maximal kompakte Konnektivität
- Redundanzfreiheit

Maximal kompakte Konnektivität bedeutet dabei, dass Dokumente mit dieser Eigenschaft aus so wenig Hierarchieebenen wie möglich bestehen müssen und dass Datenwerte nicht ohne Informationsverlust aus dem Dokument entfernt werden können.

Ein zweiter wichtiger Beitrag der Arbeit ist, dass eine modellbasierte Methodik entwickelt wurde, die eine automatische Erzeugung einer, die Normalform erfüllende, DTD garantieren kann.

Intuitiv empfunden, sollte ein gutes Design für XML-Inhalt sich durch kompakte, so wenig Hierarchien wie möglich auszeichnen. Daten sollten nicht entfernbar sein, ohne einen Informationsverlust zu bedeuten. Im Artikel wurde eine Formalisierung dieser Ideen vorgenommen.

Eine Vorbemerkung ist, dass die vorgestellte Methodik nicht für alle Arten von XML-Inhalt anwendbar ist. Speziell dokumentenzentriertes XML ist als Anwendungsbereich ausgeklammert, da der Reihenfolge der Wörter große Wichtigkeit zukommt. Als Zielanwendung sind vor allem sogenannte "real-world-Anwendungen" geeignet, für die ein konzeptuelles Modellieren Sinn macht.

Die grobe Vorgehensweise beim Anwenden der vorgestellten Methodik ist

1. für eine konkrete Applikation ein konzeptuelles Modell erstellen
2. Transformation des erstellten Modelles in eine XNF erfüllende DTD

Die oben erwähnten qualitativen Eigenschaften der Normalform können garantiert werden, nicht aber die Eindeutigkeit der Normalform. Der Grund ist, dass im Allgemeinen mehrere "gute" DTD's für ein Modell existieren. Unter diesen DTD's zu wählen, ist abhängig von *Nutzungsanforderungen* und *Perspektiven* der Anwender.

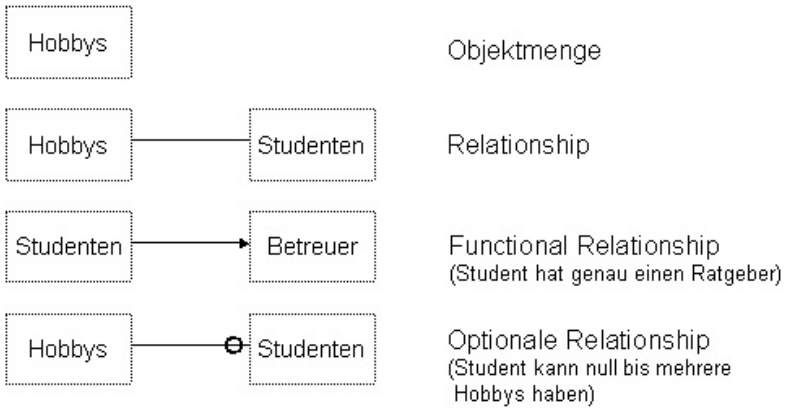
Ein möglicherweise resultierendes Problem ist, dass eben diese Nutzungsanforderungen die aufgestellten Prinzipien der XNF verletzen können. Um solche Probleme in den Griff zu bekommen, werden Heuristiken eingesetzt. Die so entstehenden Synergien bieten die

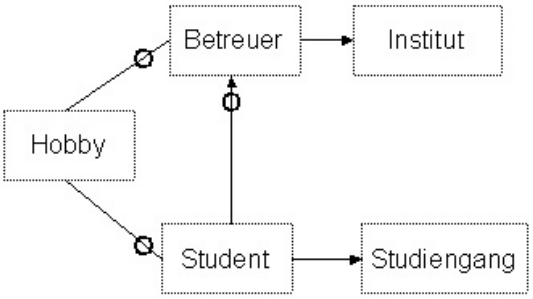
besten Ergebnisse. Die im Rahmen dieser Arbeit entwickelte Umgebung unterstützt eine Interaktion mit den generierten DTD's. Interaktion bedeutet dabei, dass die vom System vorgeschlagenen DTD's, basierend auf der Modellbeschreibung, durch Experten geändert, angepasst oder abgewiesen werden können. Änderungen werden vom System überwacht und die Auswirkungen dem Nutzer unmittelbar präsentiert.

5.1.1 Das verwendete Modell

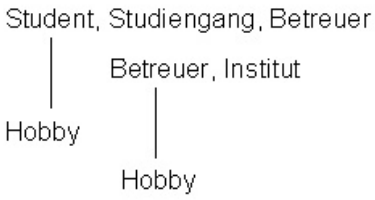
Das verwendete Modell ist im Ansatz sehr allgemein. Es handelt sich um eine Graphennotation, ähnlich der von UML. Weiter unter werden die wichtigsten Konzepte vorgestellt. An dieser Stelle werden Experten benötigt, die die entsprechende Anwendung durch einen um Zusicherungen erweiterten Hypergraphen modellieren. Es handelt sich genauer um CM-Hypergraphen, (conceptual-model).

Grundlegend hat ein XML-Dokument eine hierarchische Struktur mit genau einer Wurzel. Die Menge der Strukturen unmittelbar unterhalb dieser Wurzel bildet einen Wald hierarchischer Bäume. Genau dieser Wald soll vom konzeptuellen Modell entwickelt werden. Jeder Baum dieses Waldes wird durch einen Schema-Baum dargestellt. An dieser Stelle folgt eine Vorstellung der wichtigsten Konzepte anhand eines Beispieles.





a) Konzeptuell-Modell-Diagramm



b) Schemabäume

c) Text-Schemabaumnotation

(Student, Studiengang, Betreuer, (Hobby))**

(Betreuer, Institut, (Hobby))**

Um die Frage der Qualität dieser Bäume bezüglich eines Modelles zu klären, wurden von den Autoren hilfreiche Definitionen formuliert. Diese Definitionen beschäftigen sich mit der theoretischen Strukturen, die in jedem XML-Dokument vorhanden sind. Das ist auch der Grund, warum sämtliche Artikel, die Themen wie Normalform und Abhängigkeiten in XML-Dokumenten behandeln, gleiche oder geringfügig verschiedene Definitionen für Redundanz in Dokumenten, für funktionale und mehrwertige Abhängigkeiten entwickeln. Unterschiede existieren aufgrund der verwendeten Notationen, respektive der verwendeten Modelle, die meist auf eine unterschiedliche Allgemeinheit des Ansatzes zurückzuführen sind.

Die wichtigsten grundlegenden Bezeichnungen in [EmM] sind:

M	- ist ein CM-Hypergraph
T	- ist ein Schema-Baum für M
t	- ist ein Instanz-Baum über T
Pfad in T	- ist eine Folge von Knoten in T, vom Wurzelknoten bis zu einem Blattknoten
T ist zu M passend konstruiert	- wenn jeder Pfad von T eine Folge von Kanten aus M einbettet

Signifikante Eigenschaften von Bäumen sind :

Redundanz
Ein Datum, bezeichnet mit v, kommt in t redundant bezüglich einer konkreten Constraint vor, wenn v in t durch ein in t nicht vorkommendes Zeichen, hier mit $\perp$ bezeichnet, ersetzt werden kann, und durch das Constraint $\perp$ = v impliziert wird. Das heißt, Redundanz muss immer in Verbindung mit einer der unten folgenden Constraint gesehen werden.

Obwohl verschiedene Constraints denkbar sind, werden in diesem Artikel nur funktionale, mehrwertige, und Enthaltensein-Constraints betrachtet. Dennoch ist diese Menge für die meisten Anwendungen ausreichend.

Im direkten Anschluss folgen einige formale Definitonen, die für ein grundlegendes Verständnis für den weiteren Verlauf wichtig sind.

5.1.2 Redundanz durch Funktionale Abhängigkeit

T ist passend konstruiert für M. X $\rightarrow$ Y ist eine funktionale Kante in M, die auch in T enthalten ist. Bei dieser funktionalen Abhängigkeit, handelt es sich um eine injektive Abbildung. Weiter sei s ein Subtupel über XY in t. A ist ein Attribut in Y und a ist der Wert von A in s.

t ist genau dann redundant bezüglich der funktionalen Abhängigkeit X $\rightarrow$ Y, wenn der a-Wert in s mehr als einmal in t vorkommt. Das bedeutet im Grunde, dass eine Instanz der funktionellen Abhängigkeit zweimal oder häufiger im Dokument gespeichert wurde. Falls eine solche Instanz t existieren kann, sagt man, T ist potentiell redundant bezüglich der funktionalen Abhängigkeit X $\rightarrow$ Y.

5.1.3 Redundanz durch mehrwertige Abhängigkeiten

Voraussetzungen wie bei der funktionalen Abhängigkeit. Der Unterschied ist, dass keine funktionale Kante existiert. Dem "Argument" der Beziehung, hier X, sind mehrere Werte zugeordnet. Ansonsten sind wieder unser Schema-Baum T und das Subtupel s gegeben, sowie eine Kante X — Y. y ist das Y-Subtupel in s.

t ist redundant bezüglich der mehrwertigen Abhängigkeit X — Y, wenn y in s mehr als einmal in t vorkommt. Falls eine solche Instanz t existieren kann, sagt man, T ist potentiell redundant bezüglich der mehrwertigen Abhängigkeit X — Y.

Überdeckung von M
F ist ein Wald von Schema-Bäumen, der zu M korrespondiert, wenn jeder Baum von F passend konstruiert ist, und die Vereinigung aller Kanten aller Bäume die Kanten von M überdeckt.

Redundanz bezüglich Enthaltensein
F ist ein Schema-Baum-Wald korrespondierend zu M. T ist ein Schema-Baum aus F mit

1. einem Wurzelknoten
2. einer Objektmenge im Wurzelknoten

Gibt es Objektmengen S_1, ... , S_n in den Knoten von F, und die S_i's sind Spezialisierungen von G in M und G $=\cup_{i=1}^{n} S_i$ für alle T-Instanzen von F, dann sind die Werte von G redundant.

5.1.4 Normalform XNF

Aufbauend auf diesen Definitionen kann man die Normalform XNF definieren. Für die Normalform brauchen wir einen CM-Hypergraphen M. F ist ein Schema-Baum-Wald der zu M korrespondiert. Der Wald F ist in XNF-Normalform, wenn

- kein Baum aus F potentiell redundant bezüglich einer funktionalen, mehrwertigen oder Enthaltensein-Constraint ist.
- F so wenig oder weniger Schema-Bäume hat, als jeder andere zu M korrespondierende Wald F'.

Der im Artikel vorgestellte *Algorithmus* soll hier nicht vollständig präsentiert werden, anstelle wird auf den Originalartikel verwiesen [Quelle]. Die Eingabedaten des Algorithmus sind ein binäre CM-Hypergraph H. Als Ausgabe wird ein Schema-Baum-Wald erzeugt, für den eine potentielle Redundanz bezüglich funktionaler und mehrwertiger Abhängigkeiten nicht möglich ist.

Die Arbeitsweise des Algorithmus kann grob wie folgt charakterisiert werden.

- Der Algorithmus arbeitet mit dem Markieren von Knoten und Kanten, dies solange bis alle Knoten und Kanten des Modells markiert sind.
- Zu Beginn wird überprüft, ob unmarkierte Knoten in H existieren. Ist das der Fall wird ein Knoten gewählt und anschliessend markiert.
- Dieser Knoten wird zum Wurzelknoten eines neuen Baumes T. Anschliessend wird der Knoten noch als continuation-Attribute des Baumes T bezeichnet.
- Im Folgenden wird der Baum weiter konstruiert über ein Markieren von noch nicht markierten Kanten aus H, für die ein inzidenter Knoten bereits continuation-Attribute ist.
- Der Algorithmus arbeitet dann weiter und nutzt die unterschiedlichen Arten von Beziehungen zwischen Knoten und Kanten. Das Verhalten ist abhängig von obligatorischen oder optionalen Kanten, und ob Knoten als continuation-Attribute bezeichnet sind

Zu bewältigende Probleme bei der Entwicklung des Algorithmus waren, dass der Algorithmus nicht für alle Hypergraphen geeignet war. So wurden zunächst zwei Bedingungen entwickelt, von denen eine später wieder aufgehoben werden konnte. Die Bedingungen sind, dass Hypergraphen

1. kanonisch
2. binär

seien müssen. Ein Hypergraph ist dann kanonisch, wenn

1. es in H keine redundante Kante gibt
2. es in H keinen redundanten Knoten gibt
3. bidirektionale Kanten auch Bijektionen repräsentieren

Um einen Hypergraphen H in einem kanonischen Hypergraphen H' umzuwandeln bedarf es Experten oder einer Werkzeuguntersützung. Erweiterte Versionen des Algorithmus unterstützen nun auch n-äre Relationen.

Der Gewinn dieser Arbeit ist die Formulierung einer Normalform, die garantierte Eigenschaften aufweist. Als weiterer wichtiger Beitrag, ist die Methodik der automatische Umwandlung von Modellen zu nennen. Außerdem wurden zwei Tools entwickelt, die zum einen das Entwickeln der Modelle in der entsprechenden Notation unterstützen, inklusive der Konvertierung. Das andere Tool konvertiert einen CM-Hypergraphen in einen kanonischen Hypergraphen. Außerdem entwickeln die Tools mit erfahrenen Nutzern synergetische Effekte, die das Design zusätzlich unterstützen.

5.2 Ein semantisch "reiches" Datenmodell

In einem anderem Artikel [DLL03], wird ein semantisch reiches Datenmodell für semistrukturierte Daten vorgestellt. Das präsentierte Modell hat den Namen ORA-SS-Modell, was für Object-Relationship-Attribute model for semistructured data steht.

Der Ausgangspunkt für die Entwicklung war, dass aktuelle Datenmodelle für semistrukturierte Daten nicht in der Lage sind, die für ein effizientes Management der Daten wichtigen semantischen Informationen zu beschreiben.

5.2.1 Das ORA-SS-Modell

- kann die Schachtelungsstruktur der Daten darstellen
- unterscheidet zwischen
  - Objektklassen
  - Beziehungstypen

– Attributen

Der Hauptbeitrag des Modells ist jedoch

- die Möglichkeit zur Spezifizierung der Teilnahme von Eltern- und Kindobjekten an Beziehungstypen
- die Unterscheidung zwischen Attributen von Objekt- und Beziehungstypen
- die explizite Notation des Grades von Beziehungen

Gerade diese Informationen sind wesentlich zur Kontrolle von Redundanz und Konsistenz in XML-Inhalten.

Ein Teil der Motivation der Arbeit war, Redundanzen in Daten zu entdecken. Die Verfahrensweise von redundanten Daten ist nicht immer gleich. Es ist auch möglich, Daten aus Performancegründen in Dokumenten redundant zu halten. Dies birgt jedoch die Gefahr von Update-Anomalien. Ein Hinweis zur Behandlung von Redundanzen ist:

1. Anzahl der dublizierten Daten ist fest → nicht eliminieren
2. Daten werden niemals aktualisiert → nicht eliminieren
3. Anzahl redundanter Daten ist nicht fest → eliminieren

Für 1. kann ein solches Vorgehen empfohlen werden, da die Anzahl der redundanten Informationen fest im Dokument verankert ist. Der Anwender bzw. die Anwendung ist sich bei Änderungen im Dokument bewusst, wo überall Änderungen zu vollziehen sind. Es gibt niemals einen Kontrollverlust, somit ist eine unbedingte Eliminierung der Redundanz nicht notwendig und bringt keinerlei Vorteile.

Im 2. Fall ist eine Elimination auch nicht sinnvoll, da das Dokument per Voraussetzung niemals geändert werden soll. Es besteht keine Gefahr des Auftretens von Update-Anomalien.

Im 3. Fall ist ein Eliminieren notwendig, da die redundanten Daten nicht wie im ersten Fall kontrolliert werden. Vor Update-Operationen müsste jedesmal überprüft werden welche Daten wo redundant auftreten. Dies ist zwar möglich, ist aber keinesfalls effizient und sollte deshalb vermieden werden.

Redundanz entsteht im Allgemeinen aus (*,*)Beziehungen, die hierarchisch, dass heisst durch Schachtelung repräsentiert werden. In allen semistrukturierten Schema-Definitions-Sprachen und -Modellen werden (*,*)Beziehungen durch (1,1)- und (1,*)Beziehungen modelliert oder aber bei der Übersetzung des Modells entsprechend behandelt.

Ein anderer wichtiger Aspekt ist der Konsistenzerhalt bei Sichten und bei Transformationen von XML-Inhalt. Sichten auf Daten sind ein üblicher Schutzmechanismus für die Basisdaten, wie aus dem Bereich der Datenbanken bekannt ist. Die Idee ist, nur die für den Nutzer relevanten Daten zugänglich zu machen. Andere Daten, die für die Arbeit nicht von Interesse sein sollten werden ausgeblendet. Außerdem erlauben Sichten eine komfortable Beschränkung von Inhalt. Transformation von Inhalt ist eine typische Anwendung im XML-Bereich. Bei diesen beiden Anwendungen ist darauf zu achten, dass die Semantik der Ursprungsdaten nicht verletzt wird. Verletzungen der Konsistenz

treten insbesondere dann auf, wenn Subelemente und deren Attribute unbeobachtet in der Hierarchie der Struktur in Richtung Wurzel gehoben werden. Attribute, die syntaktisch in einem Subelement gespeichert sind, eigentlich aber zu dem Subelement und dem Beziehungstyp gehören, verursachen eine solche Verletzung der Konsistenz. Doch dazu später bei der Thematik der Attribute mehr.

5.2.2 Das Datenmodell

Das Daten-Modell beinhaltet drei wesentliche Konzepte:

1. Objektklassen : beschreiben real-world-Entities
2. Beziehungstypen : verbinden mehrere Objektklassen miteinander
3. Attribute : beschreiben Eigenschaften von Objekten oder Beziehungstypen

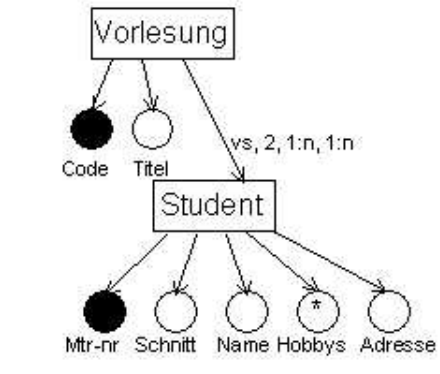
Das Daten-Modell umfasst weiter vier unterschiedliche Diagramm-Typen.

1. Instanz-Diagramm
2. Schema-Diagramm
3. Funktionale-Abhängigkeiten-Diagramm
4. Vererbungs-Diagramm

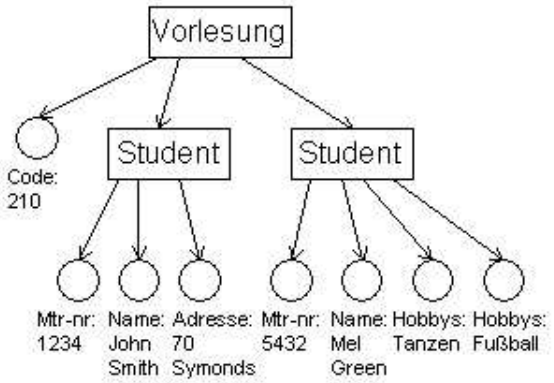
In dem Kontext sind die Namen alle selbsterklärend. Die Diagramme aus dem ersten und zweiten Punkt kommt hier die größte Wichtigkeit zu. Die anderen Diagramme unterstützen zusätzliche semantische Informationen und verhindern somit ein Überladen einzelner Diagramme mit zu vielen Informationen.

Das Schema-Diagramm ist ein gerichteter Graph, in dem innere Knoten Objektklassen modellieren und die Blätter Attribute von Objekten oder Beziehungen darstellen.

Das Instanz-Diagramm wird benötigt, wenn ein Dokument-Schema über bereits existierende Instanzdaten aufgebaut werden soll. Das Instanz-Diagramm ist dem Schema-Diagramm sehr ähnlich. Auch hier handelt es sich um einen gerichteten Graphen, in dem interne Knoten Objekte markieren und die Blätter die Instanz-Daten der Attribute darstellen. Ein Beispiel unterlegt die Erklärungen.



a) ORA-SS Schema-Diagramm

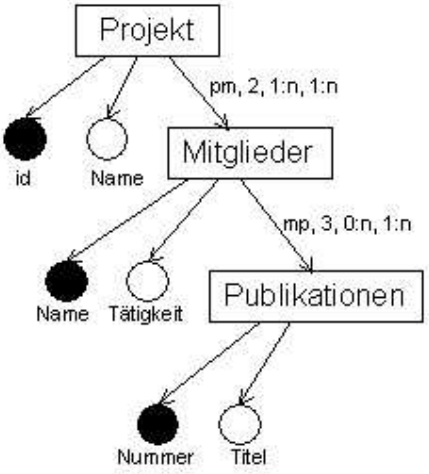


b) ORA-SS Instanz-Diagramm

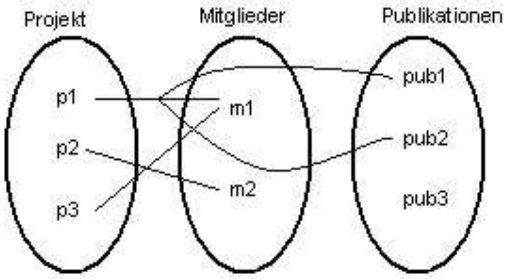
Im Folgenden sollen die wichtigsten Konzepte des Modells erklärt werden und wie mit ihnen die besonderen semantischen Informationen dargestellt werden können.

Die Objektklasse entspricht einer Entität, gleich der im ER-Diagramm oder respektive dem Typ eines Elementes im semistrukturierten Datenmodell. Da semistrukturierte Daten weniger regulär sind als strukturierte Daten, werden sie auch anders als strukturierte Daten nicht durch ihre Attribute charakterisiert, sondern vielmehr durch ihren Namen.

Beziehungstypen repräsentieren geschachtelte Beziehungen. Sie haben einen Grad, der binäre, ternär oder höher sein kann und es sind explizite numerische Teilnahmezusicherungen der beteiligten Objektklassen möglich. Die Beschriftung von Beziehungen wird in folgender Form vorgenommen: *name, n, a:b, b:c.* Name bezeichnet den Namen der Beziehung, n gibt den Grad der Beziehung an, x:y ist die übliche Notation für Kardinalitäten von Entitäten, die an Beziehungen teilnehmen. a:b gilt für das Elternobjekt und c:d für das Kindobjekt. Ein Beispiel unterstreicht die Ausführungen.



a) ORA-SS Schema-Diagramm



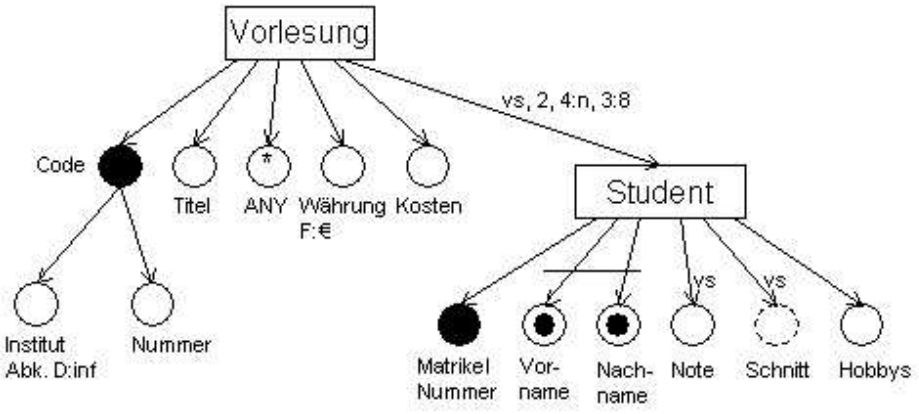
b) Instanz-Schema-Diagramm

In dem Beispiel wird die Notation für Beziehungen anhand einer binären und einer ternären Beziehung deutlich. Die binäre Beziehung besteht zwischen einem Projekt und den Mitgliedern. Die Beziehung trägt den Namen 'pm', und es wird ausgesagt, dass an einem Projekt ein bis n-viele Mitglieder beteiligt sind. Umgekehrt kann ein Mitarbeiter an mindestens einem und beliebig vielen Projekten arbeiten.

Die zweite in dem Schema vorkommende Beziehung ist ternärer Art. Das heißt, sie besteht zwischen Projekten, Mitgliedern und Publikationen. Durch die notierten Kardinalitäten wird ausgesagt, dass ein Mitglied keine bis n-viele Publikationen haben kann. Eine Publikation muss von mind. einem, bis beliebig vielen Mitgliedern verfasst werden. Der Grad der Beziehung impliziert aber weiter, dass die Beziehung mit Mitgliedern und Projekten verbunden ist. Genauer gesagt, es besteht eine Beziehung zwischen einer binären Beziehung und einer Objektklasse. Das wird durch die Abbildung b) noch

einmal verdeutlicht. Der Grad einer Beziehung ist wichtig und notwendig zur Modellierung unterschiedlicher semantischer Inhalte und für die Speicherung der Daten in einer Struktur.

Attribute repräsentieren Eigenschaften, entweder von Objekten oder Beziehungstypen. Wie oben schon gesehen, ist ein Attribut ein Kreis verbunden durch einen Pfeil mit einer Objektklasse.



a) ORA-SS Schema-Diagramm inklusive ‚reicher' Attributierung

Wie im Beispiel zu sehen, kann eine Bezeichnung außerhalb des Attributkreise maximal folgende Form haben: *name, F:v1, D:v2*. F und D sind optionale Konzepte, sie unterstützen die Möglichkeiten der DTD von fixierten- und Standard-Attributwerten. v1 bezeichnet dabei einen festen Attributwert, v2 den default-mäßigen Attributwert, hier im Beispiel 'inf'. Der spezielle ANY-Name kennzeichnet wie auch in DTD's eine unbekannte Struktur. Im Inneren eines Attributkreises kann man die Kardinalität eines Attributes notieren. Dafür werden die üblichen Bezeichnungen für Kardinalitäten verwendet, wie ?, +, *. Ein ausgefüllter schwarzer Kreis markiert ein Schlüsselattribut, ein solcher Kreis mit einem weiterem Kreis umgeben bezeichnet einen Schlüsselkandidaten. Ein zusammengesetzter Schlüssel wird durch eine Linie über die Verbindungspfeile von Attributen für ein Objekt markiert. Ein abgeleitetes Attribut wird durch einen gestrichelten Kreis dargestellt. In dem Beispiel kann der Schnitt durch die Noten in den entsprechenden Kursen abgeleitet werden. Beim Attribut Code der Objektklasse Vorlesung, handelt es sich um ein komplexes Attribut, das aus Attributen zusammengesetzt wurde.

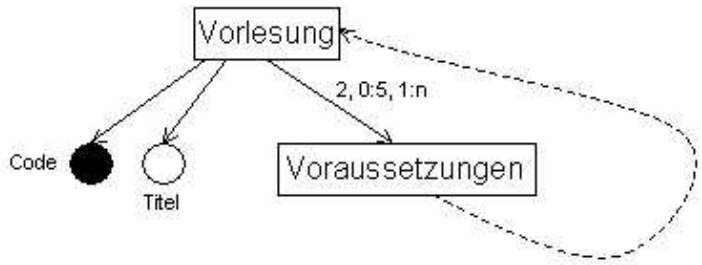
Ein wichtiges und hilfreiches Mittel zur Modellierung der Semantik ist, dass man einer Kante zwischen einem Attribut und der zugeordneten Objektklasse einen Namen geben kann. Dieser Name muss der Name einer Beziehung des Objektes sein. Auf diese Weise kann man Attribute von Objektklassen und Beziehungstypen trennen.

Zwischen Attributen von Objekten und Beziehungen zu unterscheiden ist wichtig für die Einhaltung der Konsistenz im Schema. Attribute von Objekten können beliebig mit

Objekten verschoben werden, Attribute von Beziehungen jedoch können nur mit ihren Beziehungen verschoben werden.

Referenzen dienen dem Referenzieren von Objektklassen. Sie werden durch eine gestrichelte Linie dargestellt. Es ist auf die Unterscheidung zwischen referenzierenden und referenziertem Objekt zu achten, dieses wird durch die Pfeilrichtung dargestellt.

Falls das referenzierte Objekt über keinerlei Extraattribute oder Subelemente verfügt, ist eine Darstellung unnötig und das Elternelement des referenzierenden Elementes sollte das referenzierte Objekt gleich selbst referenzieren.



a) ORA-SS Schema-Diagramm Selbstreferenzierung

Referenzen sind unerlässlich für eine gute Modellierung, trotzdem sollten sie sparsam eingesetzt werden, denn sonst degeneriert das Schema zu einer flachen Struktur, wo alle Klassen direkt unter der Wurzel angeordnet sind. (*,*)Beziehungen zwischen Eltern- und Kindelementen sollten zur Vermeidung von Redundanzen durch Referenzen auf der gleichen Ebene modelliert werden.

5.2.3 ORA-SS nutzen

Das Vorgehen bei der Nutzung des ORA-SS-Modells ist wie folgt:

1. vorhandenes XML-Dokument auf ein Instanz-Diagramm abbilden
2. Schema-Diagramm aus Instanz-Diagramm extrahieren
3. Schema-Diagramm normalisieren (durch Experten)
4. Originales Dokument auf restrukturiertes Dokument abbilden

Der erste Schritt ist einfach und kann direkt, durch eine einfache Abbildung erfolgen. Der zweite Schritt ist dann schwierig. Um das Schema-Diagramm aus dem Instanz-Diagramm zu extrahieren, müssen die Schachtelungsstrukturen erkannt und die semantischen Informationen identifiziert werden. Die semantischen Informationen betreffen dabei im Wesentlichen den Grad von Beziehungen, die Teilnahmezusicherungen von Objekten

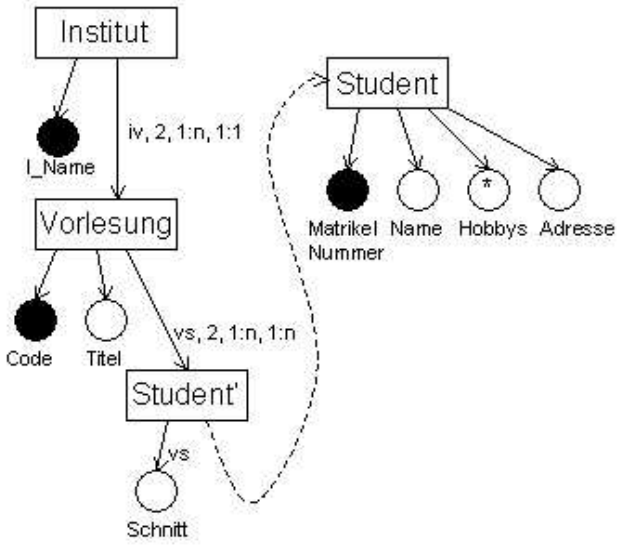
an Beziehungen, die Identifikation von Schlüsseln und die Attributzugehörigkeit. Dieser wichtige Prozess muss von Experten aus den entsprechenden Anwendungsbereichen der Applikation begleitet oder selbst durchgeführt werden.

```
<Institut name='cs'>
<Vorlesung Code='230'>
        <Titel>Software Entwicklung</Titel>
        <Student Mtr-nr='1234', Schnitt='2,3'>
                <Name>John Smith</Name>
                <Adresse>70 Symonds<Adresse>
        </Student>
        <Student Mtr-nr='5123', Schnitt='1,7'>
                <Name>Mel Green</Name>
        </Student>
</Vorlesung>
<Vorlesung Code='210'>
        <Student Mtr-nr='1234'>
                <Name>John Smith</Name>
                <Adresse>70 Symonds<Adresse>
        </Student>
        <Student Mtr-nr='5123'>
                <Name>Mel Green</Name>
                <Hobby>Fußball</Hobby>
                <Hobby>Tanzen</Hobby>
        </Student>
        </Vorlesung>
</Institut>
```

a) Original-Schema

```
<Institut name='cs'>
<Vorlesung Code='230'>
        <Titel>Software Entwicklung</Titel>
        <Student' Mtr-nr='1234', Schnitt='2,3'>
        <Student' Mtr-nr='5123', Schnitt='1,7'>
</Vorlesung>
<Vorlesung Code='210'>
        <Student' Mtr-nr='1234'>
        <Student' Mtr-nr='5123'>
        </Vorlesung>
        <Student Mtr-nr='1234'>
        <Name>John Smith</Name>
        <Adresse>70 Symonds<Adresse>
</Student>
<Student Mtr-nr='5123'>
        <Name>Mel Green</Name>
        <Hobby>Fußball</Hobby>
        <Hobby>Tanzen</Hobby>
</Student>
</Institut>
```

b) normalisiertes Schema



Normalisiertes ORA-SS Schema-Diagramm

Die Normalisierung des Schemas muss auch von Experten durchgeführt werden. Man versteht dabei unter anderem die Ersetzung von (*,*)Beziehungen durch Referenzen. Das in Abbildung a) dargestellte originale XML-Dokument wurde über ein normalisiertes Schema in ein XML-Dokument ohne Redundanz überführt.

Die wertvollsten Beiträge des Artikels sind die Modellierung von semistrukturierten Daten durch semantisch vielfältige Modellierungsmöglichkeiten. Außerdem wurden auch Konsistenzeigenschaften von Dokumenten betrachtet. Ein weiterer Vorteil ist die Unterstützung verschiedener Diagrammarten, die zum einen unterschiedliche Perspektiven auf die Anwendung erlauben und zum anderen ein Überfluten des Diagramms verhindern.

Nachteile sind, dass es zum einen keinen Algorithmus gibt, der beispielsweise eine Normalisierung des Schema-Diagramms selbstständig oder unterstützend vornimmt. Außerdem gibt es kein Werkzeug zur Unterstützung der Modellierung. Alles in allem geschieht hier ein Design der XML-Anwendung wieder per Hand, nur das durch die vielfältigen Möglichkeiten auch ein qualifiziertes Design forciert werden kann.

Beiden Artikel gemein ist, dass die Herangehensweise für ein qualitativ gutes Design von XML-Inhalt durch eine Annährung über graphische Modellierungsnotationen geschieht. Graphische Notationen gelten im Allgemeinen als bevorzugte Darstellungsvariante, da sie Menschen in ihrer Auffassungsgabe hilfreich unterstützen.

Wenn automatische Transformationen unterstützt werden, dann nur auf DTD-Strukturen. Das liegt im Wesentlichen an der Einfachheit der Konzepte der DocumentType-Definition. Höhere Schema-Sprachen sind auch komplexer beim Entwurf von Schemata für Dokument anzuwenden.

Für ein gutes Design ist selbst bei automatischer Umwandlung die Anwendung von Expertenwissen notwendig und hilfreich. Ein autonomer Prozess der Bewertung von XML-Inhalten ist somit ausgeschlossen.

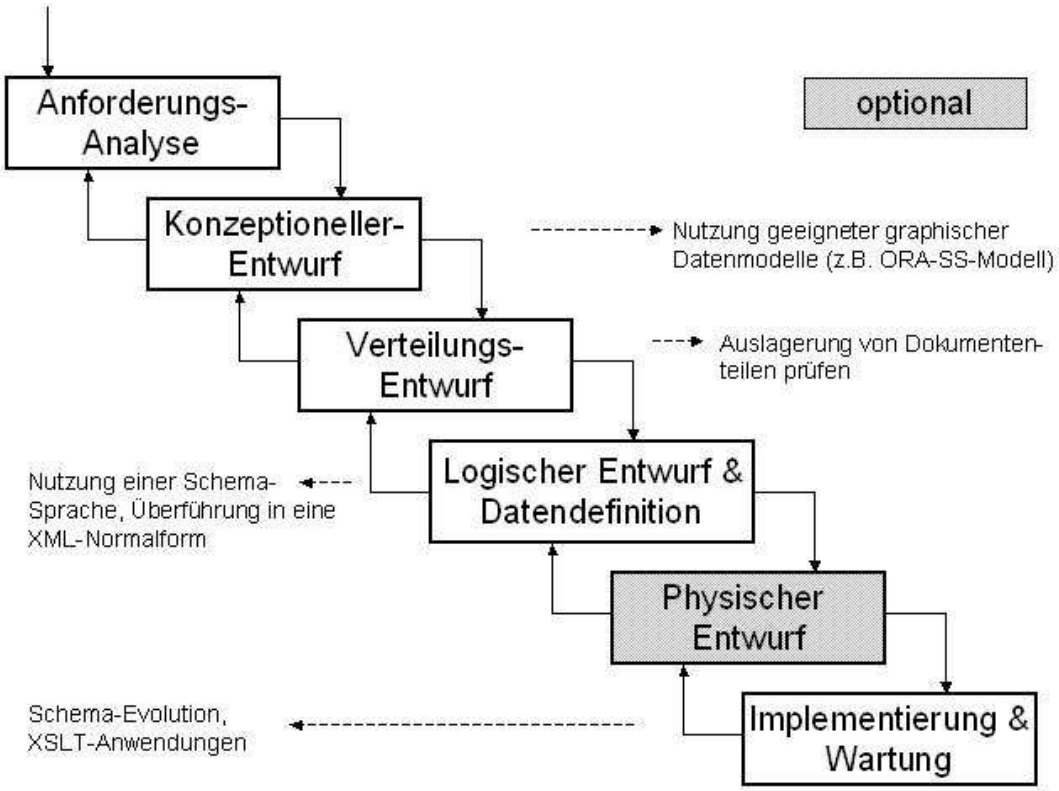
Eine weitere Einschränkung ist, dass sich die Artikel nur auf klar strukturiertes, datenzentriertes XML konzentrieren. Eine Bewertung der Struktur von dokumentzentriertem XML ist aufgrund der recht schwachen Strukturierung und der fixierten Reihenfolge der Wörter unmöglich.

6 eigene Ansätze

In diesem Abschnitt werden Empfehlungen und Techniken aus den anderen Abschnitten zu sinnvollen, sich ergänzenden Techniken zusammengefügt. Die gegebenen Richtlinien werden sich auf datenzentriertes XML beschränken, da für dokumentenzentriertes XML aufgrund der Charakteristik Strukturempfehlungen eigentlich nicht durchgesetzt werden können. Eine Ausnahme sind Metadaten über Dokumente und die semantische Kodierung von Tag-Bezeichnern. Diese beiden Punkte sind allgemein, auf alle XML-Dokumente anwendbar und sollten bei jeder Modellierung berücksichtigt werden. Die Charakteristik von dokumentenzentriertem XML ist hier gleichzusetzen mit der Struktur solcher Anwendungen, in der die Reihenfolge der Elemente nicht vertauscht werden kann, so wie in einem Text die Reihenfolge der Wörter nicht vertauscht werden kann.

6.1 Nutzung eines modifizierten Phasenmodells

Für den Entwurf von datenzentriertem XML wird ein Vorgehen ähnlich dem des Phasenmodells beim Datenbankentwurf vorgeschlagen. Genauer wird ein modifiziertes Phasenmodell vorgeschlagen, bei dem die Phasen des logischen Entwurfes und der Datendefinition zusammenfallen werden.



In der Anforderungsanalyse wird, wie auch beim Datenbankentwurf, der abzubildende Informationsgehalt in den entsprechenden Abteilungen und Bereichen des zu modellierenden Weltausschnitts zusammengetragen.

Das Ergebnis können dann informale Beschreibungen des Problems in Form von Texten, Tabellen und Formblättern sein. Die Analysephase ist auch gleichzeitig die Grundlage für die in der Struktur zu kodierende Semantik, welche ein elementarer Bestandteil der Struktur sein sollte.

Beim konzeptionellen Entwurf überführt man die informalen Beschreibungen aus der Anforderungsanalyse in ein erstes formales Modell. Ähnlich wie beim Datenbankentwurf, sollten auch hier graphisch notierte Datenmodelle zum Einsatz kommen. Im Abschnitt über aktuelle Forschungsschwerpunkte werden zwei unterschiedliche, für XML geeignete Datenmodelle und ihre Anwendungsmöglichkeiten vorgestellt.

Eine notwendige Voraussetzung für ein geeignetes Modell ist die Fähigkeit, semantische Informationen für XML-Inhalt möglichst gut darstellen zu können. Diese Anforderung schränkt die Anzahl der Kandidatenmodelle ein. Ein besonders gutes Beispiel ist das später vorgestellte ORA-SS-Modell [DLL03] von G. Dobbie und weiteren.

Als abschliessender Teil des konzeptionellen Entwurfs folgt eine Konfliktbetrachtung. Wie auch beim DB-Entwurf sollen Namenskonflikte, Typ- und Strukturkonflikte frühestmöglich identifiziert und eliminiert werden.

Dem Verteilungsentwurf sollte bei XML-Anwendung besondere Aufmerksamkeit geschenkt werden. Es gilt jene Dokumentteile zu identifizieren, für die eine Auslagerung aus unterschiedlichen Gründen Sinn macht. Wie schon im vorangegangenen Abschnitt argumentiert, ist es möglich, dass sich einzelne Dokumententeile schneller ändern als andere. Auch ist es denkbar, dass Teile von Dokumenten für eine gemeinsame Nutzung von mehreren Anwendungen ausgelagert werden können.

Aus diesen und anderen Gründen ist eine Aufteilung in einzelne Dateien sinnvoll. Durch die Aufspaltung in mehrere Dokumententeile werden zusätzliche Modellierungsmöglichkeiten offenbart. Hinweise darauf bietet auch der Abschnitt über Pattern. Modelliert man ein XML-Dokument mit einer zugehörigen DTD als ein Dokument mit interner und externer DTD, so kann durch die Nutzung von Parameterentities im Instanzdokument ein Schaltermechanismus für unterschiedliche Dokumentenmodule realisiert werden. Detailliertere Hinweise finden sich im Grundlagenabschnitt.

In der Phase des logischen Entwurfs und der Datendefinition erfolgt eine Umsetzung der formalen Beschreibung in ein resultierendes Schema. Für eine detaillierte Beschreibung der Umsetzung wird auf die vorangegangenen Abschnitte verwiesen. Dort wird unter anderem auf eine Art der Normalisierung von XML-Inhalt und eine automatische Transformation vorgestellt. Ein wichtiger Punkt, ist dass die im Grundlagenteil dieser Arbeit gegebenen Empfehlungen im resultierenden Schema auch umgesetzt werden.

Eine Phase des physischen Entwurfs ist für XML-Dokumente die als separate Struktur verwendet werden nicht möglich. Es verhält sich anders, wenn die XML-Daten in

einer Datenbank genutzt werden. Dann ist eine Indizierung von XML-Inhalt wie bei dokumentenzentriertem XML möglich.

In die Phase der Implementierung und Wartung gehören Prozesse wie Schemaevolution und die Verwendung von Dokumentinhalten in XSLT-Anwendungen. Geänderte Rahmenbedingungen und neue Erkenntnisse aus der Anwendung von Dokumenten machen eine eventuelle Anpassung und Optimierung der Struktur nötig und möglich.

7 Zusammenfassung

Die Fragestellung, was Gütekriterien für XML-Dokumente sind und wie ein "gutes" Design von XML-Dokumenten auszusehen hat, kann pauschal nicht beantwortet werden. Der Charakter von XML als Metasprache ermöglicht eine Anwendung in vielen Bereichen, wo immer Daten elektronisch gespeichert werden. So unterschiedlich der Kontext der Anwendungsbereiche ist, so unterschiedlich sind oftmals auch die Anforderungen an das Design.

Immer wieder auftretende Probleme beim Designprozess kann man in zwei Gruppen unterteilen. Die eine Gruppe beinhaltet die Probleme, die im Zusammenhang mit der Anwendungsmodellierung stehen. Hier fallen zum Beispiel Probleme wie Redundanz oder Inkonsistenz von Daten der Anwendung hinein. Die Art der Darstellung von Informationen oder der Grad der Verteilung von Dokumenten bzw. von Dokumentinhalten zählen ebenso in diese Kategorie.

Die zweite Problemgruppe der Designentscheidungen wird durch die Techniken und Konzepte von XML und seinen Erweiterungen gebildet. Das heisst, wird der Informationsgehalt der Daten durch eine XML-Technik bzw. durch eine Erweiterung optimal dargestellt. Sind Techniken der Darstellung schon allgemein akzeptiert und Standardsoftware vollständig oder nur teilweise integriert.

Im Verlaufe dieser Arbeit wurden eine Reihe von Faktoren und Designvarianten identifiziert, die die Qualität von XML-Inhalt steigern. Eine Möglichkeit, Dokumente qualitativ aufzuwerten, besteht darin, die von XML und seinen Erweiterungen dargebotenen Konzepte optimal einzusetzen. Diese Ansätze wurden versucht, im Grundlagenabschnitt aufzuzeigen. Dabei setzen Lösungen einerseits auf so triviale Sachen wie Kodierungsinformationen von Dokumenten auf, um Kombinationen von Dokumententeilen und die Portabilität optimal zu unterstützen. Andere Ansätze nutzen Schemasprachen, um die Struktur von Dokumenten mehr oder weniger restriktiv zu beschreiben.

Im weiteren Verlauf der Arbeit wurde versucht, Analogien aus dem Bereich der Softwaretechnik und der Datenbanken für XML-Dokumente sinnvoll abzubilden. Gerade aus dem Bereich der Softwaretechnik lassen sich Prinzipien wie Wiederverwendbarkeit und Techniken wie Modularisierung auch für XML-Inhalt applizieren. Dem Bereich der Datenbanken entspringt ein ganz wesentlicher Prozess, der Prozess der Normalisierung. Die Normalisierung von XML-Inhalt ist für die Klasse der datenzentrierten XML-Anwendung von größtem Interesse, da mit ihr solche Probleme wie redundante und inkonsistente Darstellung von Daten eliminiert bzw. reduziert werden können. Die Normalisierung von XML-Daten ist ein so wichtiges Gebiet innerhalb von XML, so dass sie Hauptgegenstand einiger, wichtiger Forschungsarbeiten im Bereich qualitativen XML's ist.

Ein anderes, für XML sehr gut anzuwendendes Prinzip ist die Verwendung von Pattern. Gerade im Bereich des Prototypings, des schnellen Erstellens lauffähiger Lösungen sind Pattern ein geeignetes Mittel, um den Designprozess zu unterstützen. So ist in einigen Bereichen der Anwendungsentwicklung die Verwendung von Pattern ein sogenannter 'big-time-saver'.

Im auf Pattern folgenden Abschnitt, wurden zwei aktuelle Forschungsarbeiten, die

Thematik der Qualität von XML-Anwendungen betreffend vorgestellt. In einer Arbeit wurde ein erweitertes Datenmodell entwickelt, welche die Darstellung von Informationen die Modellierung betreffend optimal unterstützen soll. Hauptgegenstand der anderen Arbeit ist der Prozess der Modellierung und die algorithmische Transformation des erstellten Modells in XML.

Im letzten Teil der Arbeit wurde versucht, die einzelnen Abschnitte zusammenzufassen und in einem Designprozess sinnvoll zu kombinieren. Als Lösung wurde ein modifiziertes Phasenmodell aus dem relationalem Datenbankentwurf vorgeschlagen. In den einzelnen Entwicklungsphasen sollten die vorgestellten Lösungen entsprechend appliziert werden. Die wichtigsten Schritte in den einzelnen Phasen, die in dieser Arbeit behandelt wurden sind, die Modellierung mittels geeigneter Modelle, die Transformation in eine Normalform und die Anwendung geeigneter Pattern. Nicht desto trotz bleibt die Optimierung von XML-Inhalt und die Bestimmung der Güte für XML-Anwendungen ein schwieriges Thema. Das große Anwendungsspektrum macht es kompliziert, allgemeingültige Aussagen zu formulieren. Außerdem sind für die Gruppe der dokumentzentrierten Anwendungen Empfehlungen kaum umzusetzen.

Literatur

[HaM03] Elliot Rusty Harold, W. Scott Means; *XML in a Nutshell*, O'Reilly, Jahr 2003, 2.Auflage

[PAT] *http://www.xmlpatterns.com/*, Jahr 2005

[ArL04] Marcelo Arenas, Leonid Libkin; *A Normal Form for XML Documents*, ACM Transactions on Database Systems Vol.29, März 2004, Seiten 195-232

[DLL03] Gillian Dobbie, T.W.Ling, M.L.Lee, x.Wu; *ORA-SS: A Conceptual Model for Semistructured Data*, ACM Journal Name Vol.V, August 2003, Seiten 1-27

[EmM] D.W.Embley, W.Y.Mok; *Developing XML Documents with Guaranteed "Good" Properties*

[Sch05] *http://www.schematron.com*, Jahr 2005

[Rel05] *http://relaxng.org/*, Jahr 2005

[Vli03] Eric van der Vlist;*XML Schema*, O'Reilly, Januar 2003

[SkW04] Marco Skulschus, Marcus Wiederstein;*XML Schema - Grundlagen, Praxis, Referenz*, Galileo Press, Mai 2004

[ARC05] *http://www.xml.com/pub/a/2003/03/26/patterns.html*,o'Reilly xml.com, Jahr 2005