

Anwendbarkeit von XML Schema für Daten und Metadaten im Bereich digitaler Bibliotheken

Magisterarbeit

Zur Erlangung des Magister Artium
durch den Fachbereich
Historisch-Kulturwissenschaftliche Informationsverarbeitung
der Universität zu Köln

vorgelegt von:

Martin Iordanidis

Metzer Str. 6

58332 Schwelm

Tel.: 02336/442848

E-Mail: martin@iordanidis.de

Gutachter

Prof. Dr. Manfred Thaller

Prof. Dr. Wolfgang Schmitz

Inhalt

ERSTER TEIL

Einleitung	5
Kapitel 1 Metadaten im Umbruch	6
1.1 Hintergrund und Problemstellung	6
1.2 Schlussfolgerung	9
Kapitel 2 Metadatenstandards und Austauschformate	9
2.1 MARC	9
2.1.1 Hintergrund	9
2.1.2 UNIMARC	11
2.1.3 MARC21	11
2.1.4 Inhalt und Struktur	13
2.1.5 Schlussfolgerungen	16
2.2 MAB2	17
2.2.1 Hintergrund	17
2.2.2 Inhalt und Struktur	17
2.2.3 Schlussfolgerungen	18
2.3 Encoded Archival Description (EAD)	19
2.3.1 Hintergrund	19
2.3.2 Inhalt und Struktur	20
2.3.3 Schlussfolgerungen	21
2.4 Electronic Binding Project (Ebind)	21
2.4.1 Hintergrund.....	21
2.4.2 Inhalt und Struktur.....	22
2.4.3 Schlussfolgerungen.....	22

Kapitel 3	Deskriptives Markup als Grundlage von Metadaten	23
3.1	Geschichte deskriptiven Markups	23
3.2	Standard Generalized Markup Language (SGML)	24
3.3	Extensible Markup Language (XML)	25
3.4	Von Metadaten zu „Metainformation“	26
3.5	Document Type Definitions als Datenmodell	27
3.6	XML Schema als Datenmodell	29
Kapitel 4	Datenmodelle	30
4.1	Hintergrund	30
4.2	Datensemantik	31
4.3	Flexibilität von Daten	35
4.4	Vier Aspekte von Daten	36
4.5	Kategorisierung	37
4.6	Semistrukturierte Datenmodelle	39
4.7	Zusammenfassung und Anwendung	40
ZWEITER TEIL		
Kapitel 5	Anwendungsfälle für XML Schema	43
5.1	XML Schema Basisarchitektur	43
5.2	Anwendungsfälle: Encoded Archival Description	45
5.2.1	Inhaltsmodell und Typisierung	48
5.2.2	Zwei Klassen von complex types	49
5.2.3	Compositors	49
5.2.4	Kontextinformation sichern	51
5.2.5	Kardinalitätsoperatoren	52
5.2.6	Attributtypisierung	54
5.3	Anwendungsfälle: MARC	56
5.3.1	MARC-Komponenten	56

5.3.2	Syntaxvorgaben durch reguläre Ausdrücke	58
5.4	Anwendungsfälle: MAB2	60
5.4.1	Modellierung von Homogenität	60
5.4.2	Bündelung durch komplexe Typen	62
5.4.3	Content specification.....	63
5.4.4	Bündelung durch Elementengruppen	65
5.5	Anwendungsfälle: Ebind	65
5.5.1	Notwendigkeit von Namespaces.....	65
5.5.2	Deklaration und Einbindung.....	66
5.5.3	Integration von Namespaces.....	67
5.6	Anwendungsfälle: RDF	70
5.6.1	Hintergrund	70
5.6.2	Beschreibung von Ressourcen	71
5.6.3	Beschreibung von Komponenten	72
5.6.4	Application Profiles.....	73
5.6.5	Integration von RDF und XML Schema.....	73
Epilog	75
Literaturangaben	76
Anhang A	XML Schema Beispiele.....	83
Anhang B	Crosswalks	93

Einleitung

Ziel der vorliegenden Arbeit ist es, das Potenzial der XML Schema-Technologie in seiner Rolle als Strukturgrammatik und als Datenmodell für Daten und Metadaten darzulegen. Die Domäne, innerhalb derer das Thema dieser Arbeit ansetzt - ‚digitales‘ Bibliothekswesen jenseits der Schwelle zum Informationszeitalter - ist noch so dynamisch wie die zahllosen (Metadaten-) Standards des World Wide Web und bereits so komplex und vielschichtig wie ihre analoge Vergangenheit. Es ist daher so erstrebenswert wie schwierig, den Rahmen dieser Domäne genau abzugrenzen. Wenn im Kontext der vorliegenden Arbeit also von ‚digitalen Bibliotheken‘ bzw. ‚verteilten Informationssystemen‘ die Rede ist, sind damit alle denkbaren Mischformen aus ‚elektronischen Bibliotheken‘ und ‚virtuellen Bibliotheken‘ gemeint. Erstere zeichnen sich durch digitalisierte Bestände aus, während letztere die Virtualisierung bibliothekarischer Dienste anstreben [EF00, S.5]. Die Praxis hat gezeigt, dass eine solche scharfe Abgrenzung nie wirklich existiert hat und alle Konzepte einer ‚digitalen Bibliothek‘ letztlich aus den Erfahrungen, Zielsetzungen und Möglichkeiten des konventionellen Bibliothekswesens und der Informationstechnologie erwachsen.

Metadatenvokabulare und -grammatiken konstituieren die Sprachen, die in diesen beiden Bereichen gesprochen werden. XML Schema ist eine solche Grammatik, und sie verspricht, deskriptives Markup dem Ideal einer *lingua franca* des WWW noch näher zu bringen. Ihre Ausdrucksmächtigkeit gewährleistet das ‚retrieval‘ von Informationen in seinem wörtlichsten Sinne - das ‚Zurückholen‘, ‚Wiedergutmachen‘ oder gar ‚Retten‘ von Information. Ihre Formatierung - XML Schema *ist* XML - ermöglicht eine Interoperabilität, die auf die Erfordernisse der Gegenwart und Zukunft ausgerichtet ist. Die vorliegende Arbeit möchte dies in zwei separaten, aber aufeinander bezogenen Teilen vermitteln. Der erste Teil widmet sich der Definition von möglichen Anwendungsbereichen von XML Schema in vier ausgewählten Metadatenstandards und Austauschformaten. Die historische bzw. theoretische Aufarbeitung der Markuptechnologie und Datenmodellierung soll XML Schema vor dem Hintergrund der theoretischen Informatik formal legitimieren. Der zweite Teil der Arbeit zeigt Lösungen für die in Teil eins formulierten Aufgabenstellungen auf.

Alle in Anhang A aufgeführten Beispielschemata befinden sich auch auf dem beigegeführten Datenträger.

Erster Teil

Kapitel 1 - Metadaten im Umbruch

1.1 Hintergrund und Problemstellung

Am 12. März 2002 veröffentlichte die Dublin Core Metadata Initiative (DCMI) das Dublin Core Element Set¹ erstmals in seiner XML Schema-Implementation [DCM02a]. Mit diesem Schritt nähert sich die DCMI ihren beiden erklärten Zielen, sowohl die Verbreitung betriebsfähiger Metadatenstandards zu fördern, als auch spezialisierte Metadaten-Vokabulare und deren Nutzung in ‚intelligenten‘ Informationssystemen zu entwickeln [DCM02b].

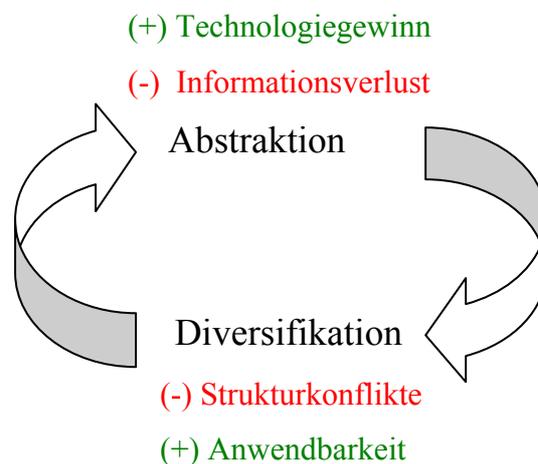
Die breite internationale wie interdisziplinäre Verbreitung des Dublin Core seit seiner Erschaffung im Jahre 1995 dokumentiert deutlich, wie groß der Bedarf an Interoperabilität inmitten eines höchst heterogenen Ökosystems von Metadatenstandards und Austauschformaten ist. Dies gilt nicht nur für das WWW, sondern für den Austausch von Metadaten auf jeder denkbaren Ebene - lokale Retrieval-Systeme in klassischen Bibliotheken sind ebenso davon betroffen wie international vernetzte Bibliotheksverbände. Untersuchungen der Bayrischen Staatsbibliothek über die Auswirkungen von stark heterogenen (hier: bibliographischen) Metadaten zeigen, dass Ergebnisse übergreifender Suchen nach Literatur aufgrund dieser Heterogenität noch unbefriedigend sind [WW01, S. 301ff]. Die konsequente Forderung: *„Sowohl die Betreiber einzelner Literaturinformationssysteme als auch die Entwickler übergreifender Suchdienste können und müssen zur Homogenisierung der Metadaten und zur weiteren Verbesserung des Retrievals beitragen“*(ebd.). Mit dem Dublin Core ist ein erster Schritt in die Richtung einer solchen Homogenisierung getan. Jedoch - und damit nähern wir uns dem Problem - vornehmlich innerhalb des Mediums World Wide Web.

Es steht außer Zweifel, dass ein globales Unterfangen wie die Durchsetzung eines universellen Metadatenstandards für das WWW mit der Implementation einiger weniger Deskriptoren beginnen muss. Den klassischen Bibliotheken und ihren prospektiven digitalen Nachfahren ist damit jedoch nur in dem Maße geholfen, wie ihre Bestandsdaten zu einer Kommunikation mit einer solchen kondensierten ‚Schnittstelle zum WWW‘ wie dem Dublin Core in der Lage sind.

¹ Das Dublin Core Element Set ist ein aus 15 Beschreibungselementen bestehendes Metadatenvokabular, mit dem die bibliographischen Mindestanforderungen an Webressourcen abgedeckt werden sollen. Ein umfassende Darstellung findet sich unter [DCM02b].

Initiativen wie Dublin Core sind über 25 Jahre an internationaler Standardisierungsarbeit im Bibliothekswesen vorausgegangen (vgl. Kap. 2.1.2). Die Expertise bibliothekarischer Arbeit konnte während dieser Zeit (zwangsläufig) umfassender in traditionelle Austauschformate wie MARC einfließen als in ein bewusst minimal angelegtes Vokabular, das primär für die Nutzung im Internet vorgesehen ist. Der Dublin Core hat in dieser Hinsicht Pionierarbeit geleistet, die im Zusammenhang mit anderen Metadatenstandards nach einer Fortführung verlangt. Gefragt ist bei diesem Unterfangen ‚das Beste von allem‘: Eine möglichst umfassende bibliographische Beschreibung der Ressource, wie sie Austauschformate wie MARC ermöglichen, *zusammen* mit den Vorteilen der Interoperabilität eines Minimalsets. All dies unter Vermeidung von Informationsverlusten, verbunden mit einer detaillierten Erschließung der Ressource ‚nach innen‘ und einer potenziell unbegrenzten Erweiterbarkeit. Document Type Definitions (DTDs) - eine lange verbreitete Methode, Metadatenvokabulare mittels einer strukturierten Grammatik zu beschreiben - können diese Idealanforderungen in ihrer Summe nicht erfüllen.

Die Konsequenzen daraus sind weitreichend. Sie haben einen Evolutionstrend im Bereich Metadaten ins Rollen gebracht, der aufgrund der bislang genutzten Mittel an seinem eigenen Wachstum zu scheitern scheint. Reflektiert man die Geschichte der Beschreibung von Daten durch Metadaten, kristallisiert sich dabei ein scheinbar endloses Wechselspiel von Abstraktion und Diversifikation heraus:



Beide Prinzipien scheinen sich gegenseitig zu bedingen und sind jeweils mit Vor- und Nachteilen verbunden. Nachfolgende Übersicht versucht, diesen Evolutionstrend anhand der Entwicklungssprünge² im Bereich der Metadatenutzung zu verifizieren:

² Auf eine genaue zeitliche Einordnung wurde dabei verzichtet.

Prinzip	Hist. Entwicklung	Technologiegewinn	Informationsverlust	Anspruch Datenmodell
(1) Abstraktion	Auszeichnung von Daten mit Metadaten	semantisches Markup	Verlust semantischer Neutralität	semantisch neutral
(2) Diversifikation	Entstehung vieler heterogener Metadatenvokabulare	-	kaum strukturelle Deckung in Instanzdokumenten	universelles Datenmodell
(3) Abstraktion	Entwicklung von Strukturgrammatiken	Document Type Definitions (DTDs)	Verlust semantischer Neutralität	semantisch neutral
(4) Diversifikation	Entwicklung spezieller Strukturgrammatiken	TEI, EAD etc.	kaum strukturelle Deckung zwischen <i>Gruppen</i> von Instanzdokumenten	universelles Datenmodell
(5) Abstraktion	Bibliographische Minimalsets	Dublin Core	Geringe Informationstiefe	hohe Informationstiefe

Die Beschreibung von Daten durch Metadaten (1) stellt eine Abstraktion dar. Da bei der Beschreibung von Phänomenen stets eine Interpretation der ‚Wirklichkeit‘ stattfindet (vgl. Kap. 4.2), wird die Semantik des Beschriebenen bei diesem Vorgang auf eine mögliche Bedeutung reduziert. Die Beschreibungen von Daten werden z.B. als Metadatenvokabulare realisiert. ‚Wirklichkeit‘ ist jedoch so vielgestaltig, wie Wahrnehmungen dieser Wirklichkeit existieren. Der Wunsch, die Welt aus verschiedenen Perspektiven heraus zu beschreiben, muss daher in einer Diversifikation (2) von vielen Metadatenvokabularen münden. Die Frühphase der Markuptechnologie sah sich mit exakt dieser Problematik konfrontiert (vgl. Kap. 3.1), da eine strukturelle Deckung von Instanzdokumenten noch nicht durch zugrundeliegende Grammatiken gewährleistet wurde. Daraus ergab sich ein Bedarf an strukturierten Grammatiken (DTDs), welche Abstraktionen (3) bestimmter Vokabulare darstellten. Die hierbei involvierte Interpretation schränkte die zulässigen Beschreibungen durch Elemente ein und deren Anordnung durch eine Hierarchie. Die Popularität des deskriptiven Markup setzte sich unterdessen interdisziplinär durch. Dies führte zu einer Diversifikation (4) von Strukturgrammatiken, da viele Fachrichtungen die Implementation eigener Metadatenvokabulare und korrespondierender Grammatiken anstrebten.

Sämtliche mit Metadaten ausgezeichnete Ressourcen haben gemein, dass sie auffindbar und identifizierbar sein sollen. Da die Codierung bibliographischer Metadaten in den unterschiedlichen Vokabularen i.d.R. heterogen ausfiel (vgl. Kap.2.1), machte man erneut vom Prinzip der Abstraktion (5) Gebrauch: die wichtigsten bibliographischen Metadaten wurden isoliert, vereinheitlicht und einer Ressource als Headerinformation hinzugefügt.

Genau dies ist der Ansatz des Dublin Core. Er sichert ein Mindestmaß an Interoperabilität, vernachlässigt jedoch den tieferen Informationsgehalt der beschriebenen Ressourcen. Daraus ergaben sich Anstrengungen der Dublin Core-Träger, das Problem mittels einer Verfeinerung der bestehenden Elemente zu entschärfen [DCM02c]. Die sog. *Qualifiers* der Dublin Core-Elemente ermöglichen eine genauere Beschreibung der DC-Elemente, in der Baker eine Parallele zu natürlichen Sprachen sieht: “*Qualifiers modify the properties of Dublin Core statements by specifying, in the manner of natural-language adjectives, ‘what kind’ of subject, date, or relation*” [Bak00].

An diesem Punkt der Analyse gelangen wir in die Gegenwart zurück. Mit der Implementation des Dublin Core als XML Schema wird deutlich, dass die Ausdrucksmächtigkeit einer Sprache nicht nur vom Umfang ihres Vokabulars abhängt. Ein Wortschatz - so groß er auch sein mag - ist wenig nutzbringend, wenn seine Verwendung nicht durch eine leistungsstarke Grammatik gestützt und reglementiert ist. XML Schema ermöglicht die Schaffung solcher ausdrucksmächtigen Grammatiken für Metadatenvokabulare, wie im zweiten Teil der vorliegenden Arbeit gezeigt wird.

1.2 Schlussfolgerung

Offensichtlich sind Strukturgrammatiken in Form von DTDs nicht in der Lage, komplexe Datenstrukturen mit einfachen Mitteln zu entwerfen und gleichzeitig deren Interoperabilität zu sichern (vgl. Kap.3.3). Die XML Schema-Technologie verspricht, diese Erwartungen leichter zu erfüllen (vgl. Kap. 4ff:).

Um eine Fokussierung auf die Anforderungen von bestehenden Austauschformaten an eine solche leistungsstarke Technologie zu bestimmen, sollen im Folgenden vier Formate untersucht werden, die für die vorliegende Arbeit relevant sind. Jede Untersuchung schließt mit möglichen Anwendungsszenarien für das genannte Austauschformat.

Kapitel 2 - Metadatenstandards und Austauschformate

2.1 MARC

2.1.1 Hintergrund

Das MARC-Format zählt zu den ältesten und umfangreichsten Standards im Bereich bibliographischer Metadaten. Das Akronym MARC steht für MACHine Readable Cataloging. In seiner ursprünglichsten Form wurde das MARC-Format Mitte der Sechzigerjahre auf Initiative der amerikanischen *Library Of Congress* entwickelt [TBL02]. Die ersten Entwicklungsschritte von MARC gehen auf das Pilotprojekt MARC I zurück, welches die

Erstellung und Nutzung von Katalogdaten in maschinenlesbarer Form erprobte. Parallel zu diesen Aktivitäten in den Vereinigten Staaten bemühte sich das britische *Council of the British National Bibliography* ebenfalls um technische Lösungen zur maschinenlesbaren Verwaltung bibliographischer Metadaten. Zielsetzung der britischen Initiative war es, die Produktion der *British National Bibliography* (BNB) mit Hilfe automatisierter Verfahren zu vereinfachen. Dieses Unterfangen firmierte unter dem Namen BNB MARC Project. Im Jahre 1968 fusionierten die amerikanischen und britischen Bestrebungen unter dem Projektnamen MARC II mit dem Ziel, ein standardisiertes Austauschformat für bibliographische Metadaten zu schaffen. Die Zielsetzung dieser Kooperation wurde aufgrund stark abweichender Praktiken bei der Katalogisierung und divergierender Anforderungen der nationalen Bibliotheken verfehlt.

Die Folge war die Herausbildung eines jeweils eigenen Standards für Großbritannien und die Vereinigten Staaten. So gingen aus dem MARC II-Projekt der UKMARC für Großbritannien sowie der USMARC für den Einsatz in Nordamerika hervor. Das im selben Jahr veröffentlichte Regelwerk zur Katalogisierung bibliographischer Daten, die *Anglo-American Cataloguing Rules* (AACR), erschienen dementsprechend einmal als britische und einmal als nordamerikanische Edition [TBL02].

Weitere nationale Ableger des MARC-Formats folgten noch Ende der Sechzigerjahre. In Australien etablierte sich der AusMARC, in Kanada bildete sich das Derivat CanMARC heraus. Trotz des von Anbeginn der MARC-Initiative an herrschenden Trends zur Diversifikation in heterogene nationale Formate³ wurde das Grundbestreben des MARC-Formats als willkommener Beitrag in der internationalen Standardisierungsarbeit wahrgenommen. Im Tagungsbericht des im Juni 1971 in Berlin veranstalteten Symposiums zum Thema „Austausch bibliographischer Daten und das MARC Format“ heißt es:

Durch die Anwendung der elektronischen Datenverarbeitung [...] sowie einer eindeutigen Kennzeichnung des einzelnen Buches [...] wird es wahrscheinlich möglich sein, nationale Netze mit einem oder mehreren bibliographischen Zentren zu schaffen, die die bibliographischen Informationen über Neuerscheinungen des eigenen Landes sammeln, maschinell lesbar machen und auch in dieser Form verteilen können. Die nationalen Zentren ihrerseits werden untereinander die Daten ihrer eigenen, nationalen Bibliotheken verteilen und austauschen. [KKP*72, S.5]

³ Ein Blick in die Formatquellen der *Library Of Congress* offenbart nicht weniger als drei Dutzend nationale Derivate des MARC [LOC03a]

Hier wird zum einen die Notwendigkeit einer Standardisierung auf internationaler Ebene unterstrichen, zum anderen die heute bestehende (und im Entstehen begriffene) Infrastruktur verteilter Bibliothekssysteme skizziert.

2.1.2 UNIMARC

Bedingt durch das ausdrückliche Bestreben, auch über nationale Grenzen hinaus einer normierten Katalogisierung zu folgen, geriet die Heterogenität der diversen MARC - Formate zu einem praktischen Problem. Als Konsequenz daraus wurde ab 1977 mit UNIMARC ein universelles MARC-Format implementiert, welches „den Datenaustausch zwischen nationalen bibliographischen Einrichtungen [...] erleichtern“ sollte [DDB01a]. UNIMARC ist ein internationales bibliographisches Format, dessen inhaltliche Konsistenz technisch wie politisch durch ein internationales Gremium gesichert werden soll. In dieser Funktion ist seit 1991 das *Permanent UNIMARC Committee (PUC)* mit der damit verbundenen Aufsichtspflicht betraut. Das PUC ist eine aus sieben bis neun Mitgliedern bestehende internationale Expertengruppe, die der *International Federation Of Library Associations and Institutions (IFLA)* angegliedert ist. Seine Aufgabe besteht darin, „to control the UNIMARC format in accordance with the principles of Universal Bibliographic Control“ [IFL02a] und gilt als das Schwerpunktprogramm der IFLA im Bereich der Standardisierungsarbeit. Erwähnenswert ist weiterhin, dass das PUC als Teil des Programms „Universal Bibliographic Control and International MARC“ (UBCIM) unter der direkten Schirmherrschaft der *Deutschen Bibliothek Frankfurt* steht [DDB01b]. Dies und die Tatsache, dass von den weltweit 50 Institutionen, die UNIMARC als Austauschformat nutzen⁴, allein sieben in Deutschland liegen, lässt die Frage nach der Notwendigkeit eines zusätzlichen, spezifisch deutschen Austauschformates, dem MAB bzw. MAB2⁵, aufkommen.

2.1.3 MARC21

Seit dem Jahr 1997 firmiert der MARC-Standard unter dem Namen MARC21 [LOC02a]. Das Label „MARC21“ soll die zukunftsorientierten Intentionen der MARC-Träger für das 21. Jahrhundert signalisieren und den zunehmend internationalen Charakter als Codierungs- und Austauschformat zum Ausdruck bringen [SLB02]. Inhaltlich betrachtet, ist MARC21 das

⁴ Als Grundlage dient eine 1998 von dem UBCIM-Programm durchgeführte Umfrage zur Nutzung des UNIMARC, s. [IFL02b]

⁵ Das Maschinelle Austauschformat für Bibliotheken (MAB) ist Gegenstand von Kap. 2.2.

Ergebnis des Bestrebens, die beiden Formate USMARC und CanMARC zu harmonisieren. Aufgrund der besonderen Ausprägung des CanMARC im zweisprachig geprägten Kanada gestaltete sich die Integration des CanMARC und des USMARC zunächst schwierig. Dennoch wurde im Zuge der 1994 begonnenen Harmonisierung eine Zusammenführung der beiden nordamerikanischen MARC-Standards konsequent verfolgt. Das Benutzerhandbuch für das MARC21-Format erschien im Jahre 1999 erstmals in der neuen Form und wird seitdem sukzessiv durch Nachträge ergänzt [SLB02]. Die englische Ausgabe wird dabei von der *Library Of Congress* herausgegeben, eine zusätzliche französische Ausgabe von der kanadischen *National Library Of Canada*.

Die Bestrebungen einer Kohärenzschaffung im Bereich internationaler Austauschformate sind nicht spurlos am deutschen Bibliothekswesen vorübergegangen. Spätestens seit Ende der Neunzigerjahre wird die Adaption des international ausgerichteten MARC21-Formats in Verbindung mit RAK (Regeln zur Alphabetischen Katalogisierung)⁶ in Deutschland konkret diskutiert - nicht selten überaus emotional⁷. Als die Notwendigkeit einer sachlichen Beurteilung der möglichen Migration zu MARC21 mit den verbundenen Konsequenzen allzu offensichtlich wurde, reagierte der Standardisierungsausschuss der *Deutschen Bibliothek* Ende 2001 mit einem bis dato ungewöhnlichen Schritt: Nachdem sich der Ausschuss „*im Grundsatz für einen Umstieg von dem deutschen Regelwerk RAK und dem Format MAB hin zu den amerikanischen Standards AACR und MARC*“ ausgesprochen hat [DDB01c], macht man die endgültige Entscheidung nun von einer Vorab-Studie abhängig, die die Rahmenbedingungen und Zeitabläufe eines Umstiegs untersuchen soll. Aufgrund des großen Publikumsinteresses - und entgegen dem sonst üblichen Ablauf - wurde der Antrag auf das DFG-Projekt "Umstieg auf internationale Formate und Regelwerke (MARC21, AACR2)" der Öffentlichkeit zugänglich gemacht. Die heikle Sachlage klingt in dem im Februar 2002 an die DFG gerichteten Antrag durch:

Das Projekt [...] untersucht *nicht* die Qualität von Regelwerken und von Formaten. Es untersucht die Rahmenbedingungen und die Konsequenzen eines Umstiegs von

⁶ RAK stellt ein in Deutschland entstandenes Regelwerk zur alphabetischen Katalogisierung dar. Es existieren viele Sonderformen des RAK, darunter für öffentliche und wissenschaftliche Bibliotheken: RAK OeB und RAK WB. Eine genaue Darlegung des Regelwerkes findet sich bei [Hac92, S. 190ff.].

⁷ Ausreichend Belege finden sich in der Inetbib-Diskussionsliste unter <http://www.inetbib.de> bzw. auf den jährlich stattfindenden Tagungen der Inetbib-Liste.

RAK und MAB auf AACR und MARC. Dabei werden bibliothekarische, organisatorische, technische und betriebswirtschaftliche Aspekte beleuchtet. Das Ergebnis der Studie ist in der Sache offen. [DDB01c]

Zusammengefasst werden in dem Antrag auf die Studie folgende Argumente *für* einen Umstieg auf internationale Formate genannt:

- Verbesserte Voraussetzungen für die Nutzung verteilter bibliographischer Datenbanksysteme
- ungehinderte wechselseitige Datenübernahme aus vernetzten Datenbanken
- bessere Verteilung deutscher Informationen und Daten
- bessere Nutzung internationaler Daten für deutsche Wissenschaftler

Kritiker der Migration befürchten dagegen v.a., dass der mit einem Umstieg verbundene Aufwand erheblich unterschätzt würde und die Zielstandards AACR2/MARC21 ihrerseits veraltet und modernisierungsbedürftig seien [DDB01d].

Derzeit werden im Rahmen der o.g. Studie verschiedene Szenarien einer Migration zu AACR2/MARC21 in Betracht gezogen. Diese reichen vom radikalen Katalogabbruch und einer Neukatalogisierung über maschinelle Übersetzungen vom MAB2 nach MARC21 bis hin zu Mischlösungen aus RAK/MARC21 bzw. AACR2/MAB2. Auch das Verbleiben bei RAK/MAB2 wird in Betracht gezogen, ebenso eine Vertagung der Entscheidung bei gleichzeitiger aktiver Beteiligung an internationalen Entwicklungen [DDB01d]. Zum Zeitpunkt des Drucks der vorliegenden Arbeit wertet *Die Deutsche Bibliothek* die Rückläufer einer Umfrage aus, in der Rahmenbedingungen wie die Systemvoraussetzungen der jeweiligen Bibliotheken, Verbände und bibliothekarische Verbände erfragt werden. Das Projekt "Umstieg auf internationale Formate und Regelwerke (MARC21, AACR2)" wird nach 18 Monaten Laufzeit am 31. Oktober 2003 enden.

2.1.4 Inhalt und Struktur

MARC21 und seine Vorgänger stellen Mechanismen bereit, mit deren Hilfe Maschinen bibliographische Informationen austauschen, lesen und interpretieren können. Die Datenelemente des MARC-Standards bilden laut der *Library Of Congress* das Fundament der meisten heute genutzten Bibliotheken [LOC02a]. Die heute existenten MARC21-Formate umfassen eine Familie von fünf Standards:

1. MARC 21 Format for Authority Data
2. MARC 21 Format for Bibliographic Data

3. MARC 21 Format for Classification Data
4. MARC 21 Format for Community Information
5. MARC 21 Format for Holdings Data

Allen MARC-Standards ist gemein, dass sie sehr viel stärker numerisch orientiert arbeiten als andere - zumeist jüngere - Metadatenformate. Dies ist an drei, teils historisch bedingten Gründen festzumachen.

Erstens ist MARC in seiner prägenden Entwicklungsphase stark der Dewey Dezimalklassifikation gefolgt. In dem bereits im 19. Jahrhundert geschaffenen Klassifikationssystem des amerikanischen Bibliothekars Melville Dewey (1851-1931) werden Hauptkategorien (z.B. von wissenschaftlichen Disziplinen) mit einer Zahl von 0 bis 9 gekennzeichnet, die durch Hinzufügen weiterer Ordnungszahlen in Unterkategorien gegliedert werden können. Diese Vorgehensweise kann bis in beliebig tiefe Ebenen fortgeführt werden und so immer genauere Kategorien beschreiben:

6:62 Technik
 622 Bergbautechnik
 622.3 Einzelne Bergbauzweige
 622.33 Kohlebergbau

nach [HBI95]

Obwohl die Dewey Dezimalklassifikation (DDK) im Wissenschaftsverständnis des 19. Jahrhunderts wurzelt, gilt sie auch über 125 Jahre nach ihrem Entstehen als die international am meisten verbreitete Universalklassifikation⁸. *Die Deutsche Bibliothek* bescheinigt der DDK einen Status, der noch immer „flexibel auf wissenschaftliche Veränderungen und Globalisierung reagiert“ und hat zur Nutzung der DDK in den deutschsprachigen Ländern das *Konsortium DDC Deutsch* ins Leben gerufen [DDB01e].

Ein zweiter Grund für die numerische Ausrichtung des MARC ist eng mit der Geschichte der Informationsverarbeitung verknüpft. Während der frühen Entwicklungsphase des MARC in den Sechzigerjahren galt die Schonung von Speicherressourcen als eine Maßgabe. Durch die Nutzung sog. Felder, die den Zahlencodes der DDK nachempfunden sind, lassen sich MARC-Records relativ kompakt zur Auszeichnung eines bibliographischen Objekts nutzen - sehr viel knapper, als dies mit textuellen Bezeichnern möglich wäre:

⁸ Dies wurde v.a. durch das weitgehend sprach- und kulturneutrale Verständnis der Dezimalzahlen ermöglicht.

MARC Felder	Bedeutung	Daten
100 1# \$a	Main entry, personal name w/ surname:	Arnosky, Jim.
245 10 \$a	Title proper:	Raccoons and ripe corn /
\$c	Statement of responsibility:	Jim Arnosky.
250 ## \$a	Edition statement:	1st ed.
260 ## \$a	Place of publication:	New York :
\$b	Name of publisher:	Lothrop, Lee & Shepard Books,
\$c	Date of publication:	c1987.
(...)	(...)	(...)

nach [LOC02b]

Drittens und letztens sind Zahlen und Symbole besser als Indizes geeignet als durch Literale beschriebene Feldeigenschaften. Das MARC-Format hat in dieser Hinsicht schon früh die Perspektive der Datenbanktechnologie berücksichtigt, indem z.B. ähnliche Felder nur einmal gespeichert und über Relationen mit Daten verknüpft werden konnten.

Jeder bibliographische MARC-Eintrag besteht aus logisch unterteilten *fields*. Manche *fields* werden in *subfields* unterteilt. Sämtliche MARC-Felder werden durch dreistellige Zeichenfolgen beschrieben, die *tags* genannt werden. Dies gilt auch, wenn den dreistelligen *tags* weitere Zeichen folgen sollten. Manche MARC-*tags* können mehrfach auftreten. Wenn ein *tag* wiederholt werden kann, wird dies in der MARC-Dokumentation durch das Symbol ‚R‘ (Repeatable) gekennzeichnet. Kann er nur einmal auftreten, wird dies mit dem Symbol ‚NR‘ (Non-Repeatable) vermerkt [LOC02b]. Zum Beispiel kann ein Katalogeintrag mehrere Themen (Subjects) haben und müsste daher mit dem Symbol ‚R‘ versehen werden.

Die *Library Of Congress* hat festgestellt, dass nur etwa 10% aller existierenden MARC-*tags* für Records immer wieder verwendet werden, während die anderen 90% nur gelegentlich auftreten [LOC02b]. Folgende Tabelle zeigt die am häufigsten verwendeten *tags*⁹ zusammen mit ihrer Bedeutung und dem Vorkommen:

⁹ Die Auflistung beruht inhaltlich auf den Angaben des *Library Of Congress* MARC-Tutorials [LOC02b]; zum besseren Verständnis der Felder wurden jedoch die korrekten deutschen Übersetzungen aus dem Handbuch der Schweizerischen Landesbibliothek herangezogen [SLB02].

MARC tag	Bezeichnetes Feld	(non-)repeatable
010	<i>Library Of Congress</i> Kontroll-Nummer	(NR)
020	Internationale Standardbuchnummer	(R)
100	Haupteintragung - Personennamen (z.B. Autor),	(NR)
245	Sachtitel und UrheberInnenangaben	(NR)
250	Ausgabebezeichnung (Edition),	(NR)
260	Erscheinungsvermerk	(NR)
300	Physische Beschreibung (bei Büchern oft als Kollation bezeichnet)	(R)
440	Gesamttitelangabe	(R)
520	Anmerkungen, Zusammenfassung usw.	(R)
650	Thema	(R)
700	Nebeneintragung – Personennamen (Co-Autor, Herausgeber, Illustrator)	(R)

Durch die in MARC erlaubten Wiederholungen von *tags* ist die Eindeutigkeit von Einträgen nicht immer gewährleistet. Sollen sie eindeutig identifiziert werden, ist dazu ein Mindestmaß an Kontextinformation nötig. Andere Austauschformate wie das deutsche MAB2 kritisieren diese Liberalität des MARC-Formates und reglementieren die Benutzung von *tags* strenger.

2.1.5 Schlussfolgerungen

Das MARC-Format markiert eine langfristige Anstrengung zur Vereinheitlichung bibliographischer Formate, welche auf internationaler Ebene Resonanz gefunden hat. Da dieser Trend in Zukunft anzuhalten scheint, sich aber inzwischen auch andere Austauschformate herausgebildet haben, ist eine Reihe von Konkordanz zur Übersetzung zwischen diesen Formaten zu erwarten. Zu einem großen Teil sind diese bereits implementiert. Es besteht Bedarf für Anwendungen aus dem Markup-Bereich, die diese Übersetzungen leisten können.

MARC beruht durch die Nähe zur DDK auf einer Systematik, die im wesentlichen auf die Abbildung strukturierter Hierarchien abzielt. Idealerweise sollte MARC daher auf Datenmodellen aufsetzen, die ebenfalls eine hierarchische Strukturierung von Daten vorsehen. MARC verfolgt genau wie die DDK das Ziel einer leichten und potenziell unbegrenzten Erweiterbarkeit.

Weiterhin ist MARC stark numerisch orientiert und sollte deshalb auf Datenmodellen beruhen, die eine ausgeprägte numerische Datentypisierung unterstützen. Die Möglichkeit präziser Syntaxvorgaben etwa durch reguläre Ausdrücke kann beim Einsatz von MARC im Verbund mit Strukturgrammatiken nur von Vorteil sein.

Warum XML Schema vor dem Hintergrund dieser Schlussfolgerungen mögliche Lösungen verspricht, wird in Kapitel 5.3 des zweiten Teils dieser Arbeit dargelegt.

2.2 MAB2

2.2.1 Hintergrund

Das Format MAB (Maschinelles Austauschformat für Bibliotheken) ist ein vor allem in Deutschland gebräuchliches Austauschformat. Es wird i.d.R. in Verbindung mit den RAK (Regeln zur Alphabetischen Katalogisierung) verwendet. Die Anfänge des MAB gehen auf das Jahr 1973 zurück, in dem unter der Federführung der *Deutschen Bibliothek* zusammen mit der *Arbeitsstelle für Bibliothekstechnik* ein nationales Austauschformat initiiert wurde [Eve99]. Eine umfassende Revision des MAB führte 1995 nach zweijähriger Entwicklungsarbeit zu der neuen Formatversion MAB2. Seitdem sind vier Ergänzungsschriften des MAB2 erschienen. Bis heute wird das Format durch den MAB-Ausschuss der *Deutschen Bibliothek* getragen, der auch für die Pflege und Weiterentwicklung des Formats Sorge trägt [DDB01f]. Eversberg bemerkt, dass MAB für *Die Deutsche Bibliothek* selbst kein Arbeitsformat darstellt, da es dort hausintern niemals eingesetzt wurde [Eve99].

2.2.2 Inhalt und Struktur

Ähnlich wie sein anglo-amerikanisches Pendant MARC besteht MAB aus fünf Formaten, die verschiedene Arten von Daten auszeichnen:

1. MAB-Format für bibliografische Daten (MAB-Titel)
2. MAB-Format für Personennamen (MAB-PND)
3. MAB-Format für Körperschaftsnamen (MAB-GKD)
4. MAB-Format für Schlagwörter (MAB-SWD)
5. MAB-Format für Lokaldaten (MAB-Lokal)

vgl. [DDB01f]

Bereits beim Vergleich mit den Bezeichnungen der MARC-Formate wird deutlich, dass MAB *kein* streng kongruentes Abbild der MARC-Architektur darstellt. Der erste Eindruck erweist sich bei näherer Analyse des MAB als richtig. Obwohl zweifelsohne von MARC beeinflusst¹⁰, weist MAB zum Teil gravierende konzeptuelle Unterschiede zu MARC auf.

Während die MARC-Formate bei der Titelaufnahme einer relativ strengen Reihenfolge von Record-Köpfen und Nebeneintragungen folgen, erlaubt MAB es, *sachlich* zueinander passende Elemente in Segmenten anzuordnen [PP02]. Der Ansatz des MAB ist also mehr auf die Verknüpfung semantisch verwandter Komponenten ausgerichtet als die vergleichsweise statische Struktur der MARC-Formate.

Einen weiteren Unterschied stellt die Zuordnung zwischen bibliographischen Elementen und Feldern dar. Während MARC in Feldern oder Unterfeldern ggf. mehrere bibliographische Elemente zusammenfasst, weist MAB jedem Feld im allgemeinen nur ein Element zu [PP02]. Zum Beispiel zeichnet Feld 245 des bibliographischen MARC-Formats einen Sachtitel zusammen mit einem Paralleltitel auf, während MAB ein eigenes Feld für den Paralleltitel definieren würde. MAB bevorzugt also insgesamt eine atomarere Aufschlüsselung und setzt die Einzelkomponenten dann in einen Zusammenhang. In MARC würden Angaben zu einem mehrbändigen Werk ggf. in einem Satz zusammengefasst werden, während MAB in diesem Fall mehrere zueinandergehörende Sätze (Hauptsätze, Untersätze und Nachsätze) bildet. Diese Sätze stehen in einem hierarchischen Verhältnis zueinander [PP02].

Weiterhin erlaubt das MAB-Format nur eine bestimmte Anzahl von Feldwiederholungen. In MARC ist die (oft beliebige) Wiederholung eines Feldes dagegen durch den Feldzusatz „repeatable“ möglich.

2.2.3 Schlussfolgerungen

Das MAB-Format erlaubt eine feinere Granularität bei der Auszeichnung bibliographischer Elemente als MARC. Die stärkere Diversifizierung kann durch die Gruppierung von ähnlichen Elementen aufgefangen werden. Zusammengehörende Elemente werden hierarchisch angeordnet. Bei einer Nutzung des MAB2-Formats in einem XML-basierten Datenumfeld sind daher alle Aspekte der Gruppierung von Elementen sowie dokumenteninterne Linking-Mechanismen von Interesse. Inwieweit XML Schema dazu geeignet ist, semantisch homogene Komponenten ohne Informationsverlust zu bündeln, wird in Kapitel 5.4 des zweiten Teils dieser Arbeit gezeigt

¹⁰ Die Kennung von Feldern durch dreistellige Ziffern wird bei MAB genauso offensichtlich wie die Anlehnung an das System der Dewey Dezimalklassifikation.

2.3 Encoded Archival Description (EAD)

2.3.1 Hintergrund

Wohl kaum ein Metadatenstandard ist so sehr auf die wirklichkeitsgetreue Abbildung realweltlicher Objekte ausgerichtet wie die Encoded Archival Description (EAD). Dies liegt v.a. darin begründet, dass der Standard maßgeblich für diesen Zweck erschaffen wurde. Die Entwickler der EAD¹¹ richteten sich dabei nach ihren Beobachtungen in Bezug auf archivalische Findbehelfe und deren Verwendung [Rut98]. Sie begannen damit, die Definition des Begriffes ‚finding aid‘ zunächst genau zu umreißen. Der Terminus ‚Findbehelf‘ kann im Bereich des Archivwesens auf eine ganze Reihe von Werkzeugen angewandt werden. Die EAD-Entwickler grenzten diesen Bereich daher auf Bestände („inventories“) und deren Register („registries“) ein, die Medien wie gedruckte und elektronische Texte, Bildmaterial jeder Art und Tonaufnahmen umfassen können.

Ein Hauptmotiv der Projektgruppe war die Entwicklung eines Metadatenvokabulars, das Informationen über rein bibliographische Beschreibungsmodi hinaus codieren konnte: „[The developers] *were keen to include information beyond that which was provided by traditional machine-readable cataloging (MARC) records*“ [LOC02d]. Diese Informationen sollten neben der möglichst exakten physischen Beschreibung von Objekten auch deren intellektuellen Inhalt präzise dokumentieren können. So ist auch die erste Revision der EAD-DTD im Jahr 1995 weniger von informationstechnologischen Gesichtspunkten geleitet als von der Expertise erfahrener Archivare¹²:

The group shared a common understanding of how and why finding aids are constructed, the kinds of information they contain (or should contain), and the uses to which they have been put. This knowledge and appreciation of traditional paper-based finding-aids, coupled with ideas for electronic enhancements, influenced the group`s review and redesign of Pittis original model and inspired the development of a new DTD [...].

[Rut98, S. 47]

11 Die Entwicklung der ersten EAD-DTD im Jahr 1993 geht auf ein Projekt an der Berkeley University Of California unter der Leitung von Daniel Pitti zurück [LOC02d].

12 Die Gruppe des sog. ‚Ann Arbor Meetings‘ im Juli 1995 bestand aus sieben Archivaren gegenüber einem SGML- Spezialisten [Rut98, S. 47]

2.3.2 Inhalt und Struktur

Die Designprinzipien der EAD-Entwickler wurden anhand existierender Findbücher in realen Archiven und Registern definiert. Es zeigte sich dabei, dass bibliographische Informationen über Findbücher gewissen Strukturmerkmalen gehorchen, die immer wieder auftreten. Dazu zählen beispielsweise das Titelblatt, Vorwort und Inhaltsverzeichnis. Auch der Inhalt eines Findbuches ist - wenn auch umfangreicher - für den praktischen Einsatz normierbar. Aus diesen Fakten leitete das EAD-Designteam die Schlussfolgerung ab, dass Findbücher auf der grundlegendsten Ebene aus zwei Segmenten bestehen:

1. Ein Segment, das Informationen über das Findbehelf selbst enthält wie z.B. Titel, Verfasser und Erstellungsdatum sowie
2. ein Segment mit Informationen über den inhaltlichen Korpus des Findbehelfs, z.B. eine Sammlung oder Serie von Einzelstücken

[Rut98, S. 54]

Diese beiden Segmente spiegeln sich in der EAD-DTD als sog. *high-level* Elemente wider, die administrative und korpusbezogene Metadaten ähnlich abbilden wie dies in Headern und Textkorpora der Text Encoding Initiative (TEI)¹³ üblich ist.

Als weitere Maßgabe galt, dass der Standard die Erzeugung sowohl neuer Findbehelfe als auch die Konversion bestehender (gedruckter) Findbehelfe erlauben sollte. Gleiches gilt für Datenbankbestände. Damit ist EAD grundsätzlich auf Erweiterbarkeit ausgerichtet, bedient sich dabei aber der unzulänglichen Mittel von DTDs. Der Einsatz von DTDs erwies sich in diesem Kontext als problematisch, da diese nur spezifische Aufgaben berücksichtigen konnten: „(...) *although a single DTD might be used for many purposes, such as data conversion, interchange, authoring etc. it could be optimized for only one function*“ [Rut98, S. 49].

Die Encoded Archival Description hat ihre interne Struktur niemals dem Diktat etablierter Standards folgen lassen. So lassen sich aus einem EAD-Record nur in dem Maße MARC-relevante Felder extrahieren, wie diese Felder auch für EAD selber sinntragend sind. Es wurde bewusst nicht versucht, vollständige MARC-Records in die EAD-Struktur einzubetten. Im Gegenzug sind bestehende semantische Überschneidungen mit anderen Formaten sehr gewissenhaft (und vergleichsweise früh) als Konkordanzen, sog. „Crosswalks“, dokumentiert

¹³ vgl. <http://www.tei-c.org/>

worden [SAA99. S. 235ff]. Dazu zählen Crosswalks zu MARC21, ISAD(G)¹⁴ und Dublin Core. Die im Kontext dieser Arbeit relevanten Crosswalks sind in Anhang B aufgeführt.

2.3.3 Schlussfolgerungen

Die Encoded Archival Description setzt idealerweise auf einem Datenmodell auf, mit dem auch komplexe Inhaltsmodelle präzise abgebildet werden können. Weiterhin sollten Mechanismen zur Verfügung stehen, die bestimmte hierarchische Anordnungen von Komponenten erzwingen können. Um der leichten Erweiterbarkeit von Inhaltsmodellen, die mit der EAD angestrebt wurde, gerecht zu werden, ist ein hoher Grad von abstrakter Typisierung empfehlenswert. Die Transkription eines EAD-Minimalsets in ein XML Schema ist Gegenstand von Kapitel 5.2.

2.4 Electronic Binding Project (Ebind)

2.4.1 Hintergrund

Die Ebind-DTD ist ein Set von SGML-codierten Metadaten, mit denen digitales Bildmaterial ausgezeichnet werden kann. Ebind wurde 1996 an der Bibliothek der Universität Berkeley im Zuge eines Umstieges von traditionellen Fotokopien auf digitale Xerox-Kopien entwickelt. Dieser Schritt wurde nötig, weil eine sehr große Anzahl vom Verfall bedrohter Bücher fortan nur noch in Form digitaler Repliken genutzt werden sollte. Aufgrund des umfangreichen Dokumentenbestandes waren Kostengünstigkeit und Zeiteffizienz die maßgeblichen Faktoren bei der Entwicklung von Ebind. Verantwortlich für das Ebind-Projekt waren Alvin Pollock und der Schöpfer der Encoded Archival Description (EAD), Daniel Pitti [BDL02].

Die in Ebind codierte Information entspricht der eines Inhaltsverzeichnisses, ergänzt um wenige bibliographische Metadaten. Ebind unterstützt aufgrund seiner bewusst kompakt konzipierten Struktur kaum eine der heute relevanten Konventionen bei Metadaten, die bei einer Langzeitarchivierung angezeigt wären. Spätere Erweiterungen wie „Ebind2HTML“ [BDL02] dehnten den Funktionsumfang von dem des Inhaltsverzeichnisses zwar auf Navigationshilfen für Online-Benutzer aus, ließen das Problem der Isolation von anderen Metadatenstandards aber unberührt. Die *Berkeley Library* bemerkt, dass Ebind an seiner Geburtsstätte nicht über die Entwicklungsphase hinauswuchs und nie voll in den Produktionsablauf in Berkeley integriert wurde. Vielmehr adaptierten andere Institutionen

¹⁴ Die General International Standard Archival Description (ISAD(G)) ist ein 1993 verabschiedeter archivarischer Standard, der eine systematische Beschreibung hierarchisch strukturierter Erschließung ermöglicht. vgl. dazu [KL03]

Ebind und werteten dessen Potenzial im Kontext anderer Projekte im Bereich digitaler Konservierung auf [BDL02]. Im Jahre 1998 wurde Ebind in Berkeley von dem Standard MOA2 („Making Of America 2“) abgelöst. MOA2 wurde im Gegensatz zu Ebind mit den administrativen, strukturellen und technischen Metadaten ausgestattet, die zu diesem Zeitpunkt als notwendig für eine Langzeitarchivierung erachtet wurden. Ende 2001 wurde MOA2 landesweit adaptiert und firmiert seitdem unter dem Namen Metadata Encoding & Transmission Standard (METS) [BDL02].

2.4.2 Inhalt und Struktur

Die Struktur von Ebind ist an den Kern des TEI-Tagsets angelehnt, folgt also dem Konzept eines bibliographischen Headers zusammen mit Korpusinformationen. Ebind ist jedoch wesentlich einfacher strukturiert und folgt einem grundsätzlich anderen Paradigma als die TEI. Während Ebind vorwiegend die *physikalische* Struktur (Seiten, Bindung etc.) eines Dokumentenverbundes beschreibt, kann die TEI auch die *intellektuelle* Struktur (Kapitel, Abschnitte etc.) eines Dokumentes auszeichnen. Umgekehrt existiert in der TEI kein Element, das explizit eine Seite beschreiben kann [BDL02]. Die TEI-DTD versucht, dieses Problem durch ein Leerelement <pb> („page break“) innerhalb eines Textabschnittes zu lösen. So lassen sich mit der TEI zwar Komponenten eines Textkorpus in Unterabschnitte („divisions“) unterteilen und eine Granularität bis hin zu einzelnen Schriftzeichen erreichen - die explizite Auszeichnung einer Seite ist dagegen anderen Vokabularen wie Ebind vorbehalten.

2.4.3 Schlussfolgerungen

Die Entwicklung der Ebind-DTD ist von einem klar definierten Einsatzzweck geprägt. Sie erlaubt weder eine tiefere Erschließung des intellektuellen Inhalts noch die Bereitstellung umfangreicherer bibliographischer Information. Im Gegenzug stellt Ebind ein Spezialvokabular für die physische Beschreibung von Ressourcen zur Verfügung.

Damit ist Ebind prädestiniert für den Einsatz zusammen mit komplementären Standards, die entweder eine tiefere Erschließung erlauben oder ihrerseits von der Zweckmäßigkeit der Ebind-DTD profitieren sollen. Mit den bestehenden SGML-Implementationen ist die Zusammenführung verschiedener Paradigmen wie die der TEI und Ebind nicht möglich [BDL02]. Inwieweit XML Schema mit Hilfe von Namespaces eine Integration von Metadatenvokabularen unterschiedlicher Designintention erlaubt, ist Gegenstand von Kapitel 5.5.

Kapitel 3 - Deskriptives Markup als Grundlage von Metadaten

3.1 Geschichte deskriptiven Markups

Viele der offensichtlichen Nachteile von Document Type Definitions erschließen sich aus dem historischen Kontext ihrer Entwicklung. Als erste nennenswerte Auszeichnungssprache entstand die Standard Generalized Markup Language (SGML) basierend auf ihrem Vorgänger, der Generalized Markup Language (GML). GML wurde seit 1969 von einem IBM-Team um Charles F. Goldfarb entwickelt [GP00, S.8]. Die GML-Entwickler Goldfarb, Mosher und Lorie waren von IBM angehalten worden, eine Beschreibungssprache für juristische Dokumente, also vorwiegend textlastige Daten zu erstellen. Die Ausgangslage der GML-Entwicklung wurde von zwei Zielsetzungen bestimmt: erstens von dem Wunsch, Dokumente nicht nur schreiben und lesen, sondern sie auch automatisiert speichern, auffinden, verwalten und veröffentlichen zu können. Zweitens sollte durch die Entwicklung eines strukturierten *generalized markup* die Kommunikationsfähigkeit der textverarbeitenden IBM-Systeme untereinander optimiert werden. Dokumenteninhalte waren in diesen Systemen bislang fest an eine Repräsentationsform gebunden gewesen. Die Verwendung jeweils unterschiedlicher Kommandosprachen zur Darstellung der Daten verursachte massive Probleme bei der Lesbarkeit systemfremder Dateien. In dieser Situation fand für das Team um Goldfarb eines der wesentlichsten Konzepte der Markuptechnologie überhaupt Anwendung: die Trennung des Dokumenteninhalts von dessen Darstellung [EF00a]. Goldfarb griff mit der GML zwei Ansätze auf, die bereits zwei Jahre zuvor von Tunnicliffe und Rice formuliert worden waren und zusammen das Grundkonzept des *generic coding* bilden. So geht die Idee der Trennung von Struktur und Layout auf einen Vorschlag von William Tunnicliffe zurück, der die bewusste Abkehr vom *specific coding* markierte:

Historically, electronic manuscripts contained control codes or macros that caused the document to be formatted in a particular way ("specific coding"). In contrast, generic coding, which began in the late 1960s, uses descriptive tags (for example, "heading", rather than "format-17"). Many credit the start of the generic coding movement to a presentation made by William Tunnicliffe, chairman of the Graphic Communications Association (GCA) Composition Committee, during a meeting at the Canadian Government Printing Office in September 1967: his topic -- the separation of information content of documents from their format.

[Go190, S. 567]

Weiterhin beeinflusst durch das Konzept der *editorial structure tags* des New Yorker Buchdesigners Stanley Rice baute Goldfarb das resultierende Markup-Vokabular aus einer Kombination in sich verschachtelter Elemente auf [Lox98]. Damit wurde die GML die erste Auszeichnungssprache, die einen formal definierten Dokumententyp mit explizit geschachtelter Struktur vorschrieb.

3.2 Standard Generalized Markup Language (SGML)

Die durch Goldfarb weitergeführte Entwicklung der GML mündete in die Entwicklung eines Standards für Auszeichnungssprachen, der nach mehreren Normentwürfen - betreut durch das American National Standards Institute (ANSI) und die International Organisation of Standardization (ISO) - im Jahr 1986 als Standard Generalized Markup Language (SGML) zum Standard erhoben wurde [ISO86]. Das Konzept einer festen Document Type Definition (DTD) wurde integraler Bestandteil des SGML-Standards; so muss ein vollständiges SGML-Dokument aus der SGML-Deklaration, einer eingebetteten oder externen Dokumententyp-Definition (DTD) und einem annotierten Text (Dokumenteninstanz) bestehen [Lob00, S.45].

Aus den technischen Möglichkeiten der SGML ergab sich eine Vielzahl von teils sehr umfangreichen und komplexen Anwendungen. SGML dient als eine Metasprache für die Beschreibung von Markup-Sprachen. Der Annotation von Texten beliebiger fachlicher Richtung und Zweckdienlichkeit wurde damit Tor und Tür geöffnet. North und Hermans schätzen die Anzahl der auf SGML beruhenden Markup-Sprachen auf mehrere hundert [NH00, S. 20f.]. Viele dieser Markup-Sprachen wurden nach den Namen, Aufgaben oder der Nutzung ihrer Document Type Definitions benannt. Allein die Text Encoding Initiative (TEI), eine der relevantesten DTDs für die Geisteswissenschaften auf SGML-Basis, ist derart umfangreich, dass den TEI-Initiatoren die Notwendigkeit von Selbstregulierungsmaßnahmen in Form vereinfachter DTDs bald bewusst wurde. Um die praktische Anwendbarkeit der Text Encoding Initiative für das WWW zu erleichtern, führte das TEI-Konsortium mit „TEI-Lite“ Mitte der Neunzigerjahre ein Subset der TEI-DTD ein. TEI-Lite ist für den Routineeinsatz beim Markup vorgesehen und enthält nur die dafür wichtigsten *tags* der Full TEI. Eine ausführliche strukturelle Analyse von TEI-Lite findet sich bei Megginson [Meg98, S.62ff].

3.3 Extensible Markup Language (XML)

Anwendungen der SGML erreichten in der Praxis mitunter so große Komplexitätsgrade, dass deren Verarbeitung und Anzeige v.a. in Webanwendungen mit großem Aufwand verbunden war. Aus dieser Situation entstand Bedarf für eine zweckmäßig Teilmenge der SGML, die speziell auf die Anforderungen und Möglichkeiten des Web zugeschnitten ist. Dazu zählen u.a. die Unterstützung für eine breites Spektrum von Anwendungen und das Ziel, die Entwicklung von interpretierenden Systemen so weit als möglich einfach zu halten. Als weiteres Entwurfsziel galt eine leichte Lesbarkeit des Markups für den Menschen¹⁵.

Zielsetzung der zuständigen Arbeitsgruppe des World Wide Web Consortiums (W3C) war ein Standard, der den einfachen Datenaustausch mittels HTML mit den Konzepten des generischen Codings der SGML verbindet. Die Extensible Markup Language (XML) entstand:

The Extensible Markup Language (XML) is a subset of SGML [...]. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. [W3C00]

Anders als die HTML, die lediglich eine Modell-DTD der SGML darstellt, sollte XML laut dem W3C eine echte Teilmenge der SGML werden. Als maßgebliches Kennzeichen dafür *muss* ein XML-Dokument eine der SGML nachempfundene Deklaration aufweisen sowie grundlegende Syntaxanforderungen erfüllen. Soll ein XML-Dokument darüber hinaus ein valides SGML-Dokument darstellen - und nur dann - muss es auch zwingend über eine DTD verfügen. Von diesem Fall abgesehen schreibt das W3C die Verwendung einer Document Type Definition nicht explizit vor. Vielmehr unterscheidet die XML-Spezifikation zwischen den Regelmengen der (obligatorischen) *Wohlgeformtheit* und der (fakultativen) *Gültigkeit* eines XML-Dokumentes, die sich jeweils durch das Vorkommen einer DTD unterscheiden. Theoretisch ist die Verwendung von Markup ohne DTD denkbar. In diesem Fall geschieht die

¹⁵ Bach [Bac00, S.30f] sieht die Integration des ‚Faktors Mensch‘ weniger in der ideellen Nähe generischen Markups zum Publikationswesen begründet als in einer Datensicherung gegen die begrenzten Lebenszyklen von Softwaresystemen. Der Einsatz von Metadaten als Maßnahme für eine dauerhafte, systemneutrale Archivierung verdeutlicht die Annäherung der Markup-Technologie an klassische Datenmodelle und die Datenbanktechnologie. Der spätere Einsatz von XML Schema hat dies noch offensichtlicher werden lassen. (vgl. Kap. 3.6)

Auszeichnung informell, d.h. durch das einfache Vorhandensein und durch die Anordnung der Elemente dort, wo sie erzeugt werden [Dün98].

In der Praxis empfiehlt sich laut Dönhöler [Dün98] jedoch der Einsatz von XML mit Document Type Definitions, da XML-verarbeitende Anwendungen wie Browser dadurch weniger belastet werden: „*Wenn ein Parser für seine Arbeit mit dem Dokument nicht die Informationen aus einer DTD nutzen kann [...], dann ermittelt er die Dokument-Struktur während er das Dokument einliest.*“ Darüber hinaus erleichtert eine vorherige Partition von Elementenstrukturen den Einsatz von Such- und Sortierverfahren, weil Instanzdokumente nicht immer vollständig geparkt werden müssen.

Auch wenn der Einsatz von Strukturgrammatiken mit der Extensible Markup Language formal liberalisiert wurde, scheint ihr Einsatz weitgehend obligatorisch geblieben zu sein. Es lohnt sich, diesen trivial erscheinenden Umstand näher zu betrachten.

3.4 Von Metadaten zu „Metainformation“

Das Verhältnis zwischen Aufwand und Nutzen der DTD-Entwicklung wurde offenbar durch die o.g. Vorteile der Dokumentenvalidierung gerechtfertigt, solange tendenziell *statische* Daten - etwa Texte - mit Markup versehen wurden. Aufwändige Konsistenzanforderungen an DTDs und Instanzdokumente entstehen aber weder durch den Textgegenstand selbst noch durch die (aus einem natürlichsprachlichen Paradigma abgeleiteten) Metadaten zu dessen Auszeichnung. Vielmehr sind es Metainformationen *über* Texte, deren semantischer Gehalt die Darstellungsmöglichkeiten von Document Type Definitions sprengt.

Es hat sich ein Wandel weg vom linearen ‚Lesen‘ eines Dokumentes hin zum ‚Verstehen‘ eines realweltlichen Phänomens vollzogen. Diese Entwicklung kann als eine Perspektivenverschiebung weg vom textuellen Objekt hin zur subjektiv konstruierten Deutung desselben verstanden werden. Die darzustellende Dimension hat sich damit verlagert und ausgedehnt – sie involviert abweichende Deutungen eines Gegenstandes, die Kategorisierung von Gegenständen, die Modellierung ihrer Beziehungen und vieles mehr.

Der Versuch, die Ebene von Metainformationen adäquat und kommunizierbar darzustellen, verlangt daher nach einem umfangreicheren, komplexeren und flexibleren System zur Datenmodellierung. Die jüngere historische Entwicklung hin zur Nutzung von XML als Datenaustauschformat hat der Realisation einer solchen Datenmodellierungssprache Vorschub geleistet. Im Folgenden soll der Stellenwert von Document Type Definitions vor diesem Hintergrund nachgezeichnet und ein Vergleich mit XML Schema in seiner Rolle als Datenmodell gezogen werden.

3.5 Document Type Definitions als Datenmodell

Das Erbe, welches SGML den ihr nachfolgenden Markup-Technologien HTML und XML hinterläßt, stellt nur unzureichend implementierte Mittel zur Datenmodellierung zur Verfügung. Diese Unzulänglichkeiten wurden nicht innerhalb des klassischen Nutzungsrahmens von DTDs offenbar, sondern zeigten sich in der Verwendung von XML-Strukturen zum Datenaustausch. Harold [Har01, p.827] erklärt dies mit der Schlüsselintention der SGML-Entwickler, Dokumente in narrativem und für den menschlichen Leser verständlichem Stil zu beschreiben. Wie oben gezeigt wurde, waren frühe Markup-Bestrebungen - siehe auch die Aufgabenstellung, der sich Goldfarb gegenüber sah - auf die Auszeichnung primär textlicher Dokumente wie Bücher, Artikel oder technischer Handbücher ausgerichtet. DTDs waren für den Einsatz mit solchen Dokumenten vorgesehen und erfüllten hierfür ihren Zweck. Der Paradigmenwechsel weg vom linearen ‚Lesen‘ eines Dokumentes hin zu einer *selektiven* Verarbeitung von Daten zeigte bald die Grenzen von DTDs auf. Dazu zählen laut Harold die nahezu fehlende Datentypisierung, die vom XML-Standard abweichende Syntax sowie die umständliche automatisierte Erweiterung [Har01, p.827f].

Als Argumente für eine Typisierung von Daten können ähnliche Gründe vorgebracht werden wie die für die Verwendung von DTDs selbst; eine Vorab-Typisierung erleichtert die Applikationsprogrammierung insofern, als Validierungsarbeit bereits durch das Datenmodell geleistet wird. Weiterhin können Daten gezielter codiert und somit auch gezielter aufgefunden und verarbeitet werden. DTDs beinhalten implizit eine schwache Form der Typisierung, da sie zumindest das *Vorkommen* von Elementen in ihrem *declaration subset*¹⁶ steuern können. Explizit stellen XML-DTDs jedoch nur die Basisdatentypen *CDATA* und *PCDATA* zur Verfügung. *CDATA* (Character Data) kodieren Verarbeitungsanweisungen, die zwar im Dokumententext enthalten, von einem XML-Parser aber ignoriert werden. *PCDATA* (Parseable Character Data) bezeichnen Zeichendaten, die als kleinste atomare Einheit einer Dokumentenstruktur fungieren. Beiden Datentypen wird i.d.R. das Pfundzeichen (#) vorangestellt, um für den Parser die Deutung als Schlüsselwort und nicht als weiteren Elementnamen zu verdeutlichen. Beiden Typen ist gemeinsam, dass sie nur Zeichenketten oder weitere Unterelemente aufnehmen können - es ist nicht möglich, den Elementinhalt auf andere Datentypen wie etwa den Typ *integer* zu beschränken. Auch inhaltliche Vorgaben für Elemente wie Wertebereiche oder reguläre Ausdrücken sind mit DTDs nicht realisierbar [Bor01, S. 176]. Einschränkend muss erwähnt werden, dass XML und SGML grundsätzlich

¹⁶ Lobin benutzt den Begriff des *declaration subset*, der die Untermenge aller Elementen-, Attribut- Notations- und Entitydeklarationen in einer DTD bezeichnet [Lob00, S. 47].

eine Datentypisierung ermöglichen. Lobin erläutert in diesem Zusammenhang, dass die Definition weiterer Datentypen neben PCDATA realisierbar ist [Lob00, S 32.ff]. Dazu müssen Datentypen zuerst über Notationen in der DTD deklariert werden:

```
<!NOTATION GIF PUBLIC
    „+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION
    CompuServe Graphic Intechange Format//EN“>
```

Beispiel: Anlegen einer Notation des Typs GIF

Eine solche Notation stellt Instanzdokumenten einen neuen Datentyp namens GIF zur Verfügung. Das Schlüsselwort PUBLIC macht die Notation anderen Dokumenten allgemein zugänglich. Notationen müssen innerhalb der DTD in Form von Entities so referenziert werden:

```
<!ENTITY bild_xy SYSTEM "bild_xy.gif" NDATA GIF>
```

Beispiel: Referenzierung einer Notation

Erst dann stehen Bilddaten mit der Bezeichnung `bild_xy` im `.gif` - Format einem Instanzdokument zur Verfügung. Zweifellos ist diese Art der Typisierung umständlich. In der Praxis ist die Konsistenz eines solchen Vorgehens außerdem von (einzuhaltenden) Absprachen abhängig, da DTDs den Inhalt einer Entity namens GIF nicht überprüfen kann. Das W3C hat infolgedessen Ansätze zur Diskussion gestellt, die die Unterstützung von Datentypen für DTDs gewährleisten. Die im Januar 2000 eingereichte Anmerkung (Note) des W3C stellt mit der Spezifikation *Datatypes for DTDs (DT4DTD)* 1.0 einen solchen Ansatz vor [W3C00a]. Aus dem Abstract der W3C-Anmerkung geht jedoch hervor, dass DT4DTD als Brückentechnologie konzipiert und bereits auf die Verwendung mit XML Schema ausgerichtet ist:

The presented specification allows legacy systems that may presently be unable to convert their DTD markup declarations to XML Schema, to utilize XML Schema conformant datatypes. With it, DTD creators can specify datatypes for attribute values and data content, thereby providing the foundation for a smoother future transition path.

[W3C00a]

Mit der W3C-Recommendation von XML Schema wurde im Mai 2001 auf diese und andere Unzulänglichkeiten von Document Type Definitions reagiert.

3.6 XML Schema als Datenmodell

Studiert man die XML Schema-Spezifikation des W3C [W3C00b], fallen zwei Dinge schnell ins Auge. Erstens: Der Standard ist so umfangreich, dass er in zwei Teile zerlegt wurde. XML Schema Part 1 dokumentiert die strukturellen Aspekten von XML Schema, während sich XML Schema Part 2 ausschließlich der Datentypisierung widmet.

Zweitens: Die Spezifikation ist stark von Konzepten und Termini aus der objektorientierten Programmierung und der Datenbanktechnologie durchdrungen. Etwa zwei Jahre vor der Verabschiedung von XML Schema als Recommendation verfasste das W3C einen allgemeinen Anforderungskatalog an die beiden Teile des prospektiven W3C-Standards [W3C99]. Demnach solle die XML Schema-Sprache:

- ausdrückstärker als XML DTDs sein
- selbst in XML geschrieben sein
- selbstbeschreibend sein
- von XML-verarbeitenden Anwendungen verstanden werden
- leicht im Internet einsetzbar sein
- für Interoperabilität optimiert sein
- mit verwandten W3C-Spezifikationen kompatibel sein, darunter XML Links, Namespaces, Pointers, sowie dem DOM, HTML, und RDF Schema

Um diese Anforderungen zu erfüllen, bediente sich das W3C bei einer Reihe langjährig bewährter Ansätze aus verschiedenen Bereichen der Informatik. Wesentlicher Bestandteil von XML Schema ist die Typisierung von Elementen und Attributen, die der Modellierung von Objekten in höheren Programmiersprachen ähnelt. So können Inhaltsmodelle als einfache und komplexe Typen bzw. Kombinationen aus beiden realisiert werden, als Gruppen im Sinne von ‚Containern‘ zusammengefasst und ganz oder teilweise an andere Schemata vererbt werden.

Weiterhin können minimale und maximale Häufigkeiten von Elementen festgelegt, Wertebereiche von Elementen eingegrenzt oder die Eindeutigkeit von Attributen und Elementen über Namensräume erzwungen werden.

Im Bereich der Datentypisierung orientiert sich XML Schema stark an den gängigen Datentypen von C++/Java und SQL. Primitive Datentypen wie `string`, `byte`, `date`,

`integer` und `sequence` machen jedoch nur einen Bruchteil der 44 Datentypen in XML Schema aus. Insgesamt bietet XML Schema Datentypen für jeden erdenklichen Zweck, darunter eine große Zahl numerischer und lexikalischer Datentypen, datums- und zeitbezogene Typen, Token, ID-Typen und eigene Datentypen für WWW-Ressourcen. Um die Validierungsmöglichkeiten durch Datentypen zu maximieren, können diese außerdem mittels *Facetten* auf bestimmte Wertebereiche („scope“) oder Muster („patterns“) eingegrenzt werden.

Selbst Raum für noch unbekannte Erweiterungen von Inhaltsmodellen kann in XML Schema reserviert werden. Dies geschieht mit Hilfe der sog. *wildcards* `<xs:any>` und `<xs:anyAttribute>`, die das Auftreten bestimmter Elemente bzw. Attribute aus spezifizierten Namensräumen in ein bestehendes Content Model erlauben. Wildcards sollten eine bestmögliche Balance zwischen der Offenheit des Content Models und der Präzisierung des zu instanzierenden Inhaltes anstreben. Gelingt dies, erlauben *wildcards* eine potenziell unbegrenzte Erweiterbarkeit zusammen mit einem Mindestmaß an Kontrolle. XML Schema nähert sich damit der Idealvorstellung von Datenmodellen schlechthin: der Koexistenz von Flexibilität und Struktur.

Kapitel 4 - Datenmodelle

A message to mapmakers: highways are not painted red,
rivers don't have county lines running down the middle,
and you can't see contour lines on a mountain.

William Kent

4.1 Hintergrund

Nachdem in den letzten Kapiteln die historische Entwicklung von Markup-Technologien sowie Metadatenstandards als deren Darstellungs- und Kommunikationsmedien herausgestellt wurde, widmet sich dieses Kapitel den theoretischen Grundlagen von Datenmodellen. Die theoretische Durchdringung der Datenmodellierung soll hierbei beweisen, dass Markuptechnologien im Allgemeinen und XML im Speziellen auch formal als Datenmodelle bezeichnet werden können. Das Kapitel schließt mit einer Liste von Indikatoren für ein nutzbringendes Datenmodell sowie möglichen Ausprägungen in XML Schema.

Um die Schlüssigkeit der Argumentation zu gewährleisten, bieten sich dazu v.a. Ansätze der Datenmodellierung an, die zeitlich *vor* dem breiten Einsatz von Metadaten und Metadatenstandards angesiedelt sind. Die historische Datierung einer ‚Blütezeit der Metadatenstandards‘ wäre bestenfalls auf einen Zeitraum von fünf bis zehn Jahren

einzugrenzen. Die Verabschiedung des XML 1.0- Standards im Jahre 1998 taugt aufgrund der aufgezeigten Verwandtschaft zwischen XML zu SGML keineswegs als historischer Markstein dieser Entwicklung. Allenfalls die Standardisierung von SGML im Jahre 1986 kann als Startpunkt einer Ära von weithin akzeptierten Metadatenstandards angenommen werden - die Anfänge der Text Encoding Initiative (TEI) ab dem Jahr 1987 sprechen für diese Annahme [TEI01].

Tsichritzis und Lochovsky [TL82] formulieren im Jahre 1982, also deutlich vor Beginn einer ‚Markup-Ära‘, wertvolle Gedanken zur Datenmodellierung und beschreiben zahlreiche Anwendungen für die Datenbanktheorie. Wenige Jahre vor dem Siegeszug des relationalen Datenbankmodells, welches später einen Siegeszug in der Datenverarbeitung antreten wird, gehen Tsichritzis und Lochovsky auch auf Datenmodelle ein, die durch das RDBM verdrängt wurden und weitgehend ins Abseits gerieten. Ihre Grundüberlegungen zu Realität, Semantik, Daten und Datenmodellen sind jedoch ausreichend abstrakt formuliert, um zeitunabhängig im Bereich der Datenspeicherung Anwendung finden zu können. Sie lassen sich sinnvoll auf Markup-Technologien anwenden und unterstreichen damit deren mögliche Ausprägung als Datenbankmodell.

4.2 Datensemantik

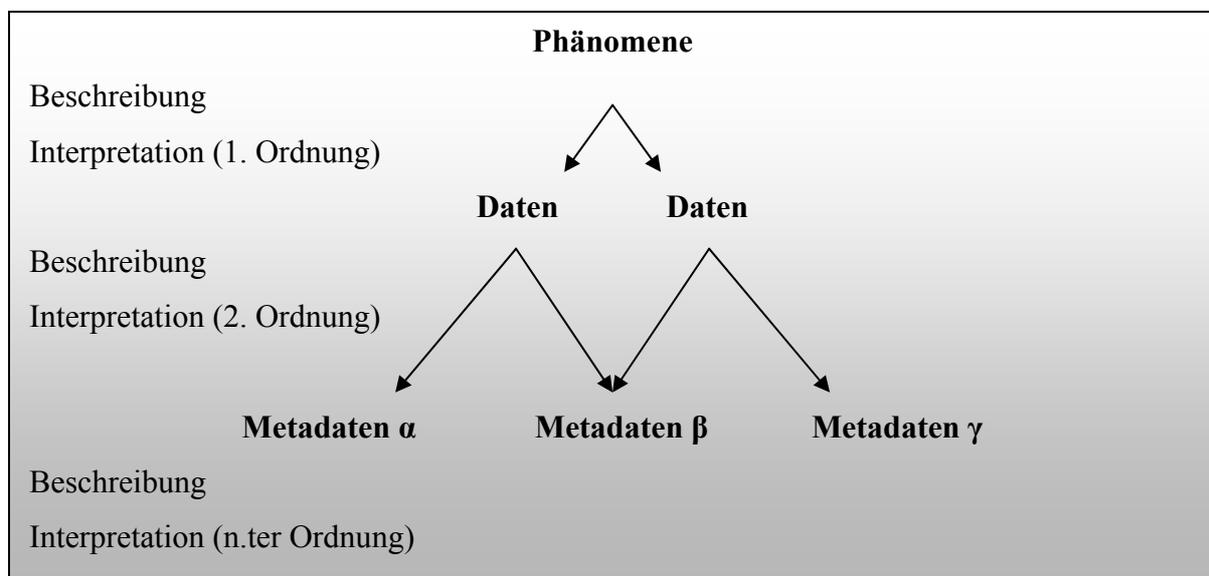
Grundlage eines jeden Datenmodells sowie von Daten als deren atomare Einheiten ist die Beschreibung von Phänomenen der äußeren Welt. Tsichritzis und Lochovsky interpretieren Wahrnehmung als eine Serie von Beschreibungen verschiedenartiger, ggf. verbundener Phänomene: „*A perception of the world can be regarded as a series of distinct although sometimes related phenomena*“ [TL82, S. 3]. Beschreibungen von Phänomenen, so unvollständig oder unverstanden sie auch sein mögen, werden in diesem Zusammenhang von den Autoren als *Daten* bezeichnet. Diese beiden Aussagen implizieren, dass die Semantik von Daten auf Wahrnehmung beruht. Seit Menschengedenken existiert das Bestreben, die Umwelt möglichst vollständig und verständlich zu beschreiben. Die Ausprägungen dieser Beschreibungen - aufgezeichnete Daten - sind dabei in dem Maße variant und von räumlicher, zeitlicher und kultureller Prägung abhängig, wie es die Wahrnehmung selbst ist. Die wörtliche Bedeutung des lateinischen Begriffs *datum* bezeichnet eine Tatsache (hier: „fact“), ein konkretes und im besten Fall genau skalierbares Ereignis. Tsichritzis et al. weisen auf die Unschärfe dieses Datenbegriffs hin. Die Beschreibung eines Phänomens zeichnet in der Regel nur ein unvollständiges Bild einer „Tat-Sache“ oder bemisst Vollständigkeit an der gewählten Perspektive des Beobachters. Weiterhin kann es sich bei Phänomenen auch lediglich um

Konzepte, etwa bei der Formulierung von Ideen handeln. Als maßgeblich für ihre theoretische Betrachtung der Datenmodellierung schränken die Autoren den Datenbegriff deshalb auf aufgezeichnete Daten ein: “[...] *data correspond to descriptions of any phenomenon or idea that a person considered worth formulating and recording*“ [TL82, S.3]. Diese Einschränkung des Datenbegriffs ist einerseits notwendig, um ihn verständlich und diskussionsfähig zu machen, betont andererseits die Rolle des Wahrnehmenden, des ‚Interpreten‘. Aus dieser Definition leitet sich weiterhin ab, dass Daten erst durch Interpretation ihren faktischen Charakter gewinnen. Die Möglichkeit, Interpretationen zusammen mit Daten abzubilden, ist demnach ein Indikator für ein Datenmodell, das ein umfassenderes Konzept eines Phänomens darstellen kann. Generisches Markup ist in besonderem Maße dazu in der Lage, wie später gezeigt wird..

Indikatoren für ein nutzbringendes Datenmodell

1) Daten und Interpretation können zusammen abgebildet werden.

Angewandt auf den Bereich Metadaten ließe sich Folgendes aus dem bisher Gesagten schlussfolgern: Metadaten sind Beschreibungen von Daten und stellen somit eine Interpretation zweiter Ordnung der Welt dar. Sie korrespondieren, um Tschirzits' Wortwahl zu adaptieren, mit einer Serie von Wahrnehmungen verschiedener, ggf. verbundener *Daten*:



In obiger Grafik wird diese Aussage visualisiert. Phänomene werden als Daten codiert. Es wurde gesagt, dass bei diesem Vorgang Interpretation involviert ist. Daten können mittels Metadaten ihrerseits beschrieben (und somit interpretiert) werden. Der Tatsache, dass heterogene Daten auch durch gleiche Metadaten beschrieben werden können - dies ist z.B.

Sinn und Zweck eines Metadatenstandards - wird durch *Metadaten* β Rechnung getragen. Die Praxis hat gezeigt (vgl. Kap. 1.2), dass die Beschreibung heterogener Daten durch universelle Metadatenvokabulare problematisch ist. Der Kreislauf aus Abstraktions- und Diversifikationsprozessen, der oben skizziert wurde, kann als Instanz dieses Problems betrachtet werden.

Eine Interpretation zweiter Ordnung (von Daten) involviert summarisch mehr subjektive Deutung als die Interpretation erster Ordnung (der Welt). Sie ist damit noch anfälliger für divergierende Sichten einer unterstellten Realität und stellt die Ursache einer Diversifizierung dar. Wie oben gezeigt wurde, wird das Mittel der Abstraktion eingesetzt, um der Diversifizierung entgegenzuwirken und Kommunikation zu gewährleisten. Ein zweiter Indikator eines nutzbringenden Datenmodells ist demnach die Möglichkeit, *abweichende* Interpretationen z.B. von Daten berücksichtigen und speichern zu können.

Indikatoren für ein nutzbringendes Datenmodell

2) Abweichende Interpretationen von Daten können abgebildet werden.

Weiterhin korrespondieren Daten “...to discrete, recorded facts about phenomena from which we gain information about the world.“ An dieser Stelle wird von Tsichritzis und Lochovsky der Begriff der ‚Information‘ eingeführt, der ein „...*increment of knowledge that can be inferred from data*“ darstellt [TL82, S.3]. Information lässt sich analog dazu also aus Daten ableiten und ist damit ebenfalls abhängig von Interpretation - sowohl von der des Datenlieferanten als auch von der subjektiven Interpretation des Wahrnehmenden.

Von Datenmodellen wird erwartet, diesen hohen Grad an Unschärfe abzumildern. Sie sollen Regeln definieren, nach denen Daten strukturiert werden. Diesem Anspruch können Datenmodelle jedoch nur in eingeschränktem Maße gerecht werden. Eine Annäherung an ein umfassendes Datenmodell ist nur bedingt möglich; einige in der Datenmodellierung gängigen Parameter dazu werden weiter unten aufgeführt. Die Unzulänglichkeit der Mittel von Datenmodellen wird auch von Tsichritzis et al. eingeräumt. Nach ihrer Ansicht ist der Rahmen der Datenstrukturierung meist unvollständig und beeinflusst außerdem Art und Weise der Nutzung von Daten:

Structures [...] do not provide complete information about the meaning of data and the way they will be used. Operations which are permitted on the data have to be specified. [TL82, S. 10]

Kent stellt schon Jahre vor Tsichritzis die Unzulänglichkeit von Datenmodellen fest, ohne sich dabei auf real implementierte Datenbanken zu beziehen. Datenmodelle seien nichts weiter als „*poor artificial approximations of some real underlying terrain*“ [Ken78, S. 5]. Auch Kents Ausführungen gehen damit ideell auf Korzybskis' Passus des ‚*The map is not the territory*‘¹⁷ zurück. Bateson [Bat73, S. 455] merkt an, dass die - längst auch in der Informationsverarbeitung verbreitete - Einsicht Korzybskis aus einer über zweitausend Jahre währenden Epoche philosophischen Denkens in Europa resultiert. Ihr liegt dieselbe Beobachterabhängigkeit der Erkenntnis zugrunde, aus der sich der seit Mitte der Achtzigerjahre¹⁸ des 20. Jhds. salonfähig gewordene Begriff des ‚Konstruktivismus‘ ableitet. Es ist eine Frage der Zeit, bis die immer stärker werdende Verbreitung ‚konstruktivistischer‘ Theorien¹⁹ auch den wissenschaftlichen Diskurs in der Informationstechnologie berührt. Als Indikator für die Richtigkeit der Subjektivität von Daten und Information stellt das Entstehen des Konstruktivismus bereits jetzt einen Gewinn für die theoretische Informatik dar. Es bleibt jedoch abzuwarten, ob der Informationstechnologie insgesamt dadurch nennenswerter technischer Fortschritt beschieden ist oder bereits gängige Anwendungen nur im Licht neuer Erkenntnisse erscheinen.

Ausgehend von letzterem dürfte es vor allem die Markuptechnologie sein, die davon berührt sein wird. Dies ist in ihrer Nähe zu natürlichen Sprachen begründet. Laut Kent sind nur natürliche Sprachen in der Lage, Daten in ausreichendem Maße zu beschreiben. Nur sie würden der amorphen, mehrdeutigen und subjektiven Natur ‚realer‘ Information gerecht [Ken78, S.6]. Auch Tsichritzis et al. sehen in der Trennung von Daten und Interpretation durch Sprache ein grundlegendes Problempotenzial bei der Nutzung von Daten. Der Einsatz von Beschreibungsmechanismen, die auf natürlicher Sprache beruhen, kann helfen, dieses Problempotenzial zu minimieren. Die Gründe für die Trennung von Daten und Interpretation sind teilweise historisch bedingt. So sind Rechner zu Beginn der Achtzigerjahre durch mangelnde Verarbeitungsfähigkeit von Sprache weit von der Mächtigkeit natürlicher Sprachen entfernt. Weiterhin werden hohe Speicherkosten als Grund für die Trennung von Daten und Interpretation angegeben [TL82, S. 4]. Beide Gründe haben sich zwischenzeitlich

¹⁷ Korzybskis Theorie konsequent folgend ist selbst Sprache nicht in der Lage, die Wirklichkeit umfassend abzubilden. Diese Aussage wird im Kontext dieses Kapitels jedoch vernachlässigt.

¹⁸ Man beachte die zeitliche Kongruenz mit Tsichritzis' Verständnis von ‚Fakten‘ und deren Interpretation.

¹⁹ Obwohl die Begriffsbildung und Formulierung ‚konstruktivistischer‘ Theorien ihre Wurzeln in den Achtziger und Neunzigerjahren haben, können retrospektiv auch Untersuchungen aus den 1960er Jahren eindeutig dazugezählt werden (v.a. die Werke von Heinz von Förster).

als hinfällig erwiesen. Speicherkosten sind drastisch gesunken, die Verarbeitung von (textlich codierter) Sprache ist durch den Einsatz von Parsern zumindest innerhalb einer gegebenen Domäne erheblich weiterentwickelt worden.

Tsichritzis et al. kommen ebenfalls zu dem Schluss, dass nur natürliche Sprachen genügend Flexibilität bieten, um Daten hinreichend zu beschreiben. Damit postuliert Tsichritzis' Standpunkt eine Verbindung von Sprache und Daten, wie sie dem Bereich des semantischen Markup zu eigen ist.

Indikatoren für ein nutzbringendes Datenmodell

3) Interpretationen von Daten können innerhalb eines natürlichsprachlichen Paradigmas abgebildet werden.

Metadaten ermöglichen insofern eine Annäherung an natürliche Sprachen, als sie Daten und deren Interpretation in einer gegebenen Struktur („Grammatik“) und mit Hilfe eines festgelegten Wortschatzes („Vokabular“) speichern können. Sie können damit den Anspruch der Nähe zu natürlichen Sprachen erfüllen.

4.3 Flexibilität von Daten

Flexibilität in der Interpretation von Daten ist notwendig, um den sich entwickelnden Aspekt der Welt gebührend zu berücksichtigen. Gleichzeitig muss ein Datenmodell jedoch eine stabile und praktisch nutzbare Datenbasis zur Verfügung stellen können:

It is apparent that an interpretation of the world is needed which is sufficiently abstract to allow minor perturbations, yet is sufficiently powerful to give some understanding concerning how data about the world are related.

[TL82, S.5]

Die Struktur eines nutzbringenden Datenmodells soll also sowohl Statik als auch Veränderung handhaben können. Semantisches Markup gewährleistet durch seine verschachtelte Struktur und die leichte Erweiterbarkeit beides. XML Schema ist stark modular konzipiert und somit in besonderem Maße dazu geeignet, diesen Anspruch an ein Datenmodell zu erfüllen. Tsichritzis et al. konkretisieren ihre Forderung nach Flexibilität in der Interpretation von Daten folgendermaßen. Daten sollen:

1. aus *verschiedenen* Perspektiven betrachtet werden können. Dazu müssen sie ausreichend abstrakt codiert sein. Sie sollen außerdem

2. auf *gleiche* Weise betrachtet werden können. Auch dazu ist ein ausreichender Abstraktionsgrad notwendig [TL82, S.5].

Die Autoren führen hierzu das Beispiel einer Personaldatenbank an. Die gespeicherten Daten über Mitarbeiter werden von verschiedenen Applikationen genutzt und werden analog zum Zweck dieser Anwendung interpretiert. So werden Mitarbeiterdaten von der Software eines Steuerbüros als Steuerpflichtige interpretiert, von einer Software für das Gesundheitswesen als Patienten usw. Verschiedene Interpretationen desselben Datenbestands sollen durch ein zugrundeliegendes Datenmodell ermöglicht werden. Gleichzeitig sollen die Daten so codiert sein, dass alle Mitarbeiter unabhängig von ihrer Funktion und Position als Angehörige desselben Unternehmens identifiziert werden können.

Indikatoren für ein nutzbringendes Datenmodell

4) Daten sollen in einem Datenmodell ausreichend abstrakt codiert werden können.

Übertragen auf den Bereich der Geisteswissenschaften ist folgendes Szenario denkbar: Ein Forscher im Bereich der Informationsverarbeitung erhält täglich eine Vielzahl von Postings diverser wissenschaftlicher Newsgroups. Die empfangenen E-Mails sind aus jeweils unterschiedlichen Perspektiven interessant für seine Forschung, da sie immer nur Teilbereiche seines Interesses betreffen. Indessen soll von einem Systemadministrator zu statistischen Zwecken der Durchsatz von E-Mails pro Mailbox aller Universitätsangehörigen ermittelt werden. In diesem trivialen Beispiel wird von einem Datenmodell erwartet, sowohl Interpretationen von atomaren Einheiten des Datenbestandes zuzulassen, als auch den Datenbestand in seiner Gesamtheit interpretieren zu können.

Um möglichst viele der in diesem Kapitel dargestellten theoretischen Sachverhalte zusammenfassend in einem Beispiel zu behandeln, wird die Implementation eines Datenmodells als XML Schema an den Schluss dieses Kapitels gestellt.

4.4 Vier Aspekte von Daten

Tsichritzis et al. definieren ein Datenmodell als ein intellektuelles Werkzeug, das eine Interpretation von Daten erlaubt: *“It is a model about data by which reasonable interpretation of data can be obtained”* [TL82, S.5]. Vollständiges Wissen wird in diesem Zusammenhang als *“open ended“* bezeichnet. Es wurde gezeigt, dass ein allumfassendes Datenmodell nicht realisierbar ist. Die Annäherung an ein allumfassendes Datenmodell kann also nur darin

bestehen, ihm die Nicht-Endlichkeit von Wissen so gut als möglich zu eigen zu machen. Auch diese lange Reise beginnt mit einem ersten Schritt. In der Datenmodellierung bestand dieser darin, zunächst elementare Objekte zu bestimmen und diese zu beschreiben. In Anlehnung an Langefors [Lan77, S. 207ff] werden hierzu vier Aspekte von Daten zu einem Tupel zusammengefasst:

`<object name, object property, property value, time>`

Tsichritzis und Lochovsky sehen in diesem Tupel eine mögliche Grundeinheit vieler Datenmodelle, klammern den Faktor Zeit jedoch aufgrund seines absoluten Charakters aus. Folgende Liste zeigt die vier Aspekte von Daten und ihre mögliche Entsprechung in Markupsprachen:

- | | |
|-------------------|---|
| · object name | Element |
| · object property | Attribut |
| · property value | Attributwert |
| · (time | z.B. "dateTime" - Datentyp in XML Schema) |

Dies soll verdeutlichen, dass semantisches Markup den früh formulierten Grundkonzepten der Datenmodellierung vollauf genügt. Es ist vielmehr davon auszugehen, dass die Syntax-Entwicklung des generischen Markups mehr oder weniger bewusst von den o.g. Ansätzen der Datenmodellierung beeinflusst wurde.

4.5 Kategorisierung

Ein weiteres ausdrucks mächtiges Werkzeug der Datenmodellierung ist die Kategorisierung. Dabei wird davon ausgegangen, dass Daten, die derselben Kategorie angehören, Ähnlichkeiten aufweisen. Ihre mithin populärste Ausprägung erfährt diese Annahme in den *entities* des relationalen Datenbankmodells. In den später nachfolgenden objektorientierten Datenbanken ist sie durch die Gruppierung in Klassen ebenfalls gegenwärtig. Eine Entsprechung der Kategorisierung in der Markuptechnologie findet sich in der Verschachtelung (*nesting*) von Informationseinheiten. Die verschachtelte Struktur von Elementen ist geeignet, Zugehörigkeiten zu Kategorien ähnlich abzubilden, wie Tabellen dies in relationalen Datenbanken tun. Direkt untergeordnete Elemente entsprechen dabei den Spalten einer Tabelle, welche durch das übergeordnete Element repräsentiert ist. Die *records*, die dieser Tabelle zugehörig sind, werden in XML durch die Daten innerhalb der

untergeordneten Elemente repräsentiert. Obwohl durch die Tiefe des *nestings* in XML-Strukturen die Reichweite eines streng relationalen Datenmodells gesprengt wird, ist das korrekte *nesting* Ansatzpunkt für das Abbilden („Mapping“) von Tabellenrelationen auf XML-Strukturen²⁰ bzw. umgekehrt.

Das Maß, in dem bei der Zuordnung von Daten auf einer Kategorisierung beharrt wird, führt laut Tschritzis et al. zu zwei Klassen von Datenmodellen:

- *strictly typed data models*: Jedes Datum *muss* zu einer Kategorie gehören. Daten, bei denen dieses nicht der Fall ist, müssen entweder abgewandelt werden oder können nicht in das Datenmodell aufgenommen werden. Weiterhin gehen einige dieser Datenmodelle davon aus, dass Kategorien nicht dynamisch erweitert werden können.
- *loosely typed data models*: Kategorien werden nur in dem Maße eingesetzt, in dem sie nützlich sind. Daten können isoliert existieren, ohne in eine Kategorie zu fallen. [TL82]

XML Schema steht gemäß dieser Definition einem *loosely typed data model* am nächsten. Zwar können durch die starke Anwendung einer Datentypisierung Kategorisierungen im Sinne eines *strictly typed data models* erzwungen werden - die historischen Wurzeln der XML Schemata liegen jedoch trotz aller logischen Nähe zu klassischen Datenmodellen im Bereich generischen Markups²¹. Auch Kazakos et al. sehen in der Nutzung XML-basierter Standards als Datenmodell eine späte Renaissance. So sei die XML „*ursprünglich als Metasprache für Dokumentenauszeichnungssprachen und nicht als Datenmodell konzipiert worden. Erst als XML massiv für datenorientierte Anwendungen eingesetzt wurde, wurde eine formellere Definition eines XML-Datenmodells ‚nachgerüstet‘.*“ [KST02, S.31] Das XML Information Set [W3C01], auf das sich die Autoren mit dieser Aussage beziehen, wurde erst im Oktober 2001 als W3C-Empfehlung verabschiedet - also zu einem Zeitpunkt, zu dem XML als Metasprache längst Verbreitung gefunden hatte.

²⁰ Kazakos et al. [KST02] liefern einen Band, der die konzeptuellen Gemeinsamkeiten von XML und Datenbanken vor dem Hintergrund ihrer Anwendung auch *jenseits* von XML als reinem Datenaustauschformat beleuchtet.

²¹ Eine Behauptung, der von einigen Vertretern der Datenbanktechnologie womöglich widersprochen würde.

4.6 Semistrukturierte Datenmodelle

Parallel zu der Entwicklung einer solchen Zwitterexistenz zwischen den Extremen eines streng strukturierten Datenmodells einerseits und vollkommen unstrukturierten Texten andererseits werden seit Mitte der Neunzigerjahre sog. ‚semistrukturierte Datenmodelle‘ diskutiert. Die ihnen zugrunde liegenden Konzepte gelten als maßgeblicher Einfluss für die heute existierenden XML-Standards [KST02,S.26]. Die Arbeiten von Abiteboul haben sich ausführlich mit den Charakteristika semistrukturierter Datenmodelle (hier v.a. [ABS99]) sowie der assoziierten Problematik einer Datenabfragesprache für solche Modelle [Abi97, S. 1-18] beschäftigt. Nach Abiteboul zeichnen sich semistrukturierte Datenmodelle dadurch aus, dass sie zwar eine Struktur besitzen aber:

- diese Struktur Variationen aufweisen kann
- die Struktur implizit in die Daten eingebettet ist und erst *a posteriori*, d.h. nach Vorliegen der Daten, angegeben werden kann
- neu eingetroffene Daten nicht der bislang entstandenen Struktur entsprechen müssen
- die Strukturen nur partiell gültig sind, d.h. ein Teil der Daten wird nicht durch sie strukturiert

[KST02, S.28]

Kazakos et al. führen das Object Exchange Model (OEM) der Stanford University als prominentes Beispiel für die Einfachheit semistrukturierter Datenmodelle an und sehen in dem 1995 durchgeführte Projekt einen der Vorläufer der XML [KST02, S.28]. Ohne hier näher auf die genaue Syntax des OEM einzugehen, zeigen sich auch bei diesem semistrukturierten Datenmodell die Grundelemente `<Objekt, beschreibender Bezeichner, Wert>`, die voll und ganz in der Tradition der von Langefors formulierten Datenaspekte stehen. Den Faktor ‚Zeit‘ außer acht lassend, wird das OEM um einige einfache Datentypen (`int`, `real`, `string`) sowie einen komplexen Typ `set`, ähnlich den *complex types* in XML Schema, bereichert [KST02, S.29]. Anhand dieser Parallelen wird deutlich, dass auch semistrukturierte Datenmodelle den schon früh entstandenen Konzepten der Datenmodellierung folgen. Die Möglichkeit der Kategorisierung bzw. Strukturierung von Daten kann, unabhängig von ihrem Strukturierungsgrad, also als ein weiterer Indikator für ein Datenmodell angeführt werden.

Indikatoren für ein nutzbringendes Datenmodell

5) Ähnlichkeiten können durch eine Kategorisierung von Daten abgebildet werden.

4.7 Zusammenfassung und Anwendung

Anhand von grundlegenden Konzepten der Datenmodellierung wurden in diesem Kapitel fünf Indikatoren für ein nützliches Datenmodell ausgemacht:

- (1) Daten und Interpretation können zusammen abgebildet werden.
- (2) Abweichende Interpretationen von Daten können abgebildet werden.
- (3) Interpretationen von Daten können innerhalb eines natürlichsprachlichen Paradigmas abgebildet werden.
- (4) Daten müssen in einem Datenmodell ausreichend abstrakt kodiert werden können.
- (5) Ähnlichkeiten können durch eine Kategorisierung von Daten abgebildet werden.

In den zurückliegenden Kapiteln wurde stellvertretend für heute übliche Markupssprachen wie XML gezeigt, dass deren Syntax diesen Indikatoren weitestgehend folgt. Der Aspekt der gemeinsamen Speicherung von Inhalt und Interpretation von Daten (1) ist durch die Verwendung deskriptiver *tags* so essentiell in der Markuptechnologie, dass hier nicht weiter darauf eingegangen wird. Die Nähe der XML-Syntax zu natürlichen Sprachen (3) wird durch den ‚sprechenden‘ Charakter von *tags* ebenfalls deutlich. Verschiedene Interpretationen von Daten (2) können so als Objekteigenschaften codiert und mit Hilfe von Attributen dargestellt werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<postings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
          xsi:noNamespaceSchemaLocation="...\postings_schema.xsd">
  <posting id="#106" newsgroup="Humanist Discussion Group" subject="16.386
    Museums and the Web 2003" timestamp="2002-12-14T06:07:29">
    Museums and the Web 2003(...)
  </posting>
</postings>
```

Analog zu dem in Kap. 4.3 beschriebenen Szenario ist diese einfache XML-Struktur in der Lage, sowohl für den Forscher als auch für den Systemadministrator relevante Interpretationen desselben Datums abzubilden. Das Attribut ‚id‘ drückt eine Interpretation des Eintrags als laufende Nummer eines Index‘ aus, ‚newsgroup‘ und ‚subject‘ beruhen

auf der intellektuellen Interpretation des Posting-Inhaltes. Das Attribut 'timestamp' ist ggf. aus mehreren Perspektiven interpretierbar. Da XML Schema selbst der XML-Syntax folgt, aber auch von ‚Markup-fremden‘ Konzepten z.B. aus der Datenbanktechnologie beeinflusst wurde, soll abschließend nur auf die letzten beiden Indikatoren (4), (5) ausführlicher eingegangen werden.

(4) Daten müssen in einem Datenmodell ausreichend abstrakt codiert werden können.

Der Abstraktionsgrad der Datencodierung steht in Zusammenhang mit dem Grad der Flexibilität, den ein Datenmodell zulässt. Je konkreter die Ausprägung eines Datenmodells gefasst ist, desto weniger Spielraum bietet es für Anwendungen, die vom ursprünglichen Zweck der Datenmodellierung abweichen. Umgekehrt gilt: Je abstrakter ein Datenmodell gehalten ist, desto leichter können Komponenten dieses Modells erweitert, eingeschränkt oder für spezielle Anwendungsfälle angepasst werden. XML Schema hat viele Ansätze des objektorientierten Denkens übernommen, um die Flexibilität von Datenmodellen zu gewährleisten. Eine der wichtigsten Parallelen ist die Nutzung benutzerdefinierter Typen - eine Entsprechung der Klassen in der objektorientierten Programmierung.

XML Schema arbeitet mit zwei Kategorien von benutzerdefinierten Typen - *simple types* und *complex types*²². Beiden liegt der Zweck zugrunde, abstrakte Typen von Inhaltsmodellen zu strukturieren. Ihr Hauptunterschied liegt in dem Komplexitätsgrad, der dabei jeweils abgebildet werden kann. *Simple types* können lediglich den Namen des Elementes sowie den zulässigen Typ des Elementes deklarieren. Attribute des generierten Elementes sowie Kindelemente sind innerhalb von *simple types* nicht zulässig:

```
<element name="posting" type="string"/>
```

Complex types können dagegen Text, beliebige Attribute, Kindelemente sowie Kombinationen all dessen aufnehmen. Sobald ein Element Attribute oder Kindelemente enthalten soll, muss zwingend ein *complex type* definiert werden [DGM*01, S. 15]:

```
<complexType name="postingType">  
  <attribute name="id" type="string" />  
</complexType>
```

²²In diesem Kapitel wird nur in der Ausführlichkeit auf das Typenkonzept in XML Schema eingegangen, wie es für das Verständnis der Beispiele erforderlich ist. Teil zwei dieser Arbeit widmet sich einer ausführlicheren Darstellung.

```

    <attribute name="newsgroup" type="string" />
    <attribute name="subject" type="string" />
    <attribute name="timestamp" type="dateTime" />
</complexType>

```

In einem korrespondierenden XML Schema für das oben gezeigte XML-Dokument wird das Element `<posting>` somit nicht als statische Informationseinheit, sondern als Instanz des Typs `<postingType>` realisiert:

```

<?xml version="1.0" encoding="UTF-8"?>
<schema>
  <element name="postings">
    <complexType>
      <sequence>
        <element name="posting" type="postingType"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="postingType">
    <attribute name="id" type="string" use="required"/>
    <attribute name="newsgroup" type="string" use="required"/>
    <attribute name="subject" type="string" use="required"/>
    <attribute name="timestamp" type="dateTime" use="required"/>
  </complexType>
</schema>

```

Warum wird hier vom Einsatz eines *complex type* Gebrauch gemacht? Zum einen schreibt der Einsatz der Attribute `id`, `newsgroup`, `subject` und `timestamp` die Deklaration eines *complex type* formal vor. Weiterhin können alle Eigenschaften des Typs `<postingType>` zentral verwaltet und an beliebiger Stelle von anderen Schema-Elementen referenziert und erweitert werden - etwa dann, wenn andere Gesichtspunkte wie Dateigröße, Absenderinformationen oder das Routing der Nachricht von Interesse sind. Der Hauptvorteil liegt jedoch in der größeren Ausdrucksmächtigkeit eines *complex type*. Sollte sich der gegebene Anwendungsfall ändern - beispielweise soll der Inhalt der Postings künftig verschlagwortet und in einer Datenbank abgelegt werden können - so muss auch das Inhaltsmodell des Elementes `<posting>` leicht für diesen Zweck angepasst werden können. Ein erster Schritt dazu liegt in der Erweiterung des *complex type* um Kindelemente:

(...)

```

<complexType name="postingType">

```

```

<sequence>
  <element name="keyword" type="string"/>
  <element name="note" type="string"/>
</sequence>
  <attribute name="id" type="string" use="required"/>
  (...)

```

Die neuen Kindelemente `<keyword>` und `<note>` werden hier in ihrer Atomizität belassen; der Datentyp `string` zeigt an, dass diese Elemente ausschließlich Zeichendaten enthalten dürfen. Selbstverständlich könnten jedoch auch sie einen abstrakten Typen referenzieren, der seinerseits einem zweckdienlichen Inhaltsmodell gehorcht. XML Schema ist vor allem durch seine starke Typisierung darauf ausgerichtet, ein Mindestmaß an Strukturiertheit mit größtmöglicher Erweiterbarkeit zu verbinden. Inwieweit insbesondere die *complex types* weiterführende Techniken der Datenmodellierung ermöglichen, zeigt der zweite Teil dieser Arbeit.

(5) Ähnlichkeit zwischen Daten kann mit Hilfe einer Kategorisierung abgebildet werden

Auch der letzte genannte Indikator für ein Datenmodell kann mittels Typenbildung realisiert werden, indem ähnliche Elemente unter einem Typ zusammengefasst werden. XML Schema ermöglicht eine Kategorienbildung außerdem durch die Gruppierung von Elementen oder Attributen mittels des Elementes `<group>`. Ein Anwendungsbeispiel sog. *named model groups* findet sich in Kapitel 5.4.

Zweiter Teil

Kapitel 5 - Anwendung von XML Schema

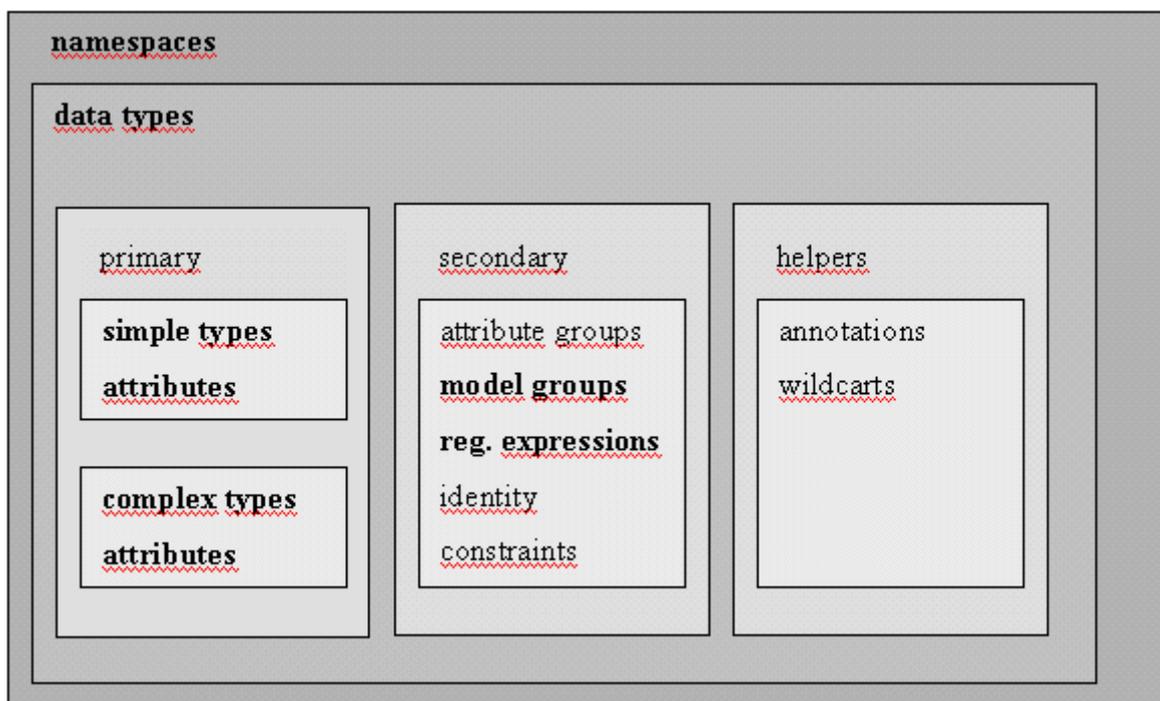
5.1 XML Schema Basisarchitektur

Der Rahmen der vorliegenden Arbeit ermöglicht keine vollständige kursorische Einführung in die XML Schema-Spezifikation. Dennoch sollen die in den folgenden Kapiteln skizzierten Anwendungen von XML Schema auf Metadatenstandards und Austauschformate zumindest grob in den Kontext der W3C-Spezifikation gesetzt werden können. Die gezeigten Beispiele folgen bewusst *nicht* der Reihenfolge ihrer Domänen, wie sie in Kapitel 2 behandelt wurden. So wurde gewährleistet, dass die wesentlichen Aspekte von XML Schema in etwa entsprechend ihrem Schwierigkeitsgrad eingeführt werden.

XML Schema besteht aus zwölf unterschiedlichen Komponenten. Ahmed et al. ordnen diese Komponenten in die drei Kategorien *primary*, *secondary* und *helpers* ein [AAC*01, S. 203ff].

Konzeption und Verständnis von XML Schemata können anhand einer elementaren Architektur erleichtert werden. Die Einordnung in primäre, sekundäre und Hilfskomponenten legt dabei eine Reihenfolge nahe, nach der ein komplexer Designprozess in viele kleine Schritte heruntergebrochen werden kann. Folgende Grafik ist an das Komponentenmodell von Ahmed et al. angelehnt, erweitert es aber um die Bereiche der XML Schema Datentypen und -Namespaces. Dies v.a. deshalb, weil XML Schema-Komponenten selbst Datentypen annehmen können und meist im Kontext des XML Schema-Namensraums verwendet werden. Fettgedruckte Komponenten finden im Rahmen der vorliegenden Arbeit Erwähnung.

XML Schema Komponenten



Zu den primären Komponenten zählen einfache und komplexe Typen sowie Elementen- und Attributdeklarationen. Sie sind die grundlegenden Bestandteile von XML Schemata und werden daher im nächsten Kapitel zuerst behandelt. Zu den sekundären Komponenten zählen Elementen- und Attributgruppen sowie Beschränkungen von Wertebereichen wie etwa reguläre Ausdrücke. Sie sind von Primärkomponenten abhängig, da sie diese genauer beschreiben oder sie als Verbünde von Komponenten gruppieren können. Hilfskomponenten können im Gegensatz zu anderen Komponenten nicht unabhängig existieren. Ihre Bedeutung erschließt sich nur aus dem Kontext ihres Vorkommens und ist lokal beschränkt. Weiterhin können Hilfskomponenten nicht mit einem eigenen Namen versehen und damit auch nicht referenziert werden.

5.2 XML Schema Anwendungsfälle: Encoded Archival Description

Wie im ersten Teil der vorliegenden Arbeit gezeigt wurde, haben sich die Ansprüche an Metadaten und assoziierte Datenmodelle bedingt durch historische und technische Entwicklungen sowie praktische Erwartungen an verteilte Informationssysteme gewandelt.

Was bedeutet dies für eine innerhalb der Geisteswissenschaften tätige Informatik? Zu dem Anspruch der maximalen Verfügbarkeit kommt hier das Streben nach größtmöglicher Vollständigkeit und der Darstellung komplexer Sachverhalte, kurzum: nach Abbildungen realweltlicher Objekte, die der Welt so ähnlich wie nur möglich sind. Naturgemäß sehen sich die Geisteswissenschaften dabei mit Datenmaterial konfrontiert, dass nur schwer in die Strukturen von relationalen Datenbankmodellen zu pressen ist. Die lange verwendeten DTD-Grammatiken sind, wenn auch aus anderen Gründen, als Datenbankmodelle ebensowenig geeignet. Document Type Definitions erweisen sich in dieser Rolle als schwerfällig, sobald der Einsatz in verteilten Systemen geplant ist (vgl. Kap. 3.5). Wie Vokabulare wie die EAD oder Ebind zeigen, liegen viele der in den Geisteswissenschaften verwendeten Metadatenvokabulare nach wie vor als DTDs vor, während Formate wie die TEI sich bereits für eine Nutzung von XML Schema öffnen [TEI03].

Einleitend wurde festgestellt, dass die ausführliche Dokumentation domänen-spezifischer Daten bei einem aus 15 Elementen bestehenden Set wie dem Dublin Core ins Hintertreffen gerät. Was bleibt ist der Wunsch nach Interoperabilität digitaler Sammlungen unter größtmöglicher Beibehaltung ihres Informationsgehalts - oder besser: der Bestände, die sie beschreiben. Dies lässt Bedarf für zweierlei erkennen: Erstens für Datenmodelle, die sich dem Wachstumsverhalten generischen Markups insofern anpassen, als sie selbst mit minimalem Aufwand oder sogar automatisiert erweitert werden können. Zweitens für Metadatenvokabulare, die mittels entsprechender Grammatiken codiert werden. Für Letzteres empfehlen sich sogenannte Minimalsets - also Teilmengen aus bestehenden Metadatenformaten, die inhaltlich entweder von den zuständigen Arbeitsgruppen definiert oder nach projektgebundenen Anwendungsfällen von Metadaten-Autoren entwickelt werden. Minimalsets zeichnen sich dadurch aus, dass sie:

- Daten mit für Anwendungssoftware verständlichen Metadaten ausstatten
- zunächst die wichtigsten Informationen verfügbar machen statt die detailliertesten
- eine tiefere Erschließung auch langfristig gewährleisten
- eine Syntax nutzen, mit der gezielt auf Teilinformationen zugegriffen werden kann

Im Zusammenhang mit Museums- und Archivbeständen unterstreicht Thaller den Vorzug der schnellen Verfügbarkeit von Ressourcen durch den Einsatz von Minimalsets:

Für große, nur zum kleinen Teil dokumentierte Objektsammlungen wäre es [...] von großem Vorteil, bewusst ‚flache‘ Beschreibungsmodi zu entwickeln, also solche, welche die rasche Anfertigung von Minimalbeschreibungen erlauben, die später ergänzt und/oder mit Softwareunterstützung miteinander verbunden werden können.

[Tha00]

Um die Anwendung der EAD-DTD mit dem geringst möglichen Aufwand zu ermöglichen, entwickelte die EAD-Arbeitsgruppe im Jahre 1998 ein Minimalset von EAD-Elementen [SAA99, S. 233f.]. Dieses Minimalset stellt eine Empfehlung der Arbeitsgruppe dar, mit dem die Auszeichnung eines EAD-codierten Findbehelfs analog zu Mindestanforderungen der EAD-DTD ermöglicht werden soll. Im Rahmen dieser Arbeit wurde eine Transkription dieses Minimalsets nach XML Schema angefertigt. Das vollständige Listing dieses XML Schema findet sich in Anhang A. Folgende schematische Darstellung der EAD-Arbeitsgruppe zeigt das Inhaltsmodell des Minimalsets:

```
<ead>
  <eadheader>
    <eadid>
    <filedesc>
      <titlestmt>
        <titleproper>
        <author>
      <publicationstmt>
        <publisher>
        <date>
    <profiledesc>
      <creation>
      <language>
      <language>
  <archdesc> mit LEVEL, LANGMATERIAL und LEGALSTATUS-Attributen
    <did>
      <repository>
        <corpname>
      <originatation>
```

<personname>, <corpname>, <famname> wenn passend
 <unittitle>
 <unitdate>
 <physdesc>
 <unitid> mit COUNTRYCODE und REPOSITORYCODE Attributen
 <abstract>
 <admininfo> Unterelemente nach Bedarf
 <bioghist>
 <scopcontent>
 <controlaccess>
 Unterelemente nach Bedarf
 <dsc> mit **TYPE** Attribut
 <c0x> oder <c>
 mit LEVEL Attribut in so vielen Ebenen wie nötig
 <did>
 <container>
 <unittitle>
 andere Unterelemente nach Bedarf

In den *Application Guidelines* der EAD-Arbeitsgruppe wird die Implementation des Minimalsets mit folgenden Beschränkungen bzw. Empfehlungen kommentiert:

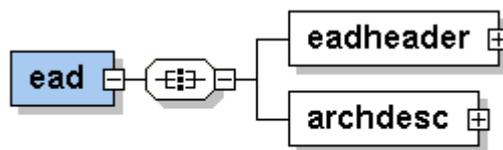
1. Fettgedruckte Elemente müssen auftreten, um gegen die EAD-DTD validiert werden zu können.
2. Die angezeigte Verschachtelung muss eingehalten werden
3. Es müssen mindestens die folgenden Attribute eingebunden werden
 - a. LEVEL (nötig für Validierung)
 - b. TYPE (nötig für Validierung)
 - c. LANGMATERIAL (nötig für ISAD(G)-Entsprechung)
 - d. LEGALSTATUS (nötig für ISAD(G)-Entsprechung)
 - e. COUNTRYCODE (nötig für ISAD(G)-Entsprechung)
 - f. REPOSITORYCODE (nötig für ISAD(G)-Entsprechung)
 empfohlen werden darüber hinaus die Attribute:
 - g. ENCODINGANALOG (als sinnvolle Erweiterung)
 - h. PARENT (als sinnvolle Erweiterung)
 - i. ID (als sinnvolle Erweiterung)
4. Bei folgenden Elementen muss die Reihenfolge aller Kindelemente zwingend eingehalten werden:

- a. <eadheader>
- b. <did> (unter <archdesc>)
- c. <c>, <c0x>

In allen anderen Fällen ist keine spezielle Reihenfolge vorgegeben. Die Ansprüche der Zielgruppe bestimmen die weitere Gestaltung des Minimalsets [SAA99, S. 233ff].

5.2.1 Inhaltsmodell und Typisierung

Obwohl XML Schema besonders gut dazu geeignet ist, einfache und komplexe Inhaltsmodelle darzustellen, orientiert sich auch eine Transkription in XML Schema weitgehend an den Vorgaben, die seinerzeit für Document Type Definitions getroffen wurden. Die wichtigsten konstituierenden Elemente des XML Schema-Designs sind einfache und komplexe Typen. Zunächst muss deshalb entschieden werden, welcher Art von Typ für das Wurzelement <ead> gewählt werden sollte.



Ein Blick in die *Tag Library* [SAA02] der EAD-Arbeitsgruppe offenbart, dass das Element <ead> sowohl Kindelemente als auch Attribute enthalten kann. Laut der EAD-Spezifikation sind drei Kindelemente zulässig. Für das Minimalset werden lediglich die Elemente <eadheader> und <archdesc> empfohlen. XML Schema schreibt bei einer Kombination von Binnenelementen und Attributen zwingend die Verwendung eines *complex type* vor. Folglich muss <ead> als ein *complex type* definiert werden:

```
<element name="ead">
  <complexType>
    <sequence>
      <element name="eadheader" type="eadheaderType"/>
      <element name="archdesc" type="archdescType"/>
    </sequence>
    <attribute name="ALTRENDR" type="string" use="optional"/>
    <attribute name="AUDIENCE" type="string" use="optional"/>
    <attribute name="ID" type="string" use="optional"/>
    <attribute name="RELATEDENCODING" type="string" use="optional"/>
  </complexType>
</element>
```

5.2.2 Zwei Klassen von complex types

XML Schema unterscheidet zwei Arten von komplexen Typen: *anonymous complex types* und *named complex types*. Bei dem im Codeausschnitt gezeigten Typ handelt es sich um einen *anonymous complex type*, da der komplexe Typ einer Elementendeklaration untergeordnet ist und über kein eigenes Namensattribut verfügt. Ein entsprechender *named complex type* hätte die Form `<complexType name="ead"/>`. Hintergrund der Unterscheidung zweier *complex types* ist die Frage, ob das Inhaltsmodell - die Kombination der enthaltenen Elemente und Attribute - an anderer Stelle einem anderen Typ zur Verfügung stehen soll oder nicht. Ist dies gewünscht, empfiehlt sich ein *named complex type*, damit der komplexe Typ über den Wert seines Namensattributes anderorts referenziert werden kann. Ist dies nicht der Fall, steht das Inhaltsmodell ausschließlich dem tragenden Element `<ead>` zur Verfügung.

Die Auszeichnung des `<ead>`-Elementes als *anonymous complex type* reicht an dieser Stelle aus. Das Element fungiert als *root*-Element mit festgelegten Kindelementen, daher muss das Inhaltsmodell keinen anderen Elementen zur Verfügung stehen. Die Reihenfolge der Unterelemente `<eadheader>` und `<archdesc>` ist in der EAD-DTD formal nicht definiert, macht aber nur so Sinn, da `<eadheader>` - *nomen est omen* - Headerinformationen über ein Findbehelf enthalten soll. Zur Modellierung der Reihenfolge auftretender Elemente benutzt XML Schema sog. *compositor*-Elemente.

5.2.3 Compositors

Die Reglementierung der Reihenfolge von Elementen eines Inhaltsmodells ist zwar schon in XML-DTDs existent, aber nur unzureichend umgesetzt worden. Eine Elementendeklaration der Form

```
<!ELEMENT eadheader (eadid, filedesc, profiledesc)>
```

legt folgendes Inhaltsmodell fest: Ein Element `eadheader` muss die Elemente `eadid`, `filedesc` und `profiledesc` in genau dieser Reihenfolge enthalten. Die Kindelemente gehorchen durch die Kommata einer UND-Verknüpfung und müssen vorkommen, sofern sie nicht durch die Kardinalitätsoperatoren `,?` oder `,*` anderweitig gekennzeichnet sind. Alternativ könnte eine DTD auch eine ODER-Verknüpfung vorschreiben, indem statt Kommata das pipe-Symbol (`|`) eingesetzt wird:

```
<!ELEMENT eadheader (eadid | filedesc | profiledesc)>
```

In diesem Fall kann nur eines der Kindelemente auftreten. Die Restmenge der möglichen Kombinationen - das Auftreten aller Elemente in beliebiger Reihenfolge - ist in DTDs nicht ohne weiteres formulierbar. Natürlich können unter Einsatz ‚kombinatorischer Mathematik‘ schwerfällige Konstrukte wie das folgende erstellt werden - der praktische Einsatz ist aber wegen seiner Fehleranfälligkeit wenig empfehlenswert:

```
<!ELEMENT eadheader( (eadid, filedesc, profiledesc) |
                      (eadid, profiledesc, filedesc) |
                      (filedesc, eadid, profiledesc) |
                      (filedesc, profiledesc, eadid) |
                      (profiledesc, eadid, filedesc) |
                      (profiledesc, filedesc, eadid) ) >
```

XML Schema geht mit seinen drei *compositor*-Elementen *sequence*, *choice* und *all* über die Möglichkeiten von DTDs hinaus. Zum einen sind die Bezeichnungen für den Menschen leichter verständlich, zum anderen wird mit dem compositor *all* der Fall abgedeckt, der im letzten Beispiel modelliert werden sollte. Das Element *sequence* besagt, dass eingeschlossene Elemente in der angegebenen Reihenfolge auftreten müssen. *choice* stellt eines der Elemente zur Auswahl. Das Element *all* erlaubt das Auftreten aller Elemente in jeder Reihenfolge:

Compositor	Entsprechung in DTDs	Definition
<i>sequence</i>	Gruppe, durch Kommata getrennt	Alle Elemente, feste Reihenfolge
<i>choice</i>	Gruppe, durch pipes () getrennt	Auswahl eines Elementes
<i>all</i>	-	Alle Elemente, beliebige Reihenfolge

nach[Sko02]

Angaben zu Inhaltsmodellen werden in XML Schema innerhalb eines *complex type* eingebunden. Skonnard [Sko02] sieht in diesem Vorgehen einen Indikator für XML Schemas Verwandtschaft zu Programmiersprachen. Die *compositors* agierten ähnlich wie Variablen, die an formale Typdefinitionen gebunden werden: „*Thinking in terms of type definitions is a major paradigm shift from the way DTDs work*“ [Sko02].

Da für das EAD-Minimalset die Maßgabe gilt, dass fettgedruckte Elemente in der angegebenen Verschachtelung erscheinen müssen, werden die Unterelemente des *<ead>* - Rootelementes innerhalb einer *<sequence>* codiert:

```
(...)
```

```
<element name="ead">
```

```
  <complexType>
```

```
    <sequence>
```

```
      <element name="eadheader" type="eadheaderType"/>
```

```
      <element name="archdesc" type="archdescType"/>
```

```
    </sequence>
```

```
(...)
```

5.2.4 Kontextinformation sichern

Die Typisierung der Kindelemente `<eadheader>` und `<archdesc>` muss wiederum als *complexType* erfolgen, da auch diese Elemente bzw. Attribute enthalten können. Die *complex types*, welche `<eadheader>` und `<archdesc>` repräsentieren, sollten im Gegensatz zum Elternelement `<ead>` als *named complex types* deklariert werden. Das Inhaltsmodell eines *named complex types* ist beliebig wiederverwendbar und über einen Namen adressierbar. Somit ist gewährleistet, dass diese Elemente anderorts referenziert und Veränderungen am Inhaltsmodell dezentral vorgenommen werden können.

Grundsätzlich wird diese Vorgehensweise für alle Elemente des Minimalsets mit Kindelementen fortgesetzt. Dies geschieht weniger aufgrund der Annahme, dass einmalig auftretende Elemente wie `<eadheader>` und `<archdesc>` sich innerhalb eines Dokumentes wiederholen könnten²³ - dieser Fall ist in der Tat unwahrscheinlich. Vielmehr wird dieser Schritt mit Rücksicht auf mögliche Erweiterungen des Minimalsets getan. Die EAD-Arbeitsgruppe hat ihren Standard nicht nur inhaltlich sehr vollständig dokumentiert, sondern stellt auch umfangreiche Kontextinformationen der einzelnen Elemente zur Verfügung. Dazu zählt, dass nicht nur die Kindelemente jedes EAD-Elementes aufgeführt sind, sondern auch alle möglichen Elternelemente. Folgender Ausschnitt aus der EAD-Tag Library zeigt, dass das Element `<phystech>` als direktes Kind der `<archdesc>`, `<c>`, und `<c0x>` - Elemente existieren kann (ohne dabei Teil des EAD - Minimalsets zu sein):

`<phystech>` - Physical Characteristics and Technical Requirements

May contain: address, blockquote, chronlist, head, list, note, p, phystech, table

May occur within: **archdesc**, archdescgrp, c, **c01**, **c02**, **c03**, **c04**, **c05**, **c06**, **c07**, **c08**, **c09**, **c10**, **c11**, **c12**,

descgrp, phystech

[SAA02]

²³ Das Element `<did>` ist im vorliegenden Minimalset die einzige implementierte Wiederholung eines *tags* - es tritt sowohl als Kind von `<archdesc>` als auch von `<c0x>` bzw. `<c>` auf, s.o.

Wann immer Kontextinformation über Elternelemente für einen bestimmten Anwendungsfall wertvoll werden sollten, ist durch Verwendung von *named complex types* in den übergeordneten Typen gewährleistet, dass deren Inhaltsmodell eindeutig adressierbar ist. Potenziell ist dazu jeder Vorgang geeignet, der einer Datenbankabfrage gleicht oder anderweitig von der Nutzung modellierter Relationen profitiert. Im Bereich der Wahrung des Kulturerbes ist folgender Anwendungsfall denkbar: das Element `<phystech>` beschreibt *“important physical conditions or characteristics that affect the storage, preservation, or use of the materials described”* [SAA02]. Da die durch EAD beschriebenen Originalmaterialien ggf. sehr alt sind und nach speziellen Gesichtspunkten gelagert werden müssen, wäre eine skizzierte Datenbankabfrage²⁴ der Form

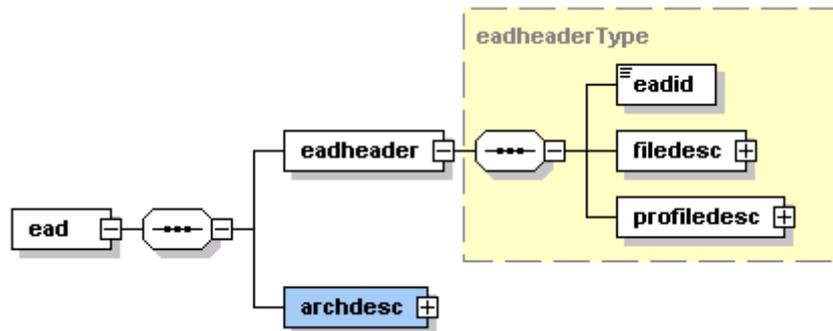
```
SELECT * FROM //ead/archdesc WHERE <phystech> some oxydization on aluminum layer</phystech>
```

in der Lage, Materialien wie mittelalterliche Bücher nach ihrem Zustand auszusortieren und der angemessenen Lagerungsstätte zuzuordnen. Ein EAD-basiertes Metadaten-System ist also über den Einsatz in verteilten Informationssystemen hinaus durchaus auch für den internen Gebrauch in Archiven einsetzbar.

5.2.5 Kardinalitätsoperatoren

Dem Typ-Charakter der referenzierten Kindelemente wurde auch in der Namenswahl Rechnung getragen (`type="eadheaderType"`), um Verwechslungen zwischen Elementenname und Elemententyp vorzubeugen. Weiterhin unterstreicht diese Konvention den „Klassen“-Charakter des Elements gegenüber dem „Konstruktor“-Charakter im referenzierenden Elternelement (`xs:element name="eadheader"`). Als nächstes müssen die *complex types* der Elemente der zweiten Hierarchieebene deklariert werden:

²⁴ Das Beispiel verwendet die an SQL angelehnte Notation einer Datenbankabfrage nur aus Gründen der Verständlichkeit. Querymechanismen für XML-Strukturen sind noch in der Entwicklung begriffen und daher nicht so allgemeinverständlich wie SQL-Syntax.



Der komplexe Typ `<eadheadertype>` enthält drei Kindelemente; laut den Vorgaben des EAD-Minimalsets müssen die Elemente `<eadid>` und `<filedesc>` vorkommen, während das Element `<profiledesc>` nicht obligatorisch ist. Das Element beinhaltet Informationen über die Erzeugung der Ressource, die den Archivgegenstand beschreibt. Dazu zählen Angaben über den Autor, das Erstellungsdatum und die Sprachversion der Ressource [SAA02].

Um in XML Schema das Vorkommen von Elementen zu bestimmen, wurde das Konzept der Kardinalitätsoperatoren von DTDs übernommen und stark vereinfacht. Über die Attribute `minOccurs` und `maxOccurs` kann die Kardinalität eines Elementes in seiner Deklaration genau bestimmt werden. Neu in XML Schema ist der Wert `unbounded`, der jede Kardinalität erlaubt. Folgende Tabelle veranschaulicht dies:

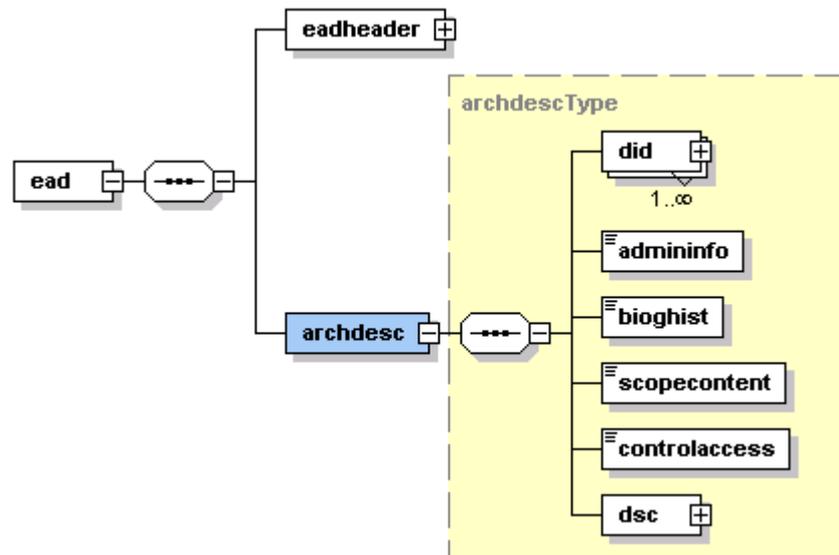
DTDs	Bedeutung	XML Schema
+	eins oder mehr	<code>minOccurs="1" maxOccurs="unbounded"</code>
?	gar nicht oder einmal	<code>minOccurs="0" maxOccurs="1"</code>
*	gar nicht, einmal, mehrmals	<code>minOccurs="0" maxOccurs="unbounded"</code>
(-)	genau einmal	<code>minOccurs="1" maxOccurs="1"</code>

`MinOccurs` und `maxOccurs` können zusammen oder voneinander unabhängig genutzt werden. Werden zum Vorkommen gar keine Angaben gemacht, muss das Element gemäß default-Wert genau einmal erscheinen [DGM*01, S. 20]. Die Kardinalitätsoperatoren der Elemente `<eadid>` und `<filedesc>` könnten daher wegfallen und müssten lediglich im Element `<profiledesc>` angegeben werden:

```

(...)
<xs:complexType name="eadheaderType">
  <xs:sequence>
    <xs:element name="eadid" type="eadidType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="filedesc" type="filedescType" minOccurs="1" maxOccurs="1"/>
    <xs:element name="profiledesc" type="profiledescType" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
(...)
  
```

Das Element `<archdesc>` („Archival Description“) beschreibt den Inhalt, Kontext und Umfang einer Sammlung von Materialien. Um die Einzelteile einer Sammlung genauer zu beschreiben, kann `<archdesc>` das Element `<did>` („descriptive identification“) untergeordnet werden [SAA02]. Durch `<did>` werden alle die Unterelemente gruppiert, die zusammen eine gute Basisbeschreibung von Einzelteilen darstellen.



Da eine Archivalie ggf. aus sehr vielen Einzelteilen besteht, muss das Element `<did>` mindestens einmal vorkommen und kann potenziell unendlich oft wiederholt werden:

```

<xs:complexType name="archdescType">
  <xs:sequence>
    <xs:element name="did" type="didType" minOccurs="1"
                maxOccurs="unbounded"/>
    (...)
  </xs:sequence>
</xs:complexType>

```

5.2.6 Attributtypisierung

XML Schema sieht auch für Attribute umfangreiche Möglichkeiten der Typisierung vor. Ihre Modellierung ist gegenüber der von Elementen jedoch eingeschränkt; so können Attributtypen weder als *complex types* deklariert noch in eine definierte Reihenfolge gebracht werden. Weiterhin ist der Gebrauch von *minOccurs* und *maxOccurs* in Verbindung mit Attributtypen nicht zulässig, da diese höchstens einmal innerhalb eines instanziierten Elementes auftreten dürfen. Attributtypen werden mit dem Element `<attribute>` deklariert

und können ihrerseits Typattribute besitzen²⁵. Diese dienen vor allem der Bestimmung von Wertebereichen, Einschränkungen und Standardwerten von Attributen [Hol01, S. 257].

Die EAD Application Guidelines machen konkrete Vorgaben bezüglich der Verwendung von Attributen. Ein Großteil davon geht aus obigem Modell des Minimalsets hervor, jedoch empfiehlt die Spezifikation darüber hinaus, „...that you consistently utilize the *ENCODINGANALOG*, *PARENT* and *ID* attributes, if your system can make effective use of them“ [SAA99, S. 233]. Um die Flexibilität des hier beschriebenen XML Schemas noch weiter zu erhöhen, wurden alle zulässigen Attribute der EAD-DTD in den betreffenden Elementen des Minimalsets hinzugefügt.

Maßgebliche Eigenschaften der Attribute des EAD-Minimalsets sind ihr Vorkommen (occurrence) und ihre Wertezuweisungen. Über das Typattribut *use* wird festgelegt, ob ein Attribut erforderlich, optional oder auf einen Wert beschränkt ist. Folgende Tabelle zeigt einen Überblick der in der EAD-DTD verwendeten Attributtypisierung und deren Aufschlüsselung in XML Schema:

Attributtypen EAD	Bedeutung	Typattribute XML Schema
#IMPLIED	Attribut kann auftreten	use="optional"
#REQUIRED	Attribut muss auftreten.	use="required"
#FIXED	default-Wert (schreibgeschützt)	use="fixed"
-	default-Wert (überschreibbar)	use="default"
-	darf nicht auftreten	use="prohibited"

Im EAD-Minimalset ist das Vorkommen fast aller Attribute optional. Die einzigen Ausnahmen betreffen das Attribut `LEVEL` im Element `<archdesc>` sowie das Attribut `TYPE` im Element `<dsc>`.

```
<xs:attribute name="LEVEL" type="xs:string" use="required"/>
<xs:attribute name="TYPE" type="xs:string" use="required"/>
```

Es ist keineswegs Zufall, dass gerade diese beiden Elemente ihre Attribute erzwingen, denn `<archdesc>` und `<dsc>` stehen in einem besonderen Verhältnis zueinander.

²⁵ Der Verwendung der - formal korrekten - Bezeichnung „Attributtyp-Attribut“ wird aus Gründen der Sprachhygiene das Wort „Typattribut“ vorgezogen.

<archdesc> beinhaltet allgemeine, übergeordnete Informationen über Materialienverbünde. Die untergeordneten Komponenten dieser Verbünde werden durch das Element <dsc> spezifiziert, wobei Datenelemente vom Element <archdesc> auf den beliebig tiefen Unterebenen von <dsc> wiederholt werden können:

Information is organized in unfolding, hierarchical levels that allow for a descriptive overview of the whole to be followed by more detailed views of the parts, designated by the element Description of Subordinate Components <dsc>

[SAA02]

In der Hauptsache wird durch das LEVEL-Attribut im übergeordnete Element <archdesc> die Klasse der beschriebenen Materialien festgelegt. Dabei kann es sich z.B. um Sammlungen, Serien oder andere Arten von Materialien handeln. Das Element <dsc> beschreibt dagegen nicht die Klasse der Materialie, sondern erschließt diese in ihrer Tiefe. Das Verhältnis der Elemente <archdesc> und <dsc> ist ein gutes Beispiel für die ‚Wirtschaftlichkeit‘ des EAD-Minimalsets. Es zeigt, wie umfangreiche Informationen über Materialien auf sinnvolle Weise über mehrere Elementenebenen verteilt werden. Völlig unklar ist lediglich, warum die beiden Attribute TYPE und LEVEL genau entgegengesetzt ihrer Funktion benannt wurden. Falls es sich hierbei um einen Fehler seitens der EAD-Entwickler handelt, ist dieser auch in der Online-Dokumentation der Encoded Archival Description noch nicht behoben [SAA02].

5.3 Anwendungsfälle: MARC

5.3.1 MARC-Komponenten

Die *Library Of Congress* hat schon früh den Nutzen von XML-Technologien für das MARC-Format erkannt. Als Konsequenz daraus wird seit etwa zwei Jahren eine MARC XML-Architektur entwickelt, mit der MARC-Daten in einer XML-Umgebung eingesetzt werden können [LOC02c]. Die XML ist gut für den Datenaustausch geeignet, da hierarchische Datenstrukturen, wie MARC sie verwendet, mit ihr leicht codiert und verarbeitet werden können. Außerdem wollte die LOC mit diesem Schritt den vielfältigen Einsatzmöglichkeiten von MARC-Daten gerecht werden:

This framework is intended to be flexible and extensible to allow users to work with MARC data in ways specific to their needs [...]. By using XML as the structure for MARC records, users of the MARC in the XML framework can more easily write their own tools to consume, manipulate, and convert MARC data.

Zu den Komponenten des *MARC XML Framework* zählen neben Stylesheets und Softwaretools auch XML Schemata, darunter ein Schema zur Erstellung eines MARC-Records in XML²⁶. Um die Anwendungsmöglichkeiten von XML Schema auf die Syntaxvorgaben von MARC-Feldern besser verständlich zu machen, sollen zunächst kurz deren Komponenten vorgestellt werden [LOC02e].

Tags

tags bezeichnen MARC-Felder. Sie bestehen aus einer dreistelligen Zeichenkette (i.d.R. Ziffern) und stehen immer am Anfang eines MARC-Records. Das *tag* 245 macht Angaben zum Titel einer Ressource [SLB02].

245 10 \$a	Raccoons and ripe corn /
\$c	Jim Arnosky.

Indicators

Jedem *tag* folgen zwei weitere Zeichen, die als *indicators* fungieren. Sie speichern zusätzliche Information darüber, wie genau ein Record anzulegen ist:

245 10 \$a	Raccoons and ripe corn /
\$c	Jim Arnosky.

Der erste *indicator* (1) gibt im Kontext des MARC-*tags* 245 an, dass eine separate Titeltkarte geführt werden muss. Der zweite (0) gibt an, wie viele Zeichen beim Lesevorgang eines Eintrags vom Rechner übersprungen werden sollen. Auf diese Weise lassen sich Artikel wie ‚The‘ oder ‚A‘ entfernen, da sie für eine Schlagwortsuche meist irrelevant sind. Ein zweiter *indicator* mit dem Wert 4 würde den Sachtitel „The Emperor“ also auf „Emperor“ verkürzen. Der *indicator* in obigem Beispiel besagt, dass eine separate Titeltkarte geführt werden muss und kein Zeichen des Eintrags übersprungen wird. Wird eine *indicator*-Position nicht genutzt, gilt sie als „undefined“ und wird mit dem Rautensymbol # besetzt. *indicators* können einen Wert zwischen null und neun annehmen. Ihr Auftreten ist nicht vorgeschrieben [LOC02e].

Subfields

subfield codes bezeichnen logische Untereinheiten eines MARC-*tags*. Sie beziehen sich stets auf das betreffende Feld und haben daher unterschiedliche Bedeutungen. Im Kontext des MARC-Feldes 245 bezeichnet das *subfield* a den Sachtitel und c den Verfasser.

²⁶ Das vollständige MARC XML Schema, aus dem im Folgenden zitiert wird, befindet sich in Anhang A.

245 10 \$a	Raccoons and ripe corn /
\$c	Jim Arnosky.

Jedes *subfield* wird durch einen sog. *delimiter* eingeleitet, in diesem Fall durch ein Dollarzeichen. Weitere *subfields* für das MARC-Feld 245 bezeichnen Untertitel, Art des Mediums oder Angaben zur Reihe [LOC02e].

5.3.2 Syntaxvorgaben durch reguläre Ausdrücke

XML Schema unterstützt den Einsatz von regulären Ausdrücken, um präzise Syntaxvorgaben für Elemente oder Attribute treffen zu können. Reguläre Ausdrücke geben Muster für Zeichenfolgen vor und können die Validität von Datensätzen bereits bei deren Erstellung gewährleisten. Das von der *Library Of Congress* entwickelte MARC XML Schema ist bewusst sehr offen konzipiert, um möglichst vielfältig eingesetzt werden zu können. Es geht daher liberal mit den Vorgaben des MARC-Formates um.

Syntaxvorgaben für *tags*

Folgender Ausschnitt aus einem MARC XML-Record [LOC02c] beschreibt Feld 245 durch das Attribut *tag*:

```
<datafield tag="245" ind1="1" ind2="0">
```

Im dem zugrundeliegenden XML Schema ist das Element `<datafield>` eine Instanz des komplexen Typs `<datafieldType>`. In diesem komplexen Typ wird das Attribut *tag* als eine Instanz des Typs `<tagdataType>` deklariert:

```
<xs:simpleType name="tagDataType" id="tag.st">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
    <xs:pattern
value="(0([1-9A-Z][0-9A-Z])|0([1-9a-z][0-9a-z]))|((([1-9A-Z][0-9A-
Z]{2})|([1-9a-z][0-9a-z]{2}))"/>
  </xs:restriction>
</xs:simpleType>
```

Der Inhalt des Typs `<tagDataType>` ist aus dem Basisdatentyp `string` im XML Schema-Namensraum abgeleitet, kann also alle Zeichendaten enthalten. Der Wertebereich wird durch

das XML Schema-*facet* `<whiteSpace>` dahingehend spezifiziert, dass Leerzeichen, Tabs und Zeilenumbrüche an ihrem Ort belassen werden. Über das Element `<pattern>` können reguläre Ausdrücke in das Schema eingebunden werden. Der obige reguläre Ausdruck besagt:

```
(
0 ([1-9A-Z] [0-9A-Z])           Enthalte die Ziffer 0 mit zwei Zeichen zwischen 1 und 9 oder A und Z
    |                           oder
0 ([1-9a-z] [0-9a-z])           enthalte die Ziffer 0 mit zwei Zeichen zwischen 1 und 9 oder a und z
)
    |                           ODER
(
([1-9A-Z] [0-9A-Z] {2})         enthalte zwei Zeichen zwischen 1 und 9 oder A und Z
    |                           oder
([1-9a-z] [0-9a-z] {2}))       enthalte zwei Zeichen zwischen 1 und 9 oder a und z.
```

Durch diesen regulären Ausdruck werden also auch zweistellige *tags* ermöglicht und Buchstaben als *tag*-Bezeichner zugelassen. Es ist leicht möglich, die `regEx` so zu verschärfen, dass nur dreistellige, aus Ziffern bestehende *tags* legal sind: `([1-9]{3})`. Durch die Definition regulärer Ausdrücke lassen sich kontextabhängig natürlich auch *tag*-Bezeichner für bestimmte MARC-Bereiche freigeben - etwa durch `(2[1-9]{2})` nur für MARC-Felder, die Daten zu Titeln und Editionen aufnehmen. Der Bereich der Benutzerrechtevergabe ist ein gutes Anwendungsfeld für diesen Fall. Gerade im Zusammenhang mit Informationssystemen im Bibliothekswesen ist zu erwarten, dass die ehemals scharf verlaufenden Grenzen zwischen Bibliotheksnutzern und -betreibern verschwimmen werden. Für wissenschaftliche Bibliotheken gilt dies in besonderem Maße, da hier die Datennutzer nicht selten auch Datenprovider sind. So können für bestimmte Benutzer/Provider XML Schemata entworfen werden, die nur die Editierung ‚risikoarmer‘ Bereiche eines Records erlauben.

Syntaxvorgaben für Indicators

Ähnlich wie für die *tags* lassen sich auch reguläre Ausdrücke für *indicators* und *subfields* nutzen. Folgender Codeausschnitt zeigt einen regulären Ausdruck, der numerische Zeichen und Kleinbuchstaben als einstellige Werte für *indicators* zulässt:

```
<xsd:simpleType name="indicatorDataType" id="ind.st">
  <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="preserve"/>
    <xsd:pattern value="\[da-z ]{1}"/>
  </xsd:restriction>
```

```
</xsd:simpleType>
```

[LOC02c]

Der Ausdruck [d] umfasst sämtliche numerischen Zeichen, ist also gleichbedeutend mit [0-9] oder [0123456789].

Syntaxvorgaben für Subfields

Für *subfields* erlaubt das MARC XML Schema ebenfalls sämtliche numerischen Zeichen sowie eine Vielzahl von *delimiter* zur Abgrenzung einzelner *subfields*. Das MARC XML Schema macht damit eine Nutzung in den verschiedensten Softwareumgebungen möglich.

```
<xsd:simpleType name="subfieldcodeDataType" id="code.st">
  <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="preserve"/>
    <xsd:pattern
      value="[\da-z!&quot;#%&amp;'()*+,-./:;&lt;=&gt;?{}_`~\[\]\{1}"/>
  </xsd:restriction>
</xsd:simpleType>
```

[LOC02c]

5.4 Anwendungsfälle: MAB2

5.4.1 Modellierung von Homogenität

Das MAB2-Format ermöglicht oft eine präzisere Aufschlüsselung bibliographischer Daten als die MARC-Formate. Um die dadurch bedingten höheren Komplexitätsgrade aufzufangen, gruppiert MAB2 ähnliche Elemente in hierarchisch angeordneten Datenstrukturen. Den fünf MAB2-Formaten dient das Segment 0 als gemeinsame Grundlage. Allen MAB2-Dateien steht eine maschinenlesbare Satzkenung voran, die durch das Segment 0 definiert wird. Die auf dem Segment 0 aufsetzenden MAB-Formate bestehen jeweils aus zusätzlichen Segmenten [DDB01f]. „MAB-TITEL“, das MAB2-Format für den Austausch bibliographischer Daten, verfügt über neun Segmente, die jeweils durch eine führende Ziffer identifiziert werden:

- 1-- SEGMENT PERSONENNAMEN
- 2-- SEGMENT KOERPERSCHAFTSNAMEN
- 3-- SEGMENT SACHTITEL**
- 4-- SEGMENT VEROEFFENTLICHUNGSVERMERK, UMFANG, BEIGABEN
- 5-- SEGMENT FUSSNOTEN
- 6-- SEGMENT PAUSCHALVERWEISUNGEN UND SIEHE-AUCH-HINWEISE
- 7-- SEGMENT SACHERSCHLIESSUNG

- 8-- SEGMENT NICHTSTANDARDMÄSSIGE NEBENEINTRAGUNG (NE)
- 9-- SEGMENT RSWK-SCHLAGWORTKETTEN

nach [DDB01f]

Weiter aufgeschlüsselte MAB-Felder können dadurch immer einem Segment zugeordnet werden; so gehört das Feld 310 (Hauptsachtitel) dem Segment 3 an usw. MAB2 folgt in seiner Formalerschließung einer weltweit etablierten Konvention zur bibliographischen Beschreibung, der *International Standard Bibliographic Description* (ISBD).²⁷ Das Segment 3 der MAB2-Formats beschreibt Sachtitel analog zu den Vorgaben der ISBD, wie die unten stehenden Tabelle zeigt²⁸:

Elemente gemäß der ISBD	MAB2 – Felder	
1. Hauptsachtitel	310	Hauptsachtitel in Ansetzungsform
2. Allgemeine Materialbenennung	334	Allgemeine Materialbenennung
3. Parallelsachtitel	340	1. Parallelsachtitel in Ansetzungsform
4. Zusätze zum Sachtitel	335	Zusätze zum Hauptsachtitel
5. Verfasserangabe	359	Verfasserangabe

Da MAB2 noch nicht offiziell mit XML-Mitteln abgebildet wird, soll folgendes Beispiel einen möglichen Einsatz skizzieren. Das Beispiel zeigt den Ausschnitt aus einem fiktiven MAB2-Record. Im Segment ‚Sachtitel‘ finden sich die fünf Elemente wieder, die gemäß der ISBD den Bereich der Sachtitel- und Verfasserangabe abdecken.

```
<?xml version="1.0" encoding="UTF-8"?>
<mabrecord>
<!-- SEGMENT PERSONENNAMEN          Felder 1xx-->
<!-- SEGMENT KOERPERSCHAFTSNAMEN    Felder 2xx-->
<!-- SEGMENT SACTITEL                Felder 3xx-->
  <fields seg="3xx">
    <!-- 310 Hauptsachtitel in Ansetzungsform -->
    <datafield tag="310">The Electric Soft Parade</datafield>
```

²⁷ Inzwischen existiert eine ganze Familie von ISBD-Standards, die jedwede Materialien außer Monographien - darunter auch Tonträger, fortlaufende Publikationen oder elektronische Dokumente - bibliographisch beschreiben. Siehe dazu [IFL02c].

²⁸ Die Bedeutungen der MAB2-Felder wurden der Kurzreferenz der *Deutschen Bibliothek* entnommen [DDB01f]. Auf die Miteinbeziehung der Unterfelder und Indikatoren wurde im Rahmen dieses Beispiels verzichtet. Eine Auflistung der ISBD-Elemente in deutscher Sprache findet sich unter [Pay99].

```

<!-- 334 Allgemeine Materialbenennung -->
<datafield tag="334">Tonträger</datafield>
<!-- 340 1. Parallelsachtitel in Ansetzungsform -->
<datafield tag="340">Holes In The Wall</datafield>
<!-- 335 Zusätze zum Hauptsachtitel -->
<datafield tag="355">POP</datafield>
<!-- 359 Verfasserangabe -->
<datafield tag="359">db records/BMG Entertainment</datafield>
</fields>
<!-- SEGMENT VERÖFFENTLICHUNGSVERMERK Felder 4xx-->
(usw.)
</mabrecord>

```

XML Schema bietet zwei Möglichkeiten, solche homogenen Komponenten zu bündeln, ohne dass deren Informationsgehalt eingeschränkt wird.

5.4.2 Bündelung durch komplexe Typen

Das Element `<fields>` wird als Instanz eines komplexen Typs `<fieldsType>` definiert. Die Modellierung eines komplexen Typen ist obligat, da `fieldsType` sowohl Kindelemente als auch das Attribut `seg` (für Segment) aufnehmen muss:

```

<xs:complexType name="fieldsType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="datafield" type="datafieldType"/>
  </xs:choice>
  <xs:attribute name="seg" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="1xx"/>
        <xs:enumeration value="2xx"/>
        <xs:enumeration value="3xx"/>
        (usw.)
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

```

Das Kindelement `<datafield>` verweist auf den korrelierenden komplexen Typ `<datafieldType>`. Um die Zuordnung zu einem MAB2-Segment zu sichern, wird das Attribut `seg` über den *occurrence*-Indikator `required` erzwungen.

5.4.3 Content specification

Die semantische Identität des Segmentes ‚Sachtitel‘ wird schließlich im komplexen Typ `<datafieldType>` festgelegt. Innerhalb eines komplexen Typs kann zwischen mehreren Inhaltsformen unterschieden werden:

- simple content
- complex content
- mixed content

Inhalt des Typs *simple content* erlaubt lediglich die Aufnahme von Attributen und Zeichendaten („text only content“), während *complex content* Attribute und Kindelemente aufnehmen kann („element content“). Als dritte Inhaltsform existiert sog. *mixed content* – hier kann eine Mischung aus Zeichendaten und Elementen auftreten [DGM*01, S. 138ff].

Da für das Element `<datafield>` keine Kindelemente vorgesehen sind, reicht die Deklaration als *simple content* an dieser Stelle aus. Der Attributname `tag` wird aus dem XML Schema-eigenen Datentyp `xs:string` abgeleitet und verfügt damit über alle Zeichen dieses Datentyps. Das Attribut `tag` wird durch den *occurrence*-Indikator `required` erzwungen und erhält einen Wertebereich, der den im Segment 3 relevanten MAB2-Feldern entspricht:

```
<xs:complexType name="datafieldType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="tag" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="310"/>
            <xs:enumeration value="334"/>
            <xs:enumeration value="340"/>
            <xs:enumeration value="355"/>
            <xs:enumeration value="359"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```

    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

Die Vorteile dieser Lösung liegen in der abstrakten Codierung eines Datenfeldes. Für andere Inhaltsmodelle (konkreter: andere Segmente) können so leicht andere MAB2-Felder integriert werden. Eine sinnvolle Erweiterung stellt die eindeutige Verbindung von den Datenfeldern zu den MAB2-Segmenten dar. Dazu sollte das Attribut `seg` aus dem komplexen Typ `<fieldsType>` ‚ausgelagert‘ und als eigener Typ realisiert werden:

```

<xs:attribute name="seg" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="1xx"/>
      <xs:enumeration value="2xx"/>
      <xs:enumeration value="3xx"/>
      (usw.)
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

```

Auf diese Weise kann das Attribut `seg` sowohl von Typen für Feldgruppen im Allgemeinen (`<fieldsType>`) als auch von Typen für MAB2-Felder (`<datafieldType>`) referenziert werden. Innerhalb eines Records lässt sich die Zugehörigkeit eines Feldes zu einem MAB2-Segment so nicht nur durch den Kontext bestimmen, sondern auch explizit ausdrücken:

```

<datafield tag="310" seg="3xx">The Electric Soft Parade</datafield>

```

Die aufgezeigte Lösung mittels komplexer Typen stellt eine flexible, leicht erweiterbare und zukunftsorientierte Methode dar. Sie ist neutral formuliert, da sie Datenfelder weitgehend vom Kontext des benutzten Austauschformates loslöst. Im konkreten Fall des MAB2-Segmentes beruht das Inhaltsmodell jedoch auf dem festgelegten Muster der *International Standard Bibliographic Description* (ISBD). In einer solchen Situation ist u.U. auch eine weniger abstrakte Lösung denkbar.

5.4.4 Bündelung durch Elementengruppen

Die gezeigte Lösung mittels komplexer Typen ist variabel nutzbar, führt aber zu einem kryptischen Datenmodell. Soll einem leicht lesbaren Datenmodell der Vorzug gegeben werden, empfiehlt sich dafür die wörtliche Verwendung der MAB2-Terminologie. XML Schema bietet neben *complex types* auch sog. *named model groups*. Diese erlauben es ebenfalls, homogene Komponenten für wiederverwendbare Inhaltsmodelle zu bündeln [DGM*01, S. 81ff]. Eine *named model group* wird durch das Element `<group>` eingeleitet. Der Vorteil neben der besseren Lesbarkeit besteht darin, dass *compositor*-Elemente wie `<sequence>` innerhalb von *element groups* unkomplizierter eingebaut werden können als in den Inhaltsmodi *simple content* oder *complex content*.

```
<xs:group name="sachtitel">
  <xs:sequence>
    <xs:element name="hauptsachtitel" type="xs:string"/>
    <xs:element name="materialbenennung" type="xs:string"/>
    <xs:element name="parallelsachtitel" type="xs:string"/>
    <xs:element name="hauptsachtitel_zusaetze" type="xs:string"/>
    <xs:element name="verfasser" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

Nachteilig wirkt sich bei dieser Lösung aus, dass immer nur exakt das angegebene Inhaltsmodell verwendet werden kann. Es müssten also verschiedene *named model groups* für sämtliche MAB2-Segmente erstellt werden. Genau wie komplexe Typen sind auch Elementgruppen global, d.h. als direkte Kinder des Wurzelementes `<schema>` zu deklarieren, wenn ihr Inhaltsmodell wiederverwendbar sein soll.

5.5 Anwendungsfälle: Ebind

5.5.1 Notwendigkeit von Namespaces

Durch die Liberalität, die die XML beim Erstellen eigener Elemente und Attribute walten lässt, muss eindeutig festgestellt werden können, woher Markup stammt. Es muss erstens erkannt werden können, zu welchem Vokabular Markup gehört. Zweitens müssen mögliche Namensgleichheiten von Elementen bzw. Attributen eindeutig aufgelöst werden können. Dazu ist ein universeller Identifizierungsmechanismus notwendig. XML Namespaces leisten genau dies. Namespaces sind keine Errungenschaft von XML Schema, sondern entstanden bereits im Jahre 1999 als Erweiterung der XML-Recommendation des W3C:

XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

[W3C99a]

Hintergrund der Namespaces ist das Bestreben, Markupvokabulare wiederzuverwenden statt sie neu zu erfinden. Dies macht immer dann Sinn, wenn Vokabulare bereits auf breiter Ebene genutzt werden und/oder Softwareapplikationen für sie existieren [W3C99a]. Weiterhin lassen sich mit Namespaces Vokabulare verbinden, die unterschiedliche Betrachtungsweisen eines Phänomens erlauben. Markupvokabulare können hierdurch semantisch erheblich aufgewertet werden. Ein Beispiel für die Integration verschiedener Namensräume zu diesem Zweck ist Gegenstand dieses Kapitels.

Der Einsatz von Namespaces wird von DTDs nicht berücksichtigt. Anpassungen für einen Gebrauch von Namespaces können zwar als „Workarounds“ erzwungen werden, komplizieren DTDs aber bis hin zur Unlesbarkeit [DGM*01, S. 183]. Die Verabschiedung der XML Namespaces fällt genau in die Entwicklungsphase von XML Schema und hat Namensräume zu einem integralen Bestandteil von dessen W3C -Spezifikation werden lassen.

5.5.2 Deklaration und Einbindung

Ein Namespace wird über einen Uniform Resource Identifier (URI) eindeutig identifiziert. Er muss nicht zwingend mit einer Datei assoziiert sein; der URI stellt lediglich einen eindeutigen Identifikator dar. Dennoch spricht nichts dagegen, dass der URI aus einem gültigen Uniform Resource Locator (URL) besteht. In einem XML Schema werden Namespaces als Attribute des Wurzelementes `<schema>` deklariert:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Der gezeigte Namespace hat den Namen ‚xs‘ und ist der Standard-Namespace von XML Schema. Die Elemente und Attribute, die mit diesem Namespace assoziiert sind, werden über ein Präfix, in diesem Fall ‚xs‘, in ihrer Zugehörigkeit kenntlich gemacht. Während das Namespace-Präfix in der Deklaration des Namespaces *hinter* den Doppelpunkt gestellt wird, wird es bei den Komponenten des XML Schema-Namensraumes dem sog. *local name* vorangestellt [DGM*01, S. 186]:

```
<xs:element name="...">
```

```
Namespace Präfix:local name
```

XML Schema unterscheidet drei Arten von Namespaces, die verschiedenen Zwecken dienen [DGM*01, S. 184]:

1. Es muss unterschieden werden können zwischen den Elementen und Attributen, die vom Schemaautor selbst erstellt werden und denen, die in der XML Schema Recommendation definiert sind. Dies wird durch den Standard-Namespace geleistet.
2. Eine Untermenge des Standardnamespace lautet `http://www.w3.org/2001/XMLSchema-datatypes`. Dieser Namespace definiert die nativen Datentypen von XML Schema. Er wird nur angewandt, wenn die *built-in datatypes* von XML Schema in Verbindung mit alternativen Schemaanwendungen wie *RELAX* oder *Schematron* angewandt werden sollen.
3. Der dritte Namespace `http://www.w3.org/2001/XMLSchema-instance` wird nur in XML-Instanzdokumenten eingesetzt. Er stellt einige Attribute für besondere Aufgaben zur Verfügung. Wofür diese Attribute nützlich sind, wird weiter unten gezeigt. Mit dem sogenannten `xsi`-Namespace lassen sich mehrere Namensräume in Instanzdokumente importieren. Über die `xsi`-Attribute können lokale Elemente außerdem abstrakten Typen anderer Namespaces genau zugeordnet werden.

5.5.3 Integration von Namespaces

Im Zusammenhang mit der Ebind-DTD (vgl. Kap. 2.4) wurde festgestellt, dass Ebind hauptsächlich auf die physische Beschreibung von Ressourcen ausgerichtet ist. Eine Auszeichnung des intellektuellen Inhaltes kann dagegen von Metadatenvokabularen wie der TEI weitaus besser geleistet werden. Es liegt daher nahe, komplementäre Formate wie Ebind und TEI gemeinsam zu nutzen und deren unterschiedliche Paradigmen innerhalb eines Datenmodells verfügbar zu machen.

Folgendes Beispiel zeigt die Integration von Ebind-Elementen in ein TEI-Instanzdokument. Seitenumbrüche und fortlaufende Kapitel lassen sich ineinander verschachtelt abbilden.

Der Verdienst der zugrundeliegenden XML Schemata liegt *nicht* in der Verschachtelung heterogener Elemente, sondern darin, dass Elemente aus mehreren Namensräumen für das Markup zur Verfügung stehen. Die Namespace-Deklarationen im Wurzelement `<TEI>` sind der Schlüssel dazu:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- XML Instanzdokument -->
(1)<TEI xmlns="http://www.tei-c.org"
(2)   xmlns:eb="http://sunsite.berkeley.edu/Ebind/"
```

```

(3)   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)   xsi:schemaLocation="http://sunsite.berkeley.edu/Ebind   ebind.xsd">
<eb:page id="15">   Seite 15 beginnt
  <div id="3">       Kapitel 3 auf Seite 15
  </div>
</eb:page>         Seite 15 geht zu Ende
<eb:page id="16">   Seite 16 beginnt
  <div id="3">       Kapitel 3 geht auf Seite 16 weiter
  </div>
  <div id="4">       Kapitel 4 beginnt auf Seite 16
  </div>
</eb:page>         Seite 16 geht zu Ende
<eb:page id="17">   Seite 17 beginnt usw.
(...)
</TEI>

```

Die Namensraumdeklarationen lösen sich folgendermaßen auf: (1) bezeichnet den *default*-Namespace des Dokumentes. Dieser besagt, dass alle Elemente, die nicht mit einem Namensraumpräfix versehen sind - etwa `<div>` - automatisch dem Namensraum des URI `http://www.tei-c.org` angehören.

Mit (2) wird der Namespace für alle Elemente des Namensraums `eb` deklariert. Diese Namespacedeclaration sorgt dafür, dass Elemente mit dem Präfix `eb:` überhaupt in diesem Dokument benutzt werden können. Die Deklaration gibt weder Auskunft darüber, wo das passende Schema lokalisiert ist noch wie Elemente des Namensraumes `eb` beschaffen sind.

(3) deklariert den XML Schema-Namensraum für Instanzdokumente. Dieser erfüllt lediglich den Zweck, Attribute mit dem Präfix `xsi:` in dem Dokument zuzulassen. Mit dem `xsi`-Namensraum sind insgesamt vier Attribute assoziiert, die Verweise auf externe Schemata oder Typen in externen Schemata ermöglichen. Eines davon, das Attribut `schemaLocation`, dient hier der Referenzierung des Namensraumes (4), in dem die durch `eb` qualifizierten Elemente zu finden sind. Dem URI des Namensraumes folgt der Dateiname des Schemas. Es ist wichtig, dass sie durch mindestens ein Leerzeichen getrennt sind. Auf diese Weise lassen sich beliebige weitere Namenräume einbinden. Dabei ist darauf zu achten, dass sowohl die Namensraumpräfixe über (2) legalisiert als auch die externen Schemata selbst eingebunden werden (4).

Namensräume müssen nicht nur den Instanzdokumenten bekannt sein, sondern auch den referenzierten Schemata selbst. Dies gilt für jeden zusätzlichen Namensraum neben dem

XML Schema-Standardnamespace, in dem ‚fremde‘ Elemente implizit oder explizit verwendet werden. Es wird gleich offensichtlich werden, was dies für das angeführte Beispiel bedeutet. Zunächst zu den Namespacedeklarationen, die im XML Schema für den TEI-Namensraum notwendig sind:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML Schema für den TEI-Namensraum -->
(1) <xs:schema targetNamespace="http://www.tei-c.org"
(2)           xmlns:eb="http://sunsite.berkeley.edu/Ebind/"
(3)           xmlns:tei="http://www.tei-c.org"
(4)           xmlns:xs="http://www.w3.org/2001/XMLSchema"
(5) <xs:import namespace="http://sunsite.berkeley.edu/Ebind/"
schemaLocation="ebind.xsd"/>
  <xs:element name="TEI">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="eb:page" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Der sogenannte *targetNamespace*(1) gibt an, für welchen Namensraum die Schemakomponenten bestimmt sind - in diesem Fall also für den TEI-Namensraum. Durch (2) wird lediglich gewährleistet, dass Elemente mit dem Präfix *eb* grundsätzlich in diesem Schema verwendet werden dürfen. Die Deklaration des Namensraums (3) mag an dieser Stelle überraschen - warum sollte ein Namensraum eingebunden werden, der durch den *targetNamespace* bereits definiert ist? Im vorliegenden Beispiel liegt eine wechselseitige Abhängigkeit von Namensräumen vor. Das bedeutet, dass in einem referenzierten Namensraum Elemente existieren, die dem lokalen Namensraum angehören. Neben dem XML Schema-Namespace (4) muss über das Element `<xs:import>` auch der referenzierte Namensraum (5) importiert werden. Dies gilt auch dann, wenn auf vermeintlich ‚eigene‘ Elemente wie z.B. `<div>` zugegriffen werden soll, die im Kontext eines fremden Namensraumes deklariert wurden. Die gezeigten Prinzipien werden genau so auf das Schema angewandt, welches den Namensraum für Ebind-Elemente definiert:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML Schema für den Ebind-Namensraum -->
```

```

<xs:schema targetNamespace="http://sunsite.berkeley.edu/Ebind/"
  xmlns:tei="http://www.tei-c.org"
  xmlns:eb="http://sunsite.berkeley.edu/Ebind/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
<xs:import namespace="http://www.tei-c.org" schemaLocation="tei.xsd"/>
  <xs:element name="page">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tei:div" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Die lauffähigen Versionen der beiden gezeigten XML Schemata sind in Anhang A aufgeführt und befinden sich auf dem beigefügten Datenträger.

5.6 Anwendungsfälle: RDF

5.6.1 Hintergrund

Das Resource Description Framwork ist eine in XML geschriebene Sprache zur Beschreibung von Ressourcen [Hol01, S. 988]. In der Regel wird RDF zur Beschreibung von Webressourcen benutzt, obwohl der Einsatz von RDF keineswegs auf Webdokumente beschränkt ist. Das W3C formuliert in seiner 1999 verabschiedeten RDF-Recommendation das hehre Ziel, die Masse an Informationen innerhalb des WWW nicht nur maschinenlesbar, sondern auch *maschinenverständlich* zu codieren. RDF fungiert als wichtigster Teil der *Semantic Web Activity* des W3C [W3C01a]. Der Begriff *Semantic Web* geht auf einen Artikel des mutmaßlichen Erfinders des World Wide Web, Tim Berners-Lee, zurück [Ber01]. Grundidee eines semantischen Web ist es, einen Mehrwert an Bedeutung innerhalb von Ressourcen zu schaffen und so die Zusammenarbeit von Menschen und Rechnern langfristig zu optimieren [Ber01]. Die Aufgabe des RDF besteht darin, den Einsatz von Metadaten so formalisieren, dass diese von Anwendungen wie Suchmaschinen, Katalogen oder intelligenten Agenten besser genutzt werden können [W3C99b]. Das RDF soll zwar domänenneutrale Prinzipien nutzen, aber auf jede Domäne anwendbar sein:

The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the

semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

[W3C99b]

Dieses Ziel des Resource Description Framework erinnert an Anforderungen, die auch an gute Datenmodelle gestellt werden. Die Balance zwischen Strukturiertheit und Spezialisierung zu finden, ist daher eine Herausforderung, der sich auch das Resource Description Framework stellen muss. Junge Technologien neigen dazu, zunächst nur einen dieser beiden Ansprüche zu erfüllen. In der Regel scheint dabei die Strukturierung von Daten vorzugehen und deren genauere Beschreibung zu folgen. Die Entwicklung von Strukturgrammatiken in der Markuptechnologie ist ein prominentes Beispiel dafür. Der Versuch, Fragmente von Ressourcen mit Hilfe des RDF auszuzeichnen, macht jedoch die Grenzen dieser Vision deutlich.

5.6.2 Beschreibung von Ressourcen

Das Resource Description Framework folgt der XML-Syntax, besitzt einen eigenen Namensraum und ein für RDF angepasstes Vokabular, um Ressourcen zu beschreiben. Dies geschieht über sogenannte RDF-Statements, die ähnlich wie bei natürlichen Sprachen über Subjekte, Prädikate und Objekte verfügen. Im folgenden Beispiel besteht das Subjekt aus der bezeichneten www-Ressource (1), das Prädikat aus der bezeichneten Eigenschaft `<dc:Creator>` (2) und das Objekt aus dem Namen des Autors. Teile des Dublin Core Element Sets lassen sich wie hier leicht über einen entsprechenden Namensraum in RDF-Statements integrieren:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
    (1) about="http://xmlbib.system-v.de/MA/ead/ead-instance.xml">
    (2) <dc:Creator> (3)Raymond S. Wright</dc:Creator>
  </rdf:Description>
```

Die Beschreibung von Ressourcen als Ganzes kann so recht unkompliziert bewerkstelligt werden. RDF-Statements werden als Headerinformationen in Webressourcen eingebunden und stellen Suchdiensten etc. die so aufbereiteten Metadaten zur Verfügung.

5.6.3 Beschreibung von Komponenten

Wie können mit Hilfe des RDF aber einzelne *Komponenten* von Ressourcen beschrieben werden? Unterschiedliche Metadatenvokabulare codieren semantisch überlappende Elemente oft anders, was die Schöpfer von Metadatenvokabularen zur Erstellung zahlreicher Konkordanzlisten²⁹ bewogen hat. Sogenannte *crosswalks* zeigen semantische Überschneidungen von Elementen verschiedener Standards auf und beziehen sich damit auf Einzelkomponenten von Ressourcen. Mechanismen im Kontext der Markuptechnologie, die diese Komponenten eindeutig adressieren können, sind zur Zeit noch unzureichend implementiert. So existieren Aspiranten für eine standardisierte Anfragesprache wie XQuery³⁰ derzeit in gleich drei Variationen und führen zu Verzögerungen bei der Verabschiedung einer W3C-Spezifikation. Die Zeitschrift iX schreibt im März 2003:

Ganze acht Entwürfe sammeln sich auf der Homepage von XQuery [...] und jeder einzelne weist eine beeindruckenden Länge auf. Jedoch sind an vielen Stellen noch offene Fragen zu klären und Inkonsistenzen auszuräumen.[...] Alles in allem [ist] XQuery noch mindestens ein Jahr von einer Recommendation entfernt.

[Zie03, S. 125]

Auch alternative Ansätze wie XML's *linking*-Mechanismen XLink bzw. XPointer verhalten sich problematisch. Mit Direktiven ähnlich den *anchors* in HTML ist zwar der *Inhalt* eines Elementes ansteuerbar, nicht aber der Wert des Elementes selbst. Modelle wie das *Document Object Model* ermöglichen zwar die gezielte Lokalisierung von Dokumentenknoten (darunter auch Elemente), erfordern dafür aber den Einsatz von Skript- oder Programmiersprachen. Eine einfache Lokalisierung z.B. der Komponente *titleproper* im Kontext eines XML-Dokumentes ist damit noch eine Zukunftsvision:

```
<rdf:Description
about="http://xmlbib.system-v.de/MA/ead.xml#ead_schema/ead/titleproper">
  <dc:title>Archival Inventories and Guides Of the World </dc:title>
</rdf:Description>
```

Doch selbst wenn solche Mechanismen zur Verfügung stünden, wäre die eindeutige Identifikation bedeutungstragender Elemente dadurch nicht immer gewährleistet. Miller merkt

²⁹ Einige Beispiele für solche Konkordanzlisten sind in Anhang B aufgeführt.

³⁰ <http://www.w3.org/XML/Query>

an, dass die Bedeutung eines Elementes nicht ausschließlich über seine Position definiert werden sollte:

The difficulty with all methods of pointing into documents [...] is that XPointer - or pointing at the first paragraph of the second page, say, depends on the syntactic representation of the document rather than its meaning. Practically this means that if the document changes, the meaning of the pointers will change, which could be an accidental effect of edits made to the document.

[Mil01]

5.6.4 Application Profiles

Sollten Komponenten von Ressourcen zukünftig leichter lokalisierbar werden, ergibt sich eine Reihe sinnvoller Anwendungsmöglichkeiten des Resource Description Framework im Bereich von *crosswalks*. Sogenannte *mappings* - Anwendungen, die die in *crosswalks* enthaltenen Informationen praktisch nutzen - dürften dann einfacher zu implementieren sein als dies heute der Fall ist. Heery und Patel [HP00] sehen umfangreiche Einsatzmöglichkeiten im Bereich der *Application Profiles*. Damit sind domänenspezifische Metadatenvokabulare gemeint, die auf bestimmte Anwendungsfälle und deren Softwarekomponenten zugeschnitten sind. *Mappings* zwischen Metadatenvokabularen sind der Schlüssel dazu, dass *Application Profiles* Interoperabilität gewährleisten können, ohne etwas von ihrem semantischen Reichtum einzubüßen. Es gibt zwei alternative Ansätze, um dieses Ziel zu erreichen - RDF und XML Schema. Beide verfügen über Vor- und Nachteile bei der Modellierung von *Application Profiles*:

- RDF ist gut geeignet, um den semantischen Gehalt von Ressourcen an interpretierende Systeme zu kommunizieren. Dagegen kann RDF lokale Inhaltsmodelle nur ungenau reproduzieren.
- Mit XML Schema lassen sich sehr explizite Inhaltsmodelle entwerfen, jedoch bietet XML Schema im Vergleich wenig Ausdrucksmöglichkeiten, das Verhältnis von Ressourcen zueinander zu beschreiben.

5.6.5 Integration von RDF und XML Schema

Es bietet sich daher an, die Vorteile beider Ansätze zu kombinieren und RDF und XML Schema als komplementäre Technologien zu nutzen. Das W3C protegiert mit *RDF Schema*

einen Ansatz, der dies versucht [W3C03]. RDF ist anfänglich nicht als typisierte Anwendung konzipiert worden. Um sich dem Typensystem von XML Schema anzunähern, ist RDF Schema als eine erweiterte Variante des RDF entwickelt worden. RDF Schema verfügt über einen Satz vordefinierter *classes* und *properties*, aus denen benutzerspezifische Instanzen abgeleitet werden können. Für RDF Schema ist der eigene Namensraum `rdfs:` festgelegt worden, der über den URI `http://www.w3.org/2000/01/rdf-schema#` definiert wird [W3C03]. Eine Klasse in RDF Schema korrespondiert mit dem generischen Konzept eines Typs oder einer Kategorie, ähnlich den Klassen in objektorientierten Programmiersprachen. Es existieren Klassen mit spracheigenen Konstrukten wie `rdfs:Class` und `rdfs:Resource`, sowie Eigenschaften (*properties*) dieser Klassenkonstrukte z.B. mit `rdf:type`³¹ und `rdfs:subClassOf`. Der Einsatz abstrakter Klassen könnte die Erstellung von *mappings* schon sehr bald vereinfachen. Folgendes Modell veranschaulicht, wie semantisch ähnliche Komponenten zueinander in Beziehung gesetzt werden könnten:

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:ead="http://lcweb.loc.gov/ead/ead-schema#"
xmlns:marc="http://www.loc.gov/standards/marcxml//schema/"
```

dc:Creator	->	rdf:type	->	rdfs:Class
Klasse	->	vom Typ	->	rdfs:Class
ead:author	->	rdf:type	->	dc:Creator
Instanz	->	vom Typ	->	Klasse dc:Creator
marc:100	->	rdf:type	->	dc:Creator
Instanz	->	vom Typ	->	Klasse dc:Creator

(usw.)

Die Definition eines ‚Bezugspunktes‘ ist bei einem solchen Ansatz formal erforderlich, um die Oberklasse `rdfs:Class` mit einem Metadatenvokabular zu assoziieren.

³¹ Die Nutzung des `rdf` - Namespacepräfix ist beabsichtigt, da die Eigenschaft `rdf:type` bereits in der regulären RDF-Spezifikation implementiert ist und in RDF Schema wiederverwendet wird.

Epilog

Auch wenn der Fokus dieser Arbeit auf den gegenwärtigen und zukünftigen Herausforderungen an die XML und seine stetig wachsende Familie liegt, sollen doch die Errungenschaften seiner Ahnen nicht dabei vergessen werden. Zum fünften ‚Geburtstag‘ der Extensible Markup Language am 10. Februar 2003 finden sich zahlreiche Gratulanten ein, und nicht alle von ihnen stammen aus der Welt des deskriptiven Markups. Gerade den älteren unter ihnen gebührt Dank, wenn man der Laudatio von Michael Sperberg-McQueen folgt:

It's important that we remember to give credit where it is due: to the foundational work of the ISO SGML working group and the hard work of the scores of experts who served on the original SGML-on-the-Web Working Group - later renamed the XML Interest Group.

Michael Sperberg-McQueen in [Dum03]

* * *

Literaturangaben

[Abi97]

Abiteboul, S.(1997). "Querying Semistructured Data" in: Proc. Intl. Conference on Database Theory (ICDT)

[ABS99]

Abiteboul, S., Buneman, P., Suciu, D.(1999). *Data on the Web - From Relations to Semistructured Data and XML*. Morgan Kaufman, München

[AAC*01]

Ahmed, K., Ancha, S., Cioroianu, A., Cousins, J., Crosbie, J., Davies, J., Gabhart, K., Gould, S., Laddad, R., Li, S., Macmillan, B., Rivers-Moore, D., Skubal, J., Watson, K., Williams, S., Hart, J.(2001). *Professional Java XML*. Wrox Press Ltd., Birmingham

[Bac00]

Bach, M. (2000). *XSL und XPath - verständlich und praxisnah*. Addison-Wesley, München

[Bak00]

Baker, T. (2000). "A Grammar of Dublin Core" in: *D-Lib Magazine*
Volume 6 Number 10/2000
URL 04.02.2003 <http://dlib.org/dlib/october00/baker/10baker.html>

[Bat73]

Bateson, G.(1973). *Steps to an Ecology of Mind: Collected Essays in Anthropology, Psychiatry, Evolution, and Epistemology*. Palladin, London-New York

[BDL02]

Berkeley Digital Library.(2002). *Digital Page Imaging and SGML - An Introduction to the Electronic Binding DTD (Ebind)*.
URL 08.02.2003 <http://sunsite.berkeley.edu/Ebind>

[Ber01]

Berners-Lee. T.(2001). "The Semantic Web" in: *Scientific American* online 05/2001.
URL 10.02.2003 <http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2>

[BM00]

Behme, H., Mintert, S.(2000). *XML in der Praxis*. Addison-Wesley, München

[Bor01]

Born, G.(2001). *Jetzt lerne ich XML - der einfache Einstieg in den neuen Dokumenten- und Web-Standard*. Markt & Technik Verlag, München.

[DCM02a]

Dublin Core Metadata Initiative.(2002). *DCMI term declarations represented in XML Schema language*. URL 07.12.2002 <http://dublincore.org/schemas/xmls>

- [DCM02b]
Dublin Core Metadata Initiative.(2002). *Dublin Core Metadata Initiative Overview*.
URL 07.12.2002 <http://www.dublincore.org/about>
- [DCM02c]
Dublin Core Metadata Initiative.(2002). *Dublin Core Qualifiers*.
URL 07.12.2002 <http://www.dublincore.org/documents/dcmes-qualifiers>
- [DDB01a]
Die Deutsche Bibliothek.(2001). *DDB professionell – UNIMARC*.
URL 02.02.2003 <http://www.ddb.de/professionell/unimarc.htm>
- [DDB01b]
Die Deutsche Bibliothek.(2001). *DDB professionell – IFLA UBCIM Programme*.
URL 02.02.2003 http://www.ddb.de/professionell/ifla_ubcim.htm
- [DDB01c]
Die Deutsche Bibliothek.(2001). *Arbeitsstelle für Standardisierung: Projekt "Umstieg auf internationale Formate und Regelwerke (MARC21, AACR2)"*
URL 04.02.2003 http://www.ddb.de/professionell/afs_projekt.htm
- [DDB01d]
Die Deutsche Bibliothek.(2001). *Arbeitsstelle für Standardisierung: Projektantrag "Umstieg auf internationale Formate und Regelwerke (MARC21, AACR2)"*
URL 04.02.2003 http://www.ddb.de/professionell/pdf/dfg_antrag.pdf
- [DDB01e]
Die Deutsche Bibliothek.(2001). *Dewey Decimal Classification*.
URL 04.02.2003 http://www.ddb.de/professionell/ddc_info.htm
- [DDB01f]
Die Deutsche Bibliothek.(2001). *MAB*.
URL 05.02.2003 <http://www.ddb.de/professionell/mab.htm>
- [DGM*01]
Ducket, J., Griffin, O., Mohr, S., Norton, F., Ozu, N., Stokes-Rees, I., Tennison, J., Williams, K.(2001). *Professional XML Schemas*. Wrox Press Ltd., Birmingham
- [Dün98]
Dönhöler, K.(1998). *Ein Glossar zu XML*.
URL 10.01.2003 <http://www.net-graphics.de/Noframe/XML/glossar.htm>
- [Dum03]
Dumbill, E.(2003). *XML at Five*.
URL 19.02.2003 <http://www.xml.com/pub/a/2003/02/12/xml-at-5.html>
- [EF00]
Endres, A., Fellner, D.W.(2000). *Digitale Bibliotheken. Informatik-Lösungen für globale Wissensmärkte*. dpunkt-Verlag, Heidelberg

- [Eve99]
Eversberg, B.(1999). *Was sind und was sollen Bibliothekarische Datenformate. WWW-Version mit Ergänzungen.* Universitätsbibliothek der TU Braunschweig.
URL 05.02.2003 <http://www.allegro-c.de/allegro/formate/formate.htm>
- [GP00]
Goldfarb, C.F., Prescod, P.(2000). *The XML Handbook 2nd Ed.* Prentice Hall, Upper Saddle River
- [Gol90]
Goldfarb, C.F.(1990). *The SGML Handbook.* Clarendon Press, Oxford
- [HBI95]
Hochschule für Bibliotheks- und Informationswesen Stuttgart.(1995). *Glossar zur Sacherschließung.* URL 04.02.2003
<http://www.hbi-stuttgart.de/hbi/glossar/regdat/dezimal.htm>
- [Hac92]
Hacker, R.(1992). *Bibliothekarisches Grundwissen.* 6., völlig Neubearb. Aufl. Saur, München, London, New York, Paris
- [Har01]
Harold, E.R. 2001). *XML Bible 2nd Edition.* Hungry Minds, New York
- [HP00]
Heery, R., Patel, M.(2000). *Application profiles: mixing and matching metadata Schemas.* URL. 15.02.2003 <http://www.ariadne.ac.uk/issue25/app-profiles>
- [Hol01]
Holzner, S.(2001). *Insider XML.* Markt & Technik Verlag, München.
- [IFL02a]
International Federation Of Library Associations and Institutions (IFLA).(2002). *IFLA Universal Bibliographic Control and International MARC Core Programme (UBCIM).* URL 02.02.2003 <http://www.ifla.org/VI/3/puc.htm>
- [IFL02b]
International Federation Of Library Associations and Institutions (IFLA).(2002). *International List of UNIMARC Users and Experts.*
URL 02.02.2003 <http://www.ifla.org/VI/3/p1996-2/iluue.htm#tab1>
- [IFL02c]
International Federation Of Library Associations and Institutions (IFLA).(2002). *Liste von Übersetzungen der ISBD ins Deutsche - Stand: März 2002*
URL 06.02.2003 <http://www.ifla.org/VII/s13/pubs/isbd.htm>
- [Jec01]
Jeckle, M.(2001). *XML Schema.*
URL 04.12.2002 <http://www.jeckle.de/files/XIAXSD.pdf>

- [ISA94]
ISAD(G).(1994). *General International Standard Archival Description*.
International Council on Archives. Ottawa, Ont.
- [ISO86]
ISO (1986). *Information Processing Systems -Text and Office Systems-
Standard Generalized Markup Language (SGML)*, ISO 8879, Oct.1986.
- [Ken78]
Kent, W.(1978). *Data and Reality*. North-Holland Publishing Company, Amsterdam.
- [KST02]
Kazakos, W., Schmidt, A., Tomczyk, P.(2002). *Datenbanken und XML*.
Springer, Berlin, Heidelberg
- [KKP*72]
Kaegbein, P., Kaltwasser, F.G., Plug, G., Striedl, H., Wieder, J.(ed.).(1972).
Austausch bibliographischer Daten und das MARC-Format. a.d. Reihe
Bibliothekspraxis, Bd. 6. Verlag-Dokumentation, München-Pullach, Berlin
- [KL03]
Koop-Litera.(2003). *Portal der österreichischen Literaturarchive*.
URL 08.02.03 <http://www.onb.ac.at/koop-litera/standards/#ISAD-G>
- [Lan77]
Langefors, B.(1977). "Information Systems Theory," in: *Information Systems*.
Volume 2, Number 4, 1977
- [Lob00]
Lobin, H.(2000). *Informationsmodellierung in XML und SGML*. Springer,
Berlin, Heidelberg
- [LOC03a]
Library Of Congress.(2003). *MARC Code List: PART V: Format Sources*.
URL 06.02.2003 <http://www.loc.gov/marc/relators/relaform.html>
- [LOC02a]
Library Of Congress.(2002). *MARC Standards FAQ*.
URL 01.02.2003 <http://www.loc.gov/marc/faq.html#1>
- [LOC02b]
Library Of Congress.(2002). *Understanding MARC Bibliographic: Machine Readable
Cataloging*. URL 04.02.2003 <http://www.loc.gov/marc/umb/um01to06.html>
- [LOC02c]
Library Of Congress.(2002). *MARC in XML*.
URL 05.02.2003 <http://www.loc.gov/marc/marcxml.html>
- [LOC02d]
Library Of Congress.(2002). *Development of the Encoded Archival Description DTD*.
URL 07.02.2003 <http://www.loc.gov/ead/eaddev.html>

[LOC02e]

Library Of Congress.(2002). *Understanding MARC Bibliographic: Machine Readable Cataloging*. URL 04.02.2003 <http://www.loc.gov/marc/umb/um07to10.html>

[Lox98]

Loxen, N.(1998). *HTML XML SGML*.

URL 01.12.2002 <http://www-is.informatik.uni-oldenburg.de/~dibo/teaching/sem/Ausarbeitungen/loxen/html/>

[Meg98]

Meggison, D.(1998). *Structuring XML Documents*. Prentice Hall, Upper Saddle River

[Mil01]

Miller, L.(2001). *Using RDF query to manage metadata vocabularies*.

URL. 15.02.03 <http://ilrt.org/discovery/2001/06/process>

[MKS96]

Maaß, S., Kreil-Sauer, A., Schröder, K.(1996). „Metadaten - Schlüssel zur Nutzung von Informationssystemen“. Diskussionspapier des Lehrstuhls für Statistik und Ökonometrie d. Friedrich-Alexander-Universität Erlangen-Nürnberg. Nürnberg, 12/1996

[NH00]

North S., Hermans, P.(2000). *XML in 21 Tagen*. Markt & Technik Verlag. München.

[Pay99]

Payer, M.(1999). *Grundlagen der Formalerschließung: Skript. Kapitel 2: Bibliographische Beschreibung*.

URL 07.02.2003 <http://www.payer.de/grundlagenfe/fegscr02.htm>

[PP02]

Payer, M., Payer, A.(2002). *Datenbankaufbau: Skript. -- Kapitel 7: Formate in bibliographischen Datenbanken*.

URL 05.02.2003 <http://www.payer.de/dbaufbau/dbauf07.html>

[RAD90]

RAD.(1990). *Rules for Archival Description*. Bureau of Canadian Archivists, section 1.0A1. Ottawa, Ont.

[Rut98]

Ruth, J. E.(1998). “Encoded Archival Description: A Structural Overview” in: *Encoded Archival Description: context, theory and case studies*. ed.: Jackie M. Dooley. The Society Of American Archivists, Chicago

[SAA98]

Society Of American Archivists. Encoded Archival Description Working Group.(1998). *Encoded Archival Description Tag Library: version 1.0*.

The Society Of American Archivists, Chicago

- [SAA99]
Society Of American Archivists. Encoded Archival Description Working Group.(1999). *Encoded Archival Description Application Guidelines: version 1.0*. The Society Of American Archivists, Chicago
- [SAA02]
Society Of American Archivists. Encoded Archival Description Working Group.(2002). *Encoded Archival Description Tag Library: version 2002*. The Society Of American Archivists, Chicago
URL 13.001.2003 <http://www.loc.gov/ead/tglib/index.html>
- [See00]
Seeboerger-Weichselbaum, M.(2000). *Das Einsteigerseminar XML*. 2. Aufl. bhv Verlag, Kaarst
- [Sko02]
Skonnard, A.(2002). "A Quick Guide to XML Schema" in: *MSDN Magazine* Vol. 7, No. 4, 2002
URL 12.01.2003 <http://msdn.microsoft.com/msdnmag/issues/02/04/xml/default.aspx>
- [SLB02]
Schweizerische Landesbibliothek.(2002). *MARC21 SWISS Version – Handbuch in deutscher Sprache*.
URL 01.02.2003 <http://www.snl.ch/marc21/dmarcein11.htm>
- [SUB03]
Staats- und Universitätsbibliothek (SUB) Göttingen.(2003). *Metadata Server der SUB Göttingen*.
URL. 15.02.2003 <http://www2.sub.uni-goettingen.de/metaform/index.html>
- [TBL02]
The British Library.(2002). *The UKMARC Manual*.
URL 01.02.2003 <http://www.bl.uk/services/bibliographic/marc/marcintro.html>
- [TEI01]
TEI - Text Encoding Initiative.(2001). *The TEI FAQ*. 01.08.2001 (revised).
URL 09.01.2003: <http://www.tei-c.org/Faq/index.html#index-div-d17e131>
- [TEI03]
TEI - Text Encoding Initiative.(2003). *Relax NG Schemas for the TEI*. 19.01.2003 (revised).
URL 13.02.2003 <http://www.tei-c.org/Schemas/RelaxNG/P4X>
- [Tha00]
Thaller, M.(2000). "Vom verschwindenden Unterschied zwischen Datenbanken und Texten: Konsequenzen neuerer www-Technologie am Beispiel von museumsnahen Datenbanken". Vortragsskript zu den EDV-Tagen Theuern 2000. URL 15.01.2003 http://www.museumtheuern.de/edvtage/index.htm?edvtage/g/g12_inh.htm

- [TL82]
Tsichritzis, D.C, Lochovsky, F.H.(1982). *Data Models*. Prentice Hall, Englewood Cliffs, N.J.
- [W3C99]
W3C.(1999).*XML Schema Requirements*. W3C Note 15. Februar1999.
URL 10.02.2003 <http://www.w3.org/TR/NOTE-xml-schema-req>
- [W3C99a]
W3C.(1999). *Namespaces in XML*. W3C Recommendation 14. Januar1999.
URL 15.02.2003 <http://www.w3.org/TR/REC-xml-names/#sec-intro>
- [W3C99b]
W3C.(1999). *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation 22. Februar 1999.
URL 09.01.2003 <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>
- [W3C00]
W3C.(2000). *Extensible Markup Language (XML) 1.0 (Second Edition)*.
W3C Recommendation 6.November 2000
URL 06.12.2002 <http://www.w3.org/TR/REC-xml>
- [W3C00a]
W3C.(2000). *Datatypes for DTDs (DT4DTD) 1.0*. W3C Note 13. Januar 2000
URL 14.12.2002 <http://www.w3.org/TR/dt4dtd>
- [W3C00b]
W3C.(2000). *XML Schema*. W3C Recommendation. 2. Mai 2001.
URL 10.02.2003 <http://www.w3.org/XML/Schema>
- [W3C01]
W3C.(2001). *XML Information Set*. W3C Recommendation 24. Oktober 2001.
URL 12.02.2003 <http://www.w3.org/TR/xml-infoset>
- [W3C01a]
W3C.(2001). *Semantic Web*. URL 12.02.2003 <http://www.w3.org/2001/sw/>
- [W3C03]
W3C.(2003). *RDF Primer*. W3C Working Draft 23. Januar 2003.
URL 17.02.2003 <http://www.w3.org/TR/2003/WD-rdf-primer-20030123/rdfschema>
- [WW01]
Wessel, C., Weiß, B.(2001).“META-LIB: Die Metadaten-Initiative deutscher Bibliotheken“ in: *Bibliothek - Forschung und Praxis*. Jahrgang 25, Nr. 3, 2001
- [Zie03]
Ziegler, C.(2003). „Anfragesprachen für XML - Nach Daten fischen“ in: *iX – Magazin für professionelle Informationstechnik*. Nr. 03/2003

Anhang A

Encoded Archival Description (EAD) - Minimalset als XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Dieses XML Schema entspricht dem EAD-Minimalset, enthält aber alle verfügbaren Attribute der EAD-DTD -->
<!-- XML Schema equivalent to the EAD Minimum Recommended Finding Aid Elements containing all -->
<!-- legal attributes of the EAD-DTD -->
<!-- 2003 by Martin Iordanidis e-mail: martin@iordanidis.de www: http://www.iordanidis.de -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <!-- <ead> -->
  <xs:element name="ead">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="eadheader" type="eadheaderType"/>
        <xs:element name="archdesc" type="archdescType"/>
      </xs:sequence>
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="RELATEDENCODING" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <!-- <eadheader> -->
  <xs:complexType name="eadheaderType">
    <xs:sequence>
      <xs:element name="eadid" type="eadidType"/>
      <xs:element name="filedesc" type="filedescType"/>
      <xs:element name="profiledesc" type="profiledescType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="FINDAIDSTATUS" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LANGENDCODING" type="xs:string" use="optional"/>
    <xs:attribute name="RELATEDENCODING" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- <eadid> -->
  <xs:complexType name="eadidType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
        <xs:attribute name="SOURCE" type="xs:string" use="optional"/>
        <xs:attribute name="SYSTEMID" type="xs:string" use="optional"/>
        <xs:attribute name="TYPE" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <!-- <filedesc> -->
  <xs:complexType name="filedescType">
    <xs:sequence>
      <xs:element name="titlestmt" type="titlestmtType"/>
      <xs:element name="publicationstmt" type="publicationstmtType"/>
    </xs:sequence>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- <titlestmt> -->
  <xs:complexType name="titlestmtType">
    <xs:sequence>
      <xs:element name="titleproper" type="titleproperType"/>
      <xs:element name="author" type="authorType"/>
    </xs:sequence>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
  </xs:complexType>

```

```

</xs:complexType>
<!-- <titleproper> -->
<xs:complexType name="titleproperType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="EXTENT" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="PUBSTATUS" type="xs:string" use="optional"/>
      <xs:attribute name="RENDER" type="xs:string" use="optional"/>
      <xs:attribute name="TYPE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <author> -->
<xs:complexType name="authorType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <publicationstmt> -->
<xs:complexType name="publicationstmtType">
  <xs:sequence>
    <xs:element name="publisher" type="publisherType"/>
    <xs:element name="date" type="dateType"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
<!-- <publisher> -->
<xs:complexType name="publisherType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <date> -->
<xs:complexType name="dateType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="CERTAINTY" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="NORMAL" type="xs:string" use="optional"/>
      <xs:attribute name="TYPE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <profiledesc> -->
<xs:complexType name="profiledescType">
  <xs:sequence>
    <xs:element name="creation" type="creationType"/>
    <xs:element name="language" type="languageType"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>

```

```

</xs:complexType>
<!-- <creation -->
<xs:complexType name="creationType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <language -->
<xs:complexType name="languageType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <archdesc -->
<xs:complexType name="archdescType">
  <xs:sequence>
    <xs:element name="did" type="didType" maxOccurs="unbounded"/>
    <xs:element name="admininfo" type="admininfoType"/>
    <xs:element name="bioghist" type="bioghistType"/>
    <xs:element name="scopecontent" type="scopecontentType"/>
    <xs:element name="controlaccess" type="controlaccessType"/>
    <xs:element name="dsc" type="dscType"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>
  <xs:attribute name="LANGMATERIAL" type="xs:string" use="optional"/>
  <xs:attribute name="LEGALSTATUS" type="xs:string" use="optional"/>
  <xs:attribute name="LEVEL" type="xs:string" use="required"/>
  <xs:attribute name="OTHERLEGALSTATUS" type="xs:string" use="optional"/>
  <xs:attribute name="OTHERLEVEL" type="xs:string" use="optional"/>
  <xs:attribute name="OTHERTYPE" type="xs:string" use="optional"/>
  <xs:attribute name="RELATEDENCODING" type="xs:string" use="optional"/>
  <xs:attribute name="TYPE" type="xs:string" use="optional"/>
</xs:complexType>
<!-- <did -->
<xs:complexType name="didType">
  <xs:sequence>
    <xs:element name="repository" type="repositoryType"/>
    <xs:element name="origination" type="originationType"/>
    <xs:element name="unittitle" type="unittitleType"/>
    <xs:element name="unitdate" type="unitdateType"/>
    <xs:element name="physdesc" type="physdescType"/>
    <xs:element name="unitid" type="unitidType"/>
    <xs:element name="abstract" type="abstractType"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
<!-- <repository -->
<xs:complexType name="repositoryType">
  <xs:sequence>
    <xs:element name="corpname" type="corpnameType"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>
  <xs:attribute name="LABEL" type="xs:string" use="optional"/>
</xs:complexType>

```

```

<!--<corpname> -->
  <xs:complexType name="corpnameType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
        <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
        <xs:attribute name="AUTHFILENUMBER" type="xs:string" use="optional"/>
        <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
        <xs:attribute name="ID" type="xs:string" use="optional"/>
        <xs:attribute name="NORMAL" type="xs:string" use="optional"/>
        <xs:attribute name="OTHERSOURCE" type="xs:string" use="optional"/>
        <xs:attribute name="ROLE" type="xs:string" use="optional"/>
        <xs:attribute name="SOURCE" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
<!-- <origination> -->
  <xs:complexType name="originationType">
    <xs:all>
      <xs:element name="personname" type="personnameType"/>
      <xs:element name="corpname" type="corpnameType"/>
      <xs:element name="famname" type="famnameType"/>
    </xs:all>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LABEL" type="xs:string" use="optional"/>
  </xs:complexType>
<!--<personname> -->
  <xs:complexType name="personnameType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
        <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
        <xs:attribute name="AUTHFILENUMBER" type="xs:string" use="optional"/>
        <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
        <xs:attribute name="ID" type="xs:string" use="optional"/>
        <xs:attribute name="NORMAL" type="xs:string" use="optional"/>
        <xs:attribute name="OTHERSOURCE" type="xs:string" use="optional"/>
        <xs:attribute name="ROLE" type="xs:string" use="optional"/>
        <xs:attribute name="SOURCE" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
<!--<famname> -->
  <xs:complexType name="famnameType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
        <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
        <xs:attribute name="AUTHFILENUMBER" type="xs:string" use="optional"/>
        <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
        <xs:attribute name="ID" type="xs:string" use="optional"/>
        <xs:attribute name="NORMAL" type="xs:string" use="optional"/>
        <xs:attribute name="OTHERSOURCE" type="xs:string" use="optional"/>
        <xs:attribute name="ROLE" type="xs:string" use="optional"/>
        <xs:attribute name="SOURCE" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
<!-- <unittitle> -->
  <xs:complexType name="unittitleType">
    <xs:sequence/>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LABEL" type="xs:string" use="optional"/>
    <xs:attribute name="TYPE" type="xs:string" use="optional"/>
  </xs:complexType>
<!-- <unitdate> -->
  <xs:complexType name="unitdateType">

```

```

<xs:simpleContent>
  <xs:extension base="xs:string">
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="CERTAINTY" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LABEL" type="xs:string" use="optional"/>
    <xs:attribute name="NORMAL" type="xs:string" use="optional"/>
    <xs:attribute name="TYPE" type="xs:string" use="optional"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>
<!-- <physdesc> -->
<xs:complexType name="physdescType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="LABEL" type="xs:string" use="optional"/>
      <xs:attribute name="OTHERSOURCE" type="xs:string" use="optional"/>
      <xs:attribute name="SOURCE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <unitid> -->
<xs:complexType name="unitidType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="COUNTRYCODE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="LABEL" type="xs:string" use="optional"/>
      <xs:attribute name="REPOSITORTYCODE" type="xs:string" use="optional"/>
      <xs:attribute name="TYPE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <abstract> -->
<xs:complexType name="abstractType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="LABEL" type="xs:string" use="optional"/>
      <xs:attribute name="TYPE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <admininfo> -->
<xs:complexType name="admininfoType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
      <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
      <xs:attribute name="Type" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- <bioghist> -->
<xs:complexType name="bioghistType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
      <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

```

        <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
        <xs:attribute name="ID" type="xs:string" use="optional"/>
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
<!-- <scopecontent> -->
<xs:complexType name="scopecontentType">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
            <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
            <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
            <xs:attribute name="ID" type="xs:string" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<!-- <controlaccess> -->
<xs:complexType name="controlaccessType">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
            <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
            <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
            <xs:attribute name="ID" type="xs:string" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<!-- <desc> -->
<xs:complexType name="dscType">
    <xs:sequence>
        <xs:element name="c0x" type="c0xType"/>
        <xs:element name="c" type="cType"/>
    </xs:sequence>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="OTHERTYPE" type="xs:string" use="optional"/>
    <xs:attribute name="TYPE" type="xs:string" use="required"/>
</xs:complexType>
<!-- <c> -->
<xs:complexType name="cType">
    <xs:sequence>
        <xs:element name="did" type="didType"/>
        <xs:element name="container" type="containerType"/>
        <xs:element name="unittitle" type="unittitleType"/>
    </xs:sequence>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LANGMATERIAL" type="xs:string" use="optional"/>
    <xs:attribute name="LEGALSTATUS" type="xs:string" use="optional"/>
    <xs:attribute name="LEVEL" type="xs:string" use="required"/>
    <xs:attribute name="OTHERLEGALSTATUS" type="xs:string" use="optional"/>
    <xs:attribute name="OTHERLEVEL" type="xs:string" use="optional"/>
</xs:complexType>
<!-- <c0x> -->
<xs:complexType name="c0xType">
    <xs:all>
        <xs:element name="did" type="didType"/>
        <xs:element name="container" type="containerType"/>
        <xs:element name="unittitle" type="unittitleType"/>
    </xs:all>
    <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
    <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
    <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
    <xs:attribute name="ID" type="xs:string" use="optional"/>
    <xs:attribute name="LANGMATERIAL" type="xs:string" use="optional"/>
    <xs:attribute name="LEGALSTATUS" type="xs:string" use="optional"/>
    <xs:attribute name="LEVEL" type="xs:string" use="required"/>
    <xs:attribute name="OTHERLEGALSTATUS" type="xs:string" use="optional"/>
    <xs:attribute name="OTHERLEVEL" type="xs:string" use="optional"/>

```

```
</xs:complexType>
<!-- <container> -->
<xs:complexType name="containerType">
  <xs:sequence>
    <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ALTRENDER" type="xs:string" use="optional"/>
  <xs:attribute name="AUDIENCE" type="xs:string" use="optional"/>
  <xs:attribute name="ENCODINGANALOG" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:string" use="optional"/>
  <xs:attribute name="LABEL" type="xs:string" use="optional"/>
  <xs:attribute name="OTHERTYPE" type="xs:string" use="optional"/>
  <xs:attribute name="PARENT" type="xs:string" use="optional"/>
  <xs:attribute name="TYPE" type="xs:string" use="optional"/>
</xs:complexType>
</xs:schema>
```

MARC 21 XML Schema [LOC02c]

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.loc.gov/MARC21/slim" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.loc.gov/MARC21/slim" elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0" xml:lang="en">
  <xsd:annotation>
    <xsd:documentation>
      MARCXML: The MARC 21 XML Schema
      Prepared by Corey Keith
      May 21, 2002 This schema supports XML markup of MARC21 records as specified in the MARC
documentation
      (see www.loc.gov). It allows tags with alphabetic and subfield codes that are symbols, neither of which are
as
      yet used in the MARC 21 communications formats, but are allowed by MARC 21 for local data. The
schema
      accommodates all types of MARC 21 records: bibliographic, holdings, bibliographic with embedded holdings,
authority, classification, and community information.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
  <xsd:element name="record" type="recordType" nillable="true" id="record.e">
    <xsd:annotation>
      <xsd:documentation>record is a top level container element for all of the field elements which compose the
record</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name="collection" type="collectionType" nillable="true" id="collection.e">
    <xsd:annotation>
      <xsd:documentation>collection is a top level container element for 0 or many records</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="collectionType" id="collection.ct">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="record"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="idDataType" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="recordType" id="record.ct">
    <xsd:sequence minOccurs="0">
      <xsd:element name="leader" type="leaderFieldType"/>
      <xsd:element name="controlfield" type="controlFieldType" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="datafield" type="dataFieldType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="recordTypeType" use="optional"/>
    <xsd:attribute name="id" type="idDataType" use="optional"/>
  </xsd:complexType>
  <xsd:simpleType name="recordTypeType" id="type.st">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="Bibliographic"/>
      <xsd:enumeration value="Authority"/>
      <xsd:enumeration value="Holdings"/>
      <xsd:enumeration value="Classification"/>
      <xsd:enumeration value="Community"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="leaderFieldType" id="leader.ct">
    <xsd:annotation>
      <xsd:documentation>MARC21 Leader, 24 bytes</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="leaderDataType">
        <xsd:attribute name="id" type="idDataType" use="optional"/>
        <xsd:attribute ref="xml:space" fixed="preserve"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:simpleType name="leaderDataType" id="leader.st">
    <xsd:restriction base="xsd:string">
      <xsd:whiteSpace value="preserve"/>
      <xsd:pattern value="[d ][5][\dA-Za-z ]{1}[\dA-Za-z ]{1}[\dA-Za-z ]{3}(2) )(2) [\d ][5][\dA-Za-z ]{3}(4500) ]"/>
    </xsd:restriction>
  </xsd:simpleType>

```


Electronic Binding Project (Ebind) - Beispielschemata

tei.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML Schema für den TEI-Namensraum -->
<xs:schema targetNamespace="http://www.tei-c.org" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tei="http://www.tei-c.org" xmlns:eb="http://sunsite.berkeley.edu/Ebind/" elementFormDefault="qualified">
  <xs:import namespace="http://sunsite.berkeley.edu/Ebind/" schemaLocation="ebind.xsd"/>
  <xs:element name="TEI">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="eb:page" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

ebind.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML Schema für den Ebind-Namensraum -->
<xs:schema targetNamespace="http://sunsite.berkeley.edu/Ebind/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eb="http://sunsite.berkeley.edu/Ebind/" xmlns:tei="http://www.tei-c.org" elementFormDefault="qualified">
  <xs:import namespace="http://www.tei-c.org" schemaLocation="tei.xsd"/>
  <xs:element name="page">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tei:div" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Anhang B

Crosswalk MARC21 – EAD [SAA99, S. 241]

MARC21	EAD
041 Language	LANGCODE attribute in <language>
100 Main entry--personal name	<origination><persname> <origination><famname>
110 Main entry--corporate name	<origination><corpname>
111 Main entry--meeting name	<origination><corpname>
130 Main entry--uniform title	<unittitle>
240 Uniform title	<controlaccess><title>
245 Title statement	<unittitle>
245\$f Title statement/inclusive dates	<unitdate type="inclusive">
245\$g Title statement/bulk dates	<unitdate type="bulk">
254 Musical presentation statement	<materialspec>
255 Cartographic mathematical data	<materialspec>
256 Computer file characteristics	<materialspec>
260\$c Date	<unitdate>
300 Physical description	<physdesc> and subelements <extent>, <dimensions>, <genreform>, <physfacet>
340 Physical medium	<phystech>
351 Organization and arrangement	<arrangement>
351\$c Hierarchical level	<archdesc> LEVEL attribute
355 Security classification control	<legalstatus>
500 General note	<odd> <note>
506 Restrictions on access note	<accessrestrict> <legalstatus>
510 Citation/references	<bibliography>
520 Summary, etc.	<scopecontent>
524 Preferred citation of described materials	<prefercite>
530 Additional physical form available	<altformavail>
535 Location of Originals/Duplicates	<originalsloc>
536 Funding information	<sponsor>
538 System Details	<phystech>
540 Terms governing use and reproduction	<userrestrict>
541 Immediate source of acquisition	<acqinfo>
544 Location of other archival materials	<relatedmaterial><separatedmaterial>

545 Biographical or historical data	<bioghist>
546 Language	<langmaterial>
555 Cumulative index/finding aids	(5)
561 Ownership and custodial history	<custodhist>
581 Publications about described materials	<bibliography>
583 Action	<appraisal> <processinfo>
584 Accumulation and frequency of use	<accruals>
600 Subject--personal name	<controlaccess><persname role="subject"> <controlaccess><famname role="subject">
610 Subject--corporate name	<controlaccess><corpname role="subject">
611 Subject--meeting	<controlaccess><corpname role="subject">
630 Subject--uniform title	<controlaccess><title role="subject">
650 Subject--topical	<controlaccess><subject>
651 Subject--geographic name	<controlaccess><geogname role="subject">
655 Genre/form	<controlaccess><genreform>
656 Occupation	<controlaccess><occupation>
657 Function	<controlaccess><function>
69x Local subject access	<controlaccess><subject source="local">
700 Added entry--personal name	<controlaccess><persname> <controlaccess><famname>
710 Added entry--corporate name	<controlaccess><corpname>
711 Added entry--meeting name	<controlaccess><corpname>
720 Added entry--uncontrolled	<name>
730 Added entry--uniform title	<controlaccess><title>
740 Added entry--uncont./related anal. title	<title>
752 Added entry--hierarchical place name	<geogname>
852 Location	<repository><physloc>

Crosswalk Dublin Core - EAD [SAA99, S. 239]

Dublin Core	EAD <eadheader>	EAD <archdesc>
Coverage		<geogname> (spatial) <unitdate> (temporal)
Description	<notestmt><note>	<abstract>

Type		<archdesc> with LEVEL attribute
Relation		
Source		
Subject	<notestmt><subject>	<controlaccess><subject>
Title	<titleproper>	<unititle>
Creator	<author>	<origination><persname> <origination><corpname> <origination><famname>
Contributor	<author>	<origination><persname> <origination><corpname> <origination><famname>
Publisher	<publisher>	<repository>
Rights		
Date	<publicationstmt> <date>	<unitdate>
Format	SGML or XML	
Identifier	<eadid>	<unitid> with COUNTRYCODE and REPOSITORYCODE attributes
Language	<language>	<archdesc> with LANGMATERIAL attribute

Crosswalk MAB2 - DC [SUB03]

MAB	Definition MAB	DC-Element	DC Definition
331 / 310	Hauptsachtitel in Vorlageform oder Mischform / Hauptsachtitel in Ansetzungsform	DC.Title	A name given to the resource
335 / 370	Zusätze zum Hauptsachtitel / Weitere Sachtitel	DC.Title.Alternative	Any form of the title used as a substitute or alternative to the formal title of the resource.
304	Einheitssachtitel	DC.Title.Alternative	Any form of the title used as a substitute or alternative to the formal title of the resource.
376	Normierte Zeitschriftentitel	DC.Title.Alternative	Any form of the title used

			as a substitute or alternative to the formal title of the resource.
670	Sachtitel in abweichender Orthographie	DC.Title.Alternative	Any form of the title used as a substitute or alternative to the formal title of the resource.
104; 108-196	Name der Personen(en)	DC.Creator	An entity primarily responsible for making the content of the resource.
204; 208-296	Name der Körperschaft(en)	DC.Creator	An entity primarily responsible for making the content of the resource.
710 / 711 / 720	Schlagwörter / Schlagwortketten / Stichwörter	DC.Subject	The topic of the content of the resource
711	Schlagwörter und Schlagwortketten nach anderen Regelwerken	DC.Subject	The topic of the content of the resource
711	Schlagwörter und Schlagwortketten nach anderen Regelwerken	DC.Subject	The topic of the content of the resource
9--	Segment RSWK Schlagwortketten	DC.Subject	The topic of the content of the resource
700 b	DDC	DC.Subject	The topic of the content of the resource
700 c	LCC	DC.Subject	The topic of the content of the resource
700 a	UDC	DC.Subject	The topic of the content of the resource
517	Angaben zum Inhalt	DC.Description	An account of the content of the resource
517 c	Inhaltsverzeichnis	DC.Description. tableOfContents	A list of subunits of the content of the resource.
750 / 753 / 756	Inhaltliche Zusammenfassung	DC.Description.abstract	A summary of the content of the resource.
412 / 417 / 418\$g	Name des Verlegers / Weitere Verleger	DC.Publisher	An entity responsible for making the resource available
100 b / 104 b / 108b-	Name der sonstigen beteiligten Person / Weitere beteiligte Personen	DC.Contributor	An entity responsible for making contributions to the content of the

196b			resource.
200 b / 204 b / 208 b - 296 b	Name der sonstigen beteiligten Körperschaft / Weitere beteiligte Körperschaften	DC.Contributor	An entity responsible for making contributions to the content of the resource.
425	Erscheinungsjahr(e)	DC.Date	A date associated with an event in the life cycle of the resource.
400 / 403	Ausgabebezeichnung in normierter Form / in Vorlageform	DC.Date.Modified	Date on which the resource was changed.
536	Voraussichtlicher Erscheinungstermin	DC.Date.Available	Date (often a range) that the resource will become or did become available.
425	Erscheinungsjahr(e)	DC.Date.Issued	Date of formal issuance (e.g., publication) of the resource.
051 / 052	Veröffentlichungsspezifische Angaben zu begrenzten Werken / Veröffentlichungsspezifische Angaben zu fortlaufenden Sammelwerken	DC.Type	The nature or genre of the content of the resource
050 / 651 / 653 / 655\$q	Datenträger / Fußnote zur Computerdatei / Physische Beschreibung der Computerdatei auf Datenträger / Elektronischer Dateiformattyp	DC.Format	The physical or digital manifestation of the resource
433 / 653\$b / 653\$d / 655\$s	Umfangsangabe / Dateiumfang / Physische Größe des Datenträgers / Größe der Datei für eine Computerdatei im Fernzugriff	DC.Format.Extent	The size or duration of the resource.
050 / 655\$q	Datenträger / Elektronischer Dateiformattyp	DC.Format.Medium	The material or physical carrier of the resource.
540-589 / 655	Segment Standardnummern / Elektronische Adresse und Zugriffsart für eine Computerdatei im Fernzugriff	DC.Identifier	An unambiguous reference to the resource within a given context
540	ISBN	DC.Identifier	An unambiguous reference to the resource within a given context
542	ISSN	DC.Identifier	An unambiguous reference to the resource

			within a given context
525	Herkunftsangaben	DC.Source	A Reference to a resource from which the present resource is derived.
516a	Angabe über die Sprache der Vorlage	DC.Language	A language of the intellectual content of the resource
037 b	Sprachencode nach ISO 639	DC.Language	A language of the intellectual content of the resource
530 / 655\$3	Titel von Bezugswerken / Bezugswerk	DC.Relation	A reference to a related resource
407	Kartographische Materialien: Mathmatische Angaben	DC.Coverage.Spatial	Spatial characteristics of the intellectual content of the resoure.
039	Zeitcode	DC.Coverage.temporal	Temporal characteristics of the intellectual content of the resource.
501 / 655\$z / 537 / 655 \$2 / 655\$k / 655\$l / MAB-LOKAL 125	Sammelfeld für unaufgegliederte Fussnoten / Allgemeine Bemerkungen zur Computerdatei im Fernzugriff / Redaktionelle Bemerkungen / Zugriffsmethode / Passwort / Login / Bemerkungen zu den titel- und lokalbezogenen Lokaldaten	DC.Rights	Information about rights held in and over the resource