Umsetzung von Datenmodellen und Methoden bei der Integration spezieller Dokumentenserver in eine IBM-Content-Manager-Umgebung

Diplomarbeit

Universität Rostock
Institut für Informatik

Vorgelegt von:	Sebastian Dolke
Geboren am:	02.03.1977 in Schwerin
Erstgutachter:	Prof. Dr. rer. nat. habil. Andreas Heuer
Zweitgutachter:	Prof. Dr.-Ing. habil. Peter Forbrig
Betreuer:	Dipl.-Ing. Temenushka Ignatova
	Dipl.-Inf. Ilvio Bruder
Abgabetermin:	12. November 2004

Kurzfassung

In der vorliegenden Diplomarbeit werden Konzepte und Verfahren zur Integration von Dokumentenarchiven in eine *IBM Content Manager*-Umgebung entwickelt. Dabei werden zwei Ansätze betrachtet: Zum Einen die Föderierung von Archiven über einen Föderierungsdienst und zum Anderen die Migration von Archiven in eine zentrale Content-Manager-Umgebung.

Als Föderierungsdienst wird der *IBM DB2 Information Integrator for Content* verwendet. In diesem Zusammenhang wird untersucht, welche Voraussetzungen ein Dokumentenarchiv zur Integration erfüllen muss und wie die lokalen Datenstrukturen in den Föderierungsdienst integriert werden können.

Als Zielplattform für das Migrationsszenario wird der *IBM DB2 Content Manager for Multiplatforms* verwendet. Zur Umsetzung der Migration wird ein Algorithmus zur Abbildung unterschiedlicher Datenmodelle auf das Datenmodell des *IBM DB2 Content Manager for Multiplatforms* entwickelt. Darüberhinaus werden ein Verfahren für die Datenmigration und Methoden für die Integration spezieller Funktionen vorgestellt.

Die theoretischen Ergebnisse dieser Arbeit werden abschließend bei der Migration eines Archivs von historischen Notenhandschriften angewendet.

Abstract

In this diploma thesis, concepts and procedures for the integration of document archives in an *IBM Content Manager* environment have been developed. Therefore, two methods of resolution have been examined: on one hand the federation of archives with the help of a federation service and on the other hand the migration of archives to a centralized content manager environment.

For the federation of archives the *IBM DB2 Information Integrator for Content* has been used, therefore the procedure of integrating local data structures into the schema of the federation service has been exemplified. Furthermore requirements that a document archive must match for integration have been surveyed.

As target platform for the migration scenario the *IBM DB2 Content Manager for Multiplatforms* has been used. In this context an algorithm which maps the data models of different platforms to the data model of the *IBM DB2 Content Manager for Multiplatforms* and a procedure for the data migration have been developed. Furthermore techniques for the integration of functions have been analysed.

Finally, the theoretical concepts of this thesis have been applied to the migration of an archive of historical handwritten music scores.

CR-Klassifikation

H.2.1 - Logical Design, H.2.5 - Heterogeneous Databases, H.3.7 - Digital Libraries

Keywords

Migration, Information Integration, IBM Content Manager, Content Management, Digital Libraries

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die mich während der Bearbeitung der vorliegenden Diplomarbeit unterstützt haben. Dabei denke ich besonders an meine Freundin, die vor allem während der letzten Wochen der Bearbeitungszeit viel Rücksicht auf meinen Tagesablauf nahm und mir dadurch zusätzliche Freiräume verschaffte.

Spezieller Dank gilt meinen Betreuern Temenushka Ignatova und Ilvio Bruder, die jederzeit ansprechbar waren und mir bei Problemen und Fragen zur Seite standen. Darüberhinaus danke ich Prof. Heuer und Prof. Forbrig für die Betreuung dieser Diplomarbeit. Bei Heike Frisch bedanke ich mich für die Unterstützung bei der Arbeit mit den AIX-Servern.

Besonderer Dank gilt meinen Korrekturlesern Matthias Nestler und meiner Schwester Kristin Dolke, die viel Zeit investierten um die vorliegende Diplomarbeit kurzfristig zu korrigieren.

Letztendlich danke ich meinen Eltern, die mich während der letzten Jahre fortwährend motiviert und in jeder erdenklichen Weise unterstützt haben und ohne die mein Studium nicht möglich gewesen wäre.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BLOB	Binary Large Object
Client for Windows	IBM DB2 Content Manager Client for Windows
CLOB	Character Large Object
DB2	IBM DB2 Universal Database
DBMS	Datenbank-Management-System
DBS	Datenbanksystem
DTD	Document Type Definition
eClient	IBM DB2 Content Manager eClient
ER-Modell	Entity-Relationship-Modell
FDBS	Foderiertes Datenbanksystem
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Businiss Machines Corporation
ICM	IBM DB2 Content Manager for Multiplatforms
IIC	IBM DB2 Information Integrator for Content
LDAP	Lightweight Directory Access Protocol
LOB	Large Object
MIME	Multipurpose Internet Mail Extensions
NSE	IBM DB2 Net Search Extender
ODMG	Object Data Management Group
OMG	Object Management Group
OODBS	Objektorientiertes Datenbanksystem
ORDBS	Objektrelationales Datenbanksystem
pid	Persistent Identifier
RDBS	Relationales Datenbanksystem
RE	Reverse Engineering
SP	Stored Procedure
SQL	Structured Query Language
TSM	IBM Tivoli Storage Manager
UDF	User Defined Function
UML	Unified Modeling Language
XML	Extensible Markup Language
XMLDBS	XML-Datenbanksystem

Kapitel 1

Einleitung

1.1 Stilkonventionen

In der vorliegenden Diplomarbeit werden folgende typographische Konventionen verwendet, um spezielle Textteile zu kennzeichnen. *Fachbegriffe* werden bei ihrer Definition einmalig durch Kursivschreibung hervorgehoben. *Produktbezeichnungen* und *Eigennamen* werden im Text durchgängig ebenfalls in kursiver Schrift dargestellt. `Schlüsselworte` und `Bezeichner` aus Quelltexten, Softwaresystemen und Diagrammen werden in einer nichtproportionalen Schrift notiert.

1.2 Motivation

In der Vergangenheit wurde Wissen in erster Linie durch Informationen auf Papier weitergegeben, verarbeitet und archiviert. Für die Verwaltung von Archiven war ein beträchtlicher Aufwand an Personal und Räumlichkeiten notwendig. Die Suche nach Informationen und der Zugriff darauf waren relativ aufwendig und nur vor Ort möglich. Die Übermittlung von Inhalten über die traditionellen Wege der Post und Versanddienste war und ist auch heute noch ein zeit- und kostenintensiver Prozess.

Die rasante Entwicklung der Informations- und Kommunikationstechnik während der letzten Jahrzehnte ermöglicht es heute, Archive von Büchern, Zeitschriften und anderen Dokumenten durch einen ungleich geringeren Aufwand zu pflegen und zu verwalten. Informationen werden dabei nicht mehr auf Papier, sondern auf digitalen Speichermedien verwaltet. Datenbank- und Information-Retrieval-Techniken ermöglichen eine gezielte Suche nach Dokumenten und einen effizienten Zugriff. Dabei können nicht nur textuelle Dokumente sondern, auch multimediale Inhalte wie Bild-, Audio- und Videomaterial archiviert werden. Durch die Entwicklung des Internets spielen geographische Entfernungen beim Zugriff auf Dokumente kaum mehr eine Rolle.

Die ständig wachsende Menge digitaler Inhalte hatte zur Folge, dass in den letzten Jahren innerhalb vieler Organisationen immer mehr Archive und Dokumentensammlungen verschiedener Projektgruppen entstanden. Auf diese Weise wurden oft Dokumentenserver auf unterschiedlichen Plattformen geschaffen, die unabhängig voneinander existieren und über die keine archivübergreifende Suche möglich ist. Dies führt dazu, dass Archive zeitaufwen-

dig nacheinander nach Dokumenten durchsucht werden. Zusätzlich müssen Nutzer für jedes Archiv einen Zugang beantragen und sich in verschiedene Oberflächen einarbeiten, da viele Archive nur über spezielle Client-Anwendungen erreichbar sind. Eine zentrale Nutzer- und Rechteverwaltung, sowie ein Zugriff auf alle gespeicherten Dokumente über eine einheitliche Oberfläche würden die Rechercheeffizienz erheblich steigern.

Zum Aufbau von Dokumentenarchiven werden vor allem relationale und objektrelationale Datenbanksysteme eingesetzt. Diese Systeme bieten einen effizienten Zugriff auf Informationen auch bei großen Datenvolumen. Nachteilig ist aber, dass diese Systeme in erster Linie zur Speicherung von Meta-Daten konzipiert wurden und die Speicherung von Dokumenten und Multimedia-Inhalten nur unzureichend unterstützen. Im Allgemeinen werden unstrukturierte Daten in konventionellen Datenbanksystemen in Form von Character Large Objects (CLOB) und Binary Large Objects (BLOB) gespeichert. Wünschenswert wäre eine Speicherung von Dokumenten mit zusätzlichen Informationen wie z.B. dem Dokumententyp. Darüberhinaus ist eine Anbindung des Dokumentenservers an Office-Anwendungen zur direkten Bearbeitung von gespeicherten Dokumenten, die Möglichkeit Multimedia-Dokumente auf spezialisierten Servern (z.B. Streaming Server) zu speichern, sowie eine spezielle Entwicklungsumgebung, die an die Bedürfnisse der Behandlung von unstrukturierten Daten angepasst ist, sinnvoll.

Enterprise-Content-Management-Systeme sind speziell auf die Archivierung und die Verarbeitung von Dokumenten und auf die Integration verschiedener Datenquellen ausgerichtet und können dazu beitragen, die genannten Ziele zu verwirklichen.

1.3 Aufgabenstellung

Die Integration komplexer Archive in eine Content-Manager-Umgebung ist nicht unproblematisch. Vor allem die Integration bzw. Abbildung unterschiedlicher Datenmodelle und die Integration spezieller Funktionen sind nicht trivial.

Im Rahmen dieser Diplomarbeit wird untersucht, inwieweit Enterprise-Content-Management-Systeme geeignet sind, Dokumentenserver verschiedener Plattformen zu integrieren und deren Inhalte über eine einheitliche Oberfläche verfügbar zu machen. Als Content-Management-System wird dabei der *IBM Content Manager* betrachtet. Ziel dieser Betrachtung ist die Konzeption einer Strategie zur Integration bzw. Migration von Dokumentenservern. Dabei wird untersucht, welche Voraussetzungen ein Dokumentenarchiv zur Integration in eine *IBM Content Manager*-Umgebung erfüllen muss und wie die Integration von lokalen Datenstrukturen in einen Föderierungsdienst bzw. die Abbildung von Datenmodellen verschiedener Plattformen auf das Datenmodell des *IBM Content Managers* umgesetzt werden kann. Darüberhinaus werden Lösungsansätze erarbeitet, welche die Integration spezieller Funktionen eines Dokumentenservers, die z.B. in Form von Stored Procedures vorliegen, ermöglichen.

Die theoretischen Ergebnisse dieser Arbeit werden abschließend bei der Migration eines Archivs von historischen Notenhandschriften (dem *eNoteHistory*-Projekt) angewendet.

1.4 Anwendungsszenario

Innerhalb von Universitäten arbeiten häufig mehrere Projektgruppen aus unterschiedlichen Fachbereichen an der Aufbereitung digitaler Inhalte wie z.B. Dissertationen, Diplomarbeiten oder wissenschaftlichen Artikeln. Das Ergebnis jeder Projektgruppe ist in der Regel ein Dokumentenserver, der die gesammelten Inhalte über eine spezielle Client-Anwendung zur Verfügung stellt.

Die Universität Rostock[1] war in den letzten Jahren unter anderem an den Projekten *eNoteHistory*[2], *Mecklenburgische Jahrbücher*[3] und *Dissertation Online* beteiligt. Als Ergebnis entstanden auch hier unabhängige Dokumentenarchive auf verschiedenen Plattformen, die jeweils über spezielle Zugänge abgefragt werden müssen.

Ziel eines neuen Projekts ist nun die Integration der verschiedenen Archivbestände in eine digitale Bibliothek auf Basis einer Content-Management-Lösung. Dabei soll eine zentrale Nutzer- und Rechteverwaltung entstehen, eine bibliothekarische Aufarbeitung der Inhalte erfolgen und ein zentraler Zugriff auf alle Dokumente realisiert werden.

Vor dem Hintergrund dieses Anwendungsszenarios werden in der vorliegenden Diplomarbeit Konzepte und Verfahren entwickelt, um Dokumentenserver verschiedener Plattformen in eine *IBM Content Manager*-Umgebung zu integrieren.

1.5 Aufbau der Arbeit

Die vorliegende Diplomarbeit ist folgendermaßen gegliedert:

Kapitel 2 - Grundlagen erläutert Begriffe und Techniken, die in den folgenden Kapiteln verwendet werden. Darüberhinaus wird das *eNoteHistory*-Projekt vorgestellt, dessen Datenbestand im Rahmen dieser Diplomarbeit in eine *IBM Content Manager*-Umgebung migriert wird.

Kapitel 3 - IBM Enterprise Content Management gibt einen Überblick über die Konzepte und die Systemarchitekturen der in dieser Arbeit verwendeten IBM Content-Management-Produkte.

Kapitel 4 - Methoden zur Archivintegration beschreibt zwei Verfahrensweisen mit denen Dokumentarchive in eine *IBM Content Manager*-Umgebung integriert werden können. Dies ist zum Einen die Föderierung von Archiven mit dem *IBM DB2 Information Integrator for Content* und zum Anderen die Migration von Archiven in den *IBM Content Manager for Multiplatforms*.

Kapitel 5 - Föderierung mit dem IBM DB2 Information Integrator for Content erläutert die Föderierung von Dokumentenarchiven mit dem *IBM DB2 Information Integrator for Content*.

[1] http://www.uni-rostock.de

[2] http://www.enotehistory.de

[3] http://www.uni-rostock.de/ub/UB_PRO_DIGB_MJ.HTM

Kapitel 6 - Migration in den IBM DB2 Content Manager for Multiplatforms erläutert die Schritte der Migration eines Dokumentenservers in den *IBM DB2 Content Manager for Multiplatforms*.

Kapitel 7 - Abbildung der Konstrukte konzeptueller Datenmodelle auf das ICM-Datenmodell definiert Vorschriften zur Abbildung der Konstrukte eines konzeptuellen Datenmodells auf das Datenmodell des *IBM DB2 Content Manager for Multiplatforms*.

Kapitel 8 - Migration des eNoteHistory-Dokumentenservers beschreibt die Migration des Dokumentenservers des *eNoteHistory*-Projekts in den *IBM DB2 Content Manager for Multiplatforms* nach der in Kapitel 6 vorgestellten Verfahrensweise.

Kapitel 9 - Zusammenfassung und Ausblick beendet die vorliegende Diplomarbeit. Es fasst die Ergebnisse dieser Arbeit zusammen und beschreibt mögliche Erweiterungen, die aufbauend darauf realisiert werden können.

Kapitel 2

Grundlagen

Im folgenden Kapitel werden Begriffe und Techniken erläutert, die zum Verständnis der vorliegenden Diplomarbeit notwendig sind. Darüberhinaus wird das *eNoteHistory*-Projekt vorgestellt, dessen Datenbestand, im Rahmen der vorliegenden Arbeit, exemplarisch in eine *IBM Content Manager*-Umgebung integriert wird.

2.1 Das eNoteHistory-Projekt

Das *eNoteHistory*-Projekt ist ein Gemeinschaftsprojekt des Instituts für Musikwissenschaft[1] der Universität Rostock, des Lehrstuhls Datenbank- und Informationssyteme[2] der Universität Rostock und des Fraunhofer IGD Rostock[3]. Das Projekt wird durch die Deutsche Forschungsgemeinschaft gefördert.

Im 18. Jahrhundert wurden Noten überwiegend handschriftlich vervielfältigt. Ein Archiv solcher handgeschriebener Noten, die sogenannte *Rostocker Sammlung* (siehe [Kr"03]), befindet sich im Besitz der Universitätsbibliothek Rostock. Ziel des *eNoteHistory*-Projekts ist die Veröffentlichung dieser Notenhandschriften sowie die Unterstützung von Musikwissenschaftlern bei der Identifizierung von Schreibern.

Dazu wurde ein Dokumentenserver zur Speicherung der digitalisierten Notenhandschriften aufgebaut, der auf einer objektrelationalen Datenbank (*IBM DB2 Universal Database*) basiert. Neben bibliografischen Daten wurden mit Hilfe des Instituts für Musikwissenschaft auch Handschriftcharakteristiken zu jedem Notenblatt erfasst. Dadurch ist es möglich Notenhandschriften nicht nur an Hand von Meta-Daten zu identifizieren, sondern auch nach Schreibern zu suchen. Diese Funktion wird als Schreibererkennung bezeichnet. Zur Realisierung der Schreibererkennung wurden Funktionen zur Berechnung des Abstands zwischen zwei Mengen von Handschriftcharakteristiken in den Dokumentenserver integriert (siehe [Mil04]). Der Abstand ist ein Maß für die Ähnlichkeit zweier Handschriften. Die Funktionen ermöglichen es, bei gegebenen Handschriftcharakteristiken Dokumente zu finden, die diesen Eigenschaften möglichst gut entsprechen.

[1] http://www.uni-rostock.de/fakult/philfak/fkw/imu/
[2] http://wwwdb.informatik.uni-rostock.de/
[3] http://www.igd-r.fraunhofer.de/IGD

In Abbildung C.3 (siehe Anhang) ist das konzeptuelle Datenmodell des Datenbestands des *eNoteHistory*-Dokumentenservers dargestellt. Folgende Zusammenhänge sind erkennbar: Die übergeordnete Dokumenteneinheit ist das musikalische Werk (`Music_Work`), dem sowohl ein Komponist (`Composer`), als auch ein Textautor (`Text_Author`) zugeordnet ist. Ein musikalisches Werk besteht aus Musikmanuskripten (`Music_Manuscript`), welche wiederum aus Sektionen (`Music_Manuscript_Section`) bestehen, die einzelne Seiten (`Music_Manuscript_Page`) enthalten. Zu jeder Manuskriptseite werden Meta-Daten und die digitalisierten Bilder der Notenhandschrift (`Page_Image`) gespeichert. Jeder Manuskriptseite ist ein Schreiber (`Scribe`) zugeordnet. Die Handschriftcharakteristik eines Schreibers einer Notenseite ergibt sich aus den einzelnen Merkmalen seiner Handschrift. Diese Informationen werden in einem Vektor (`Feature_Vector`) verwaltet. Die möglichen Merkmale einer Handschrift werden im Entity-Typ `Node` gespeichert und sind hierarchisch organisiert. Zwischen zwei Merkmalen wurde ein Abstand definiert, der die Ähnlichkeit angibt.

Eine ausführliche Beschreibung des *eNoteHistory*-Projektes befindet sich in [GVK+03]. Weitere Information sowie zusätzliche Artikel und Vorträge können auf der Projekt-Homepage (siehe [eP04]) gefunden werden.

2.2 Content Management

Der Begriff „Content Management“ ist in den letzten Jahren zu einem beliebten Schlagwort der Marketingabteilungen vieler Unternehmen der IT-Branche geworden. Wurde früher von Document Management Systemen, Redaktionssystemen und Groupware-Lösungen gesprochen, so werden viele dieser Produkte heute als Content-Management-System bezeichnet. Der schnell wachsende Markt bringt zusätzlich immer neue Produkte diverser Hersteller hervor, welche oft über eine sehr unterschiedliche Funktionalität verfügen. Dabei werden die Kunden mit einer Fülle von Akronymen und Begriffen wie „Enterprise Content Management“ (ECM), „Web Content Management“ (WCM), „Document Management“ (DM), „Document Lifecycle Management“ (DLM), „Records Management“ (RM), „Knowledge Management“ (KM) und „Information Lifecycle Management“ (ILM) konfrontiert. Erschwerend kommt hinzu, dass keine einheitliche Interpretation dieser Begriffe existiert. Entsprechend schwierig ist es, eine allgemeingültige Definition für den Begriff Content Management zu finden. Die folgenden Ausführungen beruhen auf den Definitionen aus [Kam03] und [Wer01].

2.2.1 Content

Zur Definition des Begriffs Content Management, muss zunächst der Begriff Content definiert werden.
„*Content* (engl. Inhalt) ist Information in strukturierter, schwach strukturierter und unstrukturierter Form, die in elektronischen Systemen zur Nutzung bereitgestellt wird.“ [Kam03]

- Als *strukturierter Content* werden Daten bezeichnet, die durch das Schema eines Datenbanksystems beschrieben werden können. Ein Beispiel für ein strukturiertes Datenobjekt ist ein Datensatz aus einer relationalen Datenbank.

- *Semistrukturierter Content* besteht aus Daten, die keine formale Typisierung (kein Schema) aber eine implizite Struktur (selbst beschreibend) besitzen. Die Struktur der Daten muss bei der Verarbeitung extrahiert werden. Ein Beispiel für semistrukturierte Daten sind Dateien, die von Textverarbeitungsprogrammen erstellt werden.

- Als *unstrukturierter Content* werden Daten bezeichnet, deren Inhalt auf Grund der fehlenden Struktur nicht erschlossen werden kann. Typische Beispiele dafür sind digitale Bilder, sowie Audio- oder Video-Dateien.

2.2.2 Content Management

Content Management ist ein Prozess, „...der als Teilschritte die systematische Beschaffung, Erzeugung, Aufbereitung, Verwaltung, Präsentation, Verarbeitung, Publikation und Wiederverwendung von Inhalten umfasst.“[Was02]
In den meisten Definitionen wird Content Management nicht als wohl definiertes Segment im Markt für Unternehmenssoftware betrachtet, sondern als Aktivität die alle Schritte der Verarbeitung von Content umfasst.

2.2.3 Content-Management-System

„Ein *Content-Management-System* stellt die systemtechnischen Grundlagen für das Content Management zur Verfügung. Man versteht darunter ein computergestütztes Erstellungs-, Verwaltungs- und Archivierungssystem für den Content in den unterschiedlichsten digitalen Formaten.“ [Ber04]

2.2.4 Klassifikation von Content-Management-Systemen

Innerhalb des Marktes von Content-Management-Systemen sind verschiedene Strömungen erkennbar, deren Produkte jeweils spezielle Anwendungsszenarien unterstützen. Eine allgemeine Klassifikation der Systeme ist aber auf Grund der verschiedenen Begriffsinterpretationen der Hersteller, des unterschiedlichen Funktionsumfangs der angebotenen Systeme und der Dynamik des noch jungen Marktes schwierig. Je nach Literatur (z.B. in [ZTZ04], [Kir02]) werden verschiedene Kategorien von Content-Management-Systemen abgegrenzt. In [Kam03] wird grob in drei Gruppen unterteilt:

Content Management (CM) und Content Syndication (CS)

In diese Gruppe fallen alle Systeme, die Content Management im engeren Sinne ermöglichen. Man spricht in diesem Zusammenhang auch von *Content Syndication*. Ziel ist es Verlegern bei der Verwaltung, der Abrechnung, dem Schutz und der Verteilung von digitalen Büchern, Bildern, Musikstücken und Videos zu unterstützen. Dementsprechend spielen in solchen Systemen Techniken wie Digital Rights Management (DRM), digitale Wasserzeichen, Kopierschutzmechanismen, komprimierte Bereitstellung von Inhalten und Nutzungsabrechnung eine wichtige Rolle. Im Wesentlichen geht es hierbei um die kommerzielle Nutzung und Vermarktung von Content.

Web Content Management (WCM)

Der größte Teil der Content-Management-Systeme gehört zur Gruppe der Web-Content-Management-Systeme. In *Web-Content-Management-Systemen* beschränkt sich der Veröffentlichungsprozess auf Internet-Formate. Der Ursprung dieser Systeme liegt im Bereich der dynamischen Gestaltung von Internetauftritten. Ziel dieser Systeme ist es, die Nachteile von statischen HTML-Seiten zu überwinden. Dazu bieten Web-Content-Management-Systeme Funktionen zur Versionierung und zur effizienten Pflege von Web Sites, zur Integration von eCommerce mit Bezahlfunktionalität und zur Unterstützung von editorischen Prozessen. Auch Autoren- und Redaktionssysteme können dieser Gruppe zugeordnet werden.

Web-Content-Management-Systeme sind zu einem großen Teil mit verantwortlich für den „Hype“, den die Content Management Branche in den letzten Jahren erlebte. Vielfach wird der Begriff Content Management daher auch etwas engstirnig mit Web Content Management gleichgesetzt.

Enterprise Content Management (ECM)

Enterprise-Content-Management-Systeme sind Software-Systeme zur Erfassung, Verwaltung, Speicherung, Bereitstellung und Archivierung von Informationen zur Unterstützung von Geschäftsprozessen. (vgl. [Kam03])
Systeme dieser Art umfassen herkömmliche Informationstechnologien wie Document Management, Knowledge Management, Workflow und Archivierung. Sie dienen der Erschließung von allen strukturierten und unstrukturierten Informationen im Unternehmen auf einer einheitlichen Plattform, so dass die entsprechenden Daten und Dokumente redundanzfrei intern, im Partnerverbund und extern verfügbar gemacht werden können. Unter dem Begriff Enterprise Content Management werden daher Lösungen zusammengefasst, die zwar auch Internettechnologien unterstützen, in erster Linie aber bei der unternehmensinternen Informationsaufbereitung eingesetzt werden.

2.2.5 Einordnung des IBM Content Managers

Der in dieser Arbeit zur Integration von Dokumentenarchiven verwendete *IBM Content Manager* gehört auf Grund seiner Funktionalität in die Kategorie der Enterprise-Content-Management-Systeme. Die Stärken des Systems liegen zum einen in der Repository-Komponente, die eine verteilte Speicherung von Content erlaubt und zum anderen in der Workflow-Funktionalität. Der *IBM Content Manager* bietet damit eine Grundlage zur Organisation von Geschäftsprozessen und zur Speicherung von Geschäftsdokumenten jeder Art.

Die Publikation von Content wird dagegen nur unzureichend unterstützt. In diesem Bereich wird besonders deutlich, dass der Ursprung und der Fokus des Systems im Bereich des Document Managements liegt. Während Content-Management-Systeme die Veröffentlichung von Content in verschiedenen Formaten und für verschiedene Verwendungszwecke unterstützen sollten, bietet der *IBM Content Manager* lediglich die Möglichkeit, Dokumente zu exportieren und zu drucken. Daher wird das System in [Wer01] auch kritisch als „Document Management System mit integriertem Workflow“ bezeichnet.

2.3 Information Integration

In Anwendungsszenarien in denen große Mengen von Daten verwaltet werden, sind diese Daten üblicherweise auf mehrere Standorte verteilt und werden oft in heterogenen Systemen gespeichert. Unter dem Begriff *Information Integration* werden Methoden zur Schema-Integration zusammengefasst, die verteilte Daten aus mehreren Quellen zusammenbringen, so dass diese Daten über ein einheitliches Interface abgefragt und verarbeitet werden können. Ziel des Information Integration ist es, eine ganzheitliche Sicht auf Daten zu erzeugen, die in verschiedenen Formaten in heterogenen Systemen gespeichert sind und auf die über unterschiedliche Interfaces zugegriffen wird.

Eine Möglichkeit diese Ziele zu erreichen, besteht in der Föderierung von Datenquellen. *Föderierungssysteme* sind Middleware-Lösungen, die es Applikationen ermöglichen, auf verteilte Daten zuzugreifen, als wären diese in einer einzigen Datenbank gespeichert, unabhängig vom Datenformat, dem Speicherort und der Anfragesprache. Die Autonomie und Integrität der einzelnen Datenquellen bleibt dabei vollständig erhalten und lokale Anwendungen können trotz der Föderation weitergenutzt werden.

Ein System, dass auf diese Weise arbeitet und das in der vorliegenden Diplomarbeit verwendet wird, ist der *IBM DB2 Information Integrator for Content*. Weitere Ausführungen zum Thema Information Integration befinden sich in [Mat03].

2.4 Die Unified Modeling Language (UML)

Für die Beschreibung und Notation von Datenmodellen wird in der vorliegenden Diplomarbeit die Unified Modeling Language (UML) verwendet. Dieses Kapitel gibt einen Überblick über die Konzepte dieser Modellierungssprache.

2.4.1 Einführung

Die *Unified Modeling Language (UML)* ist eine Sprache und Notation zur Modellierung, Visualisierung, Beschreibung und Dokumentation von Softwaresystemen. Die Sprache umfasst eine Sammlung von Diagrammtypen zur Erstellung von Anforderungs- und Entwurfsmodellen aus verschiedenen Perspektiven. UML ist ein offener Standard, der den gesamten Softwareentwicklungszyklus unterstützt. Durch die Standardisierung wird Entwicklern die Möglichkeit gegeben, den Entwurf und die Entwicklung von Softwaremodellen in einer einheitlichen Darstellung zu diskutieren.

Entstehung

Die UML wurde von Grady Booch, Ivar Jacobsen und Jim Rumbaugh von Rational Rose Software[4] entwickelt. In der Sprache wurden mehrere damals populäre objektorientierte Modellierungsmethoden zusammengefasst. 1994 begannen Booch und Rumbaugh ihre Ansätze OOAD (Object Oriented Analysis and Design) und OMT (Object Modeling Technique) zu vereinen, später brachte Jacobson die Konzepte von OOSE (Object Oriented Software Engineering) in den neuen Standard ein. Die Entwicklung von UML wird

[4] www.rational.com

durch ein Konsortium namhafter Firmen unterstützt, an dem sich unter anderem Microsoft, Oracle und Hewlett-Packard beteiligten. Die UML wurde 1997 bei der Object Management Group[5] (OMG) als Vorschlag für einen Standard zur objektorientierten Modellierung eingereicht und noch im selben Jahr akzeptiert. Die OMG hält heute die Rechte an der UML und entwickelt die Sprache weiter. Die bislang letzte Version der UML, Version 1.5, wurde im März 2003 verabschiedet. Die UML 2.0 befindet sich momentan im Standardisierungsprozess. Die Entstehung der UML ist in Abbildung 2.1 dargestellt.

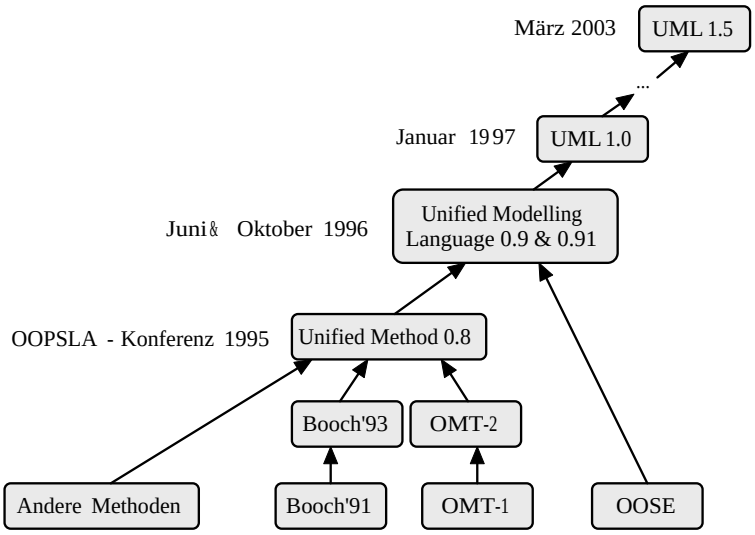


Abbildung 2.1: Historische Entwicklung der UML

Diagrammtypen

Die UML umfasst sieben Typen von Diagrammen, mit denen verschiedene Sichten auf ein System modelliert werden können. Es existieren Diagramme sowohl für die Beschreibung des statischen Aufbaus eines Softwaresystems, als auch zur Modellierung von dynamischen Abläufen. Dabei ist eine Diagrammart meistens besser zur Darstellung eines bestimmten Sachverhalts geeignet als andere. Oft kann derselbe Sachverhalt aber auch mit verschiedenen Diagrammtypen modelliert werden. In der Tabelle B.2 (siehe Anhang) sind alle Diagrammtypen der UML sowie deren Einsatzgebiete beschrieben.

2.4.2 Das Klassendiagramm

Für die Datenbankwelt und für die Beschreibung von Datenmodellen ist vor allem das Klassendiagramm der UML interessant. *Klassendiagramme* beschreiben die Struktur von Klassen sowie die Beziehung zwischen Objekten. Damit kann der statische Aufbau von Datenstrukturen in einem Softwaresystems dargestellt werden.

Nachfolgend werden alle Elemente von Klassendiagrammen beschrieben, die zur Darstellung von Datenmodellen in der vorliegenden Diplomarbeit verwendet werden. Dabei

[5] www.omg.com

wird von der UML Spezifikation 1.5 ausgegangen. Eine ausführliche Beschreibung des UML-Klassendiagramms befindet sich unter anderem im OMG-Standard (siehe [OMG03]) sowie in [BRJ99] und [WO04].

Die Basiselemente von Klassendiagrammen sind Klassen und Objekte. Bei der Beschreibung der Datenmodelle innerhalb dieser Arbeit werden aber ausschließlich Klassen dargestellt.

> Ein *Objekt* ist ein individuell erkennbares, von anderen Objekten eindeutig unterscheidbares Element des Problembereiches. (vgl. [Gli99])

> „Eine *Klasse* ist eine Menge von Objekten, in der die Eigenschaften (Attribute), Operationen und die Semantik der Objekte definiert werden. Alle Objekte einer Klasse entsprechen dieser Festlegung.“ [Dum01]

Klassen

Klassen werden in der UML durch Rechtecke dargestellt, die in drei Sektionen unterteilt sind. Die oberste Sektion enthält den Klassennamen, die mittlere die Attribute der Klasse und die unterste die Operationen der Klasse. Der Klassenname steht im Singular und wird mit einem großen Buchstaben begonnen. Attribute können durch einen Typ und einen Initialwert näher beschrieben werden. Zu einer Operation können zusätzlich zum Namen Parameter und ein Rückgabetyp angegeben werden. Besitzt eine Klasse keine Attribute bzw. Operationen so wird die entsprechende Sektion entweder leer gelassen oder gar nicht dargestellt. Abbildung 2.2 zeigt die Notation einer Klasse in UML.

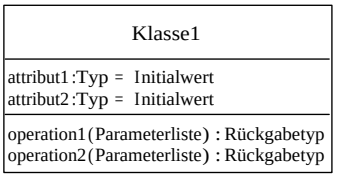


Abbildung 2.2: Darstellung einer Klasse in UML

Attribute

Attribute beschreiben die Objekte einer Klasse. Die durch eine Klassendefinition festgelegten Attribute sind in jedem Objekt dieser Klasse vorhanden. Attribute werden wie folgt definiert:

Sichtbarkeit name : Typ [Multiplizität] = Initialwert

Die Sichtbarkeit von Attributen kann durch die Sichtbarkeitskennzeichner `public`, `protected`, `private` und `package` eingeschränkt werden. Die Sichtbarkeitsangaben werden durch ein dem Attributnamen vorangestelltes + für `public`, # für `protected`, - für `private` und ~ für `package` gekennzeichnet.

Über die Multiplizität wird festgelegt, wie oft ein Attribut in einer Klasse auftritt. Die Multiplizität wird dabei in Intervallschreibweise `[untere Grenze .. obere Grenze]` angegeben.

Operationen

Operationen sind Dienstleistungen, die durch ein Objekt genutzt werden können. Die Signatur einer Operation wird in der Klasse des Objekts definiert. Neben dem Operationsnamen kann eine Liste von Parametern sowie ein Rückgabetyp festgelegt werden. Für die Sichtbarkeit von Operationen gilt dasselbe wie für die Sichtbarkeit von Attributen. Die Syntax für Operationen lautet:

Sichtbarkeit name (Parameterliste) : Rückgabetyp

Generalisierung

Als *Generalisierung* wird die Beziehung zwischen einem allgemeinen Element und einem oder mehreren spezialisierten Elementen bezeichnet. Generalisierungen werden vor allem zur hierarchischen Strukturierung von Klassen verwendet, dabei werden die Eigenschaften einer Oberklasse an die assoziierten Unterklassen vererbt. Eine Unterklasse verfügt dadurch sowohl über ihre spezielle Funktionalität und Eigenschaften, als auch über die Funktionalität und die Eigenschaften der Oberklasse.

Eine Generalisierung wird durch einen nicht ausgefüllten Pfeil von der Unterklasse zur Oberklasse dargestellt. Haben verschiedene Unterklassen eine gemeinsame Oberklasse und einen gemeinsamen Diskriminator, so können die Pfeile zu einer Linie zusammengefasst werden (Shared Target Style). Der Diskriminator definiert die für die Strukturierung der Klassen maßgebliche Eigenschaft, beispielsweise kann man Bücher nach dem Diskriminator „Buchtyp“ in Fachbuch, Roman und Lexikon gliedern. Abbildung 2.3 zeigt die Darstellung des Generalisierungskonzepts in UML.

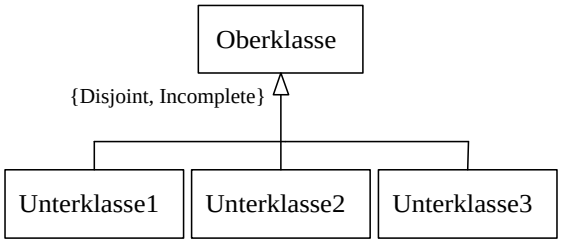


Abbildung 2.3: Darstellung einer Generalisierung in UML (Shared Target Style)

Folgende Zusicherungen können einer Generalisierung zugeordnet werden, um die Generalisierung näher zu charakterisieren:

- *overlapping (überlappend):* Die Eigenschaft `overlapping` kann in einer Generalisierungshierarchie auf zwei Arten interpretiert werden:
  - Ein Objekt kann gleichzeitig Instanz von mehreren Unterklassen sein.

  - Ein Element kann mehrere Unterklassen mit demselben Diskriminator als Vorgänger haben.

- *disjoint (disjunkt):* Die Eigenschaft `disjoint` kennzeichnet eine Generalisierungshierarchie in der die `overlapping`-Eigenschaften nicht erlaubt sind, das heißt, kein Objekt kann gleichzeitig Instanz von mehreren Unterklassen sein bzw. kein Element hat mehrere Unterklassen mit demselben Diskriminator als Vorgänger.

- *complete (vollständig):* Alle Unterklassen einer allgemeinen Klasse sind bereits definiert, unabhängig davon, ob sie im Diagramm dargestellt sind oder nicht. Es werden keine weiteren Unterklassen mehr ergänzt.

- *incomplete (unvollständig):* Einige Unterklassen der Generalisierungshierarchie wurden bereits spezifiziert, es ist aber bekannt, dass weitere Unterklassen existieren.

Eine Generalisierung ist disjunkt (`disjoint`) und vollständig (`complete`), falls keine Angabe über den Charakter der Generalisierung im Diagramm notiert ist. Für die Eigenschaften `complete`/`incomplete` existiert außerhalb der UML eine weitere Interpretation, nach der in einer vollständigen Generalisierung jedes Objekt einer Oberklasse mindestens einer Unterklasse angehören muss. Diese Interpretation von `complete` ist im UML-Standard jedoch nicht vorgesehen.

Assoziationen

Assoziationen beschreiben Beziehungen zwischen Klassen bzw. zwischen deren Objekten. Über die Multiplizität einer Assoziation wird festgelegt, mit wie vielen Objekten der assoziierten Klasse ein Objekt in Beziehung steht. Die Angabe der Multiplizität erfolgt, wie auch bei Attributen, in Intervallschreibweise, allerdings ohne umschließende eckige Klammern. Für jede Seite der Assoziation kann ein Rollenname vergeben werden, der die Rolle der jeweiligen Objekte in der Beziehung beschreibt.

Assoziationen werden durch eine Linie zwischen den an der Beziehung beteiligten Klassen dargestellt. Durch einen Pfeil am Ende der Linie wird eine gerichtete Beziehung modelliert. Über der Linie kann in kursiver Schrift ein Name notiert werden, der die Beziehung beschreibt. Neben dem Namen kann ein ausgefülltes Dreieck die Leserichtung anzeigen.

Einer Assoziation können Attribute zugeordnet werden, welche die Beziehung beschreiben. Diese Beziehungsattribute sind abhängig von der Existenz der Beziehung. Die Attribute werden innerhalb einer Klasse dargestellt, die aber keine eigenständigen Objekte beschreibt. Die Klasse wird über eine gestrichelte Linie mit der Assoziation verbunden. In Abbildung 2.4 ist die UML-Notation von Assoziationen dargestellt.

Aggregation und Komposition

Durch eine *Aggregation* wird eine Teil-Ganzes-Beziehung modelliert, das heißt, dass sich ein Objekt (das Aggregat) aus mehreren anderen Objekten zusammensetzt. Ein Objekt kann dabei zu mehreren Aggregaten gehören. Die Aggregation wird durch eine Linie zwischen den beteiligten Klassen dargestellt. Auf der Seite des Aggregats wird die Linie mit einer nicht ausgefüllte Raute versehen.

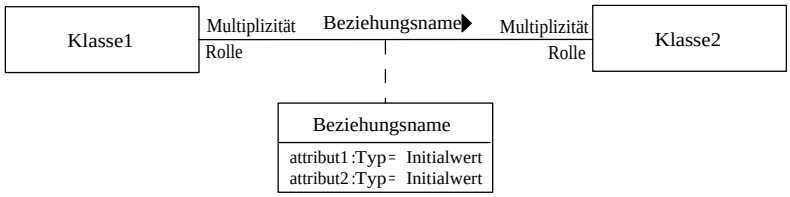


Abbildung 2.4: UML-Notation von Assoziationen

Die *Komposition* ist eine strengere Form der Aggregation, bei der die Einzelteile existenzabhängig vom Ganzen sind. Jedes Teil kann dabei nur an einer Komposition beteiligt sein. Die Lebensdauer der Einzelteile ist bei der Komposition an die Lebensdauer des Ganzen gebunden, das heißt, dass ein Teil nicht unabhängig existieren kann. Die Darstellung einer Komposition entspricht der einer Aggregation, allerdings mit einer ausgefüllten Raute. Abbildung 2.5 zeigt eine Aggregation und eine Komposition in UML-Notation.

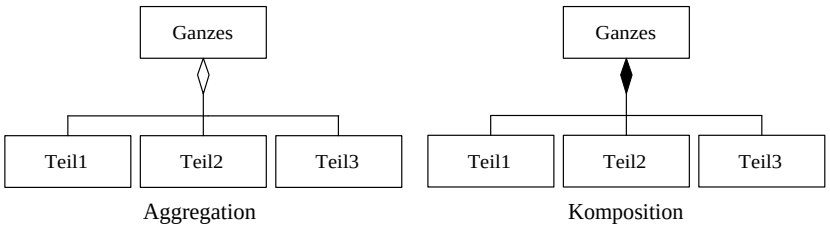


Abbildung 2.5: UML-Notation von Aggregation und Komposition

2.4.3 Erweiterungsmechanismen

Ein Vorteil der Unified Modeling Language gegenüber anderen Modellierungsmethoden sind die Konzepte zur Erweiterung der Sprache. Die folgenden Elemente können auf jedes Modellelement angewendet werden.

Zusicherungen (Constraints)

Durch *Zusicherungen* können Bedingungen oder Integritätsregeln für ein Modellelement festgelegt werden. Zusicherungen fordern oder verbieten bestimmte Eigenschaften, so kann z.B. die Wertemenge eines Attributs eingeschränkt werden. Die Formulierung von Zusicherungen kann in natürlicher oder formaler Sprache erfolgen. Eine vordefinierte formale Sprache ist die Object Constraint Language (OCL), die ebenfalls im UML-Standard beschrieben wird.

Eine Zusicherung kann zu jedem Modellelement angegeben werden und wird in geschweiften Klammern hinter dem Element notiert. Wird durch eine Zusicherung eine Abhängigkeit zwischen zwei Elementen beschrieben, so wird die Zusicherung durch eine gestrichelte Linie mit den entsprechenden Elementen verbunden. Ist dabei ein Element vom anderen abhängig, so wird dies durch einen Pfeil in Richtung des abhängigen Elements dargestellt.

Elementeigenschaften (Element Properties)

Durch *Elementeigenschaften* kann die Semantik von Modellelementen um zusätzliche Eigenschaften erweitert werden. Die Definition von Eigenschaften erfolgt durch ein Schlüsselwort-Wert-Paar. Das Schlüsselwort mit dem dazugehörigen Wert wird in geschweiften Klammern hinter dem entsprechenden Element notiert. Bei mehreren Eigenschaften werden diese durch Kommas getrennt. Ein Beispiel für eine Elementeigenschaft ist eine Versionsangabe, etwa {`Version=8.1`}. Bei Boolsche Ausdrücken können die Werte weggelassen werden, falls der Ausdruck wahr ist, das heißt {`veraltet=true`} entspricht {`veraltet`}.

Stereotypen

Mit *Stereotypen* können die vorhandenen Modellelemente (Klassen, Attribute, Assoziationen) der UML erweitert werden. Stereotypen repräsentieren eine Unterklasse eines vorhandenen Elements des Metamodells mit derselben Form, aber mit einer anderen Verwendung. Auf diese Weise können neue UML-Elemente definiert werden, welche dieselbe Form, aber eine andere Semantik als die Basiselemente haben. Zu einem Stereotyp können zusätzliche Zusicherungen sowie obligatorische Elementeigenschaften definiert werden. Stereotypen sind die Basis für Erweiterungen der UML und für die Erstellung von UML-Profilen.

Stereotypen verwenden im Allgemeinen die Notation ihres Basiselements, die um den Namen des Stereotyps erweitert wird, der über oder vor dem Elementnamen notiert wird. Der Name des Stereotyps wird dabei durch doppelte „größeralls“- und „kleineralls“-Zeichen eingerahmt. Zusätzlich kann einem Stereotyp ein Icon oder eine grafische Markierung (z.B. eine Textur oder Farbe) zugeordnet werden. Abbildung 2.6 zeigt verschiedenen Möglichkeiten zur Notation desselben Stereotyps.

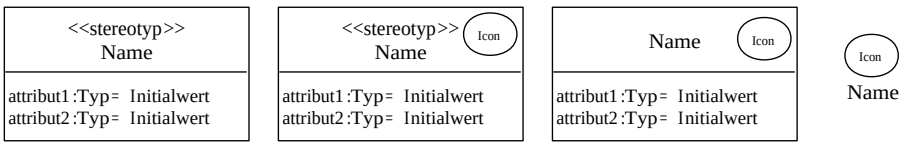


Abbildung 2.6: Darstellung eines Stereotyps in UML

2.4.4 UML-Profile

Ein *UML-Profil* besteht aus einer Menge von Stereotypen, Elementeigenschaften und Zusicherungen, mit denen die Standard-UML-Elemente zur Modellierung von speziellen Anwendungsgebieten erweitert werden. Das UML-Meta-Modell wird dabei nicht verändert. Auf diese Weise können Sammlungen verschiedener Elemente mit obligatorischen Elementeigenschaften, zusätzlichen Zusicherungen und speziellen Notationen zur Modellierung von Domänen erstellt werden, die mit der Semantik der Standardelemente nicht dargestellt werden können.

In der vorliegenden Diplomarbeit werden Profile zur Darstellung des Datenmodells des *IBM DB2 Content Managers for Multiplatforms* (siehe 3.2.7) und zur Modellierung von

konzeptuellen Datenmodellen (siehe 2.5.2) definiert. Weiterhin wird zur Notation von relationalen Datenmodellen das *UML Data Modeling Profile* (siehe Anhang A) der Firma Rational verwendet.

2.5 Konzeptuelle Datenmodellierung mit UML

2.5.1 Konzeptuelle Datenmodelle

Ein *konzeptuelles Datenmodell* beschreibt Objekte des Anwendungsszenarios und die Beziehungen zwischen diesen Objekten auf einem abstrakten Niveau mit einer anwendungsnahen Semantik, unabhängig von einem konkreten Software- oder Datenbanksystem. Durch die übersichtliche grafische Darstellung und die reiche Semantik sind Datenstrukturen, die in konzeptuellen Datenmodellen dargestellt sind, leichter zu verstehen und aussagekräftiger als Datenstrukturen, die durch ein Implementierungsmodell, wie z.B. dem relationalen Datenmodell, beschrieben werden. Auf Grund dieser Eigenschaften werden konzeptuelle Datenmodelle vor allem für den Entwurf von Datenbanken verwendet, sie können aber für Dokumentationszwecke genutzt werden.

Für die Darstellung von konzeptuellen Datenmodellen sind Entity-Relationship-Modelle (ER-Modelle), die auf einen Artikel von Peter P. Chen (siehe [Che76]) zurückgehen, weit verbreitet. Das *Entity-Relationship-Modell* basiert auf drei Grundkonzepten: Dem Entity als zu modellierende Informationseinheit, Beziehungen die zwischen Entities bestehen und Attributen, die Entities oder Beziehungen beschreiben. *Entities* sind beliebige abstrakte oder reale Dinge des Anwendungsszenarios. Entities mit gleicher Struktur werden zu einem Entity-Typ zusammengefasst. *Entity-Typen* dienen, ähnlich wie Klassen für Objekte, als Vorlage bei der Erzeugung von Entities und definieren die Struktur aller Entities eines Typs. *Schlüsselattribute* besitzen die Eigenschaft, durch ihre Werte die Instanzen eines Entity-Typs eindeutig zu bestimmen. Neben gewöhnlichen starken Entities werden abhängige bzw. schwache Entities unterschieden. Ein *schwaches Entity* kann nur in Abhängigkeit eines anderen Entity existieren, es muss also immer in Beziehung mit einem anderen Entity stehen. Der Schlüssel eines schwachen Entity-Typs setzt sich aus dem Schlüssel des Entity-Typs, von dem dieser abhängig ist und eventuell weiteren lokalen Attributen zusammen, die als *partieller Schlüssel* bezeichnet werden.

In der vorliegenden Diplomarbeit wird unter anderem untersucht, wie spezielle Funktionen von Dokumentenarchiven in eine *IBM Content Manager*-Umgebung integriert werden können. Da das ER-Modell kein Konzept zur Modellierung von Operationen bietet und auch keine Erweiterungsmöglichkeiten zur Verfügung stehen, wird statt dessen die Notation der Unified Modeling Language für die Beschreibung von konzeptuellen Datenmodellen verwendet. Dabei werden die Konzepte des klassischen ER-Modells durch UML-Konstrukte nachgebildet und erweitert. Die UML wurde gewählt, da diese Notation standardisiert, gut dokumentiert und weithin akzeptiert ist. Um alle Konstrukte des klassischen ER-Modells darstellen zu können, werden im nächsten Unterabschnitt einige Erweiterungen vorgestellt, die zu einem UML-Profil zur konzeptuellen Datenmodellierung zusammengefasst werden.

2.5.2 Ein UML-Profil zur Darstellung konzeptueller Datenmodelle

Als Basis zur Modellierung von konzeptuellen Datenmodellen werden UML-Klassendiagramme verwendet. In [Amb04] wird von Scott W. Ambler ein UML-Profil zur Datenmodellierung beschrieben. In diesem Profil werden unter anderem Stereotypen zur Modellierung von konzeptuellen Datenmodellen definiert. Leider ist es mit diesem Profil nicht möglich, partielle Schlüssel eines schwachen Entity-Typs darzustellen. Im Folgenden wird daher ein UML-Profil vorgestellt, welches das in [Amb04] beschriebene Profil erweitert. Die Definition des UML-Profils befindet sich im Anhang B.4.

Zur Modellierung von Entity-Typen wird der Stereotyp `entity` und für die Darstellung von schwachen Entity-Typen der Stereotyp `weak entity` definiert. Beide Stereotypen erweitern das UML-Element `Klasse`.

Für die Modellierung von Beziehungen zwischen Entity-Typen werden die Standardbeziehungstypen der UML: Assoziation, Aggregation und Komposition verwendet. Schwache Entity-Typen erweitern die Semantik von Kompositionen um partielle Schlüssel, die zur Identifizierung eines Entitys in Abhängigkeit von einem übergeordneten Entity genutzt werden. Da schwache Entity-Typen die durch eine Komposition modellierte Abhängigkeit von einem übergeordneten Entity-Typ beinhalten, wird zur Darstellung der Beziehung zwischen einem schwachen Entity-Typ und einem übergeordneten Entity-Typ eine Komposition verwendet.

Für die Modellierung von Schlüsseln steht in UML-Klassendiagrammen kein Konzept zur Verfügung, daher wird der Stereotype `K` (für key) und speziell für schwache Entity-Typen der Stereotyp `PK` (für partial key) definiert. Beide Stereotypen werden vom UML-Element `Attribut` abgeleitet. Da mehrere Schlüssel für einen Entity-Typ definiert und neben einfachen Schlüsseln auch zusammengesetzte Schlüssel erstellt werden können, werden zusätzlich die Elementeigenschaften `key` und `order` definiert. Die Eigenschaft `key` legt fest, zu welchem Schlüssel ein Attribut gehört. Die Eigenschaft `order` beschreibt die Position eines Attributs innerhalb eines zusammengesetzten Schlüssels. Ist ein Attribut beispielsweise am zweiten zusammengesetzten Schlüssel eines Entity-Typs, als drittes Attribut beteiligt, so werden die Elementeigenschaften {`key=K-2,order=3`} gesetzt.

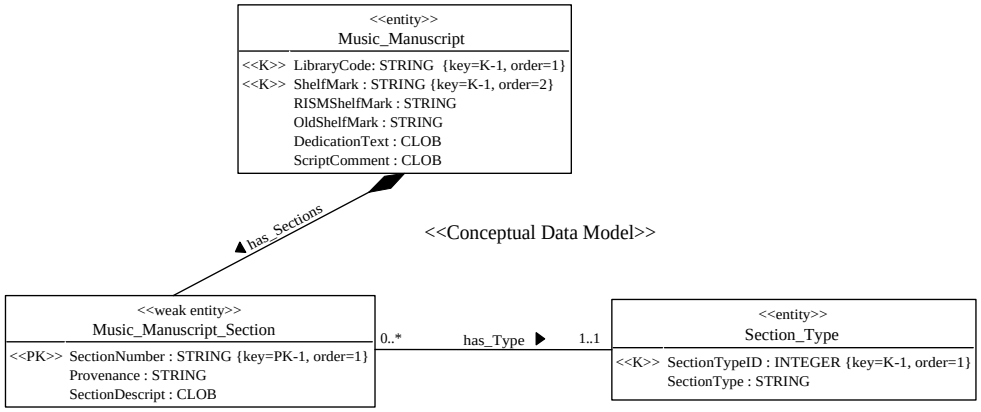


Abbildung 2.7: Beispiel für die Darstellung konzeptueller Datenmodelle mit UML

In Abbildung 2.7 ist ein vereinfachter Ausschnitt aus dem konzeptuellen Datenmodell des *eNoteHistory*-Projekts dargestellt. Das Beispieldiagramm zeigt einen Entity-Typ `Music_Manuscript`, mit einem zusammengesetzten Schlüssel, bestehend aus den Attributen `LibraryCode` und `ShelfMark`. Die Ordnung der Attribute innerhalb des Schlüssels wird durch die Elementeigenschaft `order` festgelegt. Ein Musikmanuskript besteht aus Sektionen, die nicht ohne ein Manuskript existieren können, dies wird durch eine Kompositionsbeziehung zu dem schwachen Entity-Typ `Music_Manuscript_Section` ausgedrückt. `Music_Manuscript_Section` besitzt einen partiellen Schlüssel `Section-Number`, der nur in Kombination mit dem zusammengesetzten Schlüssel von `Music_Manuscript` eindeutig ist. Einer Sektion wird ein Typ zugeordnet, wobei mehrere Sektionen vom selben Typ existieren können. Dies wird durch eine 1:n Beziehung zu dem Entity-Typ `Section_Type` modelliert. Da die Entity-Typen `Music_Manuscript_Section` und `Section_Type` jeweils nur einfache Schlüssel besitzen, hätten die Elementeigenschaften `key` und `order` nicht unbedingt aufgeführt werden müssen.

2.6 Zusammenfassung

In diesem Kapitel wurden Grundlagen gelegt, die zum Verständnis der vorliegenden Diplomarbeit notwendig sind.

Im ersten Abschnitt wurde das *eNoteHistory*-Projekt vorgestellt. Im Rahmen dieses Projekts wurde ein Dokumentenserver zur Verwaltung von handgeschriebenen Noten aufgebaut. Der Dokumentenbestand dieses Servers wird im praktischen Teil dieser Diplomarbeit in eine *IBM Content Manager*-Umgebung migriert.

Anschließend wurden die Begriffe Content Management und Information Integration erläutert.

Im letzten Teil dieses Kapitels wurde die Unified Modeling Language als Methode zur Modellierung von Softwaresystemen vorgestellt, die im Folgenden zur Darstellung von Datenmodellen verwendet wird. Dazu wurden die wichtigsten Elemente von UML-Klassendiagrammen beschrieben. Darüberhinaus wurden mit Stereotypen, Zusicherungen und Elementeigenschaften Mittel zur Erweiterung der UML vorgestellt, die als Basis für UML-Profile genutzt werden. Abschließend wurde ein UML-Profil zur Modellierung von konzeptuellen Datenmodellen beschrieben.

Kapitel 3

IBM Enterprise Content Management

Für die Integration von Dokumentenservern werden in der vorliegenden Diplomarbeit zwei Produkte der *IBM Enterprise Content Management*-Produktreihe verwendet. Dies ist zum einen der *IBM DB2 Content Manager for Multiplatforms 8.2 (ICM)* und zum anderen der *IBM DB2 Information Integrator for Content 8.2 (IIC)*. Dieses Kapitel gibt einen Überblick über die Konzepte und die Systemarchitektur dieser beiden Produkte.

3.1 IBM DB2 Content Manager for Multiplatforms

Der *IBM DB2 Content Manager for Multiplatforms* ist einer der zentralen Bestandteile der *IBM Enterprise Content Management*-Produkte. Das System basiert auf der *IBM DB2 Universal Database (DB2)* und bietet eine offene, skalierbare und erweiterbare Architektur zur Verwaltung, Archivierung und Integration von digitalem Content jeder Art. Dazu gehört unter anderem HTML und XML basierter Web Content, digitale Bilder, elektronische Office-Dokumente und Rich-Media-Content wie digitales Audio- und Videomaterial.

Neben einer parameterbasierten Suche unterstützt der *IBM DB2 Content Manager for Multiplatforms* mit Hilfe des *DB2 Net Search Extenders (NSE)* auch Volltextsuchanfragen, sowohl auf Meta-Daten, als auch auf textbasierten Dokumenten. Ein integriertes Workflow-System ermöglicht das Routing von Dokumenten zur Automatisierung von Geschäftsprozessen.

Die Plattform unterstützt unter anderem Standards wie das *Lightweight Directory Access Protocol (LDAP)* zum Import von Gruppen- und Nutzerdefinitionen sowie das *Open Document Management API (ODMA)*, das Desktop-Applikationen wie z.B. *Microsoft Word* oder *Lotus Word Pro* den direkten Zugriff auf im *IBM DB2 Content Manager for Multiplatforms* gespeicherte Dokumente ermöglicht.

Eine Zusammenfassung der Konzepte und Funktionen des *IBM DB2 Content Manager for Multiplatforms* befindet sich in [Cor03h].

3.1.1 Systemarchitektur

Die Architektur des *IBM DB2 Content Manager for Multiplatforms* basiert auf einem dreieckförmigen Client-Server-Modell bestehend aus einem Library Server zur Verwaltung der Meta-Daten, einem oder mehreren Resource Managern zur Speicherung der Dokumente sowie einer oder mehrerer Client-Applikationen.

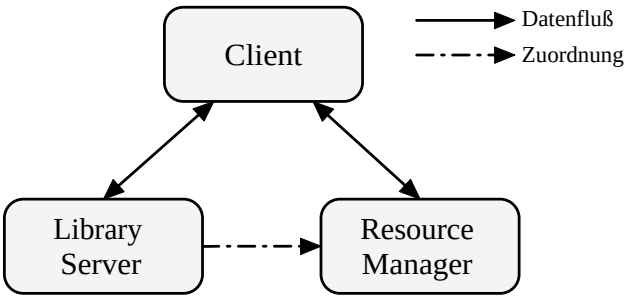


Abbildung 3.1: Dreiecksarchitektur des *IBM DB2 Content Manager for Multiplatforms*

Fordert ein Client ein Dokument an, so wird zunächst eine Anfrage an den Library Server gestellt. Der Library Server kennt die Zugriffsberechtigung des Clients sowie den Resource Manager, auf dem das angeforderte Dokument gespeichert ist. Hat der Client die notwendigen Berechtigungen, generiert und verschlüsselt der Library Server ein Token. Dieses Token wird zusammen mit den Informationen über den Resource Manager (Hostname, Port) als Antwort zum Client übertragen. Der Client sendet das erhaltene Token zum entsprechenden Resource Manager, welcher dieses entschlüsselt und auf Gültigkeit überprüft. Besteht das Token den Test, sendet der Resource Manager das angeforderte Dokument an die Client-Applikation.

Hat ein Client einmal ein gültiges Token erhalten, behält dieses seine Gültigkeit und kann für dieselbe Dokumentenanforderung beliebig oft wieder verwendet werden. Die Widerverwendbarkeit von Tokens reduziert den Datenverkehr zwischen Library Server und Client. Die Gültigkeit eines Tokens kann und sollte aber durch den Systemadministrator eingeschränkt werden, da eine Client-Applikation andernfalls auf Dokumente zugreifen kann, selbst wenn die Zugriffsrechte im Library Server geändert wurden. Ist ein Token abgelaufen, muss der Client eine neue Anfrage an den Library Server stellen.

Durch die Zuordnung mehrerer Resource Manager zu einem Library Server können Dokumente topologisch dicht bei Nutzern gespeichert werden, die besonders häufig darauf zugreifen. Wichtig ist dies vor allem bei großen Multimedia-Dokumenten. Die erweiterbare Architektur erlaubt außerdem die Anbindung von zusätzlichen Resource Managern (selbst auf anderen Plattformen), auch nachdem eine bestimmte Konfiguration erstellt wurde, falls dies durch ein unvorhergesehen hohes Datenaufkommen im Laufe der Zeit notwendig wird.

Die Folgenden Unterkapitel geben einen Überblick über die Systemkomponenten der Dreiecksarchitektur. Eine weiterführende Beschreibung befindet sich in [RDZ02] und in den IBM-Handbücher [Cor03e, Cor03c].

3.1.2 Library Server

Der *Library Server* ist die Schlüsselkomponente des *IBM DB2 Content Managers for Multiplatforms*. Anders als in der Vorgängerversion ist der Library Server in Version 8 nicht mehr durch einen eigenen Dienst oder Prozess implementiert, sondern wurde mit Hilfe von Stored Procedures (SP) vollkommen in eine DB2-Datenbank integriert. Daher identifizieren Client-Applikationen einen Library Server über den entsprechenden Datenbanknamen. Wird während der Installation nichts anderes festgelegt, so lautet der Name der Library-Server-Datenbank `ICMNLSDB`.

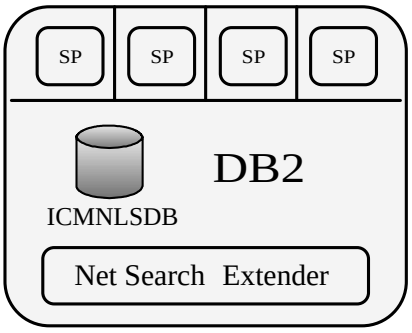


Abbildung 3.2: Library-Server-Architektur

Der Zugriff auf den Library Server erfolgt über SQL. Auf jedem Content Manager Client muss dazu mindestens ein *DB2 Runtime Client* installiert sein. Zur Unterstützung von Volltextsuchanfragen verwendet der Library Server den *DB2 Net Search Extender*. Zu den wichtigsten Aufgaben des Library Servers gehören:

- *Definition des Datenmodells:* Im Library Server werden die Struktur, die Datentypen sowie die Beziehungen der im *IBM DB2 Content Manager for Multiplatforms* gespeicherten Daten festgelegt. Das verwendete Datenmodell wird im Kapitel 3.2 ausführlich beschrieben.

- *Speicherung der Meta-Daten:* Neben den eigentlichen Objekten die im Resource Manager abgelegt sind, werden auch Meta-Daten verwaltet. Meta-Daten beschreiben die im Resource Manager gespeicherten Objekte und werden selbst in der Library-Server-Datenbank verwaltet.

- *Speicherung von Nutzer- und Gruppendefinition:* Mit dem *System Administration Client* werden vom Administrator Nutzer und Gruppen definiert, denen entsprechende System- und Objektrechte zugeordnet werden. Diese Daten werden in der Library-Server-Datenbank gespeichert.

- *Authentifizierung/Autorisierung:* Da alle nutzer- und rechterelevanten Informationen in der Library-Server-Datenbank gespeichert sind, ist der Library Server sowohl für die Authentifizierung von Nutzern als auch für die Autorisierung von System- und Objektzugriffen verantwortlich.

- *Verwaltung der Workflow Funktionalität:* Über den *System Administration Client* können Workflows definiert werden die ein Dokument durchlaufen kann, das in das System eingebracht wird. Dazu werden im Library Server verschiedene Arbeitsstationen, „Work Nodes“ genannt, definiert. Weiterhin wird die Reihenfolge, in der diese Arbeitsstationen durchlaufen werden, festgelegt.

3.1.3 Resource Manager

Der *Resource Manager* besteht aus einer Web-Applikation, die im *IBM WebSphere Application Server* installiert wird. Verwaltungsinformationen, die zum Betrieb des Resource Managers benötigt werden, speichert die Web-Applikation in einer DB2-Datenbank mit dem Namen `RMDB`.

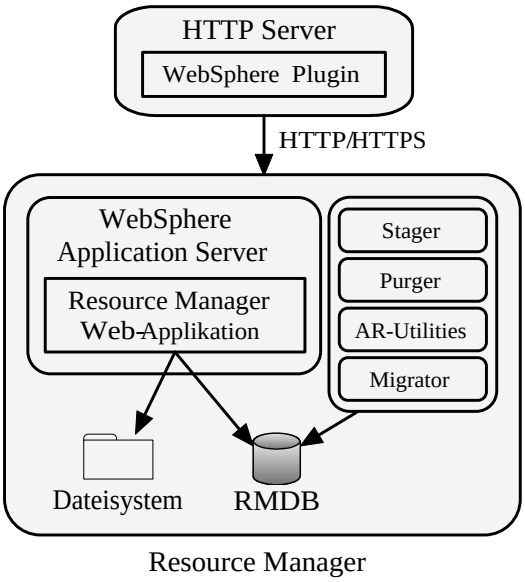


Abbildung 3.3: Resource-Manager-Architektur

Der Resource Manager ist für die Speicherung der Objekte im *IBM DB2 Content Manager for Multiplatforms* verantwortlich. Die Objekte, z.B. Texte, Bilder und Videos, werden gewöhnlich im Dateisystem der Resource-Manager-Maschine abgelegt. Es besteht aber auch die Möglichkeit, bestimmte Objekte in spezialisierten Servern zu speichern. Ein Beispiel hierfür ist der *IBM DB2 Content Manager VideoCharger*, der vom Resource Manager zur Speicherung von Audio- und Videodateien genutzt werden kann. Die Dateien können dann im Streaming-Verfahren von Client-Applikationen abgespielt werden, ohne dass die Dateien vorher komplett auf das Client-System übertragen werden müssen.

Da der *IBM DB2 Content Manager for Multiplatforms* grundsätzlich für die Verwaltung von sehr großen Objekt- und Datenmengen konzipiert wurde, werden konsequenter Weise auch spezielle Backup- und Archivsysteme, sowie die automatische Migration von Objekten unterstützt. Dies gehört ebenfalls zum Aufgabenbereich des Resource Managers. Über „Migration Policies“ genannte Regeln kann festgelegt werden, auf welchem Daten-

träger ein Objekt wie lange gespeichert wird. Beispielsweise ist es möglich, eine Klasse von Dokumenten fünf Tage lang auf einer vom Resource Manager verwalteten Festplatte zu speichern und die Dokumente dann automatisch an ein Archivsystem oder einen anderen Resource Manager zu übergeben. Archivsysteme speichern Daten im Allgemeinen auf Bandlaufwerken oder optischen Platten mit dem Ziel der langfristigen Lagerung. Ein vom *IBM DB2 Content Manager for Multiplatforms* unterstütztes Archivierungssystem ist der *IBM Tivoli Storage Manager (TSM)*.

Neben der Datenbank und der Web-Applikation gehören vier Dienste zum Resource Manager. Diese sind ebenfalls in Abbildung 3.3 dargestellt. Die Dienste sind unter anderem für die beschriebene Migration von Objekten zwischen verschiedenen Datenträgern verantwortlich. Eine ausführliche Beschreibung der Dienste befindet sich in [RDZ02] und [Cor03e]. Auf eine detaillierte Beschreibung soll an dieser Stelle verzichtet werden, da dies für das Gesamtverständnis nicht notwendig ist.

Der Zugriff auf den Resource Manager erfolgt über HTTP/HTTPS mit Hilfe eines Web-Servers und dem *WebSphere Application Server Plug-in*, dass die Aufrufe an den Application Server weiterleitet. Alternativ kann der Zugriff auch für das FTP-Protokoll konfiguriert werden.

3.1.4 System Administration Client

Der *System Administration Client* ist das zentrale Verwaltungswerkzeug für das gesamte *IBM Content Manager*-System. Sowohl der Library Server und der Resource Manager als auch der im nächsten Kapitel vorgestellte *IBM DB2 Information Integrator for Content* werden über diese Java-Applikation konfiguriert. Zu den wichtigsten Konfigurationsaufgaben, die mit dem *System Administration Client* ausgeführt werden können, zählen:

- die Definition von Nutzern und Gruppen,
- die Zuweisung von Zugriffsrechten,
- die Definition des Datenmodells,
- die Konfiguration der Workflow-Funktionalität,
- die Konfiguration des Log- und Trace-Verhaltens,
- die Konfiguration des Speichersystems und
- die Konfiguration der verteilten Suche.

3.2 Das Datenmodell des IBM DB2 Content Manager for Multiplatforms

Datenobjekte im *IBM DB2 Content Manager for Multiplatforms* werden Items genannt und nach Vorlagen erzeugt und klassifiziert, die als Item Types bezeichnet werden. In den folgenden Unterabkapiteln werden die wichtigsten Elemente des Datenmodells des *IBM DB2 Content Manager for Multiplatforms* vorgestellt. Das Datenmodell wird dabei im

Folgenden als *ICM-Datenmodell* bezeichnet. Eine weiterführende Beschreibung des ICM-Datenmodells liefern [Cor03e] und [RDZ02].

3.2.1 Attribute und Attributgruppen

Attribute dienen zum Speichern von Meta-Daten. *Meta-Daten* beschreiben Objekte wie z.B. Texte, Videos und Bilder, können aber auch selbstbeschreibend sein. Jedem Attribut wird ein Datentyp zugeordnet, der den Wertebereich festlegt. Attribute werden in Anfragen an das Content-Manager-System genutzt, um Items zu suchen. Attribute mit textbasiertem Inhalt können als `Text searchable` definiert und in Volltextsuchanfragen verwendet werden.

Neben numerischen Werten und Strings können in Attributen auch CLOBs und BLOBs gespeichert werden. Hierbei ist aber zu beachten, dass der Library Server derartige Attributwerte nur bis zu einer Größe von fünf Megabyte unterstützt, da dies die Grenze für das maximale Volumen an Daten darstellt, das zum Library Server übertragen werden kann, um ein Item zu erzeugen oder zu aktualisieren. Attribute, die ein größeres Datenvolumen speichern, müssen als Objekte des Resource Managers (Resource Items oder Document Parts) definiert werden.

Eine *Attributgruppe* besteht aus einer Menge von Attributen. Beispielsweise ist es möglich, die Attribute Postleitzahl, Ort, Straße und Hausnummer zu einer Attributgruppe Adresse zusammenzufassen. Attributgruppen besitzen aber keine eigenen Eigenschaften und dienen lediglich der Benutzerfreundlichkeit, um auf diese Weise die Zuordnung von Attributen zu vereinfachen.

3.2.2 Components

Eine *Component* besteht aus einer Menge von nutzerdefinierten und systemdefinierten Attributen, die eine Klasse von Daten beschreiben. In der Library-Server-Datenbank existiert für jede Component eine Tabelle, welche bei deren Definition erzeugt wird. Als Instanz einer Component wird eine Zeile dieser Tabelle bezeichnet. Components werden in zwei Kategorien unterteilt, Root Components und Child Components.

Eine *Root Component* ist die erste Schicht eines hierarchischen Item Types und besteht aus einer Menge von nutzerdefinierten und systemdefinierten Attributen.

Eine *Child Component* ist die optionale zweite oder tiefere Schicht eines hierarchischen Item Types und ist einer Component einer höheren Schicht zugeordnet. Die Anzahl der Child Components auf einer Ebene und die Schachtelungstiefe sind unbegrenzt.

Bei der Definition von Child Components wird eine Löschregel angegeben. Die Löschregel legt fest, ob eine Child-Component-Instanz gelöscht wird, falls die übergeordnete Component-Instanz gelöscht wird (`Cascade`) oder ob das Löschen einer übergeordneten Component-Instanz verhindert wird (`Restrict`), falls eine Child-Component-Instanz existiert. Die Anzahl der Instanzen einer Child Component, die einer übergeordneten Component-Instanz zugeordnet werden können, kann mit Hilfe von Kardinalitäten beschränkt werden. Child Components können im ICM-Datenmodell verwendet werden, um unter anderem zusammengesetzte und mehrwertige Attribute umzusetzen.

Bei der Zuordnung von nutzerdefinierten Attributen zu einer Component kann festgelegt werden, ob ein Attribut mit einem Wert belegt werden muss (`Required`) und ob ein Attribut dazu verwendet wird, ein Item in einer Client-Applikation zu repräsentieren (`Represent Item`). Weiterhin kann festgelegt werden, dass ein Attribut Schlüsselcharakter hat. Die Attributwerte eines Schlüsselattributes sind in allen Component-Instanzen verschieden. Soll die Kombination von verschiedenen Attributen einen eindeutigen Wert besitzen, so kann ein Index über die entsprechenden Attribute definiert werden. Abbildung 3.4 zeigt eine Root Component, die Kundendaten beinhaltet, der eine Child Component zugeordnet ist, in der Adressdaten gespeichert werden.

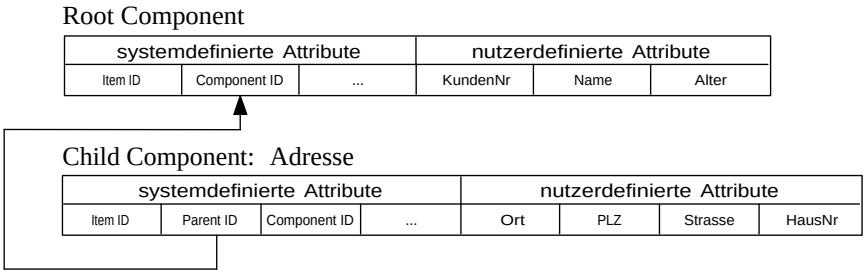


Abbildung 3.4: Root Component mit einer Child Component

3.2.3 Item Types

Item Types dienen als Vorlage, nach der Items im ICM-Datenmodell erzeugt und gesucht werden. Ein Item Type besteht aus einer Root Component, keiner oder mehreren Child Components und einer Klassifikation. Die möglichen Klassifikationen sind: `Item`, `Resource Item`, `Document` und `Document Part`. Eine Root Component wird nicht explizit vom Nutzer definiert, sondern implizit bei der Definition eines Item Types erstellt. Abbildung 3.5 zeigt einen Item Type `Kunde` der Klassifikation `Item`, dem eine Child Component `Adresse` zugeordnet ist.

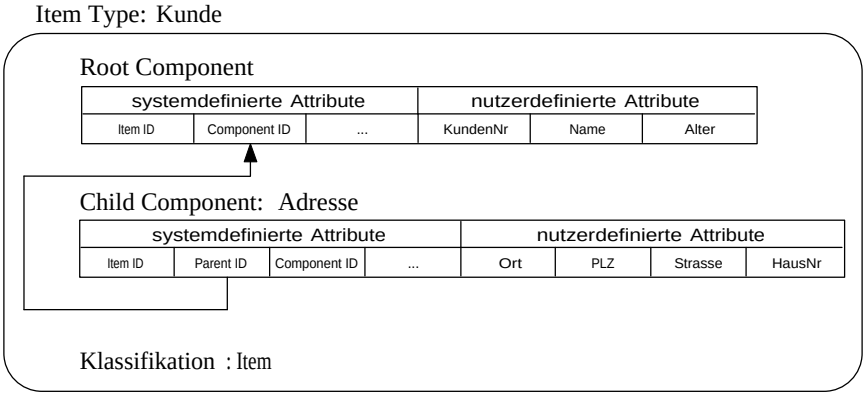


Abbildung 3.5: Item Type `Kunde` mit Root und Child Component

Bei der Suche nach Items im *IBM DB2 Content Manager for Multiplatforms* wird zunächst ein Item Type ausgewählt, anschließend können Werte für die Attribute des Item Types vorgegeben werden. Das System antwortet mit einer Liste aller Items, welche die entsprechenden Attributwerte aufweisen.

Abbildung 3.6 zeigt die Definition des Item Types `Kunde` mit dem *System Administration Client*. Bei der Definition wurde festgelegt, dass ein Kunde drei Adressen haben kann (`Maximum Cardinality: 3`). Weiterhin wurde festgelegt, dass sobald ein Kunde gelöscht wird auch alle zugehörigen Adressen gelöscht werden (`Delete rule: Cascade`).

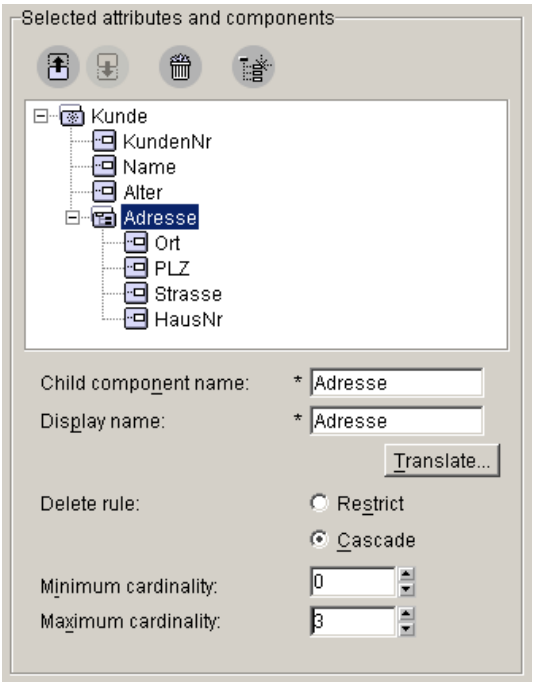


Abbildung 3.6: Item Type Definition mit dem *System Administration Client*

Item Type Klassifikationen

Die *Klassifikation* eines Item Types entscheidet über den Verwendungszweck der Items dieses Typs. Während einige Items Content repräsentieren, der in einem Resource Manager gespeichert ist, bestehen andere Items nur aus Meta-Daten.

Item Ein Item Type wird als `Item` klassifiziert, falls die Instanzen dieses Typs ausschließlich aus Attributen bestehen und ihnen kein Content in einem Resource Manager zugeordnet ist. Ein Beispiel hierfür ist der in Abbildung 3.5 dargestellte Item Type `Kunde`, bei dem ausschließlich Meta-Daten gespeichert werden.

Resource Item Items der Klassifikation `Resource Item` repräsentieren eine Referenz auf ein Objekt, dass in einem Resource Manager gespeichert ist. Die Attribute eines solchen

Item Types sollten das Objekt im Resource Manager beschreiben. Ein Beispiel hierfür ist ein Item Type `Image` mit den Attributen `Name`, `Format`, `Größe` und einem im Resource Manager gespeichertem Bild.

Document Ein Item Type der Klassifikation `Document` dient als Vorlage für Items, die aus Attributen und Document Parts bestehen. Ein Document Part enthält eine Liste von Referenzen auf Objekte in einem Resource Manager. Anders als Resource Items kann ein Document Item also nicht nur ein Objekt in einem Resource Manager referenzieren, sondern mehrere. Einem Document Item Type können die in Tabelle 3.1 aufgeführten vordefinierten Document Parts zugeordnet werden. Mit diesem Modell ist es möglich, auf einfache Weise ein Dokument zu speichern, das beispielsweise aus fünf Bildern sowie zwei Texten besteht und zu dem Annotationen erstellt werden können. Die Bilder werden im ICMBASE, die Texte im ICMBASETEXT und die Annotationen im ICMANNOTATION Part gespeichert. Ein Document Item Type ohne Document Parts entspricht in seiner Funktionalität einem Item Type der als `Item` klassifiziert ist.

Document Part Item Types, die als `Document Part` klassifiziert werden, können einem Document Item Type zugeordnet werden. Neben den fünf vordefinierte Document Parts des *IBM DB2 Content Manager for Multiplatforms* können auf diese Weise auch selbstdefinierte Parts erstellt werden.

Document Part	**Verwendung**
ICMBASE	Ist der Hauptbestandteil eines Document Items und speichert Content, der nicht in einen der spezialisierten Parts gespeichert wird, wie z.B. Bilder.
ICMBASETEXT	Ist zum Speichern von Texten vorgesehen, die für eine Volltextsuche indexiert werden sollen.
ICMBASESTREAM	Ist zum Speichern von Audio- oder Video-Streams gedacht.
ICMNOTELOG	Speichert Log-Informationen über Document Items, z.B. den Grund der letzten Änderung.
ICMANNOTATIONS	Ist zur Speicherung von Annotationen vorgesehen.

Tabelle 3.1: Vordefinierte Document Parts des *IBM DB2 Content Manager for Multiplatforms*

Custom Model vs. Document Model

Im *IBM DB2 Content Manager for Multiplatforms* wird zwischen zwei Arten von Datenmodellen unterschieden, die als Custom Model und Document Model bezeichnet werden.

Document Models sind Datenmodelle, die ausschließlich aus Document Item Types bestehen.

Custom Models sind Datenmodelle, die auch oder ausschließlich Item Types und Resource Item Types beinhalten. Custom Models sind dafür gedacht, um mit dem *IBM DB2 Content Manager for Multiplatforms* sehr spezielle, genau an den Bedarf des Nutzers angepasste, Datenmodelle zu erstellen.

Es ist wichtig, eine Entscheidung für eine der beiden Varianten vor dem Erstellen eines ICM-Datenmodells zu treffen, da die Standard-Clients des *IBM DB2 Content Manager for Multiplatforms* (siehe folgendes Kapitel) nur das Document Model unterstützen. Werden im Datenmodell Item Types und Resource Item Types verwendet, so werden diese von den Standard-Clients ignoriert.

Item Type Subsets

Das Sichtenkonzept, bekannt aus der Welt der relationalen Datenbanken, wird im *IBM DB2 Content Manager for Multiplatforms* durch Item Type Subsets nachgebildet. Mit *Item Type Subsets* ist es möglich, verschiedenen Nutzern unterschiedliche Sichten auf denselben Item Type zu ermöglichen. In einem Item Type Subset können sowohl Attribute wie auch Items eines Item Types gefiltert werden. So ist es beispielsweise möglich, einen Item Type Subset über den Item Type `Kunde` aus Abbildung 3.5 zu definieren, in dem nur die Attribute `Name` und `Alter` sichtbar sind und nur Kunden angezeigt werden, die älter als 18 Jahre sind. Im Gegensatz zu relationalen Datenbanken ist es allerdings nicht möglich, einen Item Type Subset über mehrere Item Types zu definieren.

3.2.4 Items

Als *Item* wird jede Instanz eines Item Types bezeichnet, gleich welcher Klassifikation. Beispielsweise werden alle Kunden, die unter dem Item Type `Kunde` aus Abbildung 3.5 angelegt werden, als Items bezeichnet.

Resource Items, die eine Referenz auf ein im Resource Manager gespeichertes Objekt enthalten, sind eine Repräsentation dieser Objekte, durch die diese Objekte identifiziert und gefunden werden können. Ein Item selbst stellt aber kein Objekt im Resource Manager dar.

Der *IBM DB2 Content Manager for Multiplatforms* unterstützt die Versionierung von Items. Dazu wird bei der Item Type Definition festgelegt, ob Items dieses Typs versioniert werden und ob dies automatisch passiert oder eine Nutzerbestätigung erforderlich ist. Eine Version eines Items umfasst sowohl die Attributwerte, als auch die Objekte im Resource Manager, falls es sich um ein Resource Item oder Document Item handelt. Weiterhin kann festgelegt werden, wie viele Versionen eines Items maximal erstellt werden, bevor die älteste Version überschrieben wird. Bei Document Items kann zusätzlich eine Versionierungsregel für jeden der zugeordneten Document Parts angegeben werden.

Die Semantic eines Items kann über eine Eigenschaft die als Semantic Type bezeichnet wird definiert werden. Der *Semantic Type* ermöglicht es Client-Anwendungen das Verhalten von speziellen Items zu ermitteln und zwischen verschiedenen Verwendungszwecken zu unterscheiden. Der Semantic Type wird beim Erzeugen eines Items festgelegt. Neben einigen vordefinierten Semantic Types können auch nutzerdefinierte Typen erstellt werden. Zu den vordefinierten Semantic Types gehört der Typ Folder. Items die als *Folder* deklariert wurden, können für den Aufbau von hierarchischen Strukturen zwischen Items verwendet

werden. Folder werden im Zusammenhang mit Link-Beziehungen (siehe Kapitel 3.2.6) detaillierter beschrieben.

3.2.5 Objekte

Als *Objekt* wird jede Einheit bezeichnet, die in einem Resource Manager gespeichert ist. Objekte sind z.B. Bilder, Texte oder Videos. Objekte werden durch Items im Library Server verwaltet. Ein Item besitzt alle notwendigen Informationen, um ein Objekt zu beschreiben und zu finden.

Jedem Objekt wird beim erzeugen ein *MIME Type* (Multipurpose Internet Mail Extensions) zugeordnet. Der MIME Type legt fest, wie ein Objekt von einer Applikation behandelt wird. Wurde einem Objekt beispielsweise der MIME Type `image/jpeg` zugeordnet, kann diese Information von der Client-Applikation genutzt werden, um nach dem Laden des Objekts automatisch einen Web-Browser zur Darstellung des Bildes zu starten.

Der Inhalt und das Verhalten eines Objekts des Resource Managers werden durch eine *Media Object Class* beschrieben. Wird ein Item Type vom Typ `Resource Item Typ` oder `Document Part` definiert, so muss diesem Item Type eine entsprechende Media Object Class zugeordnet werden. Wird ein Objekt von einem Resource Manager geladen, dann verwendet die Client-Applikation die zugeordnete Media Objekt Class, um das Objekt korrekt zu behandeln. Der *IBM DB2 Content Manager for Multiplatforms* kennt vier vordefinierte Media Object Classes, die in Tabelle 3.2 beschrieben sind. Darüberhinaus können eigene Klassen erstellt werden.

Media Objekt Class	**Beschreibung**
DKLobICM	Ist die Basisklasse zur Behandlung von generischen großen Objekten (generic large objects), die in einem Resource Manager gespeichert sind. Die Klasse bietet Funktionen zum Hinzufügen, Laden, Aktualisieren und Löschen von Objekten eines Resource Managers. Alle weiteren Klassen sind Unterklassen von `DKLobICM`.
DKImageICM	Repräsentiert Bilder die in einem Resource Manager gespeichert sind. Die Klasse ist als `deprecated` eingestuft und sollte daher nicht mehr verwendet werden.
DKStreamICM	Eine Klasse mit speziellen Funktionen zur Behandlung von Stream-Objekten
DKTextICM	Eine Klasse speziell für die Behandlung von Textdokumenten. Objekte dieser Art können zur Volltextsuche indiziert werden.
DKVideoStreamICM	Repräsentiert Video-Streams, die in einem Streaming Server (z.B. *IBM DB2 Content Manager VideoCharger*) gespeichert sind. Diese Klasse enthält Funktionen, um eine Streaming Session zwischen einem Server und einem Abspielprogramm aufzubauen. `DKVideoSteamICM` erweitert die Klasse `DKStreamICM`.

Tabelle 3.2: Vordefinierte Media Object Classes des *IBM DB2 Content Manager for Multiplatforms*

3.2.6 Beziehungen

Ein wichtiges Element jedes Datenmodells sind Beziehungen, die zwischen Datenobjekten erstellt werden können. Im ICM-Datenmodell existieren drei Konstrukte zum Aufbau von Beziehungen, die in den folgenden Unterabschnitten beschrieben werden.

Links

Ein *Link* ist eine gerichtete Beziehung zwischen den Root Components zweier Items. Hierfür werden Link-Typen definiert, von denen zur Laufzeit Instanzen erstellt werden.

Da es sich um eine gerichtete Beziehung handelt, wird zwischen Quell- und Ziel-Items unterschieden. Mit Hilfe von Links kann ein Item mit beliebig vielen anderen Items in Beziehung gebracht werden. Technisch werden Links mit Hilfe von Tabellen in der Library-Server-Datenbank umgesetzt, die Verweise auf die Quell- und Ziel-Items speichern.

Das Kundenbeispiel aus Abbildung 3.4 kann außer mit einer Child Component auch mit Links realisiert werden, in dem `Adresse` nicht als Child Component von `Kunde`, sondern als eigenständiger Item Type definiert wird. Über einen Link-Typen, der beispielsweise `KundeHatAdresse` genannt wird, können dann Adressen einem Kunden zugeordnet werden. Abbildung 3.7 zeigt die entsprechenden Tabellen im Library Server.

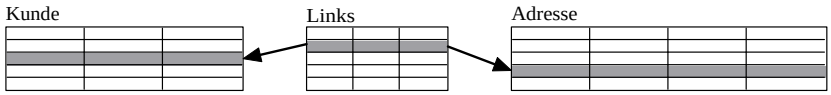


Abbildung 3.7: Link zwischen einem Kunden- und einem Adress-Item

Ein Vorteil von Links ist, dass zu einer solchen Beziehung ein beschreibendes Objekt definiert werden kann. Dafür kann jedes beliebige Item verwendet werden. Hat eine Beziehung Attribute, wie dies beispielsweise aus der UML-Modellierung bekannt ist, so kann dies durch die Zuordnung eines Items zu einem Link direkt umgesetzt werden.

Links werden im *IBM DB2 Content Manager for Multiplatforms* unter anderem dazu verwendet, um die Beziehungen zwischen Foldern und ihrem Inhalt herzustellen. Ein Folder kann eine Menge von Items sowie weitere Folder beinhalten. Jedes Item, Resource Item und Document Item kann beim Erzeugen als Folder klassifiziert werden, ohne dass es dabei in seiner gewohnten Funktionalität eingeschränkt wird. Beispielsweise kann ein Item Type `InVerzug` angelegt werden, dessen Items als Folder deklariert werden und jeweils Kunden enthalten, die mit der Zahlung ihrer Rechnung in Verzug geraten sind. Die Umsetzung der Verknüpfung eines Folders mit den darin enthaltenen Items oder Folders, erfolgt durch Links. Der Vorteil, den der Folder-Mechanismus gegenüber gewöhnlichen Links bietet, ist eine vereinfachte Handhabung auf API-Ebene.

Referenzen

Eine *Referenz* ist eine unidirektionale Beziehung zwischen der Root oder Child Component eines Items und der Root Component eines anderen Items. Hierfür wird mit dem *System Administration Client* ein Referenzattribut definiert, dass der Root oder Child Component

eines Item Types zugeordnet wird. Zur Laufzeit kann das Referenzattribut eines Items dieses Typs mit einem Verweis auf die Root Component eines anderen Items belegt werden.

Das Kundenbeispiel aus Abbildung 3.4 kann mit Referenzattributen umgesetzt werden, indem `Adresse` als eigenständiger Item Type definiert wird, und der Item Type `Kunde` ein Referenzattribut erhält. In diesem Attribut kann für jeden Kunden ein Verweis auf die entsprechende Adresse gespeichert werden.

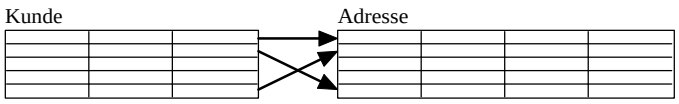


Abbildung 3.8: Referenzen zwischen Kunden- und Adress-Items

Referenzen bieten gegenüber Links zwei Vorteile: Zum einen kann über eine Referenz eine bestimmte Version eines Items referenziert werden und zum anderen können Restriktionen bezüglich des Löschens eines referenzierten Items definiert werden. Dafür kann bei der Zuordnung eines Referenzattributs zu einem Item Type angegeben werden, ob das Referenzattribut beim Löschen des Ziel-Items auf null gesetzt wird (`Set null`), das Löschen des referenzierten Items verhindert wird (`Restrict`), das Quell-Item ebenfalls gelöscht wird (`Cascade`) oder das System keine Aktion ausführt (`No action`).

Fremdschlüssel

Fremdschlüssel werden im *IBM DB2 Content Manager for Multiplatforms* nach demselben Prinzip erstellt wie Fremdschlüssel in relationalen Datenbanken. Ein *Fremdschlüssel* ist das Attribut einer Root Component, das einen eindeutigen Schlüssel eines anderen Item Types oder einer Tabelle einer externen Datenbank referenziert. Fremdschlüssel werden im *IBM DB2 Content Manager for Multiplatforms* genutzt, um die Attributwerte eines Item-Type-Attributs zu beschränken.

Sollen im Beispiel aus Abbildung 3.4 nur Adressen zugelassen werden, die eine gültige deutsche Postleitzahl haben, so kann ein Item Type `Postleitzahlen` erstellt werden, mit einem eindeutigen Attribut `PLZ`, dessen Items alle Postleitzahlen Deutschlands speichern. Das Attribut `PLZ` des Item Types `Adresse` kann anschließend als Fremdschlüssel definiert werden, der auf das Attribut `PLZ` des Item Types `Postleitzahlen` verweist. Beim Anlegen neuer Adress-Items werden Adressen mit ungültigen Postleitzahlen zurückgewiesen, wenn diese gegen die Fremdschlüsselbedingung verstoßen.

Die technische Umsetzung von Fremdschlüsseln im *IBM DB2 Content Manager for Multiplatforms* erfolgt mit Hilfe von Constraints über die Funktionalität der zu Grunde liegenden DB2-Datenbank.

In Tabelle 3.3 werden die Eigenschaften der Beziehungstypen des ICM-Datenmodells zusammengefasst.

Beziehungstyp	**zwischen welchen Components**	**Verknüpfte Items können gelöscht werden**	**kann Item-Versionen ansprechen**
Link	Root zu Root	ja	nein
Referenz	Root oder Child zu Root	kann beim Erstellen angegeben werden	kann beim Erstellen angegeben werden
Fremdschlüssel	Root zu anderen Item Type oder externer Tabelle	kann beim Erstellen angegeben werden	kann beim Erstellen angegeben werden

Tabelle 3.3: Beziehungstypen des ICM-Datenmodells

3.2.7 Ein UML-Profil zur Darstellung von ICM-Datenmodellen

Die Darstellung von ICM-Datenmodellen erfolgt in der vorliegenden Diplomarbeit, wie auch die Darstellung von konzeptuellen Datenmodellen, in UML-Notation. Dafür werden Stereotypen und Elementeigenschaften zur Modellierung der ICM-Datenmodell-Konstrukte definiert. Eine Auflistung der Erweiterungen befindet sich im Anhang B.3.

Zur Darstellung der verschiedenen Item Types des ICM-Datenmodells werden die Stereotypen `item type`, `resource item type`, `document item type` und `document part` definiert, die das UML-Element `Klasse` erweitern. Bei der Darstellung von Item Types wird die gewöhnliche Notation von UML-Klassen verwendet, wobei der Stereotyp des Item Types angegeben wird. Attribute und Methoden von Item Types werden wie in UML-Klassen üblich notiert. Attributeigenschaften wie `Text searchable`, `Required` und `Unique` können mit Hilfe von Elementeigenschaften festgelegt werden.

Für die Beziehungstypen werden die Stereotypen `link`, `reference` und `child relation` definiert, welche vom UML-Element `Assoziation` abgeleitet werden. Für die Darstellung wird folgende Notation festgelegt: Links werden durch gestrichelte Pfeile, Referenzen durch Strich-Punkt-Pfeile und die Beziehungen zwischen Root Components und Child Components durch Pfeile dargestellt.

Zur Umsetzung von Referenz- und Fremdschlüsselattributen werden die Stereotypen `RA` bzw. `FK` definiert, die das UML-Element `Attribut` erweitern. Für Fremdschlüsselattribute werden die Elementeigenschaften `constraint`, `deleteRule` und `updateRule` festgelegt. Für Referenzattribute und Child Components wird die Elementeigenschaft `deleteRule` definiert.

Zur Darstellung von Indices wird der Stereotyp `index` verwendet, der neben dem Indexnamen die am Index beteiligten Attribute enthält. Item Type Subsets werden mit Hilfe des Stereotyps `item type subset` umgesetzt. Ein Item Type Subset besteht aus einem Namen, einer Filterregel und den Attributen, die vom Basis-Item-Type übernommen wurden. Die Zuordnung von Indices und Item Type Subsets zu einem Item Type erfolgt über gewöhnliche Assoziationen.

In Abbildung 3.9 ist ein vereinfachter Ausschnitt aus dem ICM-Datenmodell des *eNoteHistory*-Projekts mit dem beschriebenen UML-Profil dargestellt. Das Datenmodell zeigt folgende Zusammenhänge: Ein Musikmanuskript besteht aus Skriptsektionen, die als Child Components modelliert wurden. Jeder Skriptsektion wird über das Referenzattribut `Typ` ein

Typ zugeordnet. Für den Item Type `MusicManuscript` wurde ein zusammengesetzter Schlüssel mit Hilfe des Indexes `ManuscriptKey` definiert. Das Attribut `LibraryCode` wird als Fremdschlüssel deklariert, welches auf das Schlüsselattribut eines Item Types `Library` (nicht dargestellt) zeigt. Dadurch wird sichergestellt, dass nur existierende Bibliothek-Codes gespeichert werden.

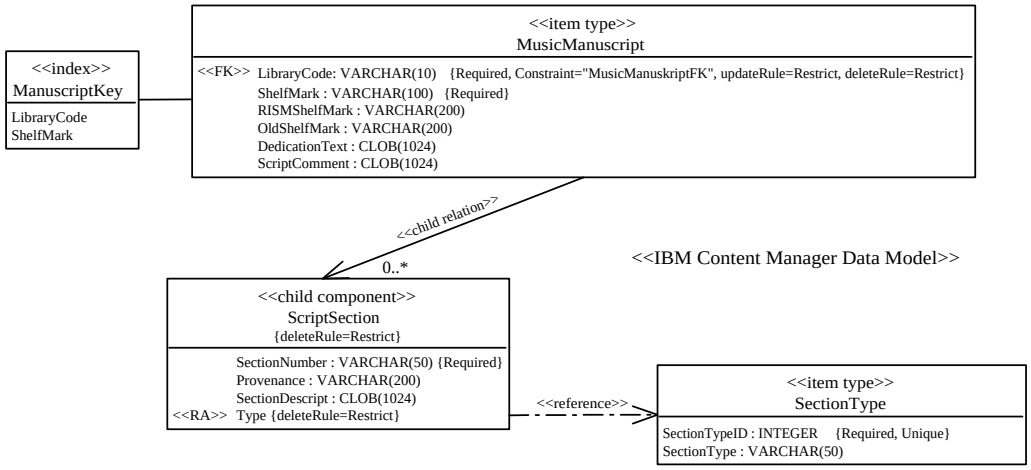


Abbildung 3.9: Beispiel für die Darstellung eines ICM-Datenmodells mit UML

3.3 IBM Content Manager Clients

Zwei Standard-Client-Anwendungen stehen für den Zugriff auf die *IBM Content Manager*-Server zur Verfügung, zum Einen der *IBM DB2 Content Manager Client for Windows* und zum Anderen der *IBM DB2 Content Manager eClient*. Beide Applikationen sind geeignet um Standardszenarios der *IBM Content Manager*-Produkte zu unterstützen und dienen als „out of the box“-Lösung zum Aufbau eines Content-Management-Systems. Darüberhinaus können eigene Client-Applikationen in Java und C++ erstellt werden.

IBM DB2 Content Manager Client for Windows

Der *IBM DB2 Content Manager Client for Windows* ist eine C++ Windows-Applikation zum Suchen und Importieren von Dokumenten. Die Applikation unterstützt unter anderem die Workflow-Funktionalität des Library Servers, Text- und Grafikannotation von Dokumenten, einfache und erweiterte Suche, Volltextsuche sowie die Versionierung von Dokumenten.

IBM DB2 Content Manager eClient

Der *IBM DB2 Content Manager eClient* ist eine Web-Applikation, die aus Java Server Pages (JSP) besteht. Anders als beim *Client for Windows* wird mit dieser Architektur das Konzept „einmal installieren - überall nutzen“ verfolgt. Der *eClient* wird als Enterprise Application

im EAR-Format ausgeliefert und in einem Application Server, wie z.B. dem *IBM WebSphere Application Server*, installiert. Nutzer können dann von jedem browserfähigen Arbeitsplatz über ein lokales Netzwerk oder das Internet auf die Anwendung zugreifen. Dies senkt den administrativen Aufwand, gleichzeitig aber auch die Performance. Der Funktionsumfang ähnelt in etwa dem der *Client for Windows*-Applikation.

Custom Clients

Für die Erstellung eigener Client-Anwendungen steht eine objektorientierte Schnittstelle sowohl für Java als auch für C++ zur Verfügung. Beide Klassenbibliotheken sind Bestandteil des im nächsten Abschnitt vorgestellten *IBM DB2 Information Integrator for Content* und werden zusammen mit diesem Produkt installiert. Über die objektorientierte Schnittstelle kann die gesamte Funktionalität des ICM-Datenmodells genutzt werden.

Zu Beachten ist, dass die Standard-Client-Anwendungen *Client for Windows* und *eClient* nur einen Teil des ICM-Datenmodells unterstützen. Die wichtigste Einschränkung ist die Festlegung auf das Document Model. Beide Client-Applikationen unterstützen keine einfachen Item Types, sondern ausschließlich Document Item Types. Außerdem fehlt die Unterstützung von Fremdschlüsseln und Referenzen. Links können nur über den Folder-Mechanismus genutzt werden. Child Components, die im ICM-Datenmodell beliebig verschachtelt werden können, zeigen beide Clients nur bis zur ersten Ebene an.

Durch diese Einschränkungen ergibt sich die Notwendigkeit, bereits beim Aufbau eines *IBM Content Manager*-Systems zu entscheiden, ob die IBM-Standard-Clients unterstützt oder ob eigene Client-Applikationen erstellt werden sollen. Bei einer Entscheidung zugunsten der Standard-Clients müssen die Restriktionen bei der Definition des Datenmodells entsprechend berücksichtigt werden. Eine Tabelle mit den Eigenschaften der Standard-Client-Applikationen bezüglich des ICM-Datenmodells befindet sich im Anhang in Tabelle B.1.

Eine ausführliche Beschreibung des *IBM DB2 Content Manager Client for Windows* und des *IBM DB2 Content Manager eClients* befindet sich in [RDZ02], [Cor03a] und [Cor03g]. Eine Beschreibung zum Erstellen eigener Client-Anwendungen befindet sich in [Cor03f].

3.4 IBM DB2 Information Integrator for Content

In Anwendungsszenarien, in denen sehr große Datenmengen verwaltet werden, wird der Datenbestand häufig nicht zentral gespeichert, sondern auf mehrere Rechner verteilt. In einigen Szenarien befinden sich diese Rechner zudem nicht im selben Rechenzentrum, sondern an verschiedenen Standorten. Trotzdem ist es wünschenswert, dass der Zugriff auf einen solchen Datenbestand, zentral administriert werden kann und dass Suchanfragen gleichzeitig an alle Server gestellt werden können.

Der *IBM DB2 Information Integrator for Content (IIC)* ist eine Middleware-Lösung zur Integration verschiedener Datenquellen, die diese Aufgaben löst. Die Software verbindet Client-Anwendungen mit unterschiedlichen Content Servern und ermöglicht Suchanfragen über Systemgrenzen hinweg. Ein *Content Server* oder Backend ist ein Software-System, dass Multimedia-Daten, Office-Dokumente oder ähnlichen Content mit den dazugehörigen Meta-Daten speichert.

Neben der Funktionalität zur Datenintegration sind im *IBM DB2 Information Integrator for Content* außerdem Werkzeuge zum Information Mining, ein Web Crawler sowie ein gegenüber dem *IBM DB2 Content Manager for Multiplatforms* erweitertes Workflow-System enthalten.

Eine ausführliche Beschreibung des *IBM DB2 Information Integrator for Content* befindet sich in den IBM-Handbüchern [Cor03d, Cor03b].

3.4.1 Systemarchitektur

Der *IBM DB2 Information Integrator for Content* besteht aus einer Administrationsdatenbank, dem *System Administration Client* zur Administration des Systems sowie verschiedenen Connectors zur Kommunikation mit der Administrationsdatenbank und den Content Servern. Die Architektur ermöglicht es Client-Anwendungen in einer Anfrage mehrere Content Server nach Dokumenten zu durchsuchen. Dazu werden Teile der lokalen Datenstrukturen der Content Server in ein föderiertes Schema integriert. Abbildung 3.10 veranschaulicht den Zusammenhang zwischen den verschiedenen Systemkomponenten.

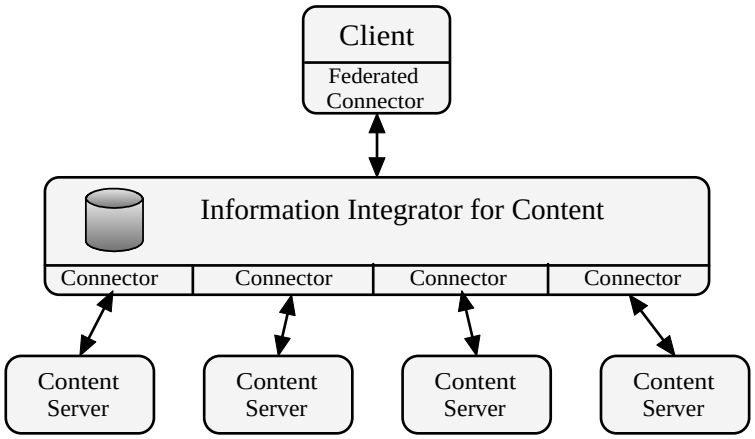


Abbildung 3.10: *IBM DB2 Information Integrator for Content* Architekturüberblick

3.4.2 Administrationsdatenbank

Die Administrationsdatenbank ist eine DB2-Datenbank, in der Nutzer und Gruppen, Zugriffsrechte, die Abbildung der loken Datenstrukturen auf das föderierte Schema und andere Verwaltungsinformationen gespeichert werden. Die Datenbank wird außerdem zur Umsetzung der erweiterten Workflow-Funktionalität und der Information-Mining-Funktionen verwendet. In einem Anwendungsszenario können verschiedene Administrationsdatenbanken eingerichtet werden, um die Rechenlast zu verteilen.

Wird der *IBM DB2 Information Integrator for Content* auf einem Rechner installiert, auf dem bereits eine *IBM DB2 Content Manager for Multiplatforms* Library-Server-Datenbank installiert ist, so kann die Administrationsdatenbank in diese Datenbank integriert werden, da die Library-Server-Datenbank viele Informationen enthält, die auch die Administrationsdatenbank benötigt.

3.4.3 System Administration Client

Der *System Administration Client* hat ähnliche Aufgaben wie der Administration Client des *IBM DB2 Content Manager for Multiplatforms*. Dazu gehören unter anderem:

- die Definition der Content Server für die verteilte Suche,
- die Definition von Nutzern und Gruppen,
- die Zuweisung von Rechten,
- die Administration der Workflow-Funktionalität und
- die Integration der lokalen Datenstrukturen in das föderierte Schema.

Ist im System bereits ein *ICM System Administration Client* installiert, kann der *IIC System Administration Client* in diese Anwendung integriert werden, so dass sowohl der *IBM DB2 Content Manager for Multiplatforms* als auch der *IBM DB2 Information Integrator for Content* über dieselbe Java-Applikation administriert werden können.

3.4.4 Connectors

Connectors bilden die Schnittstelle zwischen den Clients des *IBM DB2 Information Integrator for Contents*, den verschiedenen Content Servern und der Administrationsdatenbank. Connectors werden in einer Klassenbibliothek zur Verfügung gestellt, die sowohl in Java als auch in C++ genutzt werden kann.

Der *Federated Connector* dient zur Anbindung von Client-Applikationen an die Administrationsdatenbank. Um eine Suche über mehrere Content Server gleichzeitig zu starten, muss einer Client-Applikation dieser Connector zur Verfügung stehen, damit eine Anmeldung an die Administrationsdatenbank erfolgen kann. Der Federated Connector ist außerdem zur Ausführung des *System Administration Clients* erforderlich.

Neben dem Federated Connector sind im *IBM DB2 Information Integrator for Content* Connectors für verschiedene andere Content Server enthalten. Dazu gehören unter anderem Connectors für die *DB2 Universal Database*, den *IBM DB2 Content Manager for Multiplatforms* und *Lotus Domino.Doc*. Connectors erlauben dem *IBM DB2 Information Integrator for Content*, sich an verschiedene Content Server anzumelden und Anfragen an die Systeme zu stellen. Darüberhinaus dienen Connectors als Schnittstelle zwischen selbsterstellten Client-Applikationen und einem bestimmten Content Server. Um beispielsweise eine C++ Client-Anwendung für den *IBM DB2 Content Manager for Multiplatforms Version 8* zu erstellen, muss der entsprechende Connector des *IBM DB2 Information Integrator for Content* installiert und der Client-Anwendung zugänglich gemacht werden.

3.5 Zusammenfassung

In diesem Kapitel wurden die wesentlichen Komponenten und Konzepte der in dieser Diplomarbeit verwendeten *IBM Content Manager*-Produkte vorgestellt.

Zentraler Bestandteil der *IBM Content Manager*-Produktreihe ist der *IBM DB2 Content Manager for Multiplatforms*. Im ersten Abschnitt dieses Kapitels wurden der Funktionsumfang sowie die Systemarchitektur dieses Produktes beschrieben. Das System basiert auf einer Dreiecksarchitektur, die aus Client-Anwendungen, einem Library Server und einem oder mehreren Resource Managern besteht. Die Aufgaben und die Funktionsweise der einzelnen Systemkomponenten wurden in den entsprechenden Unterkapiteln erläutert.

Im Anschluss daran wurde das Datenmodell des *IBM DB2 Content Manager for Multiplatforms* vorgestellt. Item Types dienen im ICM-Datenmodell als Vorlagen für die Erstellung von Datenobjekten, die als Items bezeichnet werden. Mit Links, Referenzen und Fremdschlüsseln wurden Konstrukte beschrieben, die zum Aufbau von Beziehungen zwischen Items genutzt werden können.

Im dritten Abschnitt dieses Kapitels wurden die beiden IBM Standard-Client-Anwendungen vorgestellt, die als „out of the box“-Lösung beim Aufbau eines *IBM Content Management*-Systems genutzt werden können. Beide Applikationen unterstützen lediglich einen Teil des ICM-Datenmodells. Für spezielle Anwendungsszenarien müssen eigene Client-Applikationen erstellt werden.

Zum Abschluss wurde mit dem *IBM DB2 Information Integrator for Content* eine Middleware-Lösung für die Integration verschiedener Content Server vorgestellt, über die Client-Applikationen innerhalb einer Anfrage mehrere Archivsysteme durchsuchen können. Darüberhinaus wurde mit den Connector-Klassen des *IBM DB2 Information Integrator for Content* eine Schnittstelle beschrieben, die zum Erstellen von Client-Applikationen für den Zugriff auf einzelne Content Server oder eine Föderation von Content Servern genutzt werden kann.

Kapitel 4

Methoden zur Archivintegration

Für die Integration von Dokumentenarchiven werden in der vorliegenden Diplomarbeit zwei Ansätze betrachtet: Die Integration durch eine Archivföderation mit Hilfe eines Föderierungsdienstes und die Migration von Dokumentenarchiven in eine Content-Manager-Umgebung. Beide Vorgehensweisen werden in diesem Kapitel überblicksartig vorgestellt, bevor sie in den nächsten Kapiteln detailliert beschrieben werden. Dabei werden zunächst zu beiden Themen, an Hand einiger ausgewählter Arbeiten, existierende Forschungsansätze skizziert. Aufbauend darauf werden Konzepte erläutert, die speziell für die Integration von Dokumentenarchiven in eine *IBM Content Manager*-Umgebung entwickelt wurden.

4.1 Integration durch Föderation

Bei einer Föderation von Datenarchiven werden diese über eine Middleware zu einem föderierten Datenbanksystem verbunden. Dieser Ansatz der Archivintegration wird im Allgemeinen als „Information Integration“ bezeichnet. Dabei werden Teile der lokalen Schemata der meist heterogenen Archive in ein föderiertes Schema integriert. Auf diesem Schema können anschließend Anwendungen erstellt werden, die den gesamten Datenbestand nutzen.

4.1.1 Föderierte Datenbanksysteme

Bevor eine Klassifikation föderierter Datenbanksysteme vorgestellt und eine allgemeine Vorgehensweise zur Integration von Datenarchiven beschrieben wird, soll zunächst der Begriff „föderiertes Datenbanksystem“ definiert werden. Die folgenden Ausführungen basieren dabei auf [SL90] und [Con97]. Beiden Arbeiten geben einen ausführlichen Überblick über das Thema föderierte Datenbanksysteme.

Begriffsdefinition und Architektur

Ein *Datenbanksystem (DBS)* besteht aus einer Software, die als *Datenbank-Management-System (DBMS)* bezeichnet wird und einer oder mehreren Datenbanken, die davon verwaltet werden. Ein *föderiertes Datenbanksystem (FDBS)* besteht aus einem Verbund von

kooperierenden aber autonomen Komponentendatenbanksystemen (KDBS). Die Software zur Verwaltung der Komponentendatenbanksysteme wird als *föderiertes Datenbank-Management-System (FDBMS)* bezeichnet. Ein Komponentendatenbanksystem kann ein zentralisiertes, verteiltes oder wiederum ein föderiertes Datenbanksystem sein. Die Komponentendatenbanksysteme können sich unter anderem hinsichtlich des Datenmodells, der Anfragesprache und der Transaktionsverwaltung unterscheiden. Eine der wichtigsten Eigenschaften föderierter Datenbanksysteme besteht darin, dass auf einem Komponentendatenbanksystem trotz seiner Föderierung weiterhin lokale Anwendungen ausgeführt werden können, die direkt auf dem Komponentendatenbanksystem arbeiten. Abbildung 4.1 veranschaulicht die allgemeine Architektur eines föderierten Datenbanksystems.

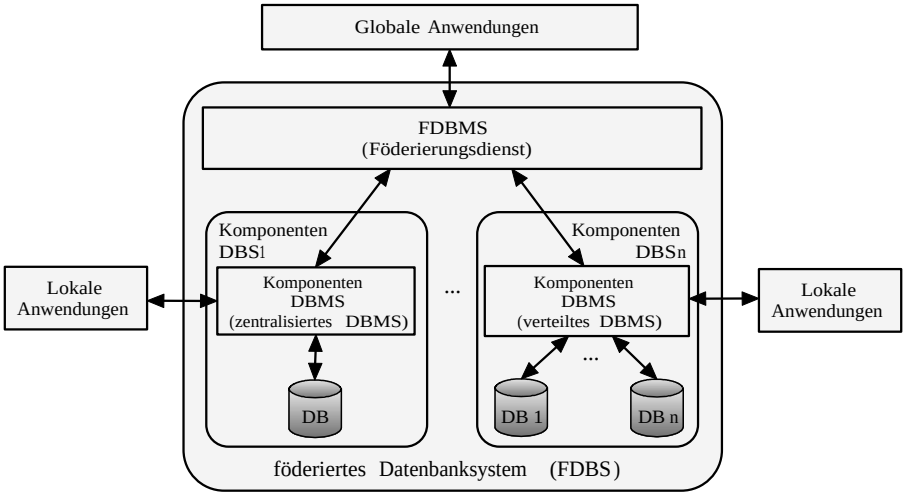


Abbildung 4.1: Allgemeine Architektur eines föderierten Datenbanksystems

Klassifikation

Ein Datenbanksystem kann entweder zentralisiert oder verteilt sein. Ein *zentralisiertes Datenbanksystem* besteht aus einem zentralisierten Datenbank-Management-System, das eine Datenbank auf dem selben Computer-System verwaltet. Ein *verteiltes Datenbanksystem* besteht aus einem verteilten Datenbank-Management-System, das mehrere Datenbanken verwaltet, die sich auf verschiedenen Computer-Systemen befinden können. Ein *Multidatenbanksystem (MDBS)* unterstützt Operationen auf mehreren Komponentendatenbanksystemen. Jedes der Komponentendatenbanksysteme wird dabei durch ein eigenes Datenbank-Management-System verwaltet.

Föderierte Datenbanksysteme gehören zur Kategorie der Multidatenbanksysteme. Diese können nach ihrer Architektur und dem Grad der Integration der Komponentendatenbanksysteme in verschiedene Klassen eingeteilt werden. In [SL90] wird eine Taxonomie vorgeschlagen, in der Multidatenbanksysteme nach der Autonomie der Komponentendatenbanksysteme unterteilt werden. Die Taxonomie ist in Abbildung 4.2 dargestellt.

Multidatenbanksysteme werden in nicht föderierte und föderierte Datenbanksysteme unterteilt. Ein *nicht föderiertes Datenbanksystem* besteht aus Komponentendatenbanksy-

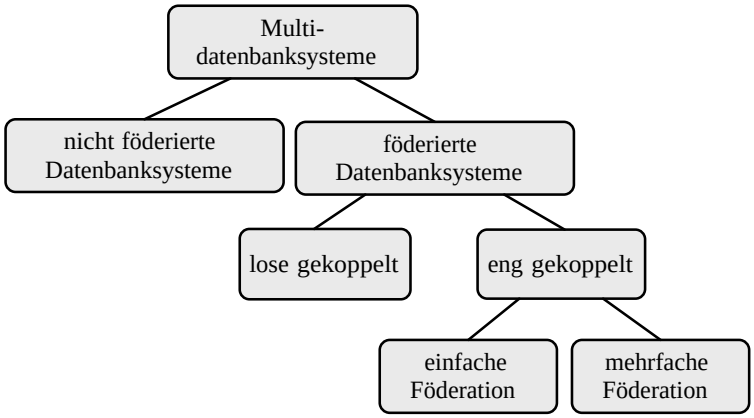


Abbildung 4.2: Taxonomie von Multidatenbanksystemen

stemen, die keine Autonomie besitzen, das heißt, das System wird vollständig auf der übergeordneten Ebene verwaltet. Im Gegensatz zu föderierten Datenbanksystemen wird nicht zwischen lokalen und globalen Anwendungen unterschieden.

In der Klasse der föderierten Datenbanksysteme werden, je nach dem wer die Föderation verwaltet, lose gekoppelte und eng gekoppelte Systeme unterschieden. In *lose gekoppelten* Systemen wird die Föderation von den Nutzern des Systems verwaltet. Dabei ist jeder Nutzer selbst für den Aufbau der Föderation und die Auswahl der daran beteiligten Datenquellen verantwortlich. In *eng gekoppelten* Systemen werden diese Aufgaben von einem Administrator übernommen.

Während in lose gekoppelten Systemen immer mehrere föderierte Schemata erstellt werden können, werden eng gekoppelte Systeme nach diesem Kriterium in einfache Föderationen und mehrfache Föderationen unterteilt. Ein eng gekoppeltes föderiertes Datenbanksystem wird als *einfache Föderation* bezeichnet, wenn lediglich ein föderiertes Schema erstellt werden kann. In einer *mehrfachen Föderation* können dagegen mehrere föderierte Schemata definiert werden.

Aufbau eines föderierten Datenbanksystems

Der Prozess des Aufbaus eines föderierten Datenbanksystems bzw. die Integration einer neuen Datenquelle wird in [SL90] in drei Phasen unterteilt: die Pre-Integration, die Entwicklung des föderierten Datenbanksystems und die Nutzung. Obwohl in [SL90] davon ausgegangen wird, dass das Datenbank-Management-System des föderierten Datenbanksystems erst implementiert werden muss, so kann die im Folgenden beschriebene Vorgehensweise auch für die Integration eines Datenarchivs mit Hilfe eines bereits existierenden Föderierungsdienstes angewendet werden.

In der *Pre-Integrations*-Phase werden Daten, die nicht durch ein Datenbank-Management-System verwaltet werden, sondern z.B. in Dateien vorliegen, aufbereitet, so dass sie in eine Föderation integriert werden können. Dafür gibt es zwei Möglichkeiten: zum Einen können die Daten in ein Datenbanksystem migriert werden und zum Anderen kann das Dateisystem so erweitert werden, dass es die Funktionen eines Datenbank-Management-

Systems zur Verfügung stellt.

In der zweiten Phase, *der Entwicklung des föderierten Datenbanksystems*, werden die lokalen Schemata, das föderierte Schema, die externen Schemata sowie Abbildungen zwischen diesen Schemata definiert. Darüberhinaus erfolgt in dieser Phase die Entwicklung der Software zur Realisierung des Föderierungsdienstes. Da in der vorliegenden Diplomarbeit der *IBM DB2 Information Integratior for Content* als Föderierungsdienst verwendet wird, entfällt dieser Schritt.

Die dritte Phase des Aufbaus eines föderierten Datenbanksystems umfasst die Verwaltung und die Nutzung der integrierten Datenbanken. Das föderierte Schema, das in der zweiten Phase erstellt wurde, erlaubt den Zugriff auf alle föderierten Komponentendatenbanksysteme. Darauf aufbauend können Anwendungen entwickelt werden, die den gesamten Datenbestand der Föderation nutzen.

4.1.2 Föderierung mit dem IBM DB2 Information Integrator for Content

Der *IBM DB2 Information Integrator for Content* ist eine Middleware-Lösung, die in der vorliegenden Diplomarbeit als Föderierungsdienst genutzt wird, um Dokumentenarchive in ein föderiertes Datenbanksystems zu integrieren. In der in Abbildung 4.2 dargestellten Taxonomie gehört das System in die Kategorie der eng gekoppelten föderierten Datenbanksysteme, die eine einfache Föderation erlauben. Die Auswahl der an der Föderation beteiligten Komponentendatenbanksysteme und die Integration der Datenstrukturen ist Aufgabe des Administrators. Der Begriff „Komponentendatenbanksystem“ wird im Umfeld des *IBM Content Manager* durch den Begriff „Content Server“ ersetzt.

Der *IBM DB2 Information Integrator for Content* bietet eine transparente Sicht auf verteilt gespeicherte Daten. Dazu wird ein föderiertes Schema erstellt, das verschiedene lokale Schemata integriert. Die Transparenz wird durch die Abbildung zwischen dem föderierten Schema und den verteilten lokalen Schemata verwaltet.

Die Integration eines Dokumentenarchivs in den *IBM DB2 Information Integrator for Content* besteht im Wesentlichen aus der Integration des lokalen Schemas in das föderierte Schema. Voraussetzung dafür ist die Installation und Konfiguration eines entsprechenden Connectors, so dass der Föderierungsdienst den Inhalt des Content Server auslesen kann. Im Anschluss daran werden auf der Föderierungsschicht föderierte Entity-Typen definiert, deren Attribute auf die Attribute eines lokalen Entity-Typs abgebildet werden. Die föderierten Entity-Typen werden in Suchvorlagen integriert, in denen festgelegt wird, wonach gesucht wird, wie die Suchergebnisse präsentiert werden und wer die Suchvorlage verwenden darf. Anschließend können Anwendungen erstellt werden, die sich mit Hilfe des Federated Connectors an die Administrationsdatenbank des *IBM DB2 Information Integrator for Content* anmelden und den gesamten Datenbestand der Föderation nutzen.

In Anlehnung an die in Abschnitt 4.1.1 beschriebene Vorgehensweise zum Aufbau eines föderierten Datenbanksystems werden im Folgenden drei Schritte zur Integration eines Dokumentenarchivs in den *IBM DB2 Information Integrator for Content* definiert.

- Installation und Konfiguration des entsprechenden Connectors
- Integration der relevanten Bestandteile des lokalen Schemas in das föderierte Schema

- Erstellung globaler Anwendungen

Der Vorteil einer Archivföderation besteht vor allem in dem relativ geringen Aufwand, der mit dieser Art der Integration verbunden ist. Die Daten der Quellarchive können in den heterogenen Systemen bleiben und müssen nicht in ein neues System übertragen werden, wie dies bei Migrationslösungen der Fall ist. Des Weiteren können alle bestehenden Anwendungen, welche für die einzelnen Archive bereits existieren, uneingeschränkt weiter genutzt werden.

Ein Nachteil bei der Integration über den *IBM DB2 Information Integrator for Content* ist das schwache Datenmodell der Föderationsschicht, beispielsweise gibt es keine Konstrukte zum Aufbau von Beziehungen zwischen den Objekten des föderierten Schemas. Darüberhinaus ergeben sich auf Grund der aufwendigen Anfrageverarbeitung, in der eine Transformation zwischen dem föderierten Schema und den entsprechenden lokalen Schemata erfolgt, Performance-Nachteile gegenüber dem direkten Zugriff auf ein Archivsystem.

4.2 Integration durch Migration

Neben der Integration über einen Föderierungsdienst kann der Datenbestand eines Dokumentenarchivs extrahiert, transformiert und in ein zentrales Datenverwaltungssystem importiert werden. Dieser Prozess wird als *Migration* bezeichnet. Obwohl das Datenmodell des *IBM DB2 Content Manager for Multiplatforms* lediglich die Grundzüge eines objektorientierten Datenmodells unterstützt, wurde das in der vorliegenden Diplomarbeit entwickelte Migrationsverfahren vor allem durch Arbeiten auf dem Gebiet der Migration in objektorientierte Systeme beeinflusst. Zwei dieser Arbeiten werden im Folgenden vorgestellt, bevor anschließend ein Verfahren zur Migration von Dokumentenarchiven in den *IBM DB2 Content Manager for Multiplatforms* entwickelt wird.

4.2.1 Migration in objektorientierte Systeme

In der Dissertation von Andreas Behm (siehe [Beh01]) wird ein Konzept zur Migration relationaler Datenbanken in objektorientierte Systeme beschrieben. Die Migration wird in zwei Phasen unterteilt. In der ersten Phase wird das relationale Datenbankschema überarbeitet und in ein objektorientiertes Datenmodell transformiert. Anschließend erfolgt die Migration der relationalen Daten in ein objektorientiertes Datenbanksystem.

Für die Umsetzung dieses Verfahrens wurde ein „Zwischenmodell“ (SOT - Semi Object Types) entwickelt, mit dem sowohl die Schema-Transformation als auch die Datenmigration realisiert wird. Das Zwischenmodell unterstützt alle objektorientierten Modellierungskonstrukte und ermöglicht eine flexible Schema-Transformation. Darüberhinaus wurde eine Algebra für die formale Definition der Datenmigration entwickelt.

Die Schema-Transformation erfolgt in drei Schritten. Im ersten Schritt wird das relationale Schema in ein SOT-Schema transformiert. Das SOT-Schema wird daraufhin umgeformt, so dass ein wohlgeformtes objektorientiertes Schema entsteht. Abschließend erfolgt die Abbildung des SOT-Schemas in ein objektorientiertes Schema, das den ODMG-Standard erfüllt. Die notwendigen Schritte zur Datenmigration werden dabei automatisch generiert.

In der Diplomarbeit von Christina Kipp (siehe [Kip96]) wird ein Verfahren zur Migration von relationalen Datenbanken in das objektorientierte Datenbanksystem O_2 beschrieben. Die Transformation der relationalen Datenstrukturen erfolgt dabei mit Hilfe von ER-Diagrammen. Dafür wurden Regeln definiert, welche die wichtigsten ER-Konstrukte auf das O_2-Datenmodell abbilden. Dieser Ansatz wird in der vorliegenden Diplomarbeit für die Migration von Dokumentenarchiven in den *IBM DB2 Content Manager for Multiplatforms* aufgegriffen.

Für die Datenmigration werden die Datenbanktabellen zunächst temporär in der O_2-Datenbank gespeichert. Dabei werden die Informationen über die Schlüssel-Fremdschlüssel-Beziehungen zwischen den Tabellen mit übertragen und in speziellen Attributen der temporären Klassen abgelegt. Anschließend werden O_2-Methoden generiert, welche die temporären Tabellenobjekte in ihre endgültigen Klassenobjekte kopieren und abschließend die Beziehungen zwischen den migrierten Objekten mit Hilfe der Schlüssel-Fremdschlüssel-Beziehungen wiederherstellen.

In beiden Arbeiten werden zur Migration einer relationalen Datenbank zwei Aufgaben identifiziert. Zum Einen muss die relationale Datenstruktur auf das objektorientierte Zieldatenmodell abgebildet werden und zum Anderen muss der Inhalt des Datenarchivs migriert werden. Das im folgenden Abschnitt vorgestellte Verfahren zur Migration von Dokumentenarchiven in den *IBM DB2 Content Manager for Multiplatforms* basiert auf dem in [Kip96] beschriebenen Ansatz.

4.2.2 Migration in den IBM DB2 Content Manager for Multiplatforms

Als Zielplattform für das Migrationsszenario wird in der vorliegenden Diplomarbeit der *IBM DB2 Content Manager for Multiplatforms* verwendet. Die Umsetzung der Migration erfolgt in zwei Schritten. Damit die Daten migriert werden können wird zunächst das Datenmodell des Quellarchivs auf das Datenmodell des *IBM DB2 Content Manager for Multiplatforms* abgebildet. Anschließend wird der Inhalt des Dokumentenarchivs migriert.

In der vorliegenden Diplomarbeit wird für die Abbildung des Datenmodells nicht das Implementierungsmodell eines Dokumentenarchivs betrachtet, sondern eine konzeptuelle Repräsentation des Datenbestandes. Dafür wird das in Kapitel 2.5.2 entwickelte UML-Profil verwendet. Durch konzeptuelle Datenmodelle können die Datenstrukturen verschiedener Archivsysteme in abstrakter Form dargestellt werden. Als zu migrierende Datenquellen kommen unter anderem relationale Datenbanken, objektrelationale Datenbanken, XML-Datenbanken sowie objektorientierte Datenbanken in Frage. Die Verwendung von konzeptuellen Datenmodellen bietet folgende Vorteile gegenüber der direkten Abbildung eines Implementierungsmodells:

- Die Qualität des entstehenden ICM-Datenmodells ist besser als die eines ICM-Datenmodells, das durch eine direkte Abbildung entsteht, da ein konzeptuelles Datenmodell vor allem bei der Modellierung von Beziehungen semantisch reichere Konzepte bietet, als die meisten Implementierungsmodelle. Ziel der Abbildung ist es, ein Datenmodell zu erzeugen, dass die selbe Qualität wie ein von Grund auf neu entworfenes ICM-Datenmodell besitzt.

- Soll ein Datenbestand in den *IBM DB2 Content Manager for Multiplatforms* integriert werden, der bisher noch nicht in einem Archiv vorliegt und dessen Struktur erst modelliert werden muss, so bietet sich ebenfalls der Weg über ein konzeptuelles Datenmodell an. Die Datenstrukturen werden zunächst auf konzeptueller Ebene modelliert und dann auf das ICM-Datenmodell abgebildet.

- Es ist nicht notwendig, je eine Abbildungsvorschrift für die Datenmodelle jedes in Frage kommenden Quellarchivs zu entwickeln, statt dessen muss lediglich eine Vorschrift zur Abbildung von konzeptuellen Datenmodellen auf das ICM-Datenmodell definiert werden.

In den Anwendungsszenarien, die in dieser Diplomarbeit betrachtet werden, wird davon ausgegangen, dass ein zu migrierendes Archiv bereits in einer der oben genannten Formen vorliegt. Zu dem Implementierungsdatenmodell des Dokumentenarchivs muss also ein konzeptuelles Datenmodell erstellt werden, falls dies nicht bereits existiert. Ein guter Datenbankentwurf beginnt in der Regel mit dem Erstellen eines konzeptuellen Datenmodells, so dass ein solches Modell für die meisten Archive vorliegen sollte. Ist kein konzeptuelles Datenmodell vorhanden, so muss dies durch Reverse Engineering aus dem Datenmodell des Dokumentenarchivs gewonnen werden.

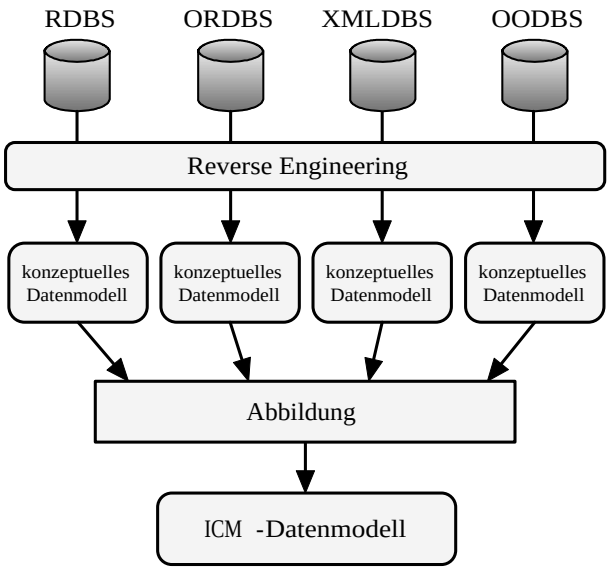


Abbildung 4.3: Migration von Datenstrukturen

Abbildung 4.3 veranschaulicht den Prozess der Migration der Datenstruktur eines Dokumentenarchivs in das ICM-Datenmodell. In Kapitel 6 wird ein Algorithmus zur Abbildung konzeptueller Datenmodelle auf das ICM-Datenmodell vorgestellt.

Nachdem die Datenstruktur eines Dokumentenarchivs auf das ICM-Datenmodell abgebildet wurde, kann der Inhalt des Dokuementenarchivs migriert werden. Dazu werden zunächst die Datenobjekte des Archivs in den *IBM DB2 Content Manager for Multiplat-*

forms übertragen. Anschließend werden die Beziehungen zwischen den Datenobjekten wiederhergestellt. Nach der Datenübertragung wird das Quellarchiv nicht mehr benötigt.

Die Anwendungen, die bisher zum Zugriff auf das Dokumentenarchiv genutzt wurden, müssen nach Abschluss der Migration entweder angepasst werden, so dass sie über die objektorientierte Schnittstelle auf den *IBM DB2 Content Manager for Multiplatforms* zugreifen können, oder die Funktionalität der Anwendungen muss in eine globale Anwendung integriert werden.

Die Migration eines Dokumentenarchivs in einen *IBM DB2 Content Manager for Multiplatforms* kann wie folgt zusammengefasst werden:

- Migration der Datenstrukturen mit Hilfe eines konzeptuellen Datenmodells
- Migration der Daten, Beziehungen und Methoden
- Anpassung der existierenden Anwendungen bzw. Integration der Anwendungsfunktionalität in eine globale Anwendung

Der Vorteil einer Migration von Archiven in den *IBM DB2 Content Manager for Multiplatforms* besteht vor allem darin, dass die Fähigkeiten des Systems, wie z.B. die automatische Migration von Objekten zwischen verschiedenen Datenträgern, die Anbindung an Archivsysteme oder die Integration von Streaming Servern voll ausgenutzt werden können. Weiterhin bietet der *IBM DB2 Content Manager for Multiplatforms* ein leistungsstärkeres Datenmodell als der *IBM DB2 Information Integrator for Content* auf der Föderationsschicht. Darüberhinaus kann der gesamte Datenbestand von verschiedenen Archiven nach einer Migration zentral administriert und verwaltet werden.

Ein Nachteil dieser Integrationslösung ist der nicht unerhebliche Aufwand der notwendig ist, um die Daten eines Dokumentenarchivs in den *IBM DB2 Content Manager for Multiplatforms* zu migrieren und die bereits existierenden Anwendungen anzupassen.

4.3 Zusammenfassung

In diesem Kapitel wurden zwei Verfahrensweisen vorgestellt, mit denen Archive in eine *IBM Content Manager*-Umgebung integriert werden können. Zum Einen die Integration über einen Föderierungsdienst und zum Anderen die Migration von der Plattform des Quellarchivs. Die Vor- und Nachteile beider Verfahren wurden analysiert. Die Entscheidung für eines der Integrationsverfahren ist unter anderem abhängig vom jeweiligen Anwendungsszenario. Eine Archivföderation ist vor allem dann interessant, wenn Dokumentenarchive mit ähnlichem oder gleichem Inhalt auf mehrere Standorte verteilt sind und der Datenbestand weiterhin lokal verwaltet werden soll. Eine Archivföderation ermöglicht in diesem Fall eine Integration der Datenbestände, bei der die Autonomie der Datenquellen erhalten bleibt.

Eine Migrationslösung dagegen gewährleistet eine zentrale Administration verschiedener Dokumentenarchive. Außerdem bieten die speziellen Dokumentenverwaltungsfunktionen des *IBM DB2 Content Manager for Multiplatforms* Vorteile gegenüber konventionellen Datenverwaltungssystemen wie z.B. relationalen Datenbanken.

Die notwendigen Schritte zur Umsetzung beider Integrationsansätze werden in den folgenden Kapiteln detailliert beschrieben.

Kapitel 5

Föderierung mit dem IBM DB2 Information Integrator for Content

Die Integration eines Dokumentenarchivs mit Hilfe des *IBM DB2 Information Integrator for Content* reduziert sich im Wesentlichen auf die Konfiguration des Föderationsdienstes, da der Archivinhalt unangetastet bleibt. Zur Integration wird das Dokumentenarchiv als Content Server definiert und der gewünschte Teil des lokalen Schemas in das föderierte Schema integriert.

5.1 Definition von Content Servern

Bevor Bestandteile der Datenstruktur eines Content Servers in das föderierte Schema integriert werden können, muss der Content Server auf der Föderationsschicht definiert werden. Dies erfolgt mit Hilfe des *System Administration Clients*. Voraussetzung dafür ist, dass der Content Server durch einen entsprechenden Connector unterstützt wird. Der *IBM DB2 Information Integrator for Content* bietet Connectors für den Zugriff auf die folgenden Content Server (vgl. [Cor03f]):

- Content Manager Version 8
- Content Manager Version 7
- Domino.Doc
- Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo for AS/400
- DB2
- DB2 DataJoiner
- Information Catalog
- DB2 Warehouse Manager Information Catalog Manager
- JDBC/ODBC Server

Falls ein Server nicht durch einen der vordefinierten Typen unterstützt wird, ist es möglich, eigene Server-Typen zu definieren. Dazu müssen ein entsprechender Connector in Java oder C++ sowie eine Server-Definitionsklasse selbst entwickelt werden.

Nachdem ein Content Server definiert wurde, kann die Datenstruktur eines Servers ausgelesen werden. Über den *System Administration Client* können anschließend Teile der lokalen Datenstruktur in das föderierte Schema integriert werden. Für den Zugriff auf einen Content Server erfolgt eine Abbildung der im *IBM DB2 Information Integrator for Content* definierten Nutzer auf einen oder mehrere lokale Nutzer des jeweiligen Content Servers. Eine Client-Anwendung muss sich daher nur gegenüber der Administrationsdatenbank authentifizieren.

5.2 Integration von lokalen Datenstrukturen

Eine *föderierte Suche* ist eine Anfrage einer Client-Anwendung, in der gleichzeitig verschiedene Content Server durchsucht werden. Die Verteilung der Daten bleibt dabei für den Nutzer transparent. Dafür werden auf der Föderationsschicht föderierte Entity-Typen und Suchvorlagen definiert, welche die lokalen Datenstrukturen verschiedener Content Server integrieren. Abbildung 5.1 veranschaulicht diesen Zusammenhang.

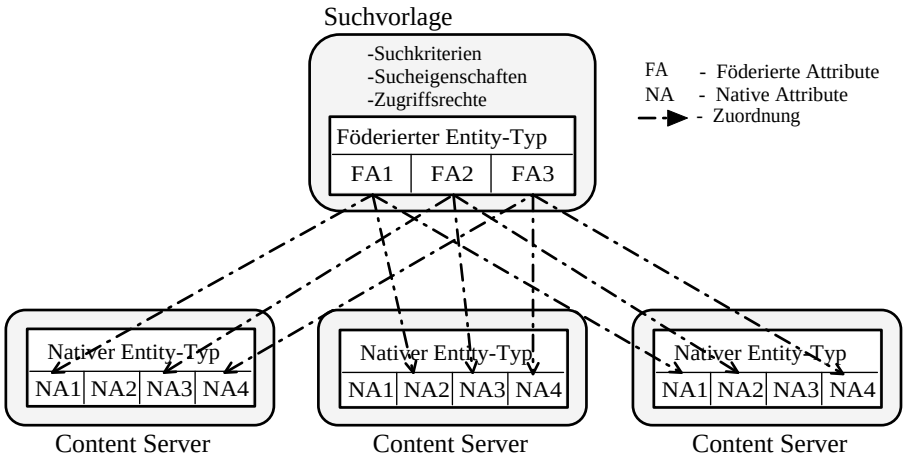


Abbildung 5.1: Datenintegration mit dem *IBM DB2 Information Integrator for Content*

5.2.1 Föderierte Entity-Typen

Nachdem eine Verbindung zu einem Content Server definiert wurde, besteht der nächste Schritt in der Definition von föderierten Entity-Typen. Mit Hilfe dieser Entity-Typen werden anschließend Suchvorlagen erstellt. Die Definition eines föderierten Entity-Typs umfasst:

- die Definition des Entity-Typs,
- die Definition von föderierten Attributen,

- die Abbildung föderierter Attribute und
- die Zuweisung von Parametern.

Ein föderierter Entity-Typ besteht aus föderierten Attributen und Eigenschaften. Die Attribute eines föderierten Entity-Typs werden auf die Attribute eines nativen Entity-Typs eines Content Servers abgebildet. Native Entity-Typen können sehr unterschiedlich strukturiert sein, da die einzelnen Content Server Informationen auf verschiedene Art speichern und organisieren. Beispiele für native Entity-Typen sind unter anderem die Tabellen einer DB2-Datenbank, Item Types eines IBM Content Managers der Version 8 oder Index-Klassen eines IBM Content Managers der Version 7. Die föderierten Attribute werden in diesen Fällen auf die Spalten einer Tabelle, auf die Attribute eines Item Types bzw. auf die Attribute einer Index Klasse abgebildet. Einem föderierten Attribut können dabei mehrere native Attribute verschiedener Content Server zugeordnet werden. Das Ergebnis ist ein föderierter Entity-Typ, mit dem ähnliche Strukturen auf verschiedenen Servern durchsucht werden können. Ein Entity des föderierten Entity-Typs repräsentiert dabei ein Objekt eines Content Servers.

5.2.2 Suchvorlagen

Mit Hilfe der föderierten Entity-Typen werden Suchvorlagen erstellt. Eine *Suchvorlage* nutzt die Abbildungsdefinition, die in einem föderierten Entity-Typ gespeichert ist, um native Attribute eines Content Server zu durchsuchen. Ein föderierter Entity-Typ kann in mehreren Suchvorlagen verwendet werden, jede Suchvorlage basiert dagegen auf genau einem föderierten Entity-Typ. Innerhalb einer Suchvorlage wird unter anderem festgelegt, wonach gesucht wird, wie die Suchergebnisse präsentiert werden und wer die Suchvorlage verwenden darf. Die Definition einer Suchvorlage beinhaltet die folgenden Schritte:

- Definition der Suchvorlage
- Definition von Suchkriterien
- Definition von Sucheigenschaften
- Zuweisung von Zugriffsrechten

Eine Suchvorlage besteht aus Suchkriterien. *Suchkriterien* haben ähnliche Aufgaben wie die Attribute eines Entity-Typs. Einem Suchkriterium wird ein Attribut des föderierten Entity-Typs zugeordnet, auf dem die Suchvorlage basiert. Dabei wird festgelegt, ob das Suchkriterium in einer Anfrage verwendet werden kann oder ob es lediglich im Ergebnis angezeigt wird und welche Vergleichsoperationen innerhalb einer Anfrage auf das Suchkriterium angewendet werden können. Darüberhinaus kann ein Default-Wert für jedes Suchkriterium definiert werden.

Für eine Suchvorlage können Sucheigenschaften definiert werden. *Sucheigenschaften* beeinflussen die Suche, die Struktur und die Anzeige der Anfrageergebnisse. Unter anderem kann festgelegt werden, ob innerhalb einer Anfrage alle Suchkriterien der Suchvorlage erfüllt sein müssen oder lediglich ein beliebiges, wie das System auf den Ausfall eines

in der Suchvorlage verwendeten Content Servers reagiert und in welcher Reihenfolge die Suchkriterien im Anfrageergebnis angezeigt werden. Darüberhinaus können Regeln definiert werden, so dass die Attributwerte verschiedener Content Server im Suchergebnis einheitlich dargestellt werden. Sind beispielsweise Wochentage auf einem Content Server mit dem vollständigen Namen und auf einem anderen als Abkürzung gespeichert (z.B. Montag und Mo), so kann für das Suchergebnis eine einheitliche Darstellung festgelegt werden.

Mit Hilfe der Suchvorlage erfolgt auch die Zugriffsverwaltung. Jeder Suchvorlage kann eine Liste von Nutzern und Gruppen zugeordnet werden, welche die Suchvorlage verwenden dürfen.

5.2.3 Beispiel

Ein Beispiel soll die Verwendung von föderierten Entity-Typen und Suchvorlagen verdeutlichen. Abbildung 5.2 zeigt drei Content Server unterschiedlichen Typs, die für eine föderierte Suche integriert werden sollen.

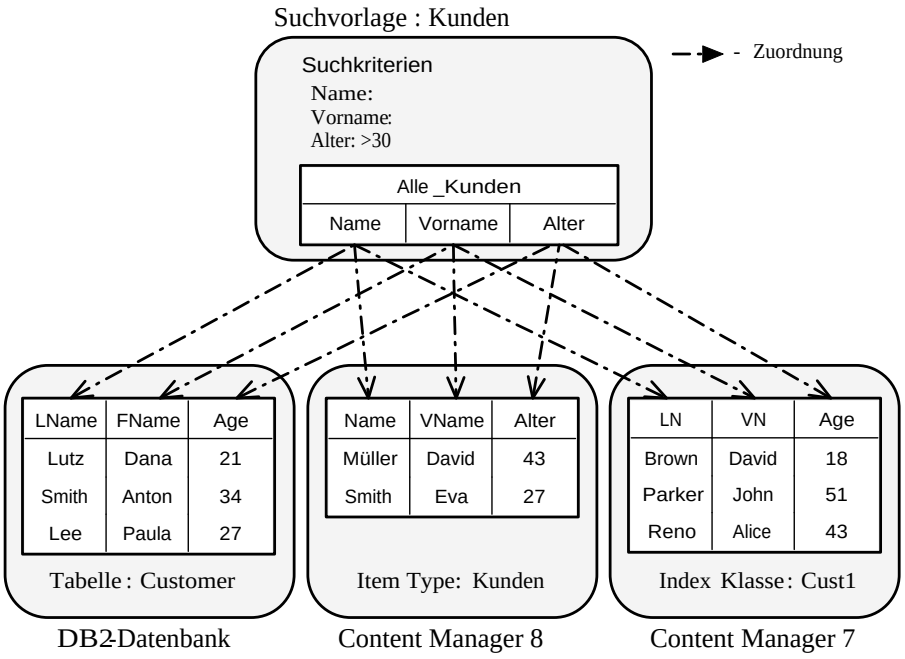


Abbildung 5.2: Beispiel für die Datenintegration mit dem *IBM DB2 Information Integrator for Content*

Obwohl alle drei Content Server ihre Daten unterschiedlich verwalten, speichern sie die selbe Art von Kundeninformationen. Ziel der Integration ist es, alle Kundendatensätze in einer Anfrage durchsuchen zu können. Dafür wird ein föderierter Entity-Typ `Alle_Kunden` definiert, dessen Attribute auf die entsprechenden Attribute der lokalen Entity-Typen der verschiedenen Content Server abgebildet werden. Die drei Attribute des föderierten Entity-Typs werden jeweils einem gleichnamigen Suchkriterium zugeordnet. Im dargestellten Beispiel sollen mit Hilfe der Suchvorlage, alle Kunden ermittelt werden, die über 30 Jahre alt

sind. Das Ergebnis dieser Anfrage ist in Tabelle 5.1 dargestellt.

Content Server	**Name**	**Vorname**	**Alter**
DB2-Datenbank	Smith	Anton	34
Content Manager 8	Müller	David	43
Content Manager 7	Parker	John	51
Content Manager 7	Reno	Alice	43

Tabelle 5.1: Anfrageergebnis für das Beispiel aus Abbildung 5.2

5.3 Zusammenfassung

In diesem Kapitel wurde die Föderierung von Dokumentenarchiven mit dem *IBM DB2 Information Integrator for Content* erläutert.

Für die Föderierung von Dokumentenarchiven werden diese auf der Föderierungsschicht als Content Server definiert, so dass die Datenstruktur ausgelesen werden kann. Zur Integration der lokalen Datenstruktur werden föderierte Entity-Typen definiert, deren Attribute auf die Attribute eines lokalen Entity-Typs abgebildet werden. Die föderierten Entity-Typen werden anschließend in Suchvorlagen integriert, mit denen festgelegt wird, wonach gesucht wird, wie die Suchergebnisse präsentiert werden und wer die Suchvorlage verwenden darf.

Kapitel 6

Migration in den IBM DB2 Content Manager for Multiplatforms

Die Migration des Datenbestands eines Dokumentenarchivs in den *IBM DB2 Content Manager for Multiplatforms* erfolgt in zwei Schritten, der Migration der Datenstruktur und der eigentlichen Datenmigration. Abbildung 6.1 veranschaulicht diese Vorgehensweise. Im Folgenden werden die einzelnen Teilschritte der Migration detailliert beschrieben.

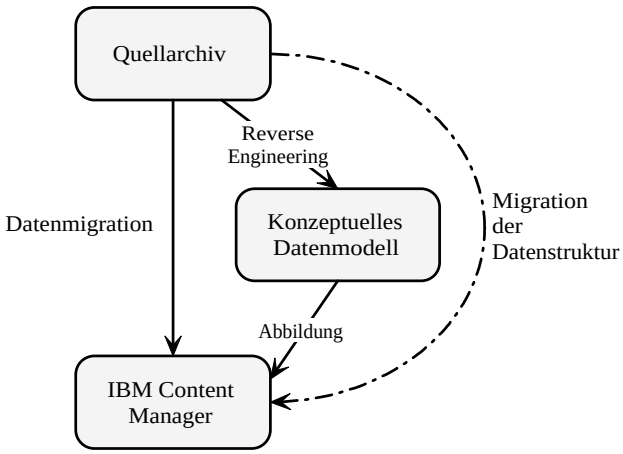


Abbildung 6.1: Migration eines Archivs in den *IBM Content Manager*

6.1 Migration der Datenstruktur

Die Migration der Datenstruktur eines Dokumentenarchivs erfolgt mit Hilfe eines konzeptuellen Datenmodells. Das konzeptuelle Datenmodell wird, falls es nicht in der Archivdokumentation enthalten ist, durch Reverse Engineering gewonnen und anschließend auf das ICM-Datenmodell abgebildet.

6.1.1 Konzeptuelle Datenmodelle durch Reverse Engineering

Wenn in der Dokumentation eines Dokumentenarchivs kein konzeptuelles Datenmodell enthalten ist, muss es durch Reverse Engineering aus den logischen Datenstrukturen gewonnen werden. Das konzeptuelle Datenmodell ist notwendig, um die Datenstrukturen des Archivs optimal auf das ICM-Datenmodell abzubilden.

Mit dem Begriff *Reverse Engineering (RE)* wird im Bereich Datenbanken der Vorgang der Rückgewinnung eines konzeptuellen Datenmodells aus den logischen Datenstrukturen eines Datenbanksystems bezeichnet. Das Ergebnis ist ein aussagekräftiges Modell der Datenbankstruktur, das auch von Außenstehenden interpretiert werden kann. Grundsätzlich versuchen RE-Verfahren, den Prozess des Forward Engineerings, mit dem ein konzeptuelles Datenmodell auf ein Implementierungsmodell abgebildet wird, umzukehren. Dazu werden Schema-Informationen, Beispieldatensätze sowie zusätzliche Informationen des Anwenders ausgewertet. Da bei der Abbildung von einem konzeptuellen Datenmodell auf ein Implementierungsmodell, wie z.B. dem relationalen Datenmodell, Informationen verloren gehen, ist die Rückgewinnung von konzeptuellen Datenmodellen nicht trivial.

Verfahren zum Reverse Engineering wurden in den letzten Jahren vor allem für den Bereich der relationalen Datenbanken entwickelt. Einen Überblick über einige bisher entwickelte Methoden gibt [dJS99]. Als eines der leistungsfähigsten Verfahren hat sich das Verfahren von Chiang/Barron/Storey (siehe [CBS94, CBS93]) erwiesen, da „...bei diesem Algorithmus mehr strukturelle Eigenschaften aus dem Relationenschema wiedergewonnen werden als bei anderen Verfahren“ [Rol96].

Der Algorithmus von Chiang/Barron/Storey arbeitet wie die meisten Verfahren mit einer Klassifikation von Tabellen. Die Tabellen des Ausgangsschemas werden an Hand der Schlüssel- und Fremdschlüsseldefinition als Entity-, Schwache-Entity- oder Beziehungsrelationen klassifiziert. Anschließend wird versucht, mit Hilfe der Klassifikation das ursprüngliche konzeptuelle Datenmodell wiederherzustellen. Das Verfahren arbeitet halbautomatisch und ist auf die Interaktion mit dem Anwender angewiesen. Beispielsweise kann oft nicht entschieden werden, ob eine Relation auf einen Entity-Typ oder eine Beziehung abgebildet werden soll. Die Kardinalitäten der Beziehungen müssen ebenfalls vom Anwender festgelegt werden.

Auch für andere Archivsysteme gibt es Lösungsansätze für die Rückgewinnung von konzeptuellen Datenmodellen. In [JMP01, JMP03] wird beispielsweise beschrieben, wie aus der DTD einer XML-Kollektion eine konzeptuelle Repräsentation in UML gewonnen wird. Dabei werden die Datenmodelle durch Mengen beschrieben und Funktionen für die Konvertierung von DTDs in UML-Klassendiagramme definiert.

Optimal ist es dennoch, wenn das konzeptuelle Datenmodell eines Archivs in der entsprechenden Projektdokumentation enthalten ist und nicht durch Reverse Engineering gewonnen werden muss, da der Reverse-Engineering-Prozess aufwendig ist und unter Umständen nicht zum gewünschten Ergebnis führt.

6.1.2 Abbildung konzeptueller Datenmodelle auf das ICM-Datenmodell

Liegt ein konzeptuelles Datenmodell vor, das die Datenstruktur eines Dokumentenarchivs beschreibt, wird ein ICM-Datenmodell zur Speicherung der Archivdaten durch die Abbil-

dung des konzeptuellen Datenmodells erstellt. Ziel der Abbildung ist es, möglichst wenig Informationen des konzeptuellen Datenmodells zu verlieren. Dies gilt vor allem für die Kardinalitäten der Beziehungskonstrukte.

Die Abbildung eines konzeptuellen Datenmodells auf das ICM-Datenmodell kann folgender Maßen zusammengefasst werden: Für jeden Entity-Typ des konzeptuellen Datenmodells wird ein Item Type im ICM-Datenmodell erstellt. Alle Attribute eines Entity-Typs, die Meta-Daten speichern, werden zu Attributen des entsprechenden Item Types. Attribute, in denen Content gespeichert wird, der von einem Resource Manager verwaltet werden soll, werden auf Resource Item Types oder Document Parts abgebildet. Die verschiedenen Beziehungstypen des konzeptuellen Datenmodells werden je nach Typ und Kardinalität durch ICM-Beziehungskonstrukte ersetzt.

Obwohl jedes Objekt im ICM-Datenmodell durch einen eindeutigen Identifikator (pid - *persistent identifier*) identifiziert wird, ist es sinnvoll, zumindest natürliche Schlüssel des konzeptuellen Datenmodells zu übernehmen. Damit wird es möglich, Duplikate auf einfache Weise beim Anlegen eines Items zu erkennen und Server-seitig zu verhindern. Künstliche Schlüssel, die aus automatisch generierten Attributwerten bestehen, müssen nicht unbedingt in das ICM-Datenmodell übernommen werden, da für den Aufbau von Beziehungen die internen Identifikatoren verwendet werden.

Falls die Schlüssel des konzeptuellen Datenmodells nicht in das ICM-Datenmodell übernommen werden oder nicht mit den Schlüsseln des Quellarchivs übereinstimmen bzw. bei der Abbildung verloren gehen, wie z.B. bei der Abbildung von schwachen Entity-Typen aus dem relationalen Modell, so ist es notwendig, die Schlüssel des Quellarchivs temporär im ICM-Datenmodell zu speichern. Dazu wird den betroffenen Item Types eine Child Component zugeordnet, welche die Schlüsselattribute aus dem Datenmodell des Quellarchivs erhält. Der ursprüngliche Schlüssel ist notwendig, um die Beziehungen zwischen den Items im ICM-Schema nach der Übernahme der Daten aus dem Quellarchiv wiederherzustellen. Nachdem die Beziehungen zwischen den Items des ICM-Schemas aufgebaut wurden, können die Child Components, die zur temporären Speicherung der Originalschlüssel verwendet wurden, wieder gelöscht werden.

In Kapitel 7 wird die Abbildung der einzelnen Konstrukte eines konzeptuellen Datenmodells detailliert beschrieben.

6.1.3 Algorithmus zur Abbildung konzeptueller Datenmodelle auf das ICM-Datenmodell

Abbildung 6.2 zeigt einen Algorithmus, in dem die in Kapitel 7 beschriebenen Abbildungsvorschriften für die Konstrukte eines konzeptuellen Datenmodells integriert werden. Der Algorithmus konvertiert ein konzeptuelles Datenmodell in ein ICM-Datenmodell.

Ein besonderes Problem bei der Konvertierung ist die Umsetzung von Beziehungen und Generalisierungshierarchien mit Hilfe von Child Components. Die Entscheidung, ob eine Beziehung durch Child Components umgesetzt werden kann oder ob eine alternative Umsetzung verwendet werden muss, hängt nicht allein von den Beziehungen eines Entity-Typs ab. Abbildung 6.3 veranschaulicht eine solche Situation. Die Komposition zwischen den Entity-Typen `E1` und `E2` könnte optimal umgesetzt werden, indem `E2` Child Component von `E1` wird. Ob `E2` als Child Component deklariert werden darf, hängt in diesem Fall nicht

1. Abbildung: Betrachte nacheinander alle Entity-Typen und bilde diese wie folgt ab:
   (a) Erzeuge Item Type
      i. Für jeden Entity-Typ, der nicht an einer 1:1-Beziehung mit beidseitig totaler Partizipation und nicht an einer Generalisierung beteiligt ist, erzeuge einen Item Type vom Typ `Item Type` mit dem Namen des Entity-Typs.
      ii. Entity-Typen, die an einer 1:1-Beziehung mit beidseitig totaler Partizipation beteiligt sind, werden zu einem Item Type vom Typ `Item Type` verschmolzen.
      iii. Entity-Typen, die an einer Generalisierung beteiligt sind, werden je nach gewünschter Umsetzung der Generalisierung zu einem Item Type mit Item Type Subsets verschmolzen, oder ebenfalls auf je einen Item Type vom Typ `Item Type` abgebildet.
   (b) Erstelle Attribute und Schlüssel
      i. Alle Meta-Attribute des Entity-Typs werden zu Attributen des Item Types. Falls ein Attribut im konzeptuellen Datenmodell einfacher Schlüssel ist, werden die Eigenschaften `Unique` und `Required` gesetzt.
      ii. Für jeden zusammengesetzten Schlüssel wird ein Index erzeugt. Die am Index beteiligten Attribute erhalten die Eigenschaft `Required`.
      iii. Für jedes mehrwertige Attribut wird eine Child Component mit entsprechender Kardinalität erstellt, die den Namen des Attributs erhält.
      iv. Falls der Entity-Typ genau ein Content-Attribut besitzt, das auf einen Resource Item Type abgebildet werden soll, setze den Type des Item Types auf `Resource Item Type`.
      v. Falls der Entity-Typ mehrere Content-Attribute besitzt, die auf Resource Item Types abgebildet werden sollen, lege für jedes Content-Attribut einen Resource Item Type an, der durch ein Referenzattribut referenziert wird. Die Resource Item Types und die Referenzattribute erhalten die Namen der Content-Attribute.
      vi. Falls der Entity-Typ ein oder mehrere Content-Attribute besitzt, die auf Document Parts abgebildet werden sollen, setze den Typ des Item Types auf `Document Item Type` und erzeuge entsprechende Document Parts.
   (c) Setze alle Beziehungen und Generalisierungen, an denen der Entity-Typ beteiligt ist, wie in Kapitel 7 beschrieben, um, ohne dabei Child Relations zu verwenden.
2. Optimierung: Kompositionen, Generalisierungen, 1:n-Beziehungen mit totaler Partizipation der 1-Seite und 1:1-Beziehungen mit einseitig partieller Partizipation können durch Child Relations umgesetzt werden, falls der Item Type, der hierfür in eine Child Component umgewandelt werden muss, vom Typ `Item Type` ist, keine eingehenden Referenzen besitzt und nicht an einer Link-Beziehung beteiligt ist.
   (a) Prüfe nacheinander, unter Beachtung der Reihenfolge, alle Kompositionen, Generalisierungen, 1:n-Beziehungen mit totaler Partizipation der 1-Seite und 1:1-Beziehungen mit beidseitig partieller Partizipation auf Optimierung.
   (b) Falls eine Optimierung möglich ist, wandle die Beziehung in eine Child Relation um.
   (c) Wiederhole Schritt `2a` bis keine Optimierungsmöglichkeit mehr gefunden wird.

Abbildung 6.2: Algorithmus zur Abbildung eines konzeptuellen Datenmodells auf das ICM-Datenmodell

nur von der Beziehung zwischen E2 und E3, sondern auch von der Beziehung zwischen E3 und E4 ab. Die Komposition zwischen E2 und E3 sollte ebenfalls durch eine Child Relation umgesetzt werden, in dem E3 Child Component von E2 wird. Dies ist wiederum abhängig von der Beziehung zwischen E3 und E4. Da n:m-Beziehungen durch Links umgesetzt werden, Child Components aber nicht an Link-Beziehungen teilnehmen können, kann E3 nicht als Child Component deklariert werden. Da die Beziehung zwischen E2 und E3 deshalb durch Referenzen realisiert werden muss, kann auch E2 nicht als Child Component verwendet werden, da Child Components nicht referenziert werden können. Abhängigkeiten mit noch tieferer Schachtelung lassen sich ebenfalls konstruieren.

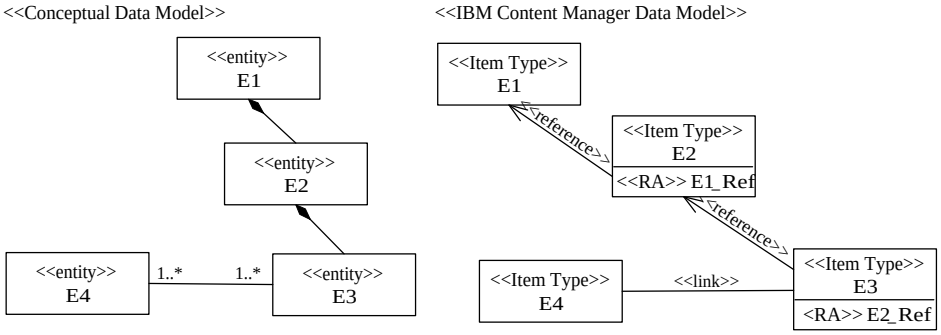


Abbildung 6.3: Beispiel für Kompositionen die nicht durch Child Components umgesetzt werden können

Der Algorithmus in Abbildung 6.2 arbeitet daher in zwei Schritten. Im ersten Schritt wird für jeden Entity-Typ ein Item Type mit den entsprechenden Attributen erstellt. Ausnahmen bilden Entity-Typen, die an einer 1:1-Beziehung mit beidseitig totaler Partizipation beteiligt sind. Diese Entity-Typen werden zu einem Item Type verschmolzen. Dasselbe gilt für Entity-Typen, die an einer Generalisierung beteiligt sind und mit Hilfe von Item Type Subsets umgesetzt werden soll.

Die Beziehungen eines Entity-Typs werden zunächst ohne Child Relations umgesetzt. Für Kompositionen, Generalisierungen, 1:1-Beziehungen mit einseitig partieller Partizipation und 1:n-Beziehungen mit totaler Partizipation der 1-Seite, werden dazu die alternativen Umsetzungen verwendet. Diese Umsetzungen können immer verwendet werden, da dabei keine Child Components entstehen.

Im zweiten Schritt des Algorithmus wird das bisher entstandene ICM-Datenmodell optimiert, indem versucht wird, die oben genannten Beziehungstypen durch ihre optimale Umsetzung mit Hilfe von Child Components zu realisieren. Dabei werden Kompositionen bevorzugt behandelt, da dieser Beziehungstyp, wenn möglich, immer durch Child Components umgesetzt werden sollte.

6.2 Datenmigration

Nachdem die Datenstruktur eines Dokumentenarchivs mit Hilfe einer konzeptuellen Repräsentation in ein ICM-Datenmodell überführt wurde, kann der Inhalt des Archivs übernommen werden. Im Folgenden wird von einer relationalen Datenbank als Quellarchiv aus-

gegangen. Die grundsätzliche Vorgehensweise lässt sich jedoch auf andere Archivformen übertragen. Die Datenmigration besteht aus zwei Schritten: der Übertragung der Daten und der Rekonstruktion der Beziehungen.

6.2.1 Übertragung der Daten

Zu jedem Item Type, der aus einem Entity-Typ hervorging, existiert im relationalen Schema eine entsprechende Tabelle. Die Zuordnung eines Item Types zu einer relationalen Tabelle kann durch eine einheitliche Namensgebung vereinfacht werden. Wenn sowohl im relationalen Datenmodell als auch im konzeptuellen Datenmodell und im ICM-Datenmodell dieselben Namen für gleiche Objekte verwendet werden, ist eine Automatisierung der Zuordnung möglich. Andernfalls ist eine manuelle Zuordnung erforderlich.

Aus jeder Zeile einer Tabelle wird ein Item des zugehörigen Item Types erzeugt. Dabei werden die Attribute der Items mit den Werten der Attribute aus der Tabelle des relationalen Schemas gefüllt. Falls der Schlüssel einer relationalen Tabelle nicht auf den zugehörigen Item Type abgebildet wurde, wird der Schlüsselwert jeder Tabellenzeile als Child Component gespeichert und dem erzeugten Item zugeordnet. Attribute die Content beinhalten, der in einem Resource Manager gespeichert werden soll, werden auf die dazu vorgesehenen Resource Item Types oder Document Parts abgebildet.

Objekte von Entity-Typen, die als Child Components umgesetzt wurden können nicht unabhängig persistent gemacht werden, da sie an ein übergeordnetes Item gebunden sind. Child Components werden daher beim Erzeugen des übergeordneten Items angelegt und mit diesem assoziiert. Falls eine Child Component weitere Child Components besitzt, wird die gesamte Hierarchie im Speicher erzeugt und anschließend mit der `add()`-Funktion der Root Component gespeichert.

Item Types, die beschreibende Objekte von Link-Beziehungen speichern, werden erst beim Aufbau der Beziehungen mit Daten gefüllt. Dasselbe gilt für Beziehungsattribute von Referenzbeziehungen, die innerhalb des Quell-Items gespeichert werden.

6.2.2 Rekonstruktion der Beziehungen

Nachdem die Daten aus dem Quellarchiv übernommen wurden, können die Beziehungen zwischen den Items aufgebaut werden. Dies erfolgt mit Hilfe der relationalen Strukturen.

Nacheinander werden die Items aller Item Types mit ausgehenden Beziehungen betrachtet. Diese Items werden als *Quell-Items* für eine Beziehung bezeichnet. Die Items die mit einem Quell-Item in Beziehung stehen, werden *Ziel-Items* genannt. Die Informationen über die Zuordnung zwischen Quell- und Ziel-Items sind innerhalb des relationalen Schemas in Schlüssel-Fremdschlüssel-Beziehungen gespeichert. Die Tabelle, welche die Schlüssel-Fremdschlüssel-Informationen enthält aus denen hervorgeht welches Quell-Item mit welchen Ziel-Items in Beziehung steht, wird als *Beziehungstabelle* für einer Beziehung bezeichnet.

Mit Hilfe der relationalen Schlüssel, die entweder auf die Item Type abgebildet oder in einer Child Component gespeichert wurden, werden die zu einem Quell-Item gehörenden Zeilen in der Beziehungstabelle ermittelt. Aus diesen Tabellenzeilen können die relationalen Schlüssel der Ziel-Items ermittelt werden, die mit dem Quell-Item in Beziehung gesetzt

werden müssen. Mit Hilfe dieser Schlüssel werden die an der Beziehung beteiligten Ziel-Items identifiziert. Beim Aufbau der Beziehung zwischen Quell- und Ziel-Item, wird der im ICM-Datenmodell vorgeschriebene Beziehungstyp verwendet. Dabei werden eventuell vorhandene Beziehungsattribute gesetzt. Bei Link-Beziehungen, die ein beschreibendes Objekt besitzen, werden dazu Items des entsprechenden Item Types erzeugt.

Nachdem alle Beziehungen aus der relationalen Datenbank übernommen wurden, können die Child Components, die zur temporären Speicherung der relationalen Schlüssel angelegt wurden, gelöscht werden.

6.2.3 Beispiel

Die Abbildungen 6.4 und 6.5 veranschaulichen an Hand eines Beispiels die Migration einer 1:n-Beziehung aus einer relationalen Datenbank in den *IBM DB2 Content Manager for Multiplatforms*.

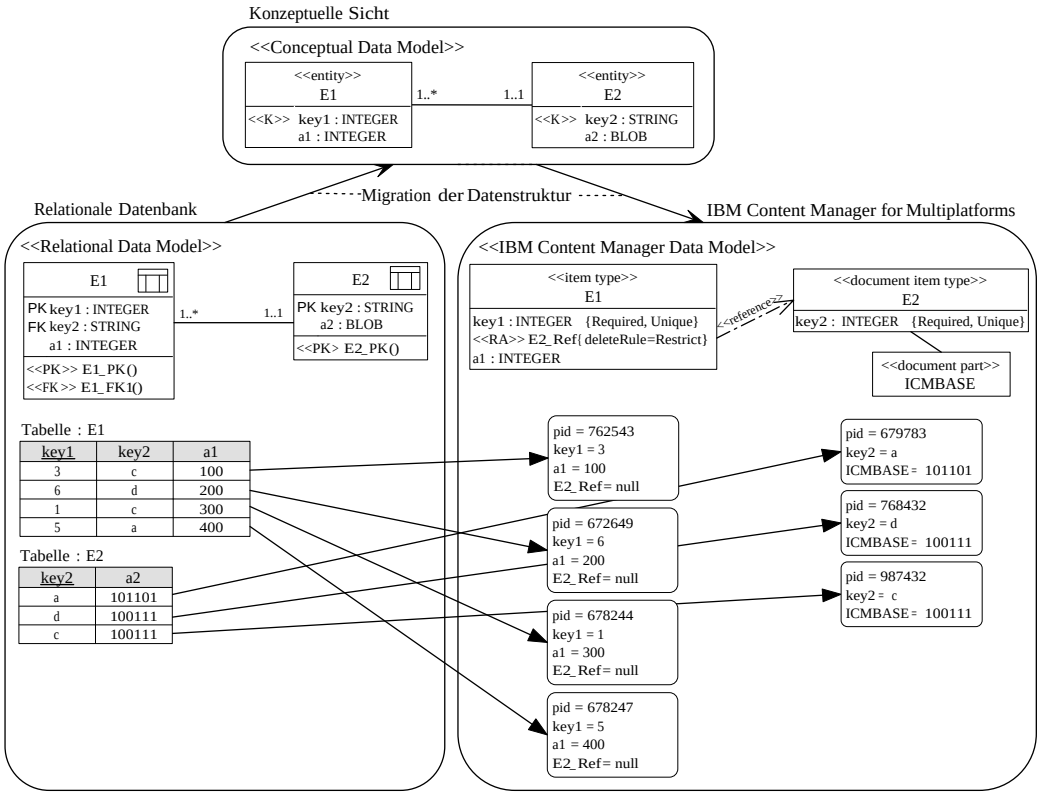


Abbildung 6.4: Beispiel für die Datenmigration einer 1:n-Beziehung

In Abbildung 6.4 ist zunächst die Migration der Daten dargestellt. Die Entity-Typen `E1` und `E2` werden im relationalen Modell durch zwei Tabellen repräsentiert. Die 1:n-Beziehung wird durch eine Schlüssel-Fremdschlüssel-Beziehung umgesetzt. Der Primärschlüssel `key2` der Tabelle `E2` wird durch einen Fremdschlüssel in der Tabelle `E1` referenziert.

Im ICM-Datenmodell werden zwei Item Types definiert, welche die Attribute der Entity-Typen erhalten. Im Entity-Typ `E2` speichert das Attribut `a2` Binärdaten, die in einem Resource Manager verwaltet werden sollen. Aus diesem Grund wird `E2` als Document Item Type definiert, dem ein `ICMBASE`-Part zugeordnet ist. In diesem Part werden die Attributwerte von `a2` gespeichert. Die Schlüsselattribute werden in das ICM-Schema übernommen und müssen daher nicht extra gespeichert werden. Die 1:n-Beziehung wurde im ICM-Datenmodell durch Referenzen umgesetzt. Dazu wurde das Referenzattribut `E2_Ref` definiert, das Instanzen von `E2` referenziert.

Bei der Migration der Daten wird für jede Tabellenzeile der Tabelle `E1` ein Item des Item Types `E1` mit den entsprechenden Attributwerten erzeugt. Das Referenzattribut, das die Beziehung speichert, wird zunächst auf `null` gesetzt. Für die Tabellenzeilen der Tabelle `E2` werden Items des Item Types `E2` erzeugt. Die Werte des Attributs `a2` werden im `ICMBASE`-Part gespeichert.

Jedes Item, das im *IBM DB2 Content Manager for Multiplatforms* erzeugt wird, erhält automatisch einen vom System erzeugten eindeutigen Identifikator. In Abbildung 6.4 wird dieser als Attribut `pid` dargestellt. Der Identifikator wird für den Aufbau der Beziehungen zwischen den Items verwendet.

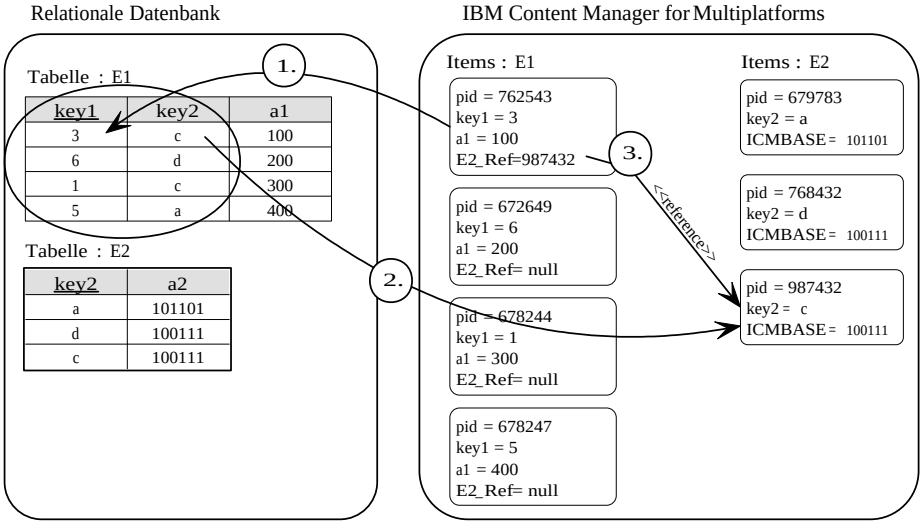


Abbildung 6.5: Beispiel für die Rekonstruktion der Beziehungen einer 1:n-Beziehung

Abbildung 6.5 zeigt den Aufbau der Beziehungen zwischen den erzeugten Items. Die Tabelle `E1` ist die Beziehungstabelle der zu rekonstruierenden 1:n-Beziehung. Aus den beiden Spalten `key1` und `key2` kann abgelesen werden, welches Item vom Typ `E1` welches Item vom Typ `E2` referenziert.

Zum Aufbau der Beziehungen werden die Items vom Typ `E1` nacheinander betrachtet. Mit Hilfe des Schlüsselattributs `key1` kann jedem Item vom Typ `E1` die entsprechende Zeile der Beziehungstabelle `E1` zugeordnet werden (Schritt 1). Das Attribut `key2` der Beziehungstabelle enthält den Schlüssel des Ziel-Items von Typ `E2`, das in Beziehung gesetzt werden muss. Mit Hilfe dieses Schlüssels wird das entsprechende Item identifiziert (Schritt

2) und über das Attribut `E2_Ref` referenziert (Schritt 3). Nachdem alle Items vom Typ `E1` betrachtet wurden, sind die Beziehungen der 1:n-Beziehung des relationalen Datenmodells auch im ICM-Datenmodell vorhanden.

6.3 Migration von Methoden

In objektrelationalen und objektorientierten Datenbanksystemen ist es möglich, nicht nur die Struktur, sondern auch das Verhalten von Objekten mit Hilfe von Methoden zu definieren. In objektrelationalen Datenbanksystemen erfolgt dies über Stored Procedures (SP) oder User Defined Functions (UDF), die global an eine Datenbank gebunden werden. In objektorientierten Datenbanksystemen kann das Verhalten von Objekten direkt definiert werden. Bei der Migration von Dokumentenarchiven aus objektrelationalen und objektorientierten Datenbanksystemen muss ein Weg gefunden werden auch Methoden in den *IBM DB2 Content Manager for Multiplatforms* zu migrieren.

Da das ICM-Datenmodell keine Konzepte bietet, um das Verhalten von Items zu definieren, ist die Integration von Methoden schwierig. Im Folgenden werden drei Ansätze zur Integration von Methoden in den *IBM DB2 Content Manager for Multiplatforms* vorgestellt.

Media Object Classes

Obwohl Methoden im ICM-Datenmodell nicht an Items gebunden werden können, ist es mit Hilfe von nutzerdefinierten Media Object Classes (siehe Kapitel 3.2.5) wenigstens möglich spezielle Methoden für die Behandlung von Objekten des Resource Managers zu definieren. Diese Möglichkeit kann in einigen Anwendungsfällen genutzt werden, um Methoden in den *IBM DB2 Content Manager for Multiplatforms* zu integrieren. Dabei handelt es sich aber nicht um eine „echte“ Server-seitige Integration der Methoden, da die Media Objekt Classes zwar im Library Server zugeordnet, die Methoden aber Client-seitig ausgeführt werden. Eine Integration von Methoden mit Hilfe von Media Object Classes ist vor allem dann sinnvoll, wenn Methoden integriert werden die ausschließlich auf Objekten des Resource Managers arbeiten.

Integration in die Client-Anwendung

Neben der Integration über Media Object Classes, besteht die Möglichkeit die zu migrierenden Methoden direkt in die Client-Anwendungen zu integrieren. Dabei wird die Logik der Methoden in die Klassen der Client-Anwendung übertragen. Wie auch bei der Integration über Media Object Classes bringt dies aber die Nachteile einer höheren Netzwerklast und eines größeren Aufwands zur Pflege der Methoden mit sich, da diese verteilt gespeichert und aktualisiert werden müssen.

Stored Procedures / UDFs

Zur Integration von Methoden die vor allem auf Meta-Daten operieren, können Methoden als Stored Procedures oder UDFs in die DB2-Datenbank integriert werden, auf die der

IBM DB2 Content Manager for Multiplatforms aufsetzt. Auf diese Weise werden Methoden Server-seitig gespeichert und der Datenverkehrs zwischen Client und Server verringert.

Eine Integration der Methoden in die Anfragesprache des *IBM DB2 Content Manager for Multiplatforms* ist nicht möglich. Der Aufruf der Methoden erfolgt daher am *IBM DB2 Content Manager for Multiplatforms* vorbei, über die SQL-Schnittstelle der Datenbank.

6.4 Zusammenfassung

In diesem Kapitel wurde eine Verfahrensweise zur Migration von Datenarchiven in den *IBM DB2 Content Manager for Multiplatforms* beschrieben. Das Verfahren besteht aus zwei Schritten, der Migration der Datenstruktur und der eigentlichen Datenmigration.

Für die Migration der Datenstruktur eines Archivs wird mit Hilfe von Reverse Engineering zunächst ein konzeptuelles Datenmodell erstellt, das anschließend auf das ICM-Datenmodell abgebildet wird. Für diese Abbildung wurde in Abschnitt 6.1.3 ein Algorithmus vorgestellt, der die einzelnen Konstrukte eines konzeptuellen Datenmodells durch Elemente das ICM-Datenmodells ersetzt.

Weiterhin wurde ein Verfahren erläutert, mit dem der Inhalt eines Archivs in einen *IBM DB2 Content Manager for Multiplatforms* migriert werden kann. Dabei werden im ersten Schritt die Datenobjekte des Quellarchivs kopiert und anschließend die Beziehungen zwischen den Objekten wiederhergestellt.

Abschließend wurden Lösungsansätze beschrieben, mit denen Methoden in den *IBM DB2 Content Manager for Multiplatforms* integriert werden können. Zur Server-seitigen Speicherung von Methoden kann die Funktionalität der DB2-Datenbank genutzt werden, auf die der *IBM DB2 Content Manager for Multiplatforms* basiert. Dabei werden Methoden als Stored Procedures oder User Defined Functions in die Library-Server-Datenbank integriert.

Kapitel 7

Abbildung der Konstrukte konzeptueller Datenmodelle auf das ICM-Datenmodell

Im folgenden Kapitel werden Abbildungsvorschriften für die einzelnen Konstrukte eines konzeptuellen Datenmodells vorgestellt. Die Abbildungsvorschriften werden innerhalb des Algorithmus zur Abbildung konzeptueller Datenmodelle auf das ICM-Datenmodell verwendet (siehe Kapitel 6.1.3).

7.1 Attribute

Attribute des konzeptuellen Datenmodells werden je nach Typ und Eigenschaften unterschiedlich auf das ICM-Datenmodell abgebildet. Dabei wird zwischen einfachen Attributen, mehrwertigen Attributen und Schlüsselattributen unterschieden.

7.1.1 Einfache Attribute

Bei der Abbildung von einfachen Attributen eines Entity-Typs, wird zwischen Meta-Attributen und Content-Attributen unterschieden.

Meta-Attribute sind Attribute, die keinen Content repräsentieren der in einem Resource Manager gespeichert werden soll, sondern lediglich Meta-Daten enthalten. Meta-Attribute werden im ICM-Datenmodell auf Attribute von Item Types abgebildet. Dabei wird die allgemeine Typdefinition des konzeptuellen Datenmodells durch einen konkreten Typ des ICM-Datenmodells ersetzt. Soll ein Attribut in Volltextsuchanfragen verwendet werden, wird die Elementeigenschaft `Text searchable` gesetzt.

Content-Attribute repräsentieren Dokumente, die in einem Resource Manager verwaltet werden sollen und werden auf Resource Item Types oder Document Item Types mit einem entsprechenden Document Part abgebildet.

7.1.2 Mehrwertige Attribute

Attribute mit Multiplizität werden mit Hilfe von Child Components umgesetzt. In der Child Component wird dazu ein Attribut vom Typ des mehrwertigen Attributs definiert. Als Name für die Child Component wird der Attributname verwendet. In jeder Instanz der Child Component wird jeweils ein Wert des mehrwertigen Attributs gespeichert. Die Multiplizität des konzeptuellen Datenmodells wird als Kardinalität für die Child Component übernommen. Abbildung 7.1 zeigt ein Beispiel für die Umsetzung von mehrwertigen Attributen.

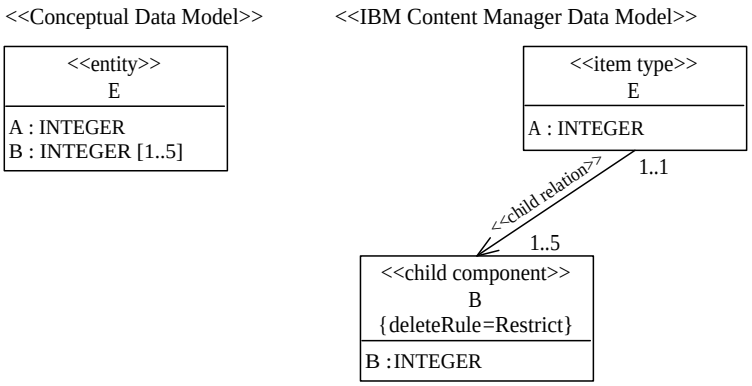


Abbildung 7.1: Abbildung von mehrwertigen Attributen auf das ICM-Datenmodell

7.1.3 Schlüsselattribute

Einfache Schlüsselattribute werden im ICM-Datenmodell als Attribute mit den Elementeigenschaften `Unique` und `Required` deklariert. Abbildung 7.2 veranschaulicht diese Vorgehensweise.

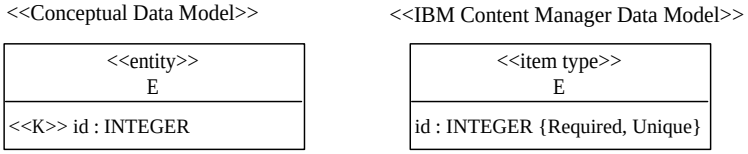


Abbildung 7.2: Abbildung von einfachen konzeptuellen Schlüsseln auf das ICM-Datenmodell

Zusammengesetzte Schlüssel werden im ICM-Datenmodell durch Indices umgesetzt. Alle am Schlüssel beteiligten Attribute werden in den Index aufgenommen und mit der Eigenschaft `Required` definiert. Dabei ist zu beachten, dass Referenzattribute nicht in einem Index verwendet werden können. Abbildung 7.3 zeigt ein Beispiel für die Umsetzung eines zusammengesetzten Schlüssels.

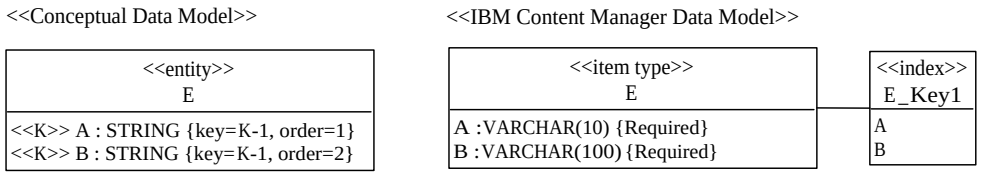


Abbildung 7.3: Abbildung von zusammengesetzten konzeptuellen Schlüsseln auf das ICM-Datenmodell

7.2 Entity-Typen

Bei der Abbildung von Entity-Typen ist die Anzahl der Content-Attribute entscheidend für die Umsetzung. Je nach dem, ob ein Entity-Typ keine, genau ein oder mehrere Content-Attribute besitzt, ist eine andere Abbildungsvorschrift notwendig. Darüberhinaus wird zwischen der Abbildung auf Resource Item Types und der Abbildung auf Document Item Types unterschieden. Werden in einem Anwendungsszenario die IBM-Standard-Clients verwendet, müssen alle Entity-Typen, die für die Standard-Client-Anwendungen sichtbar sein sollen, auf Document Item Types abgebildet werden. Bei der Verwendung von selbsterstellten Client-Anwendungen können für die Abbildung neben Document Item Types, auch einfache Item Types und Resource Item Types verwendet werden.

7.2.1 Entity-Typen ohne Content-Attribute

Entity-Typen, die ausschließlich Meta-Daten speichern und keine Attribute haben, deren Inhalt von einem Resource Manager verwaltet werden soll, werden auf einfache Item Types oder Document Item Types abgebildet. Bei Document Item Types, die von den Standard-Client-Anwendungen verwendet werden, muss der `ICMBASE`-Part angelegt werden, auch wenn in diesem Fall kein Content gespeichert wird. Andernfalls werden die Document Item Types von den Standard-Clients ignoriert. Als Name für den Item Type wird der Name des Entity-Typs verwendet. Die Attribute des Entity-Typs werden wie in Abschnitt 7.1 beschrieben definiert.

7.2.2 Entity-Typen mit einem Content-Attribut

Entity-Typen, die mindestens ein Content-Attribut beinhalten, werden entweder auf einen Resource Item Type oder einen Document Item Type mit einem entsprechenden Document Part abgebildet. Existieren zusätzlich Attribute, die Meta-Daten speichern, werden diese umgesetzt, wie im vorherigen Abschnitt beschrieben. Ist der zu speichernde Content text-basiert, kann dieser Text zur Volltextsuche indiziert werden, indem die Elementeigenschaft `Text searchable` gesetzt wird.

Die Grafiken 7.4 und 7.5 zeigen die Abbildung eines Entity-Typs mit einem Meta-Attribut und einem Attribut vom Typ `CLOB`, in dem Textdokumente gespeichert werden. Im Folgenden wird die Abbildung dieses Entity-Typs auf einen Resource Item Type und einen Document Item Type erläutert.

Abbildung auf einen Resource Item Type Resource Items repräsentieren ein im Resource Manager gespeichertes Objekt. Je nach Art des Content der gespeichert werden soll, wird eine entsprechende Media Object Class beim Erstellen des Resource Item Types festgelegt. Im Beispiel aus Abbildung 7.4 wird die Klasse `DKTextICM` verwendet, um den textbasierten Content des Attributs `B` zu speichern. Da der Content in Volltextsuchen verwendet werden soll, wird für den erstellten Resource Item Type die Eigenschaft `Text searchable` festgelegt.

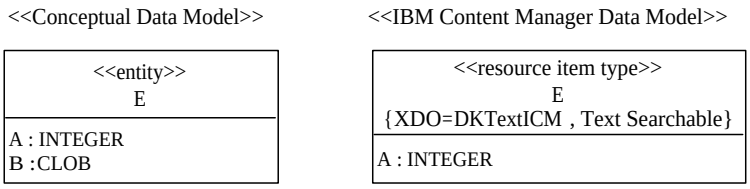


Abbildung 7.4: Abbildung eines Entity-Typs mit einem Content-Attribut auf einen Resource Item Type

Abbildung auf einen Document Item Type Bei der Abbildung auf einen Document Item Type wird, je nach Art des zu speichernden Contents, einer der vordefinierten Document Parts verwendet oder ein neuer Document Part mit einer entsprechenden Media Object Class definiert. In Abbildung 7.5 wird ein `ICMBASETEXT`-Part verwendet, um den Text des Content-Attributs `B` zu speichern.

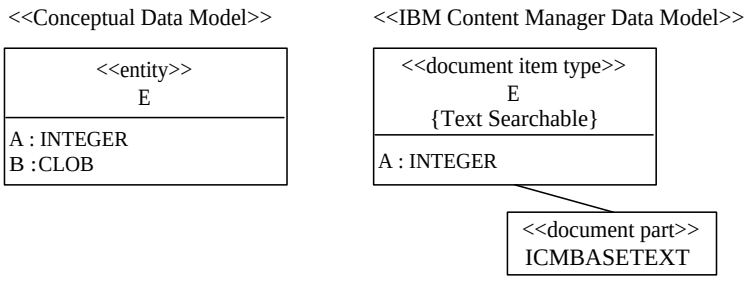


Abbildung 7.5: Abbildung eines Entity-Typs mit einem Content-Attribut auf einen Document Item Type

7.2.3 Entity-Typen mit mehr als einem Content-Attribut

Entity-Typen, die aus mehreren Content-Attributen bestehen, werden auf einen Item Type mit zugeordneten Resource Item Types oder auf einen Document Item Type mit entsprechenden Document Parts abgebildet. Die Abbildungen 7.6 und 7.7 zeigen die Umsetzung eines Entity-Typs, der aus einem Meta-Attribut vom Typ `Integer` und drei Content-Attributen in denen Bilder in verschiedener Qualität gespeichert werden, besteht. Die möglichen Abbildungen dieses Entity-Typs werden im Folgenden erläutert.

Abbildung auf mehrere Resource Item Types Bei der Abbildung auf mehrere Resource Item Types wird ein Basis-Item-Type definiert, der den Namen des Entity-Typs sowie alle Meta- und Schlüsselattribute erhält. Für jedes Content-Attribut wird ein Resource Item Type mit dem Namen des Content-Attributs erstellt, der über eine Referenz mit dem Basis-Item-Type assoziiert wird. Das Referenzattribut erhält ebenfalls den Namen des Content-Attributs. Abbildung 7.6 zeigt ein entsprechendes Beispiel.

Vorteil dieser Variante ist, dass auf Applikationsebene sehr einfach auf ein bestimmtes Objekt zugegriffen werden kann, da jedes Objekt über ein Referenzattribut direkt erreichbar ist.

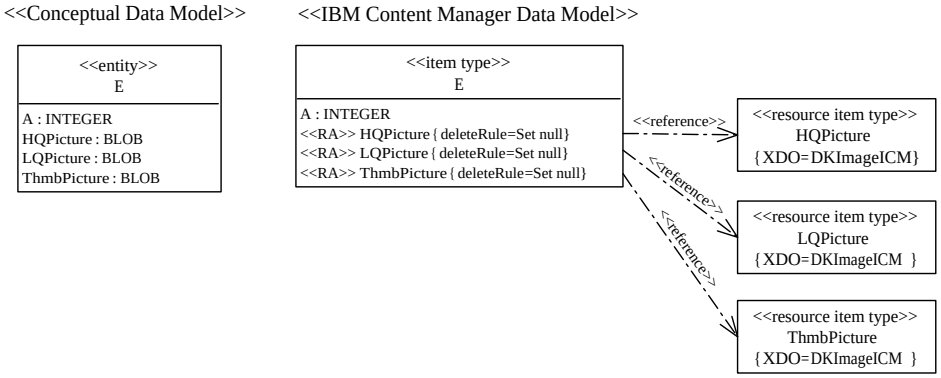


Abbildung 7.6: Abbildung eines Entity-Typs mit mehreren Content-Attributen auf einen Item Type und mehrere Resource Item Types

Abbildung auf einen Document Item Type Bei der Verwendung von Document Item Types, wird der Inhalt der Content-Attribute in einem oder mehreren Document Parts gespeichert. Im Beispiel aus Abbildung 7.7 werden alle drei Bilder in einem `ICMBASE`-Part verwaltet. Meta- und Schlüsselattribute werden wie in den vorherigen Fällen umgesetzt.

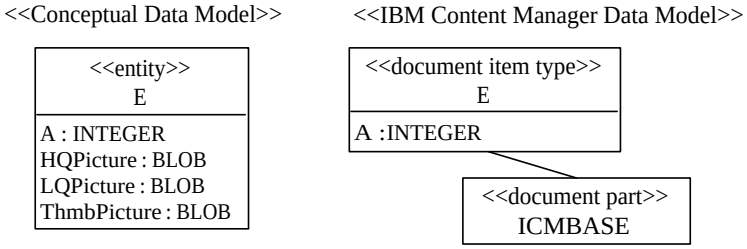


Abbildung 7.7: Abbildung eines Entity-Typs mit mehreren Content-Attributen auf einen Document Item Type

Vorteil dieser Abbildung ist, dass die Bilder an das zugehörige Item gebunden sind. Wird das Item gelöscht, so werden auch die Bilder gelöscht. Bei der Abbildung auf einen Item Type mit referenzierten Resource Item Types ist dies nicht der Fall. Darüberhinaus

können in einem Document Item Type jederzeit zusätzliche Objekte gespeichert werden, da die Anzahl der Objekte in einem Document Part nicht begrenzt ist.

Nachteil dieser Variante ist, dass alle Objekte eines Parts, auf Applikationsebene nur über eine Kollektion erreicht werden können. Wird ein ganz bestimmtes Objekt des Document Items benötigt, z.B. das Bild mit der kleinsten Auflösung, so muss umständlich durch die Kollektion iteriert werden.

7.3 Beziehungen

Die Abbildung von Beziehungen eines konzeptuellen Datenmodells, erfolgt je nach Beziehungstyps und Ausprägung der Kardinalitäten. Unterschieden wird zwischen den verschiedenen Formen einer Assoziation, sowie zwischen Kompositionen und Aggregationen.

7.3.1 1:1-Beziehungen

Als *1:1-Beziehungen* werden Assoziationen bezeichnet, bei denen die maximale Multiplizität auf beiden Seiten eins ist. Abbildung 7.8 zeigt eine allgemeine 1:1-Beziehung. Je nach Ausprägung der minimalen Multiplizitäten `p` und `r` werden drei Arten von 1:1-Beziehungen unterschieden, deren Abbildung in den folgenden Abschnitten erläutert wird.

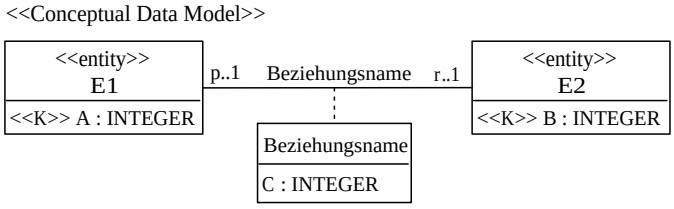


Abbildung 7.8: Allgemeine 1:1-Beziehung

Beidseitig totale Partizipation

Wenn beide Entity-Typen zwingend an einer 1:1-Beziehung beteiligt sind, das heißt die minimalen Multiplizitäten `p` und `r` sind eins, dann können die Entity-Typen zu einem Item Type verschmolzen werden. Der Item Type erhält die Attribute beider Entity-Typen und die Attribute der Beziehung. Abbildung 7.9 zeigt ein entsprechendes Beispiel.

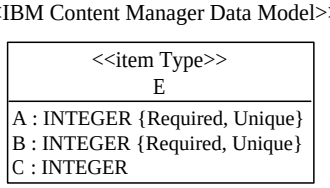


Abbildung 7.9: 1:1-Beziehung mit beidseitig totaler Partizipation im ICM-Datenmodell

Einseitig partielle Partizipation

Ist einer der in Beziehung stehenden Entity-Typen nicht zwingend an der 1:1-Beziehung beteiligt, das heißt, die minimale Multiplizität auf einer der Seiten ist null, dann spricht man von einer *einseitig partiellen Partizipation*.

Eine solche Assoziation kann optimal durch einen hierarchischen Item Type umgesetzt werden. Falls `p=1` und `r=0`, so wird der partiell an der Beziehung beteiligte Entity-Typ `E1` auf einen Item Type abgebildet, dem eine Child Component zugeordnet wird, die den total an der Beziehung beteiligten Entity-Typen `E2` repräsentiert. Die Kardinalität der Child Component wird auf `0..1` gesetzt. Damit wird die Multiplizität des Entity-Typs `E1`korrekt abgebildet. Da jede Child Component genau eine zugeordnete Root Component hat, bleibt auch die Multiplizität des Entity-Typs `E2` bewahrt. Sind der Beziehung Attribute zugeordnet, so werden diese innerhalb der Child Component definiert. In Abbildung 7.10 ist ein Beispiel für die Umsetzung einer 1:1-Beziehung mit einseitig partielle Partizipation dargestellt.

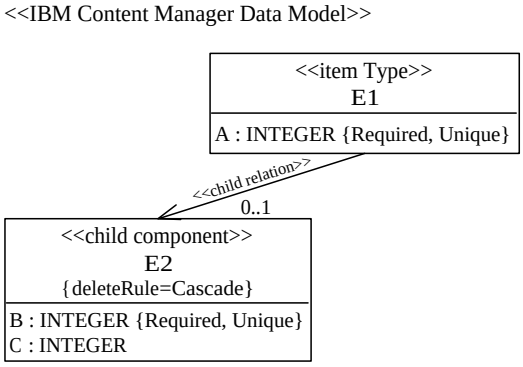


Abbildung 7.10: Umsetzungen einer 1:1-Beziehung mit einseitig partieller Partizipation durch Child Components

Nachteil dieser Umsetzung ist, dass der als Child Component modellierte Entity-Typ `E2` nicht mehr über Referenzen oder Links angesprochen werden kann (siehe Tabelle 3.3). Weiterhin kann eine Child Component jeweils nur genau einer Root Component zugeordnet werden. Ist `E2` an einer weiteren Beziehung beteiligt, dann ist `E2` eventuell bereits Child Component einer anderen Root Component, oder muss referenziert bzw. verlinkt werden können. In diesem Fall kann die beschriebene Umsetzung der Beziehung nicht verwendet werden.

Alternativ wird `E2` nicht als Child Component, sondern als Item Type mit einem Referenzattribut modelliert, das Instanzen des Item Types `E1` referenziert. Eventuell vorhandene Beziehungsattribute werden dem Item Type von `E2` zugeordnet. Die Multiplizitäten des konzeptuellen Datenmodells können auf diese Weise allerdings nicht mehr vollständig umgesetzt werden, da nicht festgelegt werden kann, wie viele Instanzen von `E1` referenziert werden und auch keine Referenz von `E2` nach `E1` erzwungen werden kann. Lediglich die maximale Multiplizität von `E2` bleibt erhalten. Für das Referenzattribut wird die Elementeigenschaft `deleteRule=Restrict` gesetzt, da dies am besten der Semantik der Mul-

tiplizität `1..1` des Entity-Typs `E2` entspricht. Abbildung 7.11 zeigt die Umsetzung einer 1:1-Beziehung mit einseitig partieller Partizipation durch Referenzen.

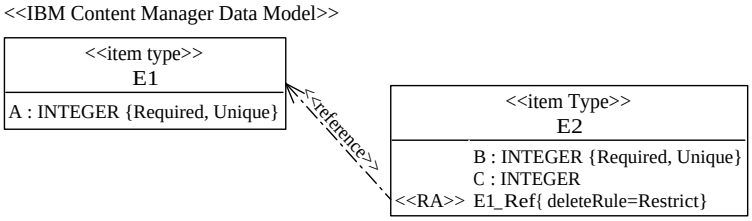


Abbildung 7.11: Umsetzungen einer 1:1-Beziehung mit einseitig partieller Partizipation durch Referenzen

Beidseitig partielle Partizipation

Ist die minimale Multiplizität auf beiden Seiten null, so spricht man von *beidseitig partieller Partizipation*. Die Abbildung dieser Beziehung ist ebenfalls durch Referenzen möglich. Diese Variante wurde bereits als Alternative im vorherigen Abschnitt erläutert. `E2` wird als Item Type definiert und referenziert Instanzen von `E1`. Alle Beziehungsattribute werden dem Item Type von `E2` zugeordnet. Als Löschregel für das Referenzattribut wird in diesem Fall die Einstellung `deleteRule=Set null` verwendet, da Items des Typs `E2` auch ohne Beziehung existieren können. Die Multiplizitäten des konzeptuellen Datenmodells werden damit bis auf die maximale Multiplizität von `E1` korrekt abgebildet. In Abbildung 7.12 ist ein Beispiel für die Umsetzung einer 1:1-Beziehung mit beidseitig partieller Partizipation durch Referenzen dargestellt.

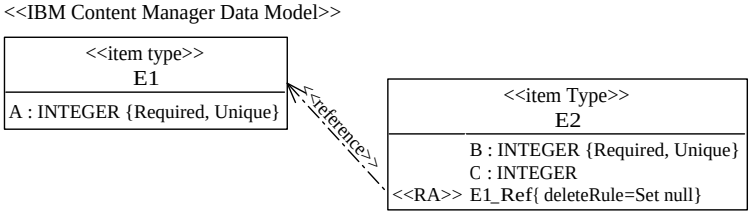


Abbildung 7.12: Umsetzungen einer 1:1-Beziehung mit beidseitig partieller Partizipation durch Referenzen

Anders als bei der Umsetzung einer Beziehung mit einseitig partieller Partizipation sind bei beidseitig partieller Partizipation auch unabhängige Instanzen von `E2` möglich. In diesem Fall muss das Referenzattribut auf `null` gesetzt werden. Den Beziehungsattributen, die ebenfalls im Item Type von `E2` gespeichert werden, muss ebenfalls ein entsprechender Wert zugewiesen werden, so dass erkennbar ist, dass keine Beziehung zu einem Item von `E1` besteht. Diese Vorgehensweise ist zwar zulässig, aber nicht besonders elegant. Insbesondere dann, wenn der Beziehung selbst Attribute zugeordnet sind, sollte über eine alternative Umsetzung mit Hilfe von Links und beschreibenden Objekten nachgedacht werden. Abbildung 7.13 zeigt ein entsprechendes Beispiel.

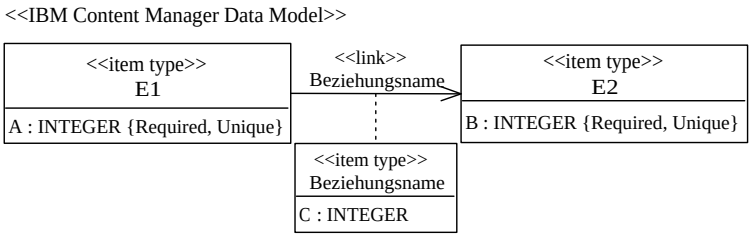


Abbildung 7.13: Umsetzungen einer 1:1-Beziehung mit beidseitig partieller Partizipation durch Links

Nachteil dieser Variante ist, dass alle Multiplizitäten des konzeptuellen Datenmodells verloren gehen, da die Anzahl der Links weder beschränkt noch erzwungen werden kann. Außerdem können Links nur zwischen Root Components erstellt werden. Falls einer der Entity-Typen auf Grund einer anderen Beziehung als Child Component modelliert wurde, so ist die Verwendung von Links nicht möglich.

7.3.2 1:n-Beziehungen

Als *1:n-Beziehungen* werden Beziehungen bezeichnet, bei denen die maximale Multiplizität auf der einen Seite eins und auf der anderen Seite unbegrenzt (* oder n) ist. Abbildung 7.14 zeigt eine allgemeine 1:n-Beziehung.

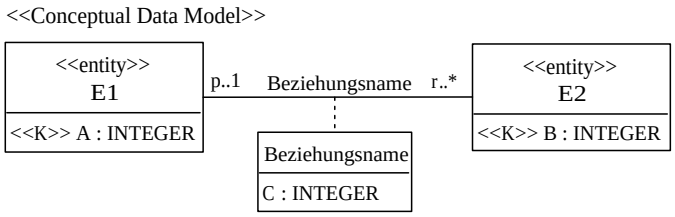


Abbildung 7.14: Allgemeine 1:n-Beziehung

Totale Partizipation der 1-Seite

Ist der Entity-Typ `E2` total an der Beziehung beteiligt, das heißt, für die minimalen Multiplizitäten gilt `p=1` sowie `r=0` oder `r=1`, dann wird `E2` als Child Component von `E1` mit der Kardinalität `r..*` definiert. Da jede Child Component genau einer Root Component zugeordnet ist, werden alle Multiplizitäten des konzeptuellen Datenmodells erfüllt. Eventuell vorhandene Beziehungsattribute werden der Child Component zugeordnet. Abbildung 7.15 zeigt ein entsprechendes Beispiel.

Ist die Umsetzung über eine Child Component nicht möglich, da `E2` an einer weiteren Beziehung beteiligt ist, wird `E2` als Item Type modelliert, der die Instanzen von `E1` referenziert. Die minimale Multiplizität `p=1` wird auf diese Weise aber nicht sichergestellt, da Referenzen nicht erzwungen werden können. Für das Referenzattribut sollte die Eigenschaft `deleteRule=Restrict` gesetzt werden, da die minimale Multiplizität von `E2`

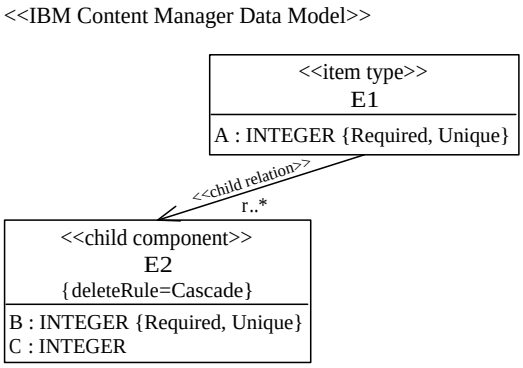


Abbildung 7.15: Umsetzung einer 1:n-Beziehung mit totale Partizipation der 1-Seite durch eine Child Component

eins ist. Abbildung 7.16 zeigt die Umsetzung einer 1:n-Beziehung mit totale Partizipation der 1-Seite durch Referenzen.

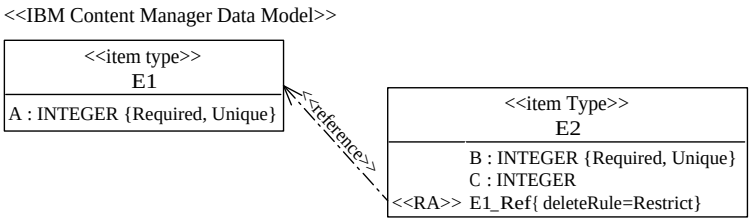


Abbildung 7.16: Umsetzung einer 1:n-Beziehung mit totale Partizipation der 1-Seite durch Referenzen

Partielle Partizipation der 1-Seite

Wenn der Entity-Typ `E2` partiell an der 1:n-Beziehung beteiligt ist, das heißt, für die minimalen Multiplizitäten gilt `p=0` sowie `r=0` oder `r=1`, wird die Beziehung auf Referenzen abgebildet. Im Item Type `E2` wird dafür ein Referenzattribut definiert, das auf Instanzen von `E1` verweist. Als Löschregel wird `deleteRule=Set null` festgelegt, da Items von `E2` auch unabhängig existieren können. Die Multiplizitäten des konzeptuellen Datenmodells können auf diese Weise exakt abgebildet werden. Eventuell vorhandene Beziehungsattribute werden dem Item Type `E2` zugeordnet. Abbildung 7.17 zeigt ein Beispiel für die Umsetzung einer 1:n-Beziehung mit partieller Partizipation der 1-Seite.

Wie auch bei der Umsetzung von 1:1-Beziehungen mit beidseitig partieller Partizipation können Items von `E2` existieren, die nicht in Beziehung mit einem Item von `E1` stehen. Dies führt auch hier zu dem Problem, dass in diesem Fall das Referenzattribut und die Beziehungsattribute auf `null` gesetzt werden müssen. Insbesondere dann, wenn Beziehungsattribute vorhanden sind, bietet sich eine alternative Umsetzung mit Hilfe von Links an. Abbildung 7.18 zeigt ein entsprechendes Beispiel.

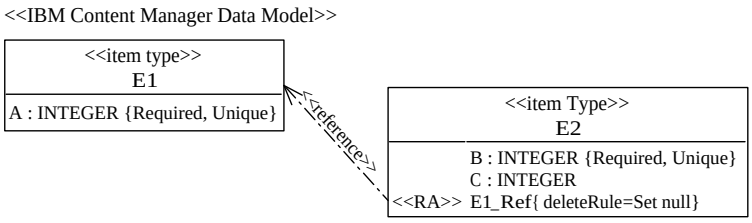


Abbildung 7.17: Umsetzungen einer 1:n-Beziehung mit partieller Partizipation der 1-Seite durch Referenzen

Nachteil dieser Variante ist, dass die maximale Multiplizität von `E2` verloren geht, da die Anzahl der Links von `E1` nach `E2` nicht beschränkt werden kann.

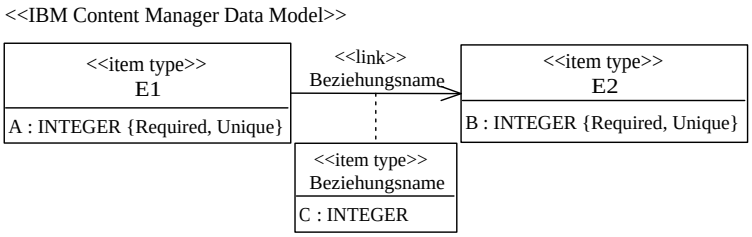


Abbildung 7.18: Umsetzungen einer 1:n-Beziehung mit partieller Partizipation der 1-Seite durch Links

7.3.3 n:m-Beziehungen

Gibt es bei einer binären Beziehung keine Einschränkungen für die maximalen Multiplizitäten, so spricht man von einer *n:m-Beziehung*. Da die Items beider Entity-Typen jeweils mit mehreren Items des anderen Entity-Typs in Beziehung stehen, erfolgt die Umsetzung von n:m-Beziehungen mit Hilfe von Links. Beziehungsattribute werden über ein beschreibendes Objekt mit dem Link assoziiert. Da nicht festgelegt werden kann, wie viele Links einem Item mindestens zugeordnet sind, gehen die minimalen Multiplizitäten des konzeptuellen Datenmodells verloren. Abbildung 7.19 zeigt die Umsetzung einer n:m-Beziehung im ICM-Datenmodell.

7.3.4 Aggregation

Aggregationen können als Menge gewöhnlicher Beziehungen betrachtet werden. Die einzelnen Beziehungen werden wie in den vorherigen Abschnitten beschrieben, auf das ICM-Datenmodell abgebildet. Abbildung 7.20 zeigt die Umsetzung einer Aggregation, die aus einer 1:n-Beziehung mit partieller Partizipation der 1-Seite und einer 1:1-Beziehung mit beidseitig partieller Partizipation besteht. Beide Beziehungen werden auf Referenzen abgebildet.

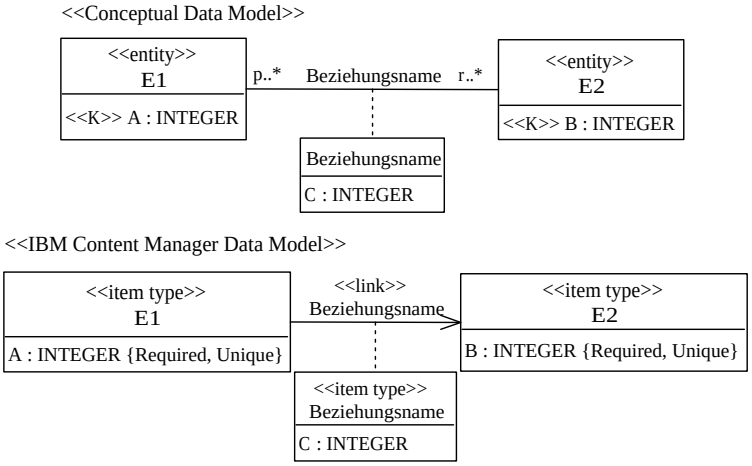


Abbildung 7.19: Umsetzung einer n:m-Beziehung im ICM-Datenmodell

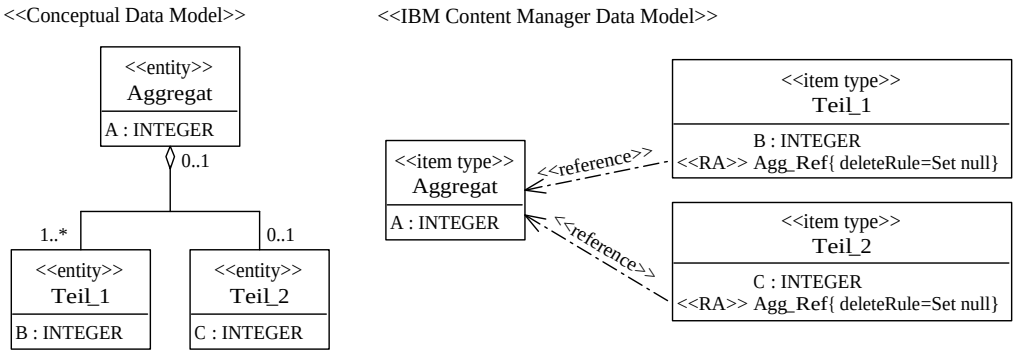


Abbildung 7.20: Umsetzung einer Aggregation im ICM-Datenmodell

7.3.5 Komposition / Schwache Entity-Typen

Die Abbildung von Kompositionen auf das ICM-Datenmodell ist schwierig. Folgender Sachverhalt muss modelliert werden: Die Items eines Item Types sind abhängig von der Existenz eines übergeordneten Items. Bei schwachen Entity-Typen können zusätzlich partielle Schlüssel modelliert werden. In diesem Fall wird ein abhängiges Item durch einen kombinierten Schlüssel, der aus dem Schlüssel des übergeordneten Items und dem eigenen partiellen Schlüssel besteht, identifiziert. In Abbildung 7.21 ist ein Beispiel für eine Komposition mit schwachen Entity-Typen im konzeptuellen Datenmodell dargestellt.

Eine Möglichkeit, Kompositionen auf das ICM-Datenmodell abzubilden, besteht darin, den oder die abhängigen Entity-Typen als Child Components zu modellieren. Damit wird sichergestellt, dass Instanzen der abhängigen Entity-Typen nicht unabhängig existieren können und immer zu einem übergeordneten Item gehören. Als Löschregel für die Child Component wird `deleteRule=Cascade` verwendet. Sobald ein übergeordnetes Item gelöscht wird, werden gleichzeitig auch alle abhängigen Items gelöscht.

Die Umsetzung von partiellen Schlüsseln bei der Verwendung von schwachen Entity-

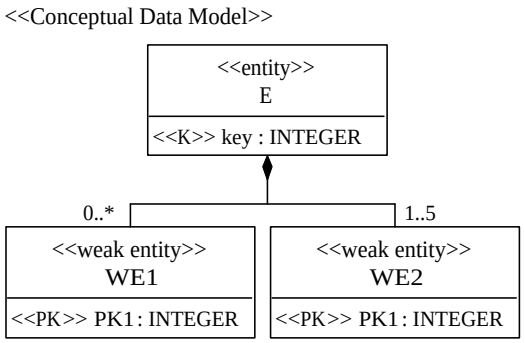


Abbildung 7.21: Komposition mit schwachen Entity-Typen

Typen ist nicht möglich, da alle Instanzen einer Child Component, unabhängig davon welchem Item sie zugeordnet sind, in derselben Datenbanktabelle gespeichert werden. Definiert man ein Attribut einer Child Component als Schlüssel, so ist dieses Attribut eindeutig für alle Instanzen. Ein partieller Schlüssel ist aber lediglich eindeutig über alle Instanzen mit demselben übergeordneten Item. Die Umsetzung von partiellen Schlüsseln über Fremdschlüsselbeziehungen nach dem Vorbild des relationalen Modells ist nicht möglich, da Fremdschlüssel nicht zwischen Root Components und Child Components definiert werden können. Der partielle Schlüssel geht also bei dieser Abbildung verloren und wird lediglich als gewöhnliches Attribut der Child Component mit der Eigenschaft `Required` definiert. Abbildung 7.22 zeigt auf der linken Seite die Umsetzung der Komposition aus Abbildung 7.21 mit Child Components.

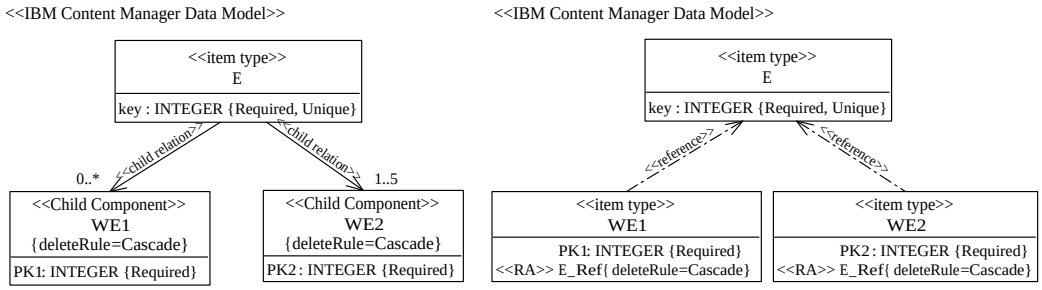


Abbildung 7.22: Umsetzung einer Komposition mit schwachen Entity-Typen im ICM-Datenmodell durch Child Components und Referenzen

Alternativ können Referenzen zur Umsetzung von Kompositionen verwendet werden. Dabei wird innerhalb der Item Types, welche die abhängigen Entity-Typen repräsentieren, jeweils ein Referenzattribut definiert, dass auf den übergeordneten Item Type zeigt. Jeder Instanz der abhängigen Item Types kann damit, wie gefordert, maximal ein übergeordnetes Item zugeordnet werden. Da Referenzen aber nicht erzwungen werden können, lässt es sich nicht verhindern, dass auch unabhängige Items auftreten können. Dies widerspricht der Semantik einer Komposition. Wird für die Referenzattribute die Löschregel `deleteRule=Cascade` verwendet, ist aber sichergestellt, dass nachdem eine Referenz

aufgebaut wurde, das abhängige Item an die Lebensdauer des übergeordneten Items gebunden ist.

Bei der Verwendung von schwachen Entity-Typen wird der partielle Schlüssel als Attribut mit der Eigenschaft `Required` definiert. Eine eindeutige Identifizierung der abhängigen Items ist durch die Kombination des partiellen Schlüssels und des Referenzattributs möglich. Ein zusammengesetzter Schlüssel würde die Eindeutigkeit dieser Attribute sicherstellen. Dies ist aber nicht möglich, da Referenzattribute nicht in einen Index aufgenommen werden können. Die Eindeutigkeit kann daher nur auf Applikationsebene, nicht aber Server-seitig überprüft werden. In Abbildung 7.22 ist auf der rechten Seite ein Beispiel für die Umsetzung einer Komposition durch Referenzen dargestellt.

7.3.6 Mehrstellige Beziehungen

Mehrstellige Beziehungen sind Beziehungen, an denen mehr als ein Entity-Typ beteiligt ist. Die Umsetzung solcher Beziehungen erfolgt im ICM-Datenmodell über einen Item Type, der für jeden an der Beziehung beteiligten Entity-Typ ein Referenzattribut erhält. Ein Item dieses Item Types repräsentiert eine Instanz der Beziehung. Abbildung 7.23 zeigt die Umsetzung einer mehrstelligen Beziehung an der drei Entity-Typen beteiligt sind.

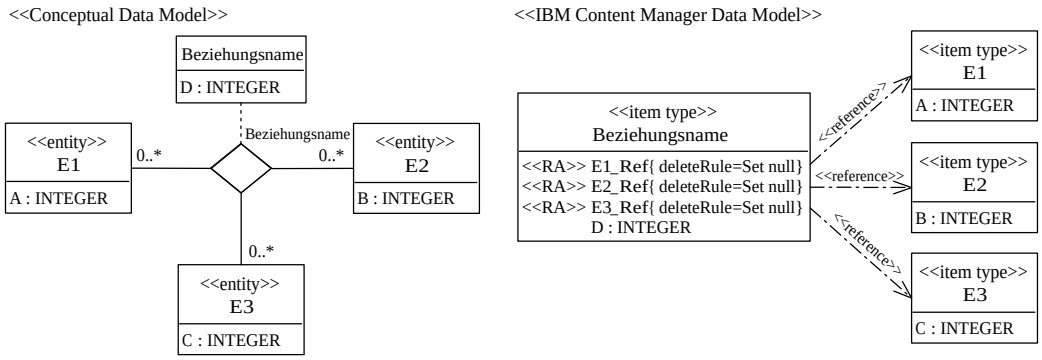


Abbildung 7.23: Umsetzung einer mehrstelligen Beziehung im ICM-Datenmodell

7.3.7 Rekursive Beziehungen

Als *rekursive Beziehungen* werden Beziehungen zwischen Entities desselben Entity-Typs bezeichnet. Rekursive Beziehungen werden auf Links abgebildet, da Referenzen nicht auf Items desselben Item Types zeigen sollten (vgl. [Cor04a]). Die Multiplizitäten des konzeptuellen Datenmodells gehen dabei verloren. Beziehungsattribute werden innerhalb eines Item Types definiert, dessen Items den Links zugeordnet werden. Abbildung 7.24 zeigt ein Beispiel für die Umsetzung einer rekursiven Beziehung durch Links.

7.4 Generalisierung

Im ICM-Datenmodell ist kein Mechanismus zur direkten Umsetzung von Vererbungshierarchien enthalten. Die Abbildung von Generalisierungshierarchien auf Item Types erfolgt

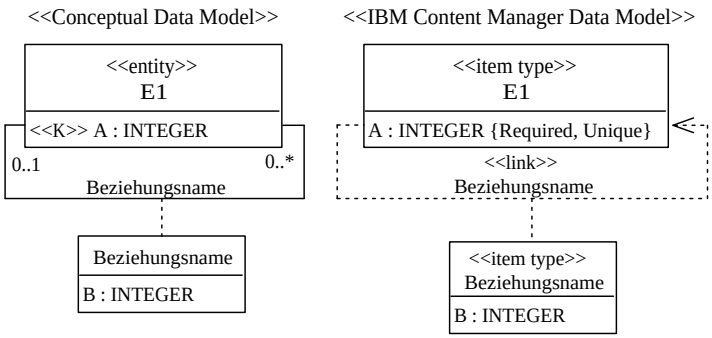


Abbildung 7.24: Umsetzung einer rekursiven Beziehung im ICM-Datenmodell

daher ähnlich, wie die Abbildung auf Tabellen des Relationenmodells. Die Umsetzung von Generalisierungshierarchien wird an Hand eines Beispiels aus dem Bibliotheksbereich erläutert. Abbildung 7.25 zeigt eine Generalisierung zwischen den Entity-Typen `Dokument`, `Buch` und `Artikel`. Zu einem Buch wird die ISBN-Nummer und der Verlag erfasst in dem es erschienen ist. Zu einem Artikel wird der Name der Zeitschrift gespeichert, in dem dieser veröffentlicht wurde. Sowohl Bücher, als auch Artikel erben das Attribut `Titel` des Entity-Typs `Dokument`.

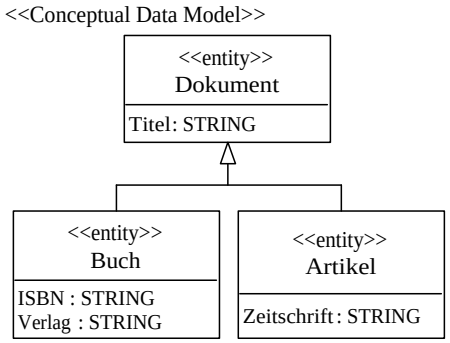


Abbildung 7.25: Beispiel einer Generalisierungshierarchie

Disjunkte Generalisierungshierarchien, in denen ein Entity jeweils in maximal einem Untertyp vorkommt, können im ICM-Datenmodell durch einen Item Type und jeweils einen Item Type Subset für jeden Untertyp dargestellt werden. Der Item Type erhält die Attribute des Obertyps und alle Attribute der Untertypen. Zusätzlich wird ein Attribut `Typ` eingeführt, das den Typ eines Items festlegt. Für jeden Untertyp wird ein Item Type Subset erstellt, welcher die jeweiligen Attribute des Untertyps erhält. Das Attribut `Typ` wird als Filter verwendet, so dass über die Item Type Subsets nur die Items der entsprechenden Untertypen erreichbar sind. Beim Anlegen von neuen Items werden die nicht verwendeten Attribute auf `null` gesetzt. Abbildung 7.26 zeigt die Umsetzung der Generalisierung aus Abbildung 7.25. Für die Untertypen `Buch` und `Artikel` werden zwei Item Type Subsets definiert. Mit den Filterregeln `Typ="Buch"` bzw. `Typ="Artikel"` wird sichergestellt, dass die Item Type Subsets nur die Dokumente des Types Buch bzw. Artikel beinhalten.

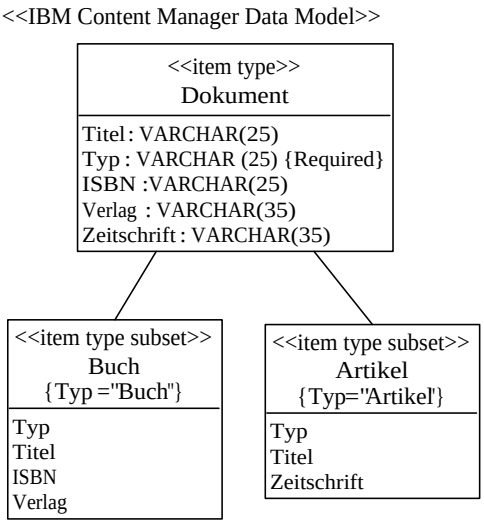


Abbildung 7.26: Umsetzung einer Generalisierungshierarchie durch Item Type Subsets

Alternativ können Generalisierungen durch Child Components, oder falls Child Components nicht verwendet werden können, durch Referenzen umgesetzt werden. Dabei wird für jeden Entity-Typ ein Item Type bzw. eine Child Component mit den lokalen Attributen des Entity-Typs erstellt. Da ein Item eines Obertyps von den Items mehrerer Untertypen gleichzeitig referenziert werden kann, können auf diese Weise auch überlappende Generalisierungen realisiert werden. Ein weiterer Vorteil dieser Umsetzung ist, dass die Struktur der Generalisierung erhalten bleibt und nachträglich relativ einfach angepasst werden kann. Änderungen in der Struktur haben lediglich lokale Auswirkungen auf den betroffenen Item Type. Darüberhinaus werden Attribute mit `null`-Werten vermieden. Abbildung 7.27 zeigt die Umsetzung des Beispiels aus Abbildung 7.25 durch Child Components auf der linken und durch Referenzen auf der rechten Seite.

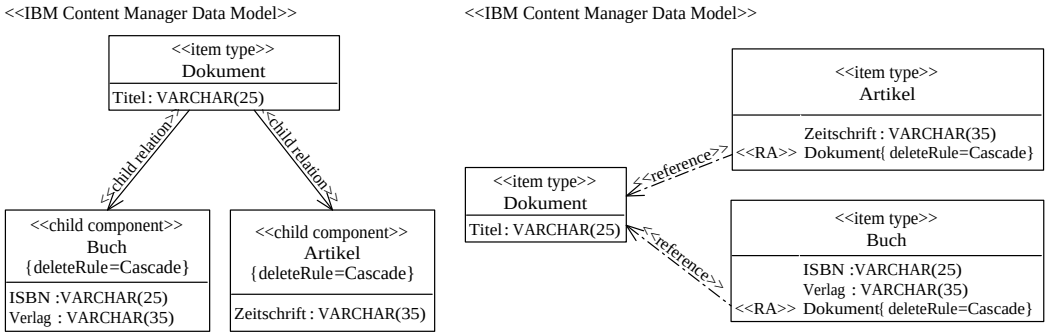


Abbildung 7.27: Umsetzung einer Generalisierungshierarchie durch Child Components und Referenzen

7.5 Operationen

Operationen können nicht ohne weiteres auf das ICM-Datenmodell abgebildet werden, da ein entsprechendes Konzept im *IBM DB2 Content Manager for Multiplatforms* nicht existiert. In Kapitel 6.3 wurde aber bereits beschrieben, wie man die Unzulänglichkeiten des ICM-Datenmodells umgehen und Methoden in den *IBM DB2 Content Manager for Multiplatforms* integrieren kann. Damit die auf konzeptueller Ebene modellierten Informationen nicht verloren gehen, bietet sich an, das ICM-Datenmodell so zu erweitert, dass einem Item Type genau wie einem Entity-Typen Operationen zugeordnet werden können. Abbildung 7.28 zeigt ein entsprechendes Beispiel.

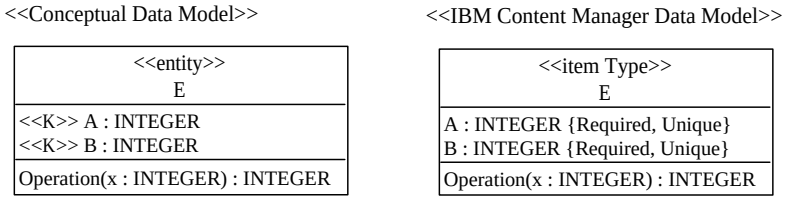


Abbildung 7.28: Abbildung von Operationen auf das ICM-Datenmodell

7.6 Zusammenfassung

In diesem Kapitel wurden Vorschriften zur Abbildung der Konstrukte von konzeptuellen Datenmodellen auf das ICM-Datenmodell vorgestellt. Bei der Anwendung der Abbildungsvorschriften müssen verschiedene Design-Entscheidungen getroffen werden, da für die meisten konzeptuellen Konstrukte verschiedene Umsetzungen möglich sind.

Entity-Typen können auf gewöhnliche Item Types oder Document Item Types abgebildet werden. Der Inhalt von Content-Attributen kann entweder in Resource Item Types oder Document Parts gespeichert werden. Welche Umsetzung gewählt wird hängt unter anderem davon ab, ob die IBM Standard-Clients verwendet werden sollen.

Für eine optimale Abbildung der Beziehungen eines konzeptuellen Datenmodells werden häufig Child Components benötigt. Child Components können aber nur verwendet werden, wenn deren Instanzen nicht referenziert oder verlinkt werden müssen und kein Content in der Child Component gespeichert wird. Für einige Beziehungskonstrukte des konzeptuellen Datenmodells wurden daher alternative Abbildungen vorgestellt, die ohne Child Components auskommen.

Die Abbildungsvorschriften dieses Kapitels wurden in den Tabellen im Anhang B.5 zusammengefasst.

Kapitel 8

Migration des eNoteHistory-Dokumentenservers

Das folgende Kapitel beschreibt die Migration des Datenbestands des *eNoteHistory*-Projekts in den *IBM DB2 Content Manager for Multiplatforms* mit Hilfe der in Kapitel 6 beschriebenen Verfahrensweise.

8.1 Integrationsstrategie

In Kapitel 4 wurden zwei Vorgehensweisen zur Integration eines Dokumentenarchivs in eine *IBM Content Manager*-Umgebung, jeweils mit ihren Vor- und Nachteilen, beschrieben. Zur Integration des *eNoteHistory*-Dokumentenservers wurde die Migrationsstrategie gewählt. Grund hierfür ist vor allem das schwache Datenmodell des *IBM DB2 Information Integrator for Content*, das nicht ausreicht um die Datenstrukturen der *eNoteHistory*-Datenbank in gewünschter Weise zur Verfügung zu stellen. Darüberhinaus sollen die speziellen Fähigkeiten des *IBM DB2 Content Manager for Multiplatforms* bezüglich der Verwaltung von Dokumenten ausgenutzt werden. Unter dem Item Type `Incipit` werden beispielsweise zu einem späteren Zeitpunkt Audio-Dateien gespeichert, die über einen Streaming Server zur Verfügung gestellt werden sollen.

8.2 Migration der Datenstrukturen

Das zur Abbildung der relationalen Datenstrukturen notwendige konzeptuelle Datenmodell befindet sich in der Dokumentation des *eNoteHistory*-Projekts. Das Datenmodell liegt allerdings in ER-Notation vor und ist veraltet. Zunächst wurde das Datenmodell daher in UML-Notation übertragen. Anschließend wurden fehlende Daten durch Reverse Engineering ergänzt, so dass das Datenmodell der aktuellen Struktur der *eNoteHistory*-Datenbank entspricht. Das aktualisierte konzeptuelle Datenmodell ist im Anhang in Abbildung C.3 dargestellt. Dieses Datenmodell wurde mit Hilfe des in Abschnitt 6.1.3 beschriebenen Algorithmus in ein ICM-Datenmodell konvertiert.

Bei der Abbildung des Datenmodells wurden Schlüsselattribute mit systemgenerierten Werten nicht übertragen, da diese im relationalen Schema der *eNoteHistory*-Datenbank aus-

schließlich für die Erstellung von Beziehungen verwendet werden. Im ICM-Datenmodell werden Beziehungen mit Hilfe des `pid`'s eines Items erstellt. Systemgenerierte Schlüsselattribute sind daher nicht notwendig. Aus diesem Grund wurden die Schlüsselattribute temporär in speziellen Child Components gespeichert und nach der vollständigen Datenübertragung wieder gelöscht.

Keine der Kompositionen, 1:n-Beziehungen mit totaler Partizipation der 1-Seite und 1:1-Beziehungen mit einseitig partieller Partizipation konnte im zweiten Schritt des Abbildungsalgorithmus in eine Child Relation umgewandelt werden. Grund hierfür ist die Tatsache, dass die Item Types, die in diesem Fall als Child Component deklariert werden müssten, an anderen Beziehungen beteiligt sind, die eine Referenzierung oder Verlinkung erfordern. Daher wurden die genannten Beziehungstypen durch ihre alternative Abbildungsvorschrift umgesetzt.

Die Dokumente des *eNoteHistory*-Projekts werden in Document Parts gespeichert. Die Item Types `Incipit`, `ManuscriptPage` und `Node` wurden daher als Document Item Types umgesetzt. Document Parts werden verwendet, da im Gegensatz zu Resource Item Types die Anzahl der gespeicherten Objekte innerhalb einer Document-Part-Kollektion nicht begrenzt ist. Falls zukünftig weitere Bilder oder andere Objekte zu einer Manuskriptseite gespeichert werden sollen, so ist dies ohne Änderungen am Datenmodell möglich.

Das temporäre ICM-Datenmodell des *eNoteHistory*-Projekts, das die Child Components zur vorübergehenden Speicherung der Primärschlüssel enthält, ist im Anhang in den Abbildungen C.4 und C.5 dargestellt. Das endgültige ICM-Datenmodell befindet sich im Anhang in Abbildung C.6.

8.3 Datenmigration

Der Datenbestands des *eNoteHistory*-Dokumentenservers wurde nach der in Abschnitt 6.2 beschriebenen Verfahrensweise in zwei Schritten migriert. Zunächst wurden die Daten der *eNoteHistory*-Datenbank übertragen, anschließend wurden die Beziehungen zwischen den importierten Items rekonstruiert.

8.3.1 Migration der Daten

Für die Migration der Daten wurde eine Java-Applikation entwickelt, die den Inhalt einer relationalen Datenbank über die Java-Database-Connectivity-Schnittstelle (JDBC) ausliest und über den entsprechenden IIC-Connector in den *IBM DB2 Content Manager for Multiplatforms* überträgt.

Das Programm erzeugt für jede Zeile einer Tabelle ein Item des entsprechenden Item Types. Die Zuordnung der einzelnen Tabellen zu einem Item Type erfolgt innerhalb einer XML-Datei, deren Struktur durch die in Abbildung 8.1 dargestellte DTD definiert wird.

Folgende Zusammenhänge sind in der DTD erkennbar: Ein Item Type besteht aus Attributen, Parts, einem Resource Content, Child Components und eventuell einem temporären Primärschlüssel. Das Item-Type-Element besitzt die XML-Attribute `name`, `source` und `property type`. Unter `name` wird ein Item-Type-Name festgelegt. Als `source` wird der Name der Tabelle angegeben, aus der die Items dieses Item Types erzeugt werden. Alternativ kann auch eine SQL-Anweisung formuliert werden, die eine entsprechende Tabelle

```
<!ELEMENT Mapping (ItemType+)>
<!ELEMENT ItemType (Attribute*, Part*, Resource?, ChildComponent*,
TempPrimaryKey?)>
<!ATTLIST ItemType name CDATA #REQUIRED
                   source CDATA #REQUIRED
                   propertyType (Document | Folder | Item) #REQUIRED>
<!ELEMENT Attribute EMPTY>
<!ATTLIST Attribute name CDATA #REQUIRED
                    source CDATA #REQUIRED>
<!ELEMENT Part EMPTY>
<!ATTLIST Part name CDATA #REQUIRED
               source CDATA #REQUIRED
               partNumber CDATA #REQUIRED
               mimeType CDATA #REQUIRED
               semanticType CDATA #REQUIRED>
<!ELEMENT Resource EMPTY>
<!ATTLIST Resource source CDATA #REQUIRED
                   mimeType CDATA #REQUIRED>
<!ELEMENT ChildComponent (Attribute+, RootItemIdentifier,
ChildComponent*)>
<!ATTLIST ChildComponent name CDATA #REQUIRED
                         source CDATA #REQUIRED>
<!ELEMENT RootItemIdentifier (IDAttribute*)>
<!ELEMENT IDAttribute EMPTY>
<!ATTLIST IDAttribute name CDATA #REQUIRED>
<!ELEMENT TempPrimaryKey (Attribute+)>
<!ATTLIST TempPrimaryKey childComponent CDATA #REQUIRED>
```

Abbildung 8.1: DTD zur Migration von relationalen Datenbanken in den *IBM DB2 Content Manager for Multiplatforms*

aus mehreren einzelnen Tabellen zusammensetzt. Das Attribut `propertyType` legt den Property Type des Item Types fest (siehe [Cor03f]).

Über das Element `Attribute` wird die Zuordnung der Tabellenattribute zu den Attributen des Item Types definiert. Mit dem XML-Attribut `name` wird der Name eines Item-Type-Attributs festgelegt. Unter `source` wird der Name des Tabellenattributs angegeben, das die Werte dieses Item-Type-Attributs liefert.

Document Item Types können Document Parts beinhalten, die über das XML-Element `Part` definiert werden. Neben dem Namen des Parts wird das Tabellenattribut festgelegt, aus dem der Content des Parts gewonnen wird. Darüberhinaus wird eine laufende Nummer, ein Mime Type sowie ein Semantic Type (siehe [Cor03f]) für jeden Part definiert.

Einem Resource Item Type wird sein Content mit Hilfe des `Resource`-Elements zugeordnet. Auch hier wird über das XML-Attribut `source` das Tabellenattribut angegeben,

das den Content des Resource Item Types liefert.

Besitzt ein Item Type Child Components, werden diese über das Element `ChildComponent` definiert. Jede Child Component besitzt einen Namen und eine Quelle. Eine Child Component entsteht bei der Migration einer relationalen Datenbank aus einer binären Beziehung, die im relationalen Schema aus zwei Tabellen besteht, die über eine Schlüssel-Fremdschlüssel-Beziehungen verbunden sind. Unter dem XML-Attribut `source` wird eine SQL-Anweisung angegeben, die einen Verbund dieser beiden Tabellen liefert. Mit dem Element `RootItemIdentifier` wird der Primärschlüssel der Tabelle der Root Component angegeben. Mit Hilfe dieser Informationen ist es möglich, festzustellen welche Child Components für welche Root Component erstellt werden müssen.

Wird zu einem Item Type der Primärschlüssel der Ursprungstabelle temporär gespeichert, so erfolgt die Deklaration mit Hilfe des `TempPrimaryKey`-Elements. Über das XML-Attribut `childComponent` wird der Name der Child Component festgelegt, die den Primärschlüssel aufnimmt. Die Zuordnung der Tabellenattribute zu den Attributen der Child Component erfolgt über `Attribute`-Elemente.

Die XML-Datei zur Abbildung der Datenstrukturen der *eNoteHistory*-Datenbank befindet sich im Anhang in Kapitel D.1.

8.3.2 Rekonstruktion der Beziehungen

Für die Rekonstruktion der Beziehungen zwischen den importierten Items der *eNoteHistory*-Datenbank wird das in Abschnitt 6.2.2 beschriebene Verfahren verwendet. Dazu wurde ebenfalls eine Java-Applikation entwickelt. Die notwendigen Informationen zur Rekonstruktion der Beziehungen aus den Schlüssel-Fremdschlüssel-Beziehungen der Tabellen des relationalen Schemas werden in einer XML-Datei spezifiziert, deren Struktur der DTD aus Abbildung 8.2 entspricht. Die XML-Datei beschreibt alle Links und Referenzen des ICM-Schemas.

Ein Link-Element enthält die XML-Attribute `name`, `sourceItemType`, `targetItemType` und `table`. Das XML-Attribut `name` legt den Link-Typ fest, unter dem die Link-Beziehung erstellt wird. Die Item Types der Quell- und Ziel-Items werden mit den XML-Attributen `sourceItemType` und `targetItemType` spezifiziert. Mit dem XML-Attribut `table` wird der Name der Tabelle festgelegt, welche die Schlüssel-Fremdschlüssel-Beziehung zwischen der Tabelle des Quell-Item-Types und des Ziel-Item-Types enthält. Die Zuordnung zwischen den Primärschlüssel- bzw. Fremdschlüsselattributen der Beziehungstabelle und den Primärschlüsselattributen der Item Types erfolgt über die Elemente `SourceItemPK` und `TargetItemPK`. Mit Hilfe dieser Informationen können die Items der Quell- und Ziel-Item-Types einer Zeile der Beziehungstabelle zugeordnet werden.

Zu einem Link kann optional ein beschreibendes Item angegeben werden. Die Definition dieses Items erfolgt mit dem Element `DescriptionItem`. Die Struktur des Elements `DescriptionItem` ähnelt der des Elements `ItemType` aus Abbildung 8.1. Lediglich das Element `TempPrimaryKey` wird nicht benötigt.

Eine Referenz wird über das Element `Reference` definiert. Wie auch bei Link-Beziehungen werden die Item Types der Quell- und Ziel-Items sowie die Tabelle, welche die relationale Beziehung speichert, angegeben. Zusätzlich wird über das XML-Attribut `referenceAttribute` das Referenzattribut des Quell-Item-Types, in dem die Refe-

```
<!ELEMENT Relations (Link*, Reference*)>
<!ELEMENT Link (SourceItemPK, TargetItemPK, DescriptionItem?)>
<!ATTLIST Link name CDATA #REQUIRED
               sourceItemType CDATA #REQUIRED
               targetItemType CDATA #REQUIRED
               table CDATA #REQUIRED>
<!ELEMENT Reference (SourceItemPK, TargetItemPK, RelationAttributes?)>
<!ATTLIST Reference sourceItemType CDATA #REQUIRED
                    referenceAttribute CDATA #REQUIRED
                    targetItemType CDATA #REQUIRED
                    table CDATA #REQUIRED>
<!ELEMENT SourceItemPK (Attribute+)>
<!ELEMENT TargetItemPK (Attribute+)>
<!ELEMENT DescriptionItem (Attribute*, Part*, Resource?,
ChildComponent*)>
<!ATTLIST DescriptionItem itemType CDATA #REQUIRED
                    propertyType (Document | Folder | Item) #REQUIRED>
<!ELEMENT RelationAttributes (Attribute+)>
<!ELEMENT Attribute EMPTY>
<!ATTLIST Attribute name CDATA #REQUIRED
                    source CDATA #REQUIRED>
<!ELEMENT Part EMPTY>
<!ATTLIST Part name CDATA #REQUIRED
               source CDATA #REQUIRED
               partNumber CDATA #REQUIRED
               mimeType CDATA #REQUIRED
               semanticType CDATA #REQUIRED>
<!ELEMENT Resource EMPTY>
<!ATTLIST Resource source CDATA #REQUIRED
                   mimeType CDATA #REQUIRED>
<!ELEMENT ChildComponent (Attribute+, RootItemIdentifier,
ChildComponent*)>
<!ATTLIST ChildComponent name CDATA #REQUIRED
                         source CDATA #REQUIRED>
<!ELEMENT RootItemIdentifier (IDAttribute*)>
<!ELEMENT IDAttribute EMPTY>
<!ATTLIST IDAttribute name CDATA #REQUIRED>
```

Abbildung 8.2: DTD zur Rekonstruktion der Beziehungen zwischen importierten Items

renz gespeichert wird, festgelegt. Die Zuordnung der Primärschlüssel- bzw. Fremdschlüsselattribute der Beziehungstabelle und den Primärschlüsselattributen der Item Types erfolgt

wie bei Links mit Hilfe der Elemente `SourceItemPK` und `TargetItemPK`. Falls die Beziehung des konzeptuellen Datenmodells, aus der die Referenzbeziehung entstanden ist, Beziehungsattribute hatte, so werden diese Attribute dem Quell-Item der Referenzbeziehung über das Element `RelationAttributes` zugeordnet.

Die XML-Datei für die Rekonstruktion der Beziehungen der *eNoteHistory*-Datenbank, befindet sich im Anhang im Kapitel D.2.

8.4 Migration der Abstandsfunktion

Der Dokumentenserver des *eNoteHistory*-Projekts beinhaltet eine Funktion zur Ermittlung der Ähnlichkeit von Handschriften. Die Funktion berechnet dazu den Abstand zwischen zwei Feature-Vektoren, in denen Handschriftcharakteristiken gespeichert sind (siehe [Mil04]). Die Funktion wurde als DB2-Table-UDF implementiert. Eine *Table UDF* wird in der `FROM`-Klausel einer SQL-Anweisung aufgerufen und liefert als Ergebnis eine Datenbanktabelle. Im Fall des *eNoteHistory*-Projekts beinhaltet die Ergebnistabelle alle Schreiber deren Handschrift dem vorgegebenen Feature-Vektor ähnelt.

Die Abstandsfunktion arbeitet ausschließlich auf Meta-Daten. Innerhalb der Funktion werden verschiedene Anfragen an die Datenbank gestellt, um alle notwendigen Informationen für die Abstandsberechnung zu ermitteln. Um den Datenverkehr zwischen Client und Server gering zu halten, wird die Funktion auch nach der Migration in den *IBM Content Manager* Server-seitig gespeichert und ausgeführt. Die Funktion wurde dafür in die Library-Server-Datenbank integriert. Damit der Umfang der notwendigen Anpassungen so gering wie möglich ausfällt, wurde die Funktion wieder als Table UDF umgesetzt. Alternativ hätte die Funktionslogik auch in eine Stored Procedure integriert werden können.

Die Datenbanktabellen, welche die Informationen zur Abstandsberechnung liefern, wurden während des Migrationsprozesses in Item Types konvertiert. Aus diesem Grund mussten alle SQL-Aufrufe, die in der Funktion verwendet werden, in die Anfragesprache des *IBM DB2 Content Manager for Multiplatforms* umgewandelt werden. Die Funktionslogik, welche die eigentlichen Berechnungen durchführt, musste nicht angepasst werden.

Der Aufruf der Funktion durch die Client-Anwendung erfolgt über eine JDBC-Verbindung mit Hilfe von SQL, da eine Integration in die Anfragesprache des *IBM DB2 Content Manager for Multiplatforms* nicht möglich ist.

8.5 Zusammenfassung

In diesem Kapitel wurde die Migration des Dokumentenservers des *eNoteHistory*-Projekts in den *IBM DB2 Content Manager for Multiplatforms* erläutert. Die Migration erfolgte nach der in Kapitel 6 beschriebenen Verfahrensweise.

Bei der Abbildung des Datenmodells des *eNoteHistory*-Projekts wurden Document Item Types zur Speicherung der Dokumente verwendet, da Document Parts flexibler sind als Resource Item Types. Schlüsselattribute mit systemgenerierten Werten wurden nicht in das ICM-Datenmodell übernommen, da sie im relationalen Schema ausschließlich zum Aufbau von Beziehungen genutzt werden.

Zur Migration der Daten und zur Rekonstruktion der Beziehungen wurden Java-Applikationen entwickelt, die mit Hilfe von XML-Dateien gesteuert werden. Die Struktur der XML-Dateien wurde in den entsprechenden Unterkapiteln erläutert.

Abschließend wurde die Migration der Abstandsfunktion in den Library Server des *IBM DB2 Content Manager for Multiplatforms* beschrieben. Die Funktion liegt als Table UDF vor und wird in dieser Form weiter verwendet. Zur Integration in die Library-Server-Datenbank mussten lediglich die SQL-Aufrufe in der Funktion durch Aufrufe der *IBM DB2 Content Manager for Multiplatforms*-Anfragesprache ersetzt werden.

Kapitel 9

Zusammenfassung und Ausblick

Ziel der vorliegenden Diplomarbeit war die Konzeption einer Strategie zur Integration von Dokumentenservern in eine *IBM Content Manager*-Umgebung. Zur Lösung dieser Aufgabe wurden zwei Integrationsstrategien analysiert: einerseits die Föderierung von Archiven über einen Föderierungsdienst und andererseits die Migration von Archiven in eine Content-Manager-Umgebung. In diesem Kapitel werden die Ergebnisse der vorliegenden Diplomarbeit zusammengefasst und mögliche Erweiterungen, die aufbauend darauf realisiert werden können, beschrieben.

9.1 Zusammenfassung

Nachdem zunächst Begriffe und Techniken erläutert wurden, die im Zusammenhang mit der Umsetzung der Integrationsverfahren verwendet werden, wurde anschließend mit der Unified Modeling Language eine Sprache zur Modellierung der in dieser Diplomarbeit verwendeten Datenmodelle vorgestellt. Für die Darstellung der unterschiedlichen Datenmodelle wurden UML-Profile entwickelt, die mit Hilfe von Stereotypen und Elementeigenschaften UML-Klassendiagramme so erweitern, dass die Besonderheiten der einzelnen Datenmodelle dargestellt werden können.

Für die Integration von Dokumentenarchiven werden zwei Produkte der *IBM Enterprise Content Management*-Produktreihe verwendet: zum Einen der *IBM DB2 Content Manager for Multiplatforms* und zum Anderen der *IBM DB2 Information Integrator for Content*. Die Konzepte und die Systemarchitektur dieser beiden Produkte wurden in Kapitel 3 erläutert.

Die Föderierung von Dokumentenarchiven erfolgt über den *IBM DB2 Information Integrator for Content*. Durch die Föderierung von Archiven ist es möglich, Applikationen zu erstellen, die auf verteilte Daten zugreifen, als wären diese zentral gespeichert. Dafür wird auf der Föderierungsschicht ein föderiertes Schema erstellt, in das Bestandteile der lokalen Schemata der verteilten Archive integriert werden. In Kapitel 5 wurde die Integration von lokalen Datenstrukturen in den *IBM DB2 Information for Integrator Content* erläutert.

Der wesentliche Teil der vorliegenden Diplomarbeit beschäftigt sich mit dem zweiten Ansatz zur Integration von Dokumentenarchiven: der Migration in den *IBM DB2 Content Manager for Multiplatforms*. Grund hierfür ist der größere Aufwand der mit dieser Art der Datenintegration verbunden ist.

Die Migration des Datenbestands eines Dokumentenarchivs erfolgt in zwei Schritten. Zunächst wird die Datenstruktur des Archivs auf das Datenmodell des *IBM DB2 Content Manager for Multiplatforms* abgebildet. Für diese Abbildung wird ein konzeptuelles Datenmodell benötigt. Ist in der Archivdokumentation kein entsprechendes Datenmodell vorhanden, so muss dies durch Reverse Engineering gewonnen werden. Anschließend wird das konzeptuelle Datenmodell auf das ICM-Datenmodell abgebildet. Dazu wurde in Abschnitt 6.1.3 ein Algorithmus präsentiert, der die einzelnen Konstrukte eines konzeptuellen Datenmodells durch Elemente des ICM-Datenmodells ersetzt. Die Abbildungsvorschriften der verschiedenen konzeptuellen Konstrukte, die im Algorithmus verwendet werden, wurden in Kapitel 7 detailliert beschrieben.

Nach der Abbildung der Datenstruktur eines Dokumentenarchivs auf das ICM-Datenmodell erfolgt die Migration der Daten und Methoden. Dazu wurde im Abschnitt 6.2 ein Verfahren erläutert, das zunächst die Daten kopiert und anschließend die Beziehungen zwischen den Daten rekonstruiert. Für die Migration von Methoden wurden in Abschnitt 6.3 drei verschiedene Ansätze vorgestellt, die je nach Art der zu migrierenden Methoden verschiedene Vor- und Nachteile bieten.

Das entwickelte Migrationsverfahren wurde abschließend bei der Migration eines Archivs von historischen Notenhandschriften angewendet. Dabei wurden zwei prototypische Applikationen zur Migration der Daten einer relationalen Datenbank in den *IBM DB2 Content Manager for Multiplatforms* entwickelt. Die Applikationen arbeiten nach dem in Abschnitt 6.2 vorgestellten Verfahren und werden über XML-Dateien gesteuert.

9.2 Ausblick

Erweiterungen und Optimierungen der in der vorliegenden Diplomarbeit beschriebenen Integrationsverfahren sind vor allem bei der Migration von Dokumentenarchiven möglich, da bei der Föderierung lediglich die Funktionalität des *IBM DB2 Information Integrator for Content* ausgenutzt wird. Im Folgenden werden Erweiterungsvorschläge gemacht, welche die Automatisierung und die praktische Umsetzung des Migrationsverfahrens betreffen.

Implementation des Abbildungsalgorithmus

Die Abbildung konzeptueller Datenmodelle auf das ICM-Datenmodell wurde bisher lediglich als Algorithmus formuliert, der bei der Migration des *eNoteHistory*-Projekts manuell angewendet wurde. Zur Beschleunigung der Abbildung und zur Verbesserung der Benutzerfreundlichkeit sollte der Algorithmus implementiert werden. Denkbar wäre dabei eine Anbindung an UML-Modellierungswerkzeuge wie z.B. *Rational Rose Data Modeler*[1] oder *PowerDesigner*[2].

Darüberhinaus wäre es sinnvoll, aus der erzeugten grafischen Repräsentation eines ICM-Datemodells automatisch Skripte zur Erstellung des entsprechenden ICM-Schemas zu erzeugen. Der *IBM DB2 Content Manager for Multiplatforms* bietet entsprechende Schnittstellen zum Erstellen von Schemata, die in XML-Dateien beschrieben sind.

[1] http://www-306.ibm.com/software/awdtools/developer/datamodeler/

[2] http://www.sybase.com/products/developmentintegration/powerdesigner

Erweiterung der Migrationsapplikationen

Für die Migration von relationalen Datenbanken wurden im Rahmen des *eNoteHistory*-Projekts zwei prototypische Applikationen zur Migration von Datenbanktupeln und zur Rekonstruktion von Beziehungen entwickelt. Dabei wurde zunächst lediglich die für die Migration der *eNoteHistory*-Datenbank notwendige Funktionalität umgesetzt. Da das *eNoteHistory*-ICM-Schema weder Child Components noch Resource Item Types enthält, werden diese Konstrukte durch die jetzige Implementierung nicht unterstützt. In den XML-Dateien zur Steuerung der Applikationen werden Child Components und Resource Item Types aber bereits berücksichtigt. Für den universellen Einsatz der beiden Applikationen sollte die Implementation der fehlenden Funktionalität nachgeholt werden.

Darüberhinaus ist es denkbar, die Erstellung der XML-Dateien zur Migration einer Datenbank zumindest teilweise zu automatisieren. Dabei könnte z.B. die Bezeichnung von Tabellen und Item Types genutzt werden, um Zusammenhänge zwischen diesen Elementen herzustellen.

Integration der Migrationschritte

Letztendlich können die drei Applikationen zur Abbildung von konzeptuellen Datenmodellen, zur Migration der Daten und zur Rekonstruktion der Beziehungen in eine Anwendung integriert werden. Auf diese Weise ist es möglich, eine Anwendung zu erstellen, die mit Hilfe einer grafischen Oberfläche in einem halbautomatischen Verfahren eine relationale Datenbank in den *IBM DB2 Content Manager for Multiplatforms* migriert. Ausgangspunkt für diese Anwendung sind eine relationale Datenbank und ein relationales Datenbankschema sowie ein konzeptuelles Datenmodell dieser Datenbank. Über die grafische Oberfläche können die nicht automatisierbaren Teile der Migration interaktiv gesteuert werden. Die Anwendung migriert die relationale Datenbank in den *IBM DB2 Content Manager for Multiplatforms* und liefert eine grafische Repräsentation des erstellten ICM-Schemas.

Anhang A

Das UML Data Modeling Profile

Das *UML Data Modeling Profile* ist ein UML-Profil zur Modellierung von relationalen Datenbanken, das von Rational Software entwickelt wurde. Das Profil umfasst unter anderem UML-Erweiterungen zur Modellierung von Datenbankschemata, Tabellen, Schlüsselattributen, Trigger und Constraints. Auch wenn das Profil bisher nicht offiziell von der OMG bestätigt wurde, so wird es auf Grund der Verbreitung der Modellierungswerkzeuge von Rational häufig zur Modellierung von relationalen Datenbanken eingesetzt. Abbildung A.1 zeigt zwei Tabellen des *eNoteHistory*-Projekts, die mit dem *UML Data Modeling Profile* dargestellt sind.

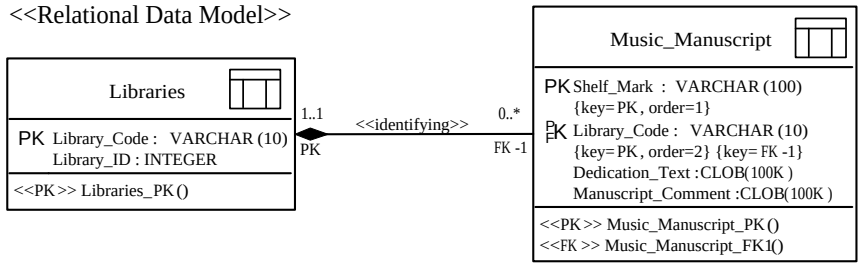


Abbildung A.1: Beispiel für die Darstellung relationaler Datenmodelle mit UML

Datenbanktabellen werden durch den Stereotyp `<<table>>` dargestellt, der das UML-Element `Klasse` erweitert. Der Name des Stereotyps wird üblicherweise nicht aufgeführt, stattdessen wird in der rechten oberen Ecke der Klassen-Box das -Symbol eingefügt.

Die Spalten einer Tabelle werden durch Stereotypen, die das UML-Element `Attribut` erweitern, dargestellt. Attributeigenschaften wie beispielsweise `nullable` oder `unique` werden durch Elementeigenschaften umgesetzt. Schlüsselattribute werden durch die dem Attributnamen vorangestellten Symbole PK für Primärschlüssel, FK für Fremdschlüssel und PK FK für Primär-Fremdschlüssel gekennzeichnet. Die Schlüsseleigenschaften werden in einer Datenbank durch Constraints umgesetzt. Für jeden Schlüssel einer Tabelle wird in der Datenbank ein Constraint erstellt, der die Struktur des Schlüssels definiert und die Schlüsseleigenschaft sicherstellt. In der UML werden Constraints als Stereotypen dargestellt, die vom Element `Operation` abgeleitet sind. Für die Umsetzung von Primär- und Fremdschlüssel-

Constraints wurden die Stereotypen `<<PK>>` und `<<FK>>` definiert. Da die Struktur eines Schlüssels aber nicht aus einem Constraint abgelesen werden kann, werden in der vorliegenden Diplomarbeit zusätzlich die in Kapitel 2.5.2 eingeführten Elementeigenschaften `key` und `order` verwendet, um zu verdeutlichen, welches Attribut zu welchem Schlüssel gehört.

Die durch Schlüssel und Fremdschlüssel definierten Beziehungen zwischen Tabellen werden durch Stereotypen der UML-Elemente Komposition bzw. Assoziation dargestellt. Dabei wird zwischen den Beziehungstypen `<<identifying>>` und `<<non-identifying>>` unterschieden. Ist der Fremdschlüssel einer Tabelle gleichzeitig auch Teil des Primärschlüssels, so ist die Tabelle abhängig von der Tabelle des durch den Fremdschlüssel referenzierten Primärschlüssels. Die Beziehung zwischen den beiden Tabellen wird in diesem Fall als „identifizierend“ bezeichnet. Im Beispiel aus Abbildung A.1 kann ein Musikmanuskript nicht ohne eine Bibliothek existieren, daher wird zur Darstellung der Beziehung der Stereotyp `<<identifying>>` verwendet. Tabellen, die über eine nicht identifizierende Beziehung verbunden sind, existieren unabhängig von einander. Als Rollen für eine Beziehung werden in der vorliegenden Diplomarbeit die an der Beziehung beteiligten Schlüssel verwendet.

In Tabelle A.1 sind die Stereotypen des *UML Data Modeling Profils* zusammengefasst. Eine ausführliche Beschreibung des Profils befindet sich in [NM01] und [Spa01] sowie in den Rational Whitepapers [Cor00, Gor03c, Gor03b] und in der Dokumentation zu *Rational Rose* (siehe [Cor03i]).

Stereotyp	**Basisklasse**	**Beschreibung**
Database	Komponente	Datenbank
Schema	Packet	Datenbankschema
Table	Klasse	Datenbanktabelle
Domain	Klasse	Domain (User Defined Type)
Non-Identifying	Assoziation	Nicht identifizierende Beziehung zwischen zwei Tabellen.
Identifying	Komposition	Identifizierende Beziehung zwischen zwei Tabellen.
Index	Operation	Index
PK	Operation	Primärschlüssel-Constraint
FK	Operation	Fremdschlüssel-Constraint
Unique	Operation	Unique Constraint
Check	Operation	Check Constraint
Trigger	Operation	Trigger
SP	Klasse	Stored Procedure

Tabelle A.1: Stereotypen des UML Data Modeling Profile (nach [Cor03i])

Anhang B

Tabellen

B.1 ICM-Datenmodell Unterstützung der IBM-Standard-Clients

Konzept	**Unterstützt durch Client for Windows**	**Unterstützt durch eClient**
Attribute	Ja[1]	Ja[1]
Attributgruppen	Nein	Ja
Root Component	Ja	Ja
Child Component	eine Ebene	eine Ebene
Item Type	Nein	Nein
Resource Item Type	Nein	Nein
Document Item Type	Ja	Ja
Document Part	Ja	Ja
Versionen	Ja	Ja
Media Object Class	Ja	Ja
Item Type Subset (Views)	Ja	Ja
Semantic Type	Document und Folder	Document und Folder
MIME Type	Ja	Ja
Links	nur Folder	nur Folder
Referenzen	Nein	Nein
Fremdschlüssel	Nein	Nein

Tabelle B.1: ICM-Datenmodell Unterstützung der IBM-Standard-Clients (nach [RDZ02])

[1] bis auf BLOB und CLOB

B.2 UML-Diagramme und Einsatzgebiete

Diagramm	**Einsatzgebiet**
Use-Case	Geschäftsprozesse, allgemeine Einsatzmöglichkeiten
Klassendiagramm	So gut wie überall, das Klassendiagramm ist das wichtigste Diagramm der UML.
Interaktionsdiagramm	Zeigt den Nachrichtenfluss und damit die Zusammenarbeit der Objekte im zeitlichen Ablauf.
- Sequenzdiagramm	Zeitliche Aufrufstruktur mit wenigen Klassen
- Kollaborationsdiagramm	Zeitliche Aufrufstruktur mit wenigen Nachrichten
Package-Diagramm	Groborientierung, in welchem Modul welche Klasse zu finden ist. Aufteilung in Unterprojekte, Bibliotheken, Übersetzungseinheiten.
Zustandsdiagramm	Darstellung des dynamischen Verhaltens.
Aktivitätsdiagramm	Bei parallelen Prozessen und anderer Parallelität, Geschäftsprozesse.
Implementierungsdiagramm	Besonders für die Darstellung von verteilten Anwendungen und Komponenten; allgemein: Darstellung von Implementierungsaspekten (Übersetzungseinheiten, ausführbare Programme, Hardwarestruktur)
- Komponentendiagramm	Zusammenhänge der Software
- Deployment-Diagramm	Hardwareaufbau

Tabelle B.2: UML Diagramme und Einsatzgebiete (nach [Wah98])

B.3 Ein UML-Profil zur Modellierung von ICM-Datenmodellen

Stereotyp	**Basisklasse**	**Textur/Symbol**	**Beschreibung**
IBM Content Manager Data Model	Modell	-	*IBM Content Manager*-Datenmodell.
item type	Klasse	-	Item Type
resource item type	Klasse	-	Resource Item Type
document item type	Klasse	-	Document Item Type
document part	Klasse	-	Document Part
child component	Klasse	-	Child Component Type
index	Klasse	-	Ein Index über die Attribute eines Item Type. Die im Index verwendeten Attribute müssen in der Definition des assoziierten Item Types als `Required` definiert sein.
item type subset	Klasse	-	Ein Item Type Subset, der eine Sicht auf einen Item Type repräsentiert.
link	Assoziation	gestrichelter Pfeil	Eine Link-Beziehung. Einem Link muss ein Name zugeordnet sein, außerdem kann optional ein Item Type mit dem Link assoziiert werden, der die Beziehung beschreibt. Der Pfeil gibt die Richtung der Beziehung an.
reference	Assoziation	Strich-Punkt-Pfeil	Eine Referenz-Beziehung. Der Anfang des Referenzpfeils sollte sich in der Nähe des entsprechenden Referenzattributs befinden.
child relation	Assoziation	Pfeil	Eine Beziehung zwischen einer Root Component und einer Child Component. Der Pfeil zeigt auf die Child Component.
FK	Attribut	-	Fremdschlüsselattribut
RA	Attribut	-	Referenzattribut

Tabelle B.3: Stereotypdefinitionen des UML-Profils zur Darstellung von ICM-Datenmodellen

Stereotyp	**Element-eigenschaft**	**mögliche Werte**	**Beschreibung**
FK	updateRule	Restrict, No action	Legt fest, ob das Zielattribut des Fremdschlüssels aktualisiert werden kann oder nicht.
	deleteRule	Restrict, Cascade, No action, Set null	Legt fest, wie das System reagiert, wenn das durch den Fremdschlüssel referenzierte Item gelöscht wird.
	constraint	*"Constraint-Name (18 Zeichen)„*	Der Name des Constraints, über den die Fremdschlüsselbeziehung in der Library-Server-Datenbank realisiert wird.
RA	deleteRule	Restrict, Cascade, No action, Set null	Legt fest, wie das System reagiert, wenn das durch das Referenzattribut referenzierte Item gelöscht wird.
child component	deleteRule	Restrict, Cascade	Legt fest, wie das System reagiert, wenn die Root Component einer Child Component gelöscht wird.
resource item type, document part	XDO	*"Name der XDO-Klasse„*	Legt die Media Objekt Class eines Resource Item Types oder eines Document Parts fest.

Tabelle B.4: Elementeigenschaften des UML-Profils zur Darstellung von ICM-Datenmodellen

B.4 Ein UML-Profil zur Modellierung von konzeptuellen Datenmodellen

Stereotyp	Basisklasse	Beschreibung
Conceptual Data Model	Modell	Ein konzeptuelles Datenmodell beschreibt Entities und die Beziehung zwischen Entities.
entity	Klasse	Ein Entity-Typ dient als Vorlage für Entities. Ein Entity-Typ beschreibt beliebige reale oder abstrakte Dinge des Anwendungsszenarios durch Attribute und Operationen.
weak entity	Klasse	Ein schwacher Entity-Typ dient als Vorlage für schwache Entities. Ein schwaches Entity kann nur in Abhängigkeit eines anderen Entity existieren.
K	Attribut	Ein Schlüsselattribut. Schlüsselattribute bestimmen durch ihre Werte die Instanzen eines Entity-Typs eindeutig.
PK	Attribut	Ein partieller Schlüssel. Partielle Schlüssel sind Teil des Schlüssels eines schwachen Entity-Typs.

Tabelle B.5: Stereotypdefinitionen des UML-Profils zur Modellierung von konzeptuellen Datenmodellen

Stereotyp	Element-eigenschaft	Beispiele	Beschreibung
K, PK	key	key = K-2	Legt fest, zu welchem Schlüssel ein Attribut gehört. (Beispiel: das Attribut gehört zum zweiten Schlüssel.)
K, PK	order	order = 3	Beschreibt die Position eines Attributs innerhalb eines zusammengesetzten Schlüssels. (Beispiel: das Attribut steht an dritter Position innerhalb eines zusammengesetzten Schlüssels.)

Tabelle B.6: Elementeigenschaften des UML-Profils zur Modellierung von konzeptuellen Datenmodellen

B.5 Abbildung von konzeptuellen Datenmodellen auf das ICM-Datenmodell

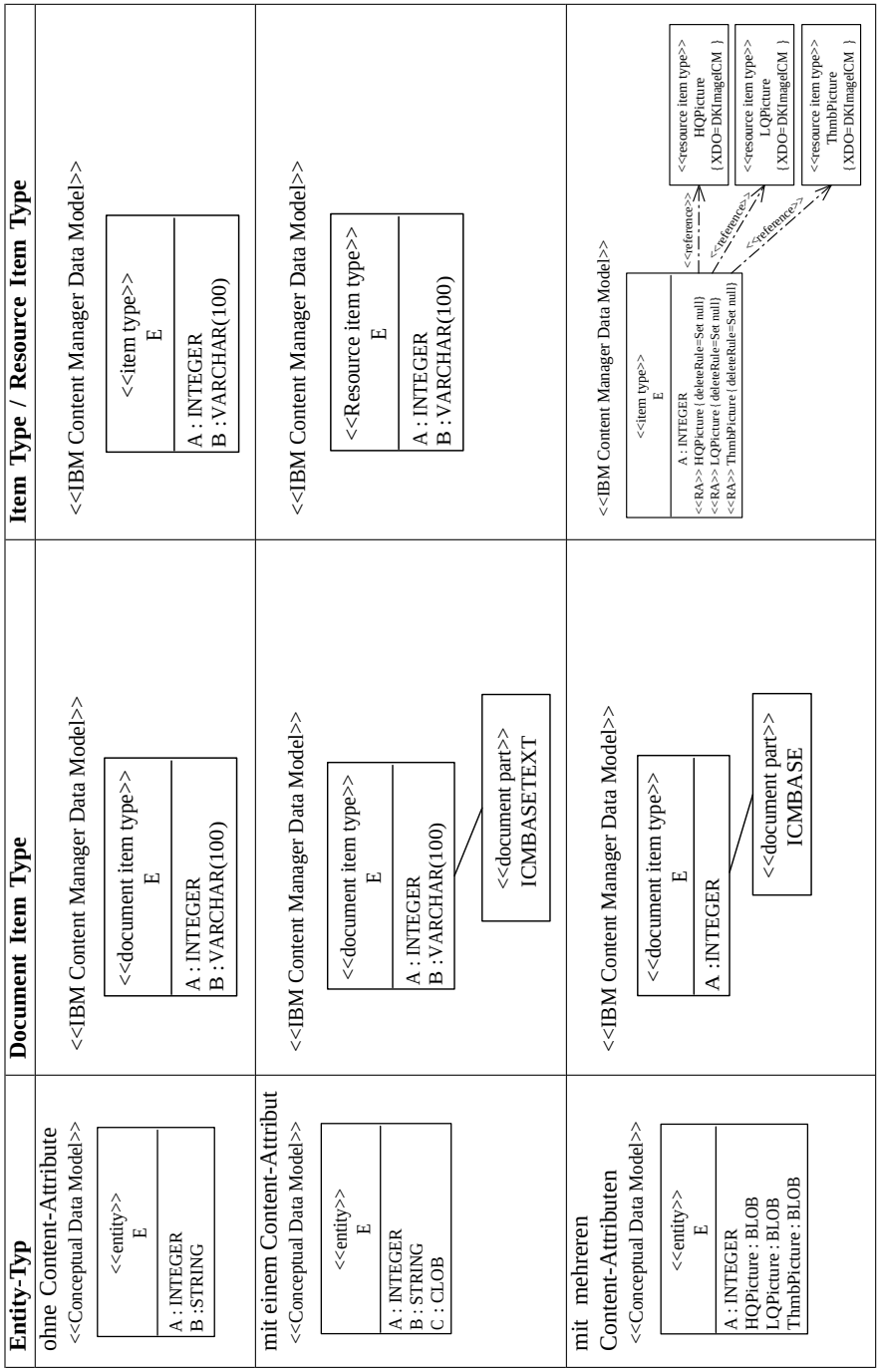


Tabelle B.7: Abbildung von Entity-Typen auf das ICM-Datenmodell

Beziehung	Ausprägung	Umsetzung	Alternative Umsetzung
1:1	beidseitig totale Partizipation	Verschmelzung	-
	einseitig partielle Partizipation	Child Component	Referenzen, falls Child Component nicht möglich
	beidseitig partielle Partizipation	Referenzen	Links, falls Beziehungsattribute vorhanden
1:n	totale Partizipation der 1-Seite	Child Component	Referenzen, falls Child Component nicht möglich

Tabelle B.8: Abbildung von konzeptuellen Beziehungen auf das ICM-Datenmodell I

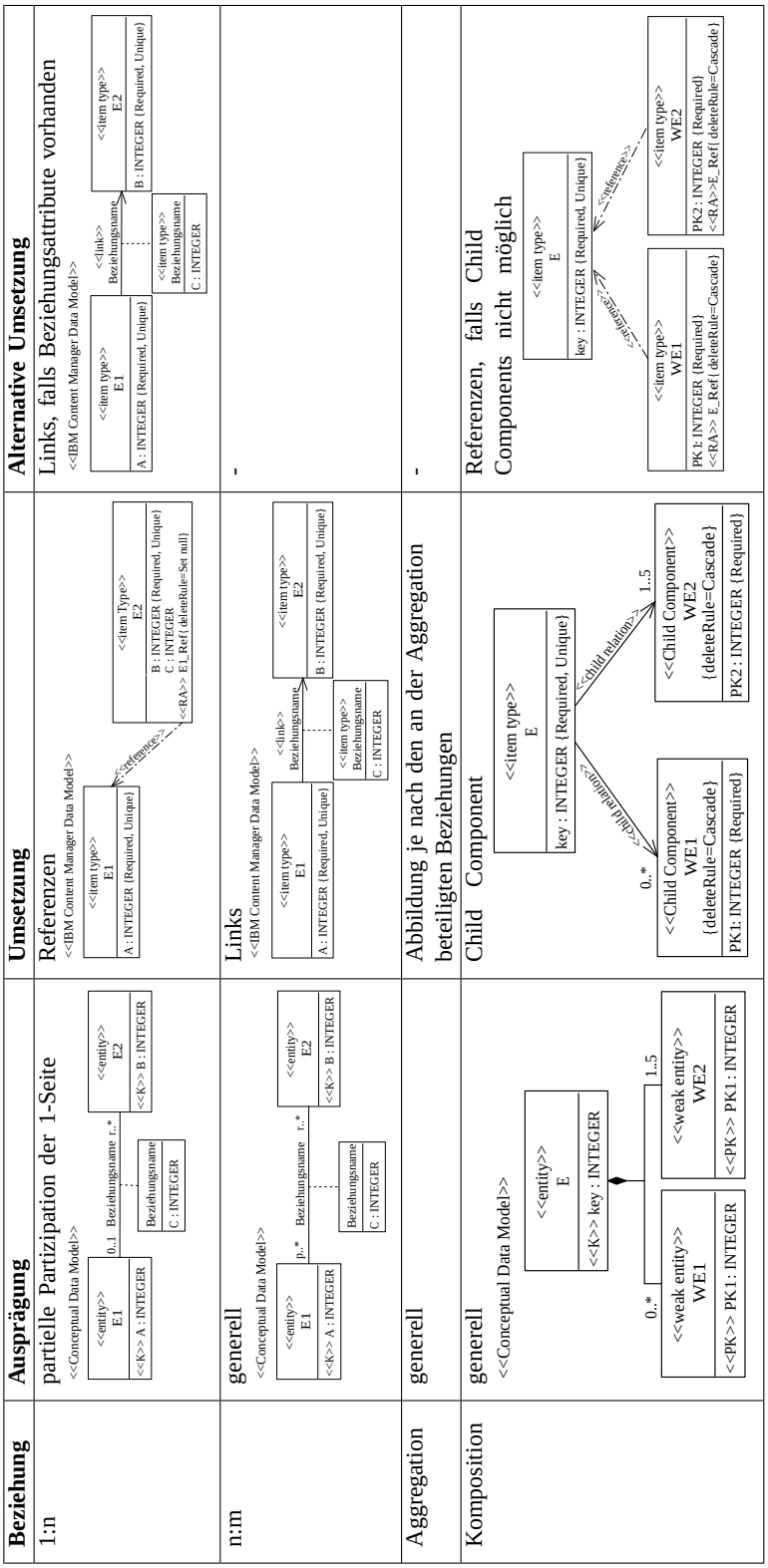


Tabelle B.9: Abbildung von konzeptuellen Beziehungen auf das ICM-Datenmodell II

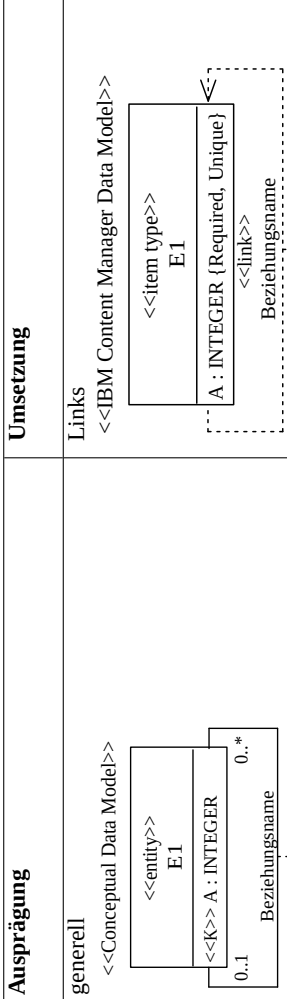
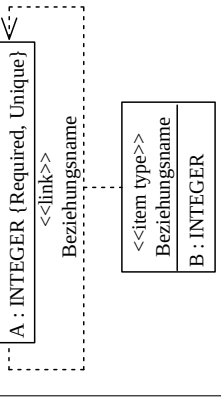
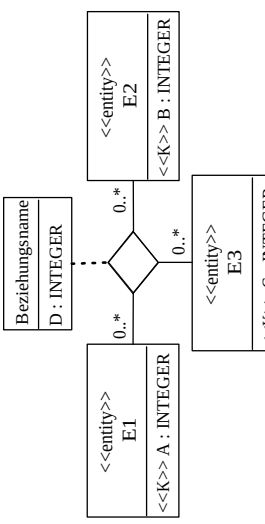
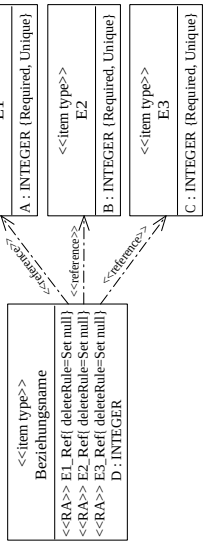
Beziehung	Ausprägung	Umsetzung	Alternative Umsetzung
rekursive Beziehungen	generell <<Conceptual Data Model>> 	Links <<IBM Content Manager Data Model>> 	-
mehrstellige Beziehungen	generell <<Conceptual Data Model>> 	Referenzen <<IBM Content Manager Data Model>> 	-

Tabelle B.10: Abbildung von konzeptuellen Beziehungen auf das ICM-Datenmodell III

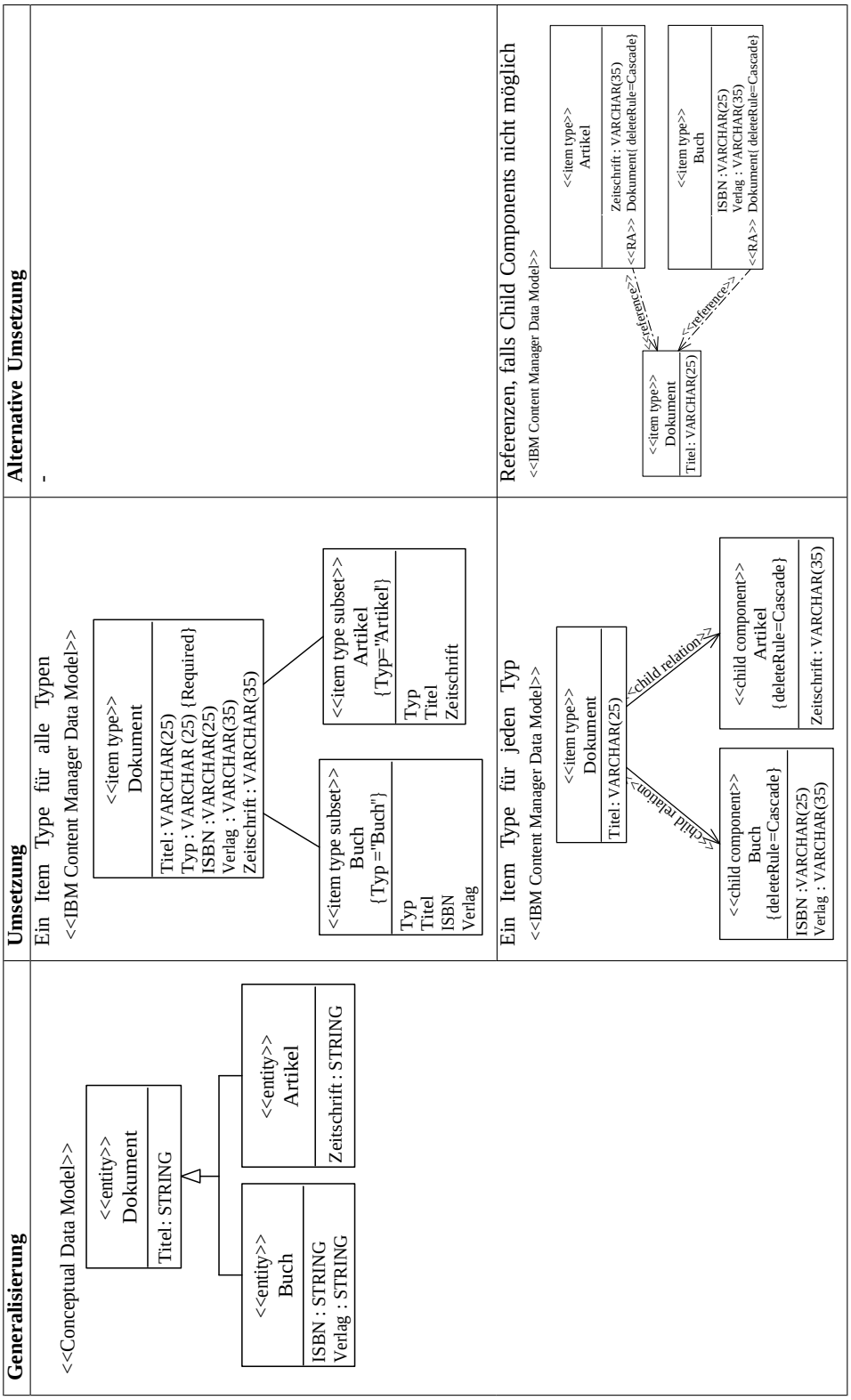


Tabelle B.11: Umsetzung von Generalisierungshierarchien im ICM-Datenmodell

Anhang C

Datenmodelle des eNoteHistory-Projekts

C.1 Relationales Datenmodell des eNoteHistory-Projekts

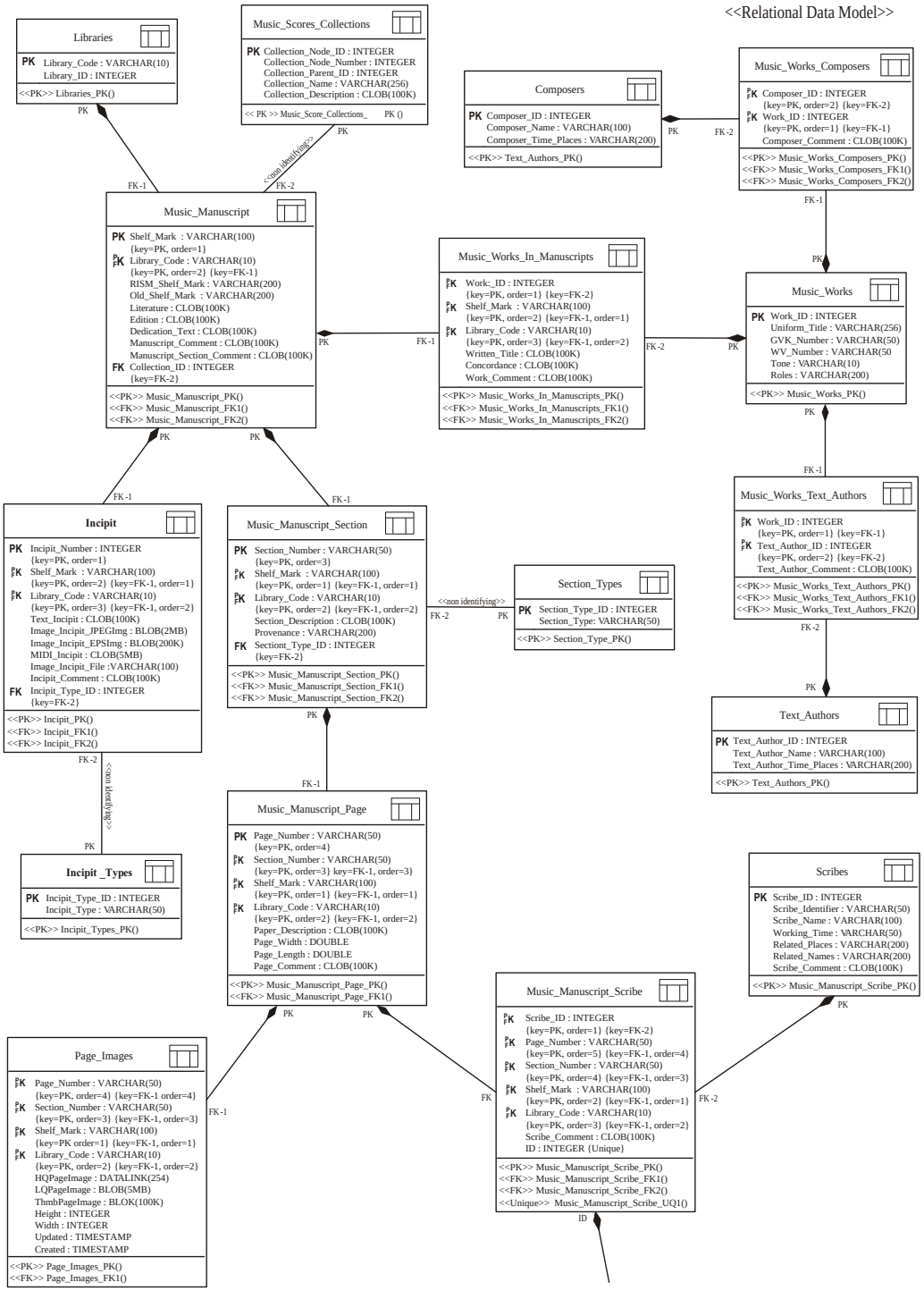


Abbildung C.1: Relationales Datenmodell des *eNoteHistory*-Projekts I

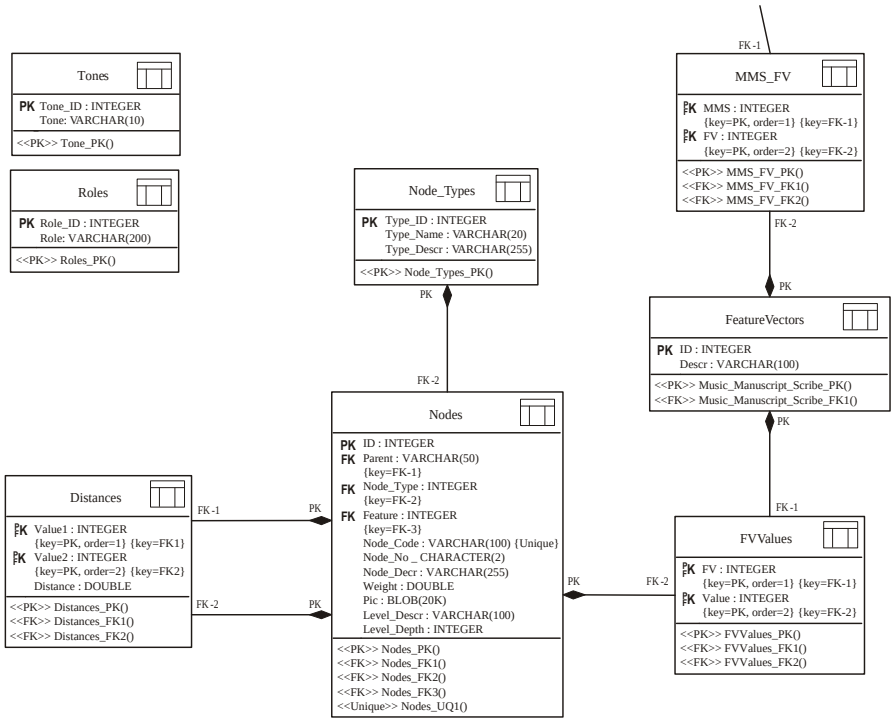


Abbildung C.2: Relationales Datenmodell des *eNoteHistory*-Projekts II

C.2 Konzeptuelles Datenmodell des eNoteHistory-Projekts

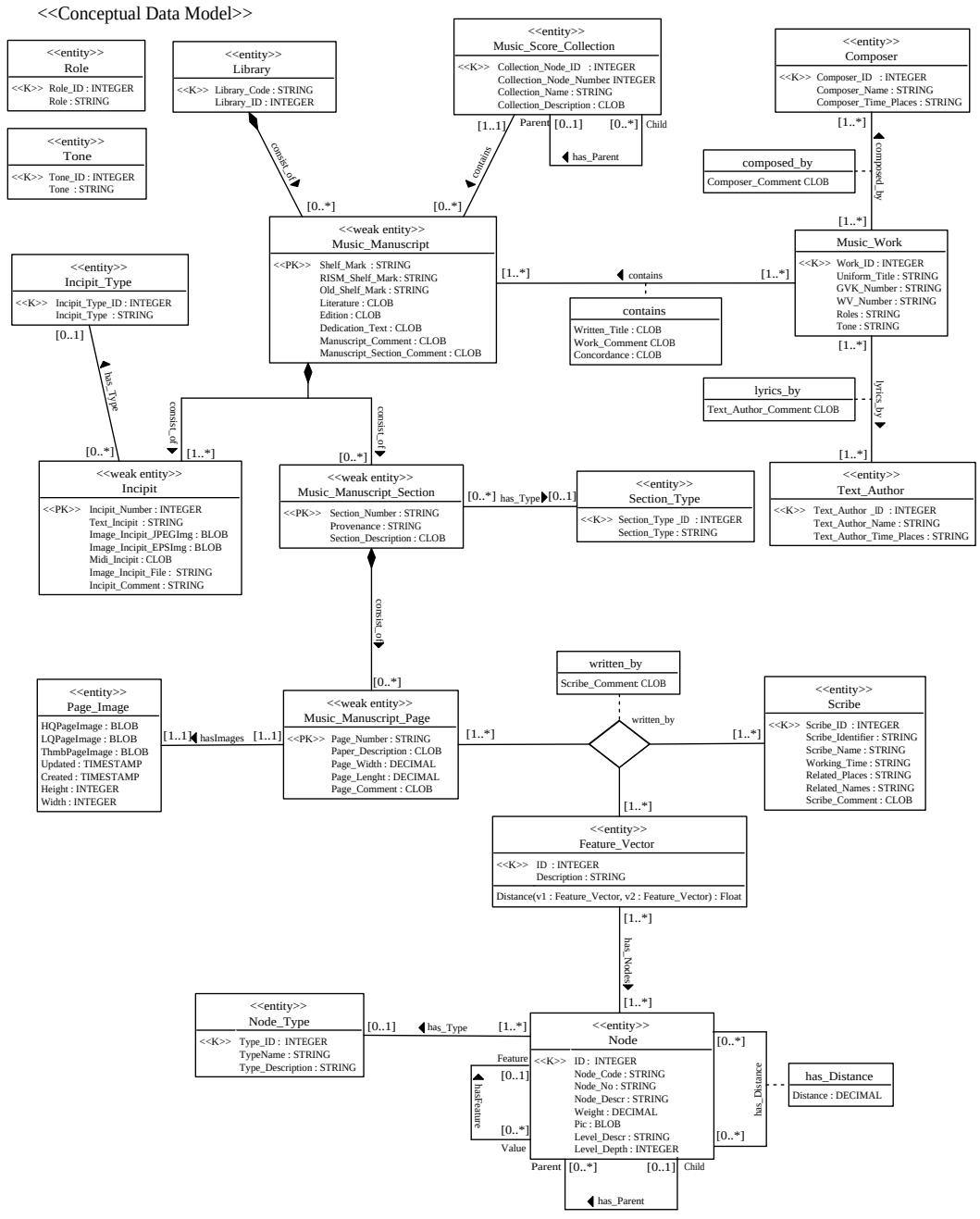


Abbildung C.3: Konzeptuelles Datenmodell des *eNoteHistory*-Projekts

C.3 ICM-Datenmodell des eNoteHistory-Projekts

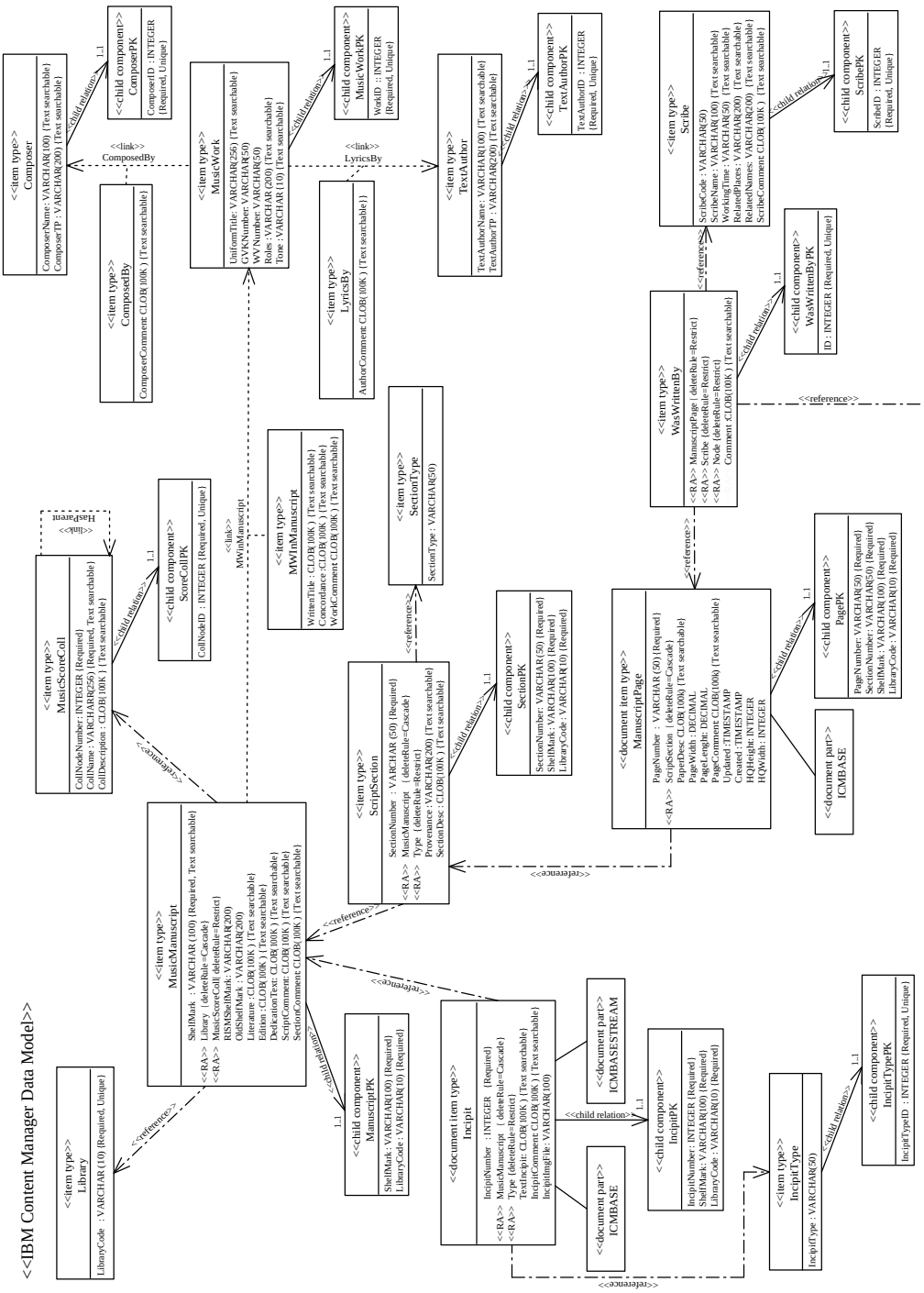


Abbildung C.4: Temporäres ICM-Datenmodell des *eNoteHistory*-Projekts I

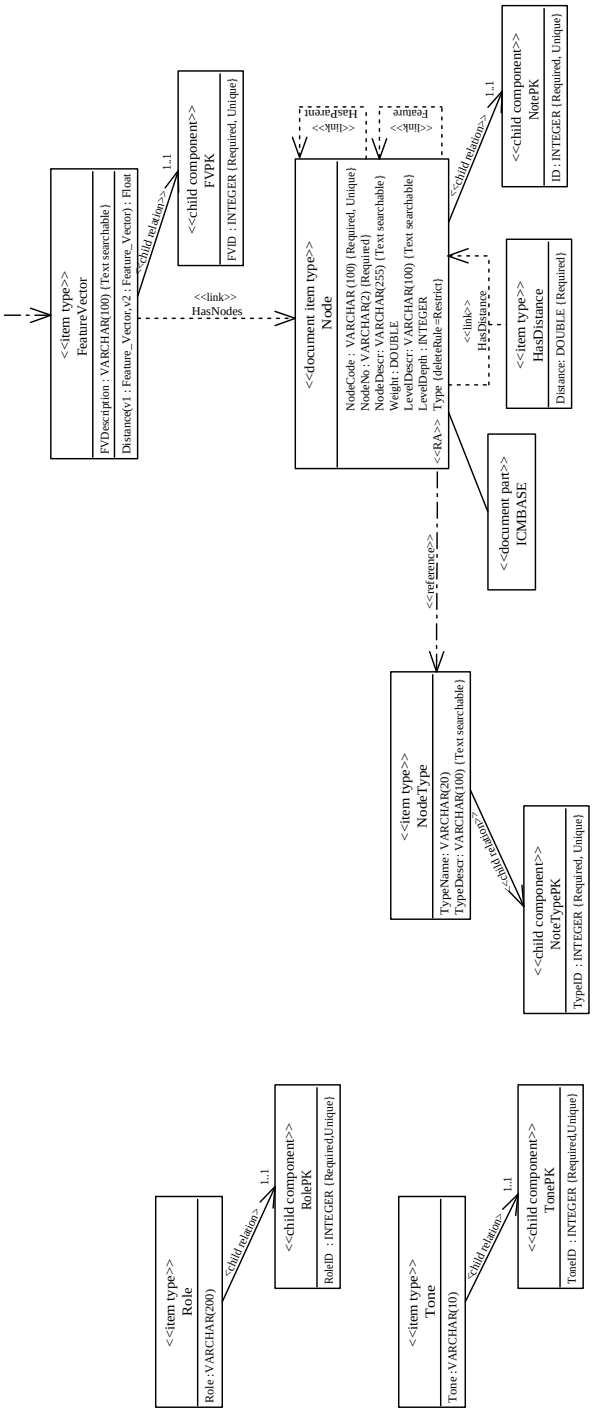


Abbildung C.5: Temporäres ICM-Datenmodell des *eNoteHistory*-Projekts II

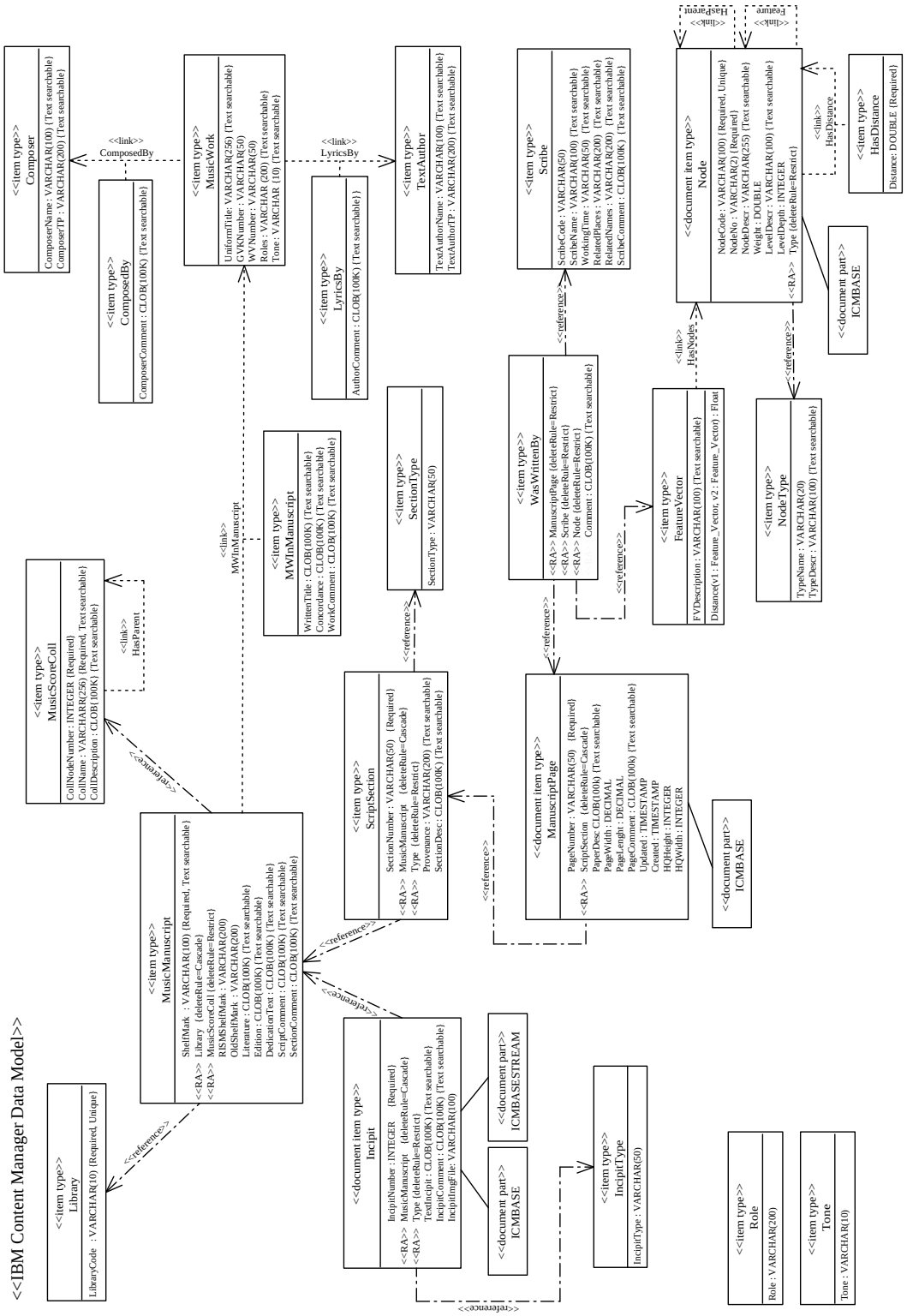


Abbildung C.6: ICM-Datenmodell des *eNoteHistory*-Projekts

Anhang D

XML-Dateien der eNoteHistory-Migration

D.1 XML-Datei zur Übertragung des Datenbankinhalts des eNoteHistory-Dokumentenservers

```
<?xml version="1.0"?>
<!DOCTYPE Mapping SYSTEM "RDBtoICMMapping.dtd">

<Mapping>
  <ItemType name="Library"
            source="METADATA.Libraries"
            propertyType="Item">
    <Attribute name="LibraryCode"
               source="LIBRARY_CODE"/>
  </ItemType>

  <ItemType name="MusicScoreColl"
            source="METADATA.MUSIC_SCORES_COLLECTIONS"
            propertyType="Item">
    <Attribute name="CollNodeNumber"
               source="COLLECTION_NODE_NUMBER"/>
    <Attribute name="CollName"
               source="COLLECTION_NAME"/>
    <Attribute name="CollDescription"
               source="COLLECTION_DESCRIPTION"/>
    <TempPrimaryKey childComponent="ScoreCollPK">
      <Attribute name="CollNodeID"
                 source="COLLECTION_NODE_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="MusicManuscript"
            source="METADATA.MUSIC_MANUSCRIPT"
            propertyType="Item">
    <Attribute name="ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="RISMShelfMark"
```

```
            source="RISM_SHELF_MARK"/>
  <Attribute name="OldShelfMark"
            source="OLD_SHELF_MARK"/>
  <Attribute name="Literature"
            source="LITERATURE"/>
  <Attribute name="Edition"
            source="EDITION"/>
  <Attribute name="DedicationText"
            source="DEDICATION_TEXT"/>
  <Attribute name="ScriptComment"
            source="MANUSCRIPT_COMMENT"/>
  <Attribute name="SectionComment"
            source="MANUSCRIPT_SECTION_COMMENT"/>
  <Attribute name="SectionComment"
            source="MANUSCRIPT_SECTION_COMMENT"/>
  <TempPrimaryKey childComponent="ManuscriptPK">
    <Attribute name="LibraryCode"
              source="LIBRARY_CODE"/>
    <Attribute name="ShelfMark"
              source="SHELF_MARK"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="Incipit"
          source="METADATA.INCIPIT"
          propertyType="Document">
  <Attribute name="IncipitNumber"
            source="INCIPIT_NUMBER"/>
  <Attribute name="TextIncipit"
            source="TEXT_INCIPIT"/>
  <Attribute name="IncipitImgFile"
            source="IMAGE_INCIPIT_FILE"/>
  <Attribute name="IncipitComment"
            source="INCIPIT_COMMENT"/>
  <Part name="ICMBASE"
        source="IMAGE_INCIPIT_JPEGIMG"
        partNumber="1"
        mimeType="image/jpeg"
        semanticType="Base"/>
  <Part name="ICMBASE"
        source="IMAGE_INCIPIT_EPSIMG"
        partNumber="2"
        mimeType="image/eps"
        semanticType="Base"/>
  <Part name="ICMBASESTREAM"
        source="MIDI_INCIPIT"
        partNumber="10"
        mimeType="audio/x-midi"
        semanticType="Base"/>
  <TempPrimaryKey childComponent="IncipitPK">
    <Attribute name="IncipitNumber"
              source="INCIPIT_NUMBER"/>
    <Attribute name="ShelfMark"
              source="SHELF_MARK"/>
    <Attribute name="LibraryCode"
```

```
                source="LIBRARY_CODE"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="IncipitType"
          source="METADATA.INCIPIT_TYPES"
          propertyType="Item">
  <Attribute name="IncipitType"
             source="INCIPIT_TYPE"/>
  <TempPrimaryKey childComponent="IncipitTypePK">
    <Attribute name="IncipitTypeID"
               source="INCIPIT_TYPE_ID"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="ScriptSection"
          source="METADATA.MUSIC_MANUSCRIPT_SECTION"
          propertyType="Item">
  <Attribute name="SectionNumber"
             source="SECTION_NUMBER"/>
  <Attribute name="Provenance"
             source="PROVENANCE"/>
  <Attribute name="SectionDesc"
             source="SECTION_DESCRIPTION"/>
  <TempPrimaryKey childComponent="SectionPK">
    <Attribute name="SectionNumber"
               source="SECTION_NUMBER"/>
    <Attribute name="ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="LibraryCode"
               source="LIBRARY_CODE"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="SectionType"
          source="METADATA.SECTION_TYPES"
          propertyType="Item">
  <Attribute name="SectionType"
             source="SECTION_TYPE"/>
  <TempPrimaryKey childComponent="SectionTypePK">
    <Attribute name="SectionTypeID"
               source="SECTION_TYPE_ID"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="ManuscriptPage"
          source="select * from METADATA.MUSIC_MANUSCRIPT_PAGE a
          full outer join IMAGES.PAGE_IMAGES b
          on a.PAGE_NUMBER=b.PAGE_NUMBER
          and a.SECTION_NUMBER=b.SECTION_NUMBER
          and a.SHELF_MARK=b.SHELF_MARK
          and a.LIBRARY_CODE=b.LIBRARY_CODE" propertyType="Document">
  <Attribute name="PageNumber"
             source="PAGE_NUMBER"/>
  <Attribute name="PaperDesc"
```

```
            source="PAPER_DESCRIPTION"/>
  <Attribute name="PageWidth"
            source="PAGE_WIDTH"/>
  <Attribute name="PageLength"
            source="PAGE_LENGTH"/>
  <Attribute name="PageComment"
            source="PAGE_COMMENT"/>
  <Attribute name="Updated"
            source="UPDATED"/>
  <Attribute name="Created"
            source="CREATED"/>
  <Attribute name="HQHeight"
            source="HEIGHT"/>
  <Attribute name="HQWidth"
            source="WIDTH"/>
  <Part name="ICMBASE"
        source="THMBPAGEIMAGE"
        partNumber="1"
        mimeType="image/jpeg"
        semanticType="Base"/>
  <Part name="ICMBASE"
        source="LQPAGEIMAGE"
        partNumber="2"
        mimeType="image/jpeg"
        semanticType="Base"/>
  <Part name="ICMBASE"
        source="HQPAGEIMAGE"
        partNumber="3"
        mimeType="image/tiff"
        semanticType="Base"/>
  <TempPrimaryKey childComponent="PagePK">
    <Attribute name="PageNumber"
               source="PAGE_NUMBER"/>
    <Attribute name="SectionNumber"
               source="SECTION_NUMBER"/>
    <Attribute name="ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="LibraryCode"
               source="LIBRARY_CODE"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="MusicWork"
          source="METADATA.MUSIC_WORKS"
          propertyType="Item">
  <Attribute name="UniformTitle"
             source="UNIFORM_TITLE"/>
  <Attribute name="GVKNumber"
             source="GVK_NUMBER"/>
  <Attribute name="WVNumber"
             source="WV_NUMBER"/>
  <Attribute name="Roles"
             source="ROLES"/>
  <Attribute name="Tone"
             source="TONE"/>
```

```
    <TempPrimaryKey childComponent="MusicWorkPK">
      <Attribute name="WorkID"
                 source="WORK_ID"/>
    </TempPrimaryKey>
  </ItemType>

 <ItemType name="TextAuthor"
           source="METADATA.TEXT_AUTHORS"
           propertyType="Item">
    <Attribute name="TextAuthorName"
               source="TEXT_AUTHOR_NAME"/>
    <Attribute name="TextAuthorTP"
               source="TEXT_AUTHOR_TIME_PLACES"/>
    <TempPrimaryKey childComponent="TextAuthorPK">
      <Attribute name="TextAuthorID"
                 source="TEXT_AUTHOR_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="Composer"
            source="METADATA.COMPOSERS"
            propertyType="Item">
    <Attribute name="ComposerName"
               source="COMPOSER_NAME"/>
    <Attribute name="ComposerTP"
               source="COMPOSER_TIME_PLACES"/>
    <TempPrimaryKey childComponent="ComposerPK">
      <Attribute name="ComposerID"
                 source="COMPOSER_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="Scribe"
            source="METADATA.SCRIBES"
            propertyType="Item">
    <Attribute name="ScribeCode"
               source="SCRIBE_IDENTIFIER"/>
    <Attribute name="ScribeName"
               source="SCRIBE_NAME"/>
    <Attribute name="WorkingTime"
               source="WORKING_TIME"/>
    <Attribute name="RelatedPlaces"
               source="RELATED_PLACES"/>
    <Attribute name="RelatedNames"
               source="RELATED_NAMES"/>
    <Attribute name="ScribeComment"
               source="SCRIBE_COMMENT"/>
    <TempPrimaryKey childComponent="ScribePK">
      <Attribute name="ScribeID"
                 source="SCRIBE_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="FeatureVector"
            source="FEATURES.FEATUREVECTORS"
```

```
          propertyType="Item">
  <Attribute name="FVDescription"
             source="DESCR"/>
  <TempPrimaryKey childComponent="FVPK">
    <Attribute name="FVID"
               source="ID"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="Node"
          source="DICT.NODES"
          propertyType="Document">
  <Attribute name="NodeCode"
             source="NODE_CODE"/>
  <Attribute name="NodeNo"
             source="NODE_NO"/>
  <Attribute name="NodeDescr"
             source="NODE_DESCR"/>
  <Attribute name="Weight"
             source="WEIGHT"/>
  <Attribute name="LevelDescr"
             source="LEVEL_DESCR"/>
  <Attribute name="LevelDepth"
             source="LEVEL_DEPTH"/>
  <Part name="ICMBASE"
        source="PIC"
        partNumber="1"
        mimeType="image/gif"
        semanticType="Base"/>
  <TempPrimaryKey childComponent="NodePK">
    <Attribute name="ID"
               source="ID"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="NodeType"
          source="DICT.NODE_TYPES"
          propertyType="Item">
  <Attribute name="TypeName"
             source="TYPE_NAME"/>
  <Attribute name="TypeDescr"
             source="TYPE_DESCR"/>
  <TempPrimaryKey childComponent="NodeTypePK">
    <Attribute name="TypeID"
               source="TYPE_ID"/>
  </TempPrimaryKey>
</ItemType>

<ItemType name="Role"
          source="METADATA.ROLES"
          propertyType="Item">
  <Attribute name="Role"
             source="ROLE"/>
  <TempPrimaryKey childComponent="RolePK">
    <Attribute name="RoleID"
```

```
                source="ROLE_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="Tone"
            source="METADATA.TONES"
            propertyType="Item">
    <Attribute name="Tone"
               source="TONE"/>
    <TempPrimaryKey childComponent="TonePK">
      <Attribute name="RoleID"
                 source="ROLE_ID"/>
    </TempPrimaryKey>
  </ItemType>

  <ItemType name="WasWrittenBy"
            source="METADATA.MUSIC_MANUSCRIPT_SCRIBE"
            propertyType="Item">
    <Attribute name="Comment"
               source="SCRIBE_COMMENT"/>
    <TempPrimaryKey childComponent="WasWrittenByPK">
      <Attribute name="ID"
                 source="ID"/>
    </TempPrimaryKey>
  </ItemType>
</Mapping>
```

D.2 XML-Datei zur Rekonstruktion der Beziehungen der eNoteHistory-Datenbank

```
<?xml version="1.0"?>
<!DOCTYPE Relations SYSTEM "RDBtoICMRelations.dtd">

<Relations>
  <Link name="HasParent"
        sourceItemType="MusicScoreColl"
        targetItemType="MusicScoreColl"
        table="METADATA.MUSIC_SCORES_COLLECTIONS">
    <SourceItemPK>
      <Attribute name="ScoreCollPK.CollNodeID"
                 source="COLLECTION_NODE_ID"/>
    </SourceItemPK>
    <TargetItemPK>
      <Attribute name="ScoreCollPK.CollNodeID"
                 source="COLLECTION_PARENT_ID"/>
    </TargetItemPK>
  </Link>

  <Link name="MWInManuscript"
        sourceItemType="MusicManuscript"
        targetItemType="MusicWork"
        table="METADATA.MUSIC_WORKS_IN_MANUSCRIPTS">
```

```
  <SourceItemPK>
    <Attribute name="ManuscriptPK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="ManuscriptPK.LibraryCode"
               source="LIBRARY_CODE"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="MusicWorkPK.WorkID"
               source="WORK_ID"/>
  </TargetItemPK>
  <DescriptionItem itemType="MWInManuscript"
                   propertyType="Item">
    <Attribute source="WRITTEN_TITLE"
               name="WrittenTitle"/>
    <Attribute source="CONCORDANCE"
               name="Concordance"/>
    <Attribute source="WORK_COMMENT"
               name="WorkComment"/>
  </DescriptionItem>
</Link>

<Link name="LyricsBy"
      sourceItemType="MusicWork"
      targetItemType="TextAuthor"
      table="METADATA.MUSIC_WORKS_TEXT_AUTHORS">
  <SourceItemPK>
    <Attribute name="MusicWorkPK.WorkID"
               source="WORK_ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="TextAuthorPK.TextAuthorID"
               source="TEXT_AUTHOR_ID"/>
  </TargetItemPK>
  <DescriptionItem itemType="LyricsBy"
                   propertyType="Item">
    <Attribute name="AuthorComment"
               source="TEXT_AUTHOR_COMMENT"/>
  </DescriptionItem>
</Link>

<Link name="ComposedBy"
      sourceItemType="MusicWork"
      targetItemType="Composer"
      table="METADATA.MUSIC_WORKS_COMPOSERS">
  <SourceItemPK>
    <Attribute name="MusicWorkPK.WorkID"
               source="WORK_ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="ComposerPK.ComposerID"
               source="Composer_ID"/>
  </TargetItemPK>
  <DescriptionItem itemType="ComposedBy"
                   propertyType="Item">
    <Attribute name="ComposerComment"
```

```
                source="COMPOSER_COMMENT"/>
  </DescriptionItem>
</Link>

<Link name="HasNodes"
      sourceItemType="FeatureVector"
      targetItemType="Node"
      table="FEATURES.FVVALUES">
  <SourceItemPK>
    <Attribute name="FVPK.FVID"
                source="FV"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="NodePK.ID"
                source="VALUE"/>
  </TargetItemPK>
</Link>

<Link name="HasDistance"
      sourceItemType="Node"
      targetItemType="Node"
      table="DICT.DISTANCES">
  <SourceItemPK>
    <Attribute name="NodePK.ID"
                source="VALUE1"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="NodePK.ID"
                source="VALUE2"/>
  </TargetItemPK>
  <DescriptionItem itemType="HasDistance"
                   propertyType="Item">
    <Attribute name="Distance"
                source="DISTANCE"/>
  </DescriptionItem>
</Link>

<Link name="HasParent"
      sourceItemType="Node"
      targetItemType="Node"
      table="DICT.NODES">
  <SourceItemPK>
    <Attribute name="NodePK.ID"
                source="ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="NodePK.ID"
                source="PARENT"/>
  </TargetItemPK>
</Link>

<Reference sourceItemType="ScriptSection"
           referenceAttribute="MusicManuscript"
           targetItemType="MusicManuscript"
           table="METADATA.MUSIC_MANUSCRIPT_SECTION">
```

```
    <SourceItemPK>
      <Attribute name="SectionPK.SectionNumber"
                 source="SECTION_NUMBER"/>
      <Attribute name="SectionPK.ShelfMark"
                 source="SHELF_MARK"/>
      <Attribute name="SectionPK.LibraryCode"
                 source="LIBRARY_CODE"/>
    </SourceItemPK>
    <TargetItemPK>
      <Attribute name="ManuscriptPK.ShelfMark"
                 source="SHELF_MARK"/>
      <Attribute name="ManuscriptPK.LibraryCode"
                 source="LIBRARY_CODE"/>
    </TargetItemPK>
  </Reference>

  <Reference sourceItemType="ScriptSection"
             referenceAttribute="Type"
             targetItemType="SectionType"
             table="METADATA.MUSIC_MANUSCRIPT_SECTION">
    <SourceItemPK>
      <Attribute name="SectionPK.SectionNumber"
                 source="SECTION_NUMBER"/>
      <Attribute name="SectionPK.ShelfMark"
                 source="SHELF_MARK"/>
      <Attribute name="SectionPK.LibraryCode"
                 source="LIBRARY_CODE"/>
    </SourceItemPK>
    <TargetItemPK>
      <Attribute name="SectionTypePK.SectionTypeID"
                 source="SECTION_TYPE_ID"/>
    </TargetItemPK>
  </Reference>

  <Reference sourceItemType="MusicManuscript"
             referenceAttribute="Library"
             targetItemType="Library"
             table="METADATA.MUSIC_MANUSCRIPT">
    <SourceItemPK>
      <Attribute name="ManuscriptPK.ShelfMark"
                 source="SHELF_MARK"/>
      <Attribute name="ManuscriptPK.LibraryCode"
                 source="LIBRARY_CODE"/>
    </SourceItemPK>
    <TargetItemPK>
      <Attribute name="LibraryCode"
                 source="LIBRARY_CODE"/>
    </TargetItemPK>
  </Reference>

  <Reference sourceItemType="Incipit"
             referenceAttribute="MusicManuscript"
             targetItemType="MusicManuscript"
             table="METADATA.INCIPIT">
    <SourceItemPK>
```

```
    <Attribute name="IncipitPK.IncipitNumber"
               source="INCIPIT_NUMBER"/>
    <Attribute name="IncipitPK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="IncipitPK.LibraryCode"
               source="LIBRARY_CODE"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="ManuscriptPK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="ManuscriptPK.LibraryCode"
               source="LIBRARY_CODE"/>
  </TargetItemPK>
</Reference>

<Reference sourceItemType="Incipit"
           referenceAttribute="Type"
           targetItemType="IncipitType"
           table="METADATA.INCIPIT">
  <SourceItemPK>
    <Attribute name="IncipitPK.IncipitNumber"
               source="INCIPIT_NUMBER"/>
    <Attribute name="IncipitPK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="IncipitPK.LibraryCode"
               source="LIBRARY_CODE"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="IncipitTypePK.IncipitTypeID"
               source="INCIPIT_TYPE_ID"/>
  </TargetItemPK>
</Reference>

<Reference sourceItemType="ManuscriptPage"
           referenceAttribute="ScriptSection"
           targetItemType="ScriptSection"
           table="METADATA.MUSIC_MANUSCRIPT_PAGE">
  <SourceItemPK>
    <Attribute name="PagePK.PageNumber"
               source="PAGE_NUMBER"/>
    <Attribute name="PagePK.SectionNumber"
               source="SECTION_NUMBER"/>
    <Attribute name="PagePK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="PagePK.LibraryCode"
               source="LIBRARY_CODE"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="SectionPK.SectionNumber"
               source="SECTION_NUMBER"/>
    <Attribute name="SectionPK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="SectionPK.LibraryCode"
               source="LIBRARY_CODE"/>
  </TargetItemPK>
```

```
</Reference>

<Reference sourceItemType="Node"
           referenceAttribute="Type"
           targetItemType="NodeType"
           table="DICT.NODES">
  <SourceItemPK>
    <Attribute name="NodePK.ID"
               source="ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="NodeTypePK.TypeID"
               source="NODE_TYPE"/>
  </TargetItemPK>
</Reference>

<Reference sourceItemType="WasWrittenBy"
           referenceAttribute="ManuscriptPage"
           targetItemType="ManuscriptPage"
           table="METADATA.MUSIC_MANUSCRIPT_SCRIBE">
  <SourceItemPK>
    <Attribute name="WasWrittenByPK.ID"
               source="ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="PagePK.PageNumber"
               source="PAGE_NUMBER"/>
    <Attribute name="PagePK.SectionNumber"
               source="SECTION_NUMBER"/>
    <Attribute name="PagePK.ShelfMark"
               source="SHELF_MARK"/>
    <Attribute name="PagePK.LibraryCode"
               source="LIBRARY_CODE"/>
  </TargetItemPK>
</Reference>

<Reference sourceItemType="WasWrittenBy"
           referenceAttribute="Scribe"
           targetItemType="Scribe"
           table="METADATA.MUSIC_MANUSCRIPT_SCRIBE">
  <SourceItemPK>
    <Attribute name="WasWrittenByPK.ID"
               source="ID"/>
  </SourceItemPK>
  <TargetItemPK>
    <Attribute name="ScribePK.ScribeID"
               source="SCRIBE_ID"/>
  </TargetItemPK>
</Reference>

<Reference sourceItemType="WasWrittenBy"
           referenceAttribute="Node"
           targetItemType="FeatureVector"
           table="FEATURES.MMS_FV">
  <SourceItemPK>
```

```
      <Attribute name="WasWrittenByPK.ID"
                 source="MMS"/>
    </SourceItemPK>
    <TargetItemPK>
      <Attribute name="FVPK.FVID"
                 source="FV"/>
    </TargetItemPK>
  </Reference>
</Relations>
```

Literaturverzeichnis

[Amb03] Scott W. Ambler. *Agile Database Techniques - Effective Strategies for the Agile Software Developer*. John Wiley & Sons, Oktober 2003.

[Amb04] Scott W. Ambler. A UML Profile for Data Modeling. WWW, April 2004. *http://www.agiledata.org/essays/umlDataModelingProfile.html*.

[Beh01] Andreas Behm. *Migrating Relational Databases to Object Technology*. Dissertation, Universität Zürich, Juni 2001.

[Ber04] Ralph Berchtenbreiter. Grundlagen von Content-Management-Systemen und Ansätze ihrer Bedeutung für das CRM. WWW, 2004. *http://www.ku-eichstaett.de/Fakultaeten/WWF/Lehrstuehle/WI/Lehre/ecrm/ Sections/content/Grundlagen von CMS_RB.pdf*.

[Bey03] Dirk Beyer. About the Unified Modelling Language (UML). WWW, 2003. *http://www-sst.informatik.tu-cottbus.de/ db/doc/UML/1ReadMe.html*.

[BLW+97] Jesus Bisbal, Deirdre Lawless, Bing Wu, Jane Grimson, Vincent Wade, Ray Richardson, and Donie O'Sullivan. A Survey of Research into Legacy System Migration, 1997.

[Boo98] Grady Booch. Software Architecture and the UML. WWW, 1998. *http://www-sst.informatik.tu-cottbus.de/ db/doc/UML/uml_architektur _original_rational.pdf*.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[BS03] Edward Bernroider and Volker Stix. Vorlesungsskript - Grundzüge der Modellierung. WWW, 2003. *http://wwwai.wu-wien.ac.at/ bernroid/lehre/gzmod/ws03/*.

[CBS93] R.H.L. Chiang, T.M. Barron, and V.C. Storey. Performance Evaluation of Reverse Engineering Relational Databases into Extended Entity-Relationship Models. In *Proceedings of the Twelfth International Conference on Entity Relationship Approach*, 1993.

[CBS94] R.H.L. Chiang, T.M. Barron, and V.C. Storey. Reverse Engineering of Relational Databases - Extraction of an EER Model from a Relational Database. *Data & Knowledge Engineering, Vol. 12*, pages 107–142, 1994.

[Che76] Peter Pin-Shan Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trancactions on Database Systems*, Band 1(Nr. 1):9–36, 1976.

[Con97] Stefan Conrad. *Föderierte Datenbanksysteme - Konzepte der Datenintegration*. Springer-Verlag, August 1997.

[Cor00] Rational Software Corporation. The UML and Data Modeling -A Rational Software Whitepaper-. WWW, 2000.
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Tp180.PDF.

[Cor03a] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - Client for Windows Programming Reference, Version 8 Release 2. WWW, März 2003.
http://www-306.ibm.com/software/data/cm/cmgr/mp/library.html.

[Cor03b] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - Managing Information Integrator for Content. WWW, Oktober 2003.
http://www-306.ibm.com/software/data/eip/library.html.

[Cor03c] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - Planning and Installing Your Content Management System, Version 8 Release 2. WWW, Oktober 2003.
http://www-306.ibm.com/software/data/cm/cmgr/mp/library.html.

[Cor03d] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - Planning and Installing Information Integrator for Content. WWW, März 2003.
http://www-306.ibm.com/software/data/eip/library.html.

[Cor03e] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - System Administration Guide, Version 8 Release 2. WWW, Oktober 2003.
http://www-306.ibm.com/software/data/cm/cmgr/mp/library.html.

[Cor03f] IBM Corporation. IBM DB2 Content Manager for Multiplatforms - Workstation Application Programming Guide, Version 8 Release 2. WWW, Oktober 2003.
http://www-306.ibm.com/software/data/cm/cmgr/mp/library.html.

[Cor03g] IBM Corporation. IBM DB2 Content Manager for Multiplatforms/IBM Information Integrator for Content - Installing, Configuring, and Managing the eClient, Version 8 Release 2. WWW, 2003.
http://www-306.ibm.com/software/data/cm/cmgr/mp/library.html.

[Cor03h] IBM Corporation. IBM DB2 Content Manager, Version 8.2 - Brochure. WWW, 2003.
http://www-306.ibm.com/software/data/cm/attach/cmv8.pdf.

[Cor03i] Rational Software Corporation. Using Rose Data Modeler. Rational Rose 2003 Documentation, 2003.
http://www.se.fh-heilbronn.de/usefulstuff/Rational Rose 2003 Documentation/rose_dm.pdf.

[Cor04a] IBM Corporation. Documentation of the SReferenceAttrDefCreationICM.java Sample, Mai 2004.

[Cor04b] IBM Corporation. IBM DB2 Universal Database Version 8.2 - Application Development Guide: Programming Server Applications. WWW, 2004.
http://www-306.ibm.com/software/data/db2/udb/support/manualsv8.html.

[dJS99] Lurdes Pedro de Jesus and Pedro Souca. Selection of Reverse Engineering Methods for Relational Databases. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, 1999.

[Dol03] Sebastian Dolke. Einsatz von ER zum Entwurf von DTDs bzw. XML-Schema, Informatikseminar (Universität Rostock). WWW, Mai 2003.
http://wwwdb.informatik.uni-rostock.de/Lehre/Vorlesungen/hs_ws2002_2003/hs_dolke.ps.

[Dum01] Reiner R. Dumke. UML - Tutorial. WWW, März 2001.
http://www-sst.informatik.tu-cottbus.de/ db/doc/UML/Dumke-UML/inhalt.htm.

[eP04] eNoteHistory Projektgruppe. Homepage des eNoteHistory Projekts. WWW, 2004.
http://www.eNoteHistory.de.

[Gil03] Frank Gilbane. Information Integration, Objects, Content Services & Infrastructures. *The Gilbane Report*, 11(1), März 2003.
http://www.gilbane.com/gilbane_report.pl/85/Information_Integration_Objects_Content_Services_amp_Infrastructures.html.

[Gli99] Martin Glinz. UML (Unified Modeling Language) im Überblick. WWW, 1999.
http://www.alumni.ch/Texte/Glinz_UML_Vortrag.pdf.

[Gor03a] Davor Gornik. Entity Relationship Modeling with UML. WWW, November 2003.
http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf.

[Gor03b] Davor Gornik. Relational Modeling with UML. WWW, November 2003.
http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2786/2786_modeling.pdf.

[Gor03c] Davor Gornik. UML Data Modeling Profile. WWW, 2003.
http://www.jeckle.de/files/RationalUML-RDB-Profile.pdf.

[GVK+03] Roland Göcke, Jörg Voskamp, Ekkehard Krüger, Tobias Schwinger, Ilvio Bruder, Andreas Finger, Andreas Heuer, and Temenushka Ignatova. Ein System zur Identifikation der Schreiber von historischen Musikhandschriften. WWW, 2003.
http://www.enotehistory.de/fhg/rgoeckeIuKTage2003.pdf.

[HS99] Andreas Heuer and Gunter Saake. *Datenbanken: Implementierungstechniken*. MITP-Verlag GmbH, 1st edition, Januar 1999.

[HS00] Andreas Heuer and Gunter Saake. *Datenbanken - Konzepte und Sprachen*. International Thomson Publishing, 2nd edition, Januar 2000.

[JMP01] Mikael R. Jensen, Thomas H. Mller, and Torben Bach Pedersen. Converting XML Data To UML Diagrams For Conceptual Data Integration. WWW, 2001.
http://www.cs.auc.dk/ mrj/publications/diweb.pdf.

[JMP03] Mikael R. Jensen, Thomas H. Mller, and Torben Bach Pedersen. FacilitatingWeb-Based OLAP By Converting XML Data To UML Diagrams. WWW, 2003.
http://www.cs.auc.dk/ mrj/publications/DKE_XML.pdf.

[Kam03] Dr. Ulrich Kampffmeyer. Enterprise Content Management - Zwischen Vision und Realität. WWW, April 2003.
http://www.contentmanager.de/magazin/artikel_398_ecm_zwischen_vision_und_realitaet.html.

[Kip96] Christina Kipp. *Migration einer relationalen Datenbank in eine objektorientierte Datenbank*. Diplomarbeit, Universität Osnabrück, Oktober 1996.

[Kir02] Michael Kirschning. *Content Management Systeme*. Projektarbeit, Fachhochschule Kiel, Juli 2002.

[Kr"03] Ekkehard Krüger. *Die Musikaliensammlungen des Erbprinzen Friedrich Ludwig von Württemberg-Stuttgart und der Herzogin Louisa Friderica von Mecklenburg-Schwerin in der Universitätsbibliothek Rostock*. Diplomarbeit, Universität Rostock, 2003.

[Mat03] Nelson Mendonca. Mattos. Integrating Information for On Demand Computing. In *Proceedings of the 29th International Conference on Very Large Databases*, 2003.

[Mil04] Lars Milewski. *Integration von Clustering-/Classification-Techniken in eine objektrelationale Datenbankumgebung*. Diplomarbeit, Universität Rostock, März 2004.

[NM01] Eric J. Naiburg and Robert A. Maksimchuk. *UML for Database Design*. Addison Wesley Publishing, July 2001.

[OMG03] OMG. OMG Unified Modeling Language Specification, Version 1.5. WWW, März 2003.
http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf.

[RDZ02] John Rodriguez, Bryan Daniel, and Lijing Zhang. *IBM Content Manager for Multiplatforms Version 8.1 - System Administration Certification Study Guide*. IBM PartnerWorld for Developers, 1st edition, Oktober 2002.

[Rol96] Stefan Rollert. *Datenbank-Entwurf durch "Reverse Engineering" von relationalen Datenbanken*. Diplomarbeit, Universität Hannover, Februar 1996.
http://www.dbs.uni-hannover.de/ftp/theses/rollert/doku_komplett.pdf.

[Sch03] Dr. Arno Schmidhauser. Skript Datenmodellierung mit UML. WWW, Dezember 2003.
http://www.sws.bfh.ch/ schmd/db.

[Sim00] Frank Simon. Einführung in UML. WWW, Mai 2000.
http://www-sst.informatik.tu-cottbus.de/ db/doc/UML/UML_Slideshow _Frank.pdf.

[SL90] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), September 1990.

[Spa01] Geoffrey Sparks. Database Modelling in UML. WWW, 2001.
http://www.sparxsystems.com.au/bin/DatabaseModelingInUML.pdf.

[Wah98] Günter Wahl. UML kompakt. *OBJEKTspektrum 2/98*, pages 22–23, Februar 1998.
http://www-sst.informatik.tu-cottbus.de/ db/doc/UML/UML-Kompakt /UML-Kompakt.htm.

[Was02] Götz Waschk. *Integration von Inhalten aus Content-Managementsystemen in lokalen Suchmaschinen*. Diplomarbeit, Universität Rostock, März 2002.

[Wer01] Daniela Wersin. *Evaluation and Comparison of Content Management Systems*. Bachelor thesis, University of Rostock, August 2001.

[WO04] Tim Weilkiens and Bernd Oestereich. *UML 2 - Zertifizierung* . dpunkt.verlag, April 2004.

[Wut03] Daniel Wutke. Content Management und Digitale Bibliotheken. WWW, 2003.
http://www.informatik.uni-stuttgart.de/ipvr/as/lehre/hauptseminar/docws03/ Contentmanagement_Handout.pdf.

[ZTZ04] Oliver Zschau, Dennis Traub, and Rik Zahradka. Klassifizierung von Content Management Lösungen. WWW, April 2004.
http://www.contentmanager.de/magazin/artikel_37_klassifizierung_von_content _management_loesungen.html.

Abbildungsverzeichnis

Tabellenverzeichnis

Selbständigkeitserklärung

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

Neubrandenburg, den 29.10.2004

_____

Sebastian Dolke

Thesen

1. Für die Integration von Dokumentenarchiven in eine *IBM Content Manager*-Umgebung können zwei Verfahren verwendet werden: Zum Einen die Föderierung von Archiven mit dem *IBM DB2 Information Integrator for Content*, und zum Anderen die Migration in den *IBM DB2 Content Manager for Multiplatforms*

2. Eine Archivföderation ist vor allem dann sinnvoll, wenn Dokumentenarchive mit ähnlichem oder gleichem Inhalt auf mehrere Standorte verteilt sind und der Datenbestand weiterhin lokal verwaltet werden soll.

3. Eine Archivmigration gewährleistet eine zentrale Administration verschiedener Dokumentenarchive. Darüberhinaus bieten die speziellen Dokumentenverwaltungsfunktionen des *IBM DB2 Content Manager for Multiplatforms* Vorteile gegenüber konventionellen Datenverwaltungssystemen wie z.B. relationalen Datenbanken.

4. Für die Migration von Datenstrukturen in den *IBM DB2 Content Manager for Multiplatforms* sind konzeptuelle Datenmodelle gut geeignet. Die erzeugten Datenstrukturen sind qualitativ besser als solche, die durch eine direkte Abbildung eines Implementierungsdatenmodells entstehen.

5. Konzeptuelle Datenmodelle können automatisch auf das ICM-Datenmodell abgebildet werden.

6. Für die Migration von Methoden in den *IBM DB2 Content Manager for Multiplatforms* kann die Funktionalität der zu Grunde liegenden DB2-Datenbank genutzt werden, indem Methoden als Stored Procedures oder UDFs in die Library-Server-Datenbank integriert werden. Darüberhinaus können Methoden Client-seitig integriert werden.

7. Die Unified Modeling Language ist geeignet, um relationale Datenmodelle, konzeptuelle Datenmodelle und ICM-Datenmodelle darzustellen.