UNIVERSITÄT GHS ESSEN

Diplomarbeit DI

Automatische Generierung von Datenbankformularen einer relationalen Datenbank

Vorgelegt dem Fachbereich Wirtschaftswissenschaften
der Universität Gesamthochschule Essen

von:	Artur Trzewik
Adresse:	...
	...
Matrikel-Nr.:	...

Gutachter:	Prof. Dr. Günther Pernul
Zweitgutachter:	Prof. Dr. Stefan Eicker

Laufendes Semester:	Wintersemester 2000/01,
Voraussichtlicher Studienabschluss:	Wintersemester 2000/01

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CGI Common Gateway Interface

DBMS Date Base Manager System

DB Datenbank

DDL Data Manipulations Language

DOM Document Object Modell

EER-Modell Extended Entity-Relationship Modell

ER-Modell Entity-Relationship Modell

GUI Graphical User Interface, graphische Benutzerschnittstelle

HTML HyperText Markup Language

NF2 Non-Firstform Relational Datamodel

ODBC Open Database Connectivity

RDBMS Relational Date Base Manager System

SQL Structured Query Language

TCL Tool Command Language

XML Extensible Markup Language

1 Einführung

Das Ziel der Diplomarbeit ist es eine Datenbank-Applikation zu entwickeln, die eine automatische Generierung von Datenbank-Formularen zu einer relationalen Datenbank erlaubt. Diese Applikation muss zuerst das Schema der Datenbank (aus Data Directory) analysieren und daraus entsprechend eine Sammlung von Formularen generieren. Diese Formulare können dann manuell mit Hilfe eines graphischen Editors angepasst werden. Schließlich werden die Formulare dargestellt und die Applikation muss die Operationen auf den Formularen richtig in die Data Manipulation Language der Datenbank (hier SQL) umsetzten.

Relationale Datenbanken sind de facto ein Standard der Datenbank-Management-Systeme. Ihr Erfolg ist unter anderem auf das einfache Konzept der Repräsentation der Information in Tabellenform zurückzuführen. Leider erlaubt das relationale Modell nicht, die erweiterte Semantik der Information und höhere Abstraktionskonzepte im Schema selbst zu speichern. Die direkte Umsetzung von Tabellen zu Formularen, die bei vielen ähnlichen DB-Werkzeugen praktiziert wird, ist oft unzulänglich. Durch den Normalisierungsprozess werden die zusammenhängenden Informationen auf mehrere Tabellen zersplittert. Zweitens unterstützt das relationale Modell keine Beziehungstypen, so dass sie nur durch die Werte der Fremdschlüssel ausgedrückt werden können. Die so entstandenen Formulare sind für den Endbenutzer fast unbrauchbar, weil für ihn die zusammenhängenden Informationen auf mehre Formulare verstreut sind, und er genau das Schema der Datenbank kennen muss, um damit auch einfache Operationen durchführen zu können.

In der Diplomarbeit wird die Möglichkeit theoretisch erörtert und praktisch umgesetzt, in den Formularen höhere Abstraktionskonzepte wie Vererbung, Aggregation und Assoziation zu verwenden. Hier wird der Grundsatz, dass die Darstellung der Information ihrer Semantik entsprechen soll, umgesetzt. Dabei wird ein Weg zu fertigen Formularen in zwei Schritten gewählt. Zuerst wird ein erweitertes Schema der Datenbank mit Hilfe von *Reverse Engineering* Techniken geschaffen. Dieses erweiterte Schema enthält neben dem relationalen Schema die weiteren semantische Informationen, wie die Relationen interpretiert werden müssen. Für den Endbenutzer liegt hier der Hauptvorteil darin, dass er ein Formular als ein Objekt und nicht die Entsprechung einer Tabelle sehen kann. Alle relevanten Informationen zu einem Objekt sollen aus einem Formular durch Verweise erreichbar sein, ohne das das Datenbank-Schema bekannt sein müsste.

Die Ausarbeitung des Themas wird von ein paar Überlegungen begleitet, die hier in Form von Thesen vorgestellt werden und im nachhinein Vertiefung finden.

- Die Entwicklung von Datenbank-Applikationen stellt besondere Anforderungen an die graphischen Benutzerschnittstellen.
- Die Darstellung der Information in den DB-Formularen sollte ihrer Semantik entsprechen.
- Man kann mit Hilfe des relationalen Modells komplexe Strukturen abbilden aber die Information, wie die Relationen interpretiert werden sollen, wird im Schema nicht festgehalten.
- Das relationale Schema enthält zu wenig semantische Informationen um aus ihm gute Formulare generieren zu können.
- Es ist sinnvoll eine Erweiterung des relationalen Modells zu schaffen, die die semantischen Ausdruckmöglichkeiten der höheren Modellierungskonzepte erhält. Solche Erweiterung kann eine Grundlage für die einfache Erstellung von benutzerfreundlichen DB-Applikationen bilden.

Die Gliederung der Diplomarbeit geht von dem Besonderen (DB-Applikationen und Formulare) zum Allgemeinen (Datenbankmodelle) und mündet in der Implementierung einer Applikation, die wiederum ein besonderer Aspekt der Problematik ist. Zuerst werden die Datenbank-Applikationen aus verschiedenen Gesichtspunkten betrachtet (Kapitel 2). Besonders werden die Techniken der Erstellung solcher Applikationen und die Anforderungen an die graphischen Oberflächen abgehandelt. Auch die Datenbank-Formulare allgemein werden theoretisch vorgestellt.

Danach werden die verschiedenen Konzepte der Datenbank-Modelle analysiert und systematisiert (Kapitel 3). Das ER-Modell, seine Erweiterungen und seine Abbildung in relationales Modell spielt hier die größte Rolle. Zusätzlich werden die Konzepte der semantischen und objektorientierten Modellen behandelt. Dabei wird gleich auf die Umsetzungsmöglichkeiten und Bedeutung dieser Konzepte für DB-Formulare und andere DB-Applikationen hingewiesen. Es wird eine Typologie von verschiedenen Meta-Informationen der Daten skizziert. Ein weiterer Teil der Diplomarbeit beschäftigt sich mit der SQL-Sprache als Data Definition Language.

Der praktische Teil der Diplomarbeit sollte die theoretischen Überlegungen konsequent umsetzten. Zuerst wird der Vorschlag eines Systems der DB-Applikationen entworfen. Ein Teil dieses Systems, der DB-Formular-Generator wird implementiert. Die Implementierung selbst wird nur teilweise dokumentiert. Besonders wird auf die Anforderungsanalyse und den Entwurf eingegangen.

Als Entwicklungsplattform dient das Betriebssystem Linux. Alle bei der Diplomarbeit benutzten Komponenten und Programmierwerkzeuge sind frei und mit Quelltexten verfügbar. Als Test-Datenbank dient eine freie relationale Datenbank mySql. Die Applikation wird in der Skriptsprache Tcl/Tk implementiert. Es werden weitere Erweiterungen der Tcl benutzt wie: Tix (Erweiterte Sammlung von GUI-Objekten), tDom (DOM Implementierung) und XOTcl (an der Uni Essen entwickelte objektorientierte Erweiterung des Tcl). Ein besonderes Augenmerk der Diplomarbeit ist es diese Komponenten vorzustellen und sie auf ihre Eignung für die Zwecke der Entwicklung zu prüfen. Ein aktueller Aspekt der Implementierung ist die Anwendung des XML-Standard und der Programmierschnittstelle DOM (Document Object Modell).

2 DB-Applikationen

Datenbank-Applikationen bestimmen in höchstem Maße wie viel Nutzen man aus den Daten ziehen kann, die in Datenbanken gesammelt sind. In der Zeiten von Anhäufung von riesigen Datenmengen, ist es besonders wichtig die Daten menschengerecht zu präsentieren. Dieses Kapitel führt in die Thematik der Datenbank-Applikationen ein. Zuerst wird eine Systematik solcher Applikationen erstellt. Die Software-Techniken der Erstellung der Applikationen finden ein eigenes Unterkapitel. Nachher wird auf die besonderen Anforderungen und Funktionen der grafischen Benutzerschnittstellen der DB-Applikationen eingegangen. Zuletzt werden die Datenbank-Formulare als spezielle Datenbank-Applikationen genauer betrachtet.

2.1 Anordnung und Systematisierung

Datenbank-Applikationen sind als eine Schnittstelle zwischen dem Benutzer und dem Datenbank-Manager-System (DBMS) definiert. Ihre Hauptaufgabe versteht David M. Kroenke in seinem Buch folgendes:

> The database application´s first task is to create, update, delete and display objects. [...] Before objects can be processed, therefore, the application must construct them from the underlying relation or as we somtimes say, the application must materialize objects. ([Kron97], S. 190)

Der Schwerpunkt liegt hier zuerst in der Schaffung eines benutzerfreundlichen Ersatzes für SQL-Sprache, möglichst mit intuitiver graphischer Benutzerschnittstelle. Die

Funktionen decken sich mit SQL-Mächtigkeit (create, update, delete and display). Der zweite Punkt betrifft die Behandlung der Daten auf einer höheren Abstraktionsebene als es von den Datenbanksystem möglich wäre. Der Autor schlägt auch eine Aufteilung der DB-Applikationen hinsichtlich des Grades der Materialisierung der Objekten vor (s. Abb. 2.1).

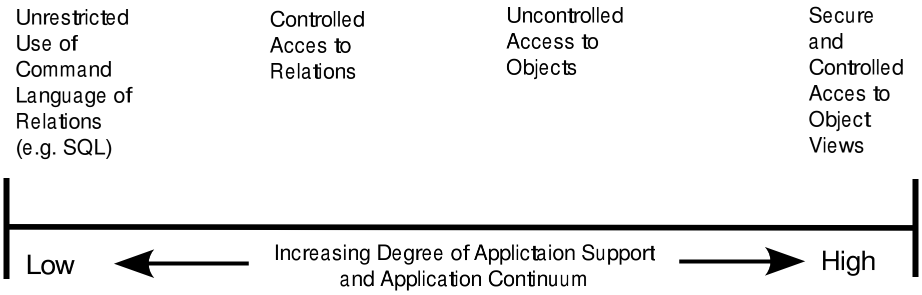

Abbildung 2.1: Aufteilung der DB-Applikationen (Quelle [Kron97])

Die Aufteilung geht von universellen Applikationen aus, die einfach die Datenstrukturen oder Dateninhalte direkt auf die Benutzer-Schnittstelle umsetzen. Dazu gehören z.B die graphischen Front-Ends zu SQL oder Abbildungen der Tabellen. Solche Datenbank-Applikationen gehören inzwischen zu fast jedem DBMS als ergänzende Werkzeuge. Am anderen Ende der Skala befinden sich die speziellen Anwendungen, die oft individuell zur Datenbeschaffenheit angefertigt werden. Nicht zu vergessen ist die Tatsache, dass die Datenbanken nur ein Teil der Informations-Systeme sind und der Übergang von Datenbank-Applikationen zu Applikationen überhaupt flüssig ist.

Oft ergänzen die Datenbank-Applikationen die Eigenschaften der Datenbanksysteme um zusätzliche Funktionalitäten wie Sicherheitsüberprüfung oder weitere Integritätsbedingungen, die vom DBMS nicht erledigt werden können. Die Entwicklung von DBMS in den letzten Jahren zeigt, dass sie immer mehr komplexere Aufgaben selbst erledigen können. Die Trigger Mechanismen bei relationalen Datenbank wie Oracle erlauben sogar die Programmierung der komplexen Logik in eigener Sprache zusätzlich zu SQL. Solche Datenbanken werden auch als aktive Datenbanken bezeichnet. Die objektorientierten Datenbanken erlauben auch die Modellierung von dynamischen Eigenschaften der Daten wie Operationen. Der Trend geht zu Datenbanksystemen, die nicht nur Daten speichern sondern auch Daten verarbeiten können. Durch die Verschiebung der Logik von Applikationen auf Datenbankmanager erzielt man zuerst einen Kostenvorteil durch wenig aufwendige Programmierung der Applikationen, zweitens entgeht man so der Gefahr, dass schlecht programmierte Applikationen die Dateninhalte vernichten.

Bei den namhaften Datenbankmanagersystemen sind die zusätzlichen Datenbank-Applikationen ein wichtiger Zusatz, der oft über den Erfolg des Produktes entscheidet. Dazu gehören:

- graphischer Abfrage-Editor (oft angelehnt an QBE Query by Example)
- Formular-Editor und Formulargenerator
- Berichtsgenerator
- Charts Generator
- 4.Generation Sprachen für die schnelle Entwicklung von DB-Applikationen

Bei der Wahl eines konkreten DBMS werden die Werkzeuge oft höher bewertet als die Leistungsfähigkeit oder Vollständigkeit des DBMS allein. Es erklärt die Popularität der Produkte wie MS Access, das zwar den pseudo RDBMS zuzuordnen ist (siehe [HeSa95], S. 40), aber reich an zusätzlichen Anwendungen ist und gut in andere hauseigene Produkte integriert ist. Es gibt auch weitere DB-Applikationen, die jedoch bis jetzt auf Grund ihres Preises nicht mit den DBMS angeboten werden. Dazu gehören komplexe statistische Auswertungsprogramme, Data-Mining Werkzeuge, graphische Visualisierungs-Werkzeuge und Data Warehouses.

Eine andere Aufteilung der Datenbank-Applikationen kann man nach der Art ihrer Benutzer vornehmen. Die Datenbankbenutzer unterteilt man oft in drei Gruppen, denen man auch bestimmte Arten von Datenbank-Applikationen zuordnen kann (siehe Tabelle 2.1). Jede Benutzergruppe hat eine eigene Spezifik, die bei der Entwicklung berücksichtigt werden muss.

Es ist auch eine Aufteilung denkbar, nach der Gewichtung von Datenstrukturen auf Dateninhalte (Abb. 2.2). Der parametrische Benutzer ist eigentlich nur auf

Benutzertypen	DB-Applikationen
gelegentlicher Benutzer (wie Manager)	Berichtsgenerator, Chartsgenerator, Datenvisualisierungsanwendungen
parametrischer Benutzer (wie Sachbearbeiter)	DB-Formulare, spezial Anwendungen für Datenerfassung
"sophiscated"Benutzer (wie Systemanalytiker, Programmierer, Wissenschaftler)	Datenbank Modellierungswerkzeuge, Anwendungsgeneratoren, Data Mining

Tabelle 2.1: DB-Applikationen und Benutzergruppen

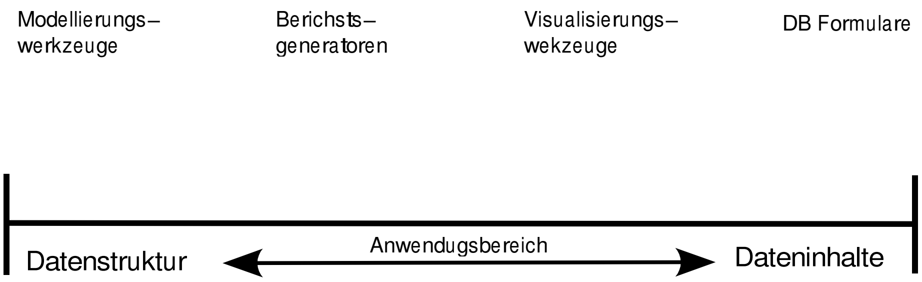

Abbildung 2.2: Aufteilung der DB-Applikationen nach Anwendungsbereich

Dateninhalte konzentriert. Der Datenbankdesigner beschäftigt sich fast nur mit Datenstruktur.

2.2 Techniken der Entwicklung

Die Entwicklung von Datenbank-Applikationen erfolgt zusammen mit der Entwicklung der Datenbank. Oft ist es sogar sinnvoll mit dem Entwurf von nötigen Applikationen zu beginnen und daraus auf die Datenbank-Struktur zu schließen. Das resultiert aus der Erfahrung, dass bei der Bedarfsanalyse die Endbenutzer besser beschreiben können was mit den Daten geschehen soll (als Funktionen und Prozesse), als der Struktur der Daten bewusst zu werden. Vorallem haben die Benutzer nur eine begrenzte Kenntnis über die Struktur der Daten (micro-level view).

Die Entwicklung von eigenen Applikationen ist sehr teuer. Man kann oft nicht auf die Standardapplikationen zurückgreifen, weil sie unzulänglich auf die Besonderheiten der Daten anzupassen sind, oder einfach die speziellen Funktionen für die Datenverarbeitung nicht anbieten. Es haben sich drei Techniken der Erstellung der

DB-Applikationen etabliert:

1. Entwicklung von Applikationen nur mit Hilfe von primären Schnittstellen zu Datenbanken wie ODBC (auf Programmiersprache-Ebene). Der Programmierer bekommt keine Unterstützung in der Bildung von datenbankorientierten GUI. Es können allgemeine GUI-Builder benutzt werden.

2. Die Verwendung von datenbankorientierten Frameworks. Die Programmlogik ist bereits vorgegeben. Es müssen nur noch datenbankspezifische Teile ergänzt werden (Die Datenstruktur wird fest programmiert). Solche Frameworks werden z.B von Borland-Delphi (zu Datenbank Paradox) oder Visual C++ angeboten. Auch die GUI-Builder, die bereits spezielle datenbankspezifische GUI-Elemente wie Tabellen enthalten, werden benutzt.

3. Benutzung von DB-Applikations-Generatoren. Meistens werden die Generatoren für die Erstellung von Formularen benutzt, dabei werden die Formulare zuerst automatisch generiert und dann manuell angepasst. Programmierkenntnisse sind für einfache Aufgaben nicht nötig. Spezielle Logik kann mit Hilfe von 4. Generation Skriptsprachen oder eigenen Programmbibliotheken programmiert werden. Solche Umgebungen repräsentieren MS-Access, Lotus-Approach. Manche große Datenbanksysteme (wie dBase, Oracle) bieten solche Entwicklungsumgebungen.

Diese Techniken unterscheiden sich vor allem in den Anforderungen an die Entwickler und der Flexibilität, die sie anbieten. Die erste Methode ist die aufwendigste und stellt die größten Anforderungen an die Entwickler, aber sie ermöglicht die Erstellung von speziellen, maßgeschneiderten und schlanken Applikationen. Die letzte Technik wiederum verlangt keine Programmierkenntnisse, die Applikationen können oft per Drag&Drop zusammengestellt werden. Die so erzeugten Applikationen benötigen eine große Laufzeitumgebung, was sich auf die Effizienz und die Stabilität negativ auswirken kann. Im Falle von speziellen Funktionen ist die Benutzung von Programmiersprachen nicht abwendbar. Hier ist man auf die Skriptsprache des Herstellers angewiesen. Aus diesen Gründen werden die Applikations-Generatoren oft nur für das Prototyping benutzt.

Das Problem, das bis jetzt nur unzureichend gelöst wurde, ist die automatische oder einfache Übertragung von evolutionären Schema-Änderungen der Daten auf die Applikationen. Zwar bieten die modernen DBMS, die Möglichkeit, das Schema nachträglich zu ändern oder zu erweitern, aber ein solches Vorgehen würde die Anpassung bei allen DB-Applikationen bedeuten, bei denen die Datenstruktur fest programmiert ist. Oft sind die Kosten für solche nachträglichen Anpassungen gleich hoch wie die primären Entwicklungskosten (bei unvollständiger Dokumentation oder bei Ausscheiden der Programmierer).

Während der gelegentliche Benutzer oder der „sophiscated" Benutzer mit Standard-DB-Applikationen oft ausreichend bedient sind, müssen für den parametrisierten Benutzer im Regelfall spezielle Applikationen aufwendig entwickelt werden. Die Computerkenntnisse der parametrisierten Benutzer sind sehr niedrig, desto aufwendiger und individueller muss die graphische Schnittstelle (GUI) der Applikation entworfen werden. Der andere Grund für die individuelle Entwicklung der Applikationen ist, dass in der Datenbank nur die statische Sicht der Daten modelliert ist[1]. Daten die vielleicht in komplexen Prozessen verarbeitet werden. Die Umsetzung der wirtschaftlichen Prozesse auf die elementaren Operationen der Datenbank-Objekte ist für den Benutzer oft nicht nachvollziehbar. Die Applikationen besitzen deswegen eine prozessorientierte Schnittstelle wie: Auftrag erfassen, Rechnung ausstellen oder Student immatrikulieren. Bei solchen Prozessen werden mehrere DB-Objekte involviert, die Reihenfolge und Logik der Operationen ist vorgegeben.

Die Softwarearchitektur der DB-Applikationen weist drei Komponenten auf: graphische Benutzerschnittstelle, Programmlogik und DB-Schnittstelle. In der Programmlogik-Komponente kann man applikationsspezifische und datenspezifische (Domain specific logic) Komponenten vorfinden. (Vergleiche [PaCh93], S. 36). Zur applikationsspezifischen Logik gehören: die Prozeduren für Ausgabe, Formatierung, Zeichnungsprozeduren, Prozeduren für autorisierten Datenbank Zugang. Die datenspezifische Logik umfasst z.B die speziellen Operationen auf Daten, Berechnung von abgeleiteten Attributen, zusätzliche Integritätsüberprüfung. Auch bei der Benutzerschnittstelle kann man applikationsspezifische und datenspezifische Elemente vorfin-

[1] Ausgenommen der objektorientierten Datenbanksystemen

den. Das Anteil der Elementen hängt von der Art der Datenbank-Applikationen ab. Bei einen graphischen Abfrage-Editor gibt es keine datenspezifische Elemente. Immer dann wenn das Anteil von datenspezifischen Elementen klein ist, ist der Einsatz von Standard-Applikationen sinnvoll.

2.3 Anforderungen und Besonderheiten der graphischen Benutzerschnittstellen

In den letzten 15 Jahren sind die graphischen Benutzerschnittstellen (GUI) zum Standard für Mensch-Computer Kommunikation geworden. Vor allen das Konzept der direkten Manipulation ermöglicht eine einfache und schematische Benutzung von Applikationen ohne eine lange Anlernphase. Die Operationen oder abstrakte Objekte werden als Metapher oder Symbole dargestellt, die durch den Benutzer manipuliert werden können. Eine Tabelle kann durch ein Symbol dargestellt werden, welches durch das Ziehen auf ein Symbol des Papierkorbs gelöscht werden kann. Auf diese Weise kann man ein Modell der internen Logik oder Daten als graphische Oberfläche abbilden. Die graphische Benutzerschnittstelle repräsentiert sichtbar die Funktionalität der Applikation, deswegen prägt sie in hohem Maße wie eine Applikation benutzt wird.

Die Entwicklung der guten graphischen Benutzerschnittstellen ist sehr aufwendig. Ungefähr 50% der Anstrengung und Kosten bei der Entwicklung der Applikationen geht auf das Konto der GUI ([PaCh93], S. 81). Die Entwickler müssen dabei psychologische Phänomene der menschlichen Wahrnehmung berücksichtigen und auf die Gewohnheiten der Benutzer eingehen. Besondere Schwierigkeiten bereitet die Tatsache, dass Menschen und Maschinen unterschiedliche Kompetenzen bei Datenwahrnehmung und Verarbeitung haben. Der Mensch kann wenig Information aus langen Zifferkolonnen auslesen, aber die gleichen Zahlen als eine Graphik dargestellt können sehr informativ sein. Der Mensch kann nur kleine Mengen von Daten gut analysieren. Im Laufe der Zeit werden immer mehr Daten angesammelt, das geschieht immer öfter automatisch, so das der Hauptzweck der DB-Applikationen wird diese Daten menschengerecht darzustellen (Datenvisualisierung). Die menschengerechte Darstellung, also die Darstellung, die der menschlichen Aufnahmekapazität und Wahrnehmungsvermögen entspricht, wird durch Segmentalisierung und Vorselektion der Daten erreicht.

Die Entwicklung von GUI der DB-Applikationen ist besonders schwierig. Die Quellen der Fehler ([Eber94], S. 565) die vom Benutzer gemacht werden könnten sind unterschiedlich. Eine Operation könnte gegen die Applikationslogik verstoßen oder auch nicht zum Datenmodell passen. Die Benutzer erwarten, dass sie durch die Interaktion mit der Applikation, ihre Bedienung erlernen, und irgendwie alle Funktionalitäten herausfinden. Der Lernprozess wird durch den ständigen Vergleich von erwartetem und tatsächlichen Output vorangetrieben.

Die DB-Applikationen sollten nicht nur einen graphischen Ersatz für SQL-Sprache anbieten, was die Kenntnisse der Konzepten der Abfragesprachen und der Datenmodellierung allgemein voraussetzten würde. Die Graphische Schnittstelle sollte den Daten selbst (Data Domain) angepasst werden. Der Benutzer sollte unmittelbar sehen welche Datenobjekte er manipulieren kann und welche Operationen er zur Verfügung hat.

2.4 Vertiefung: Datenbank-Formulare

Datenbank-Formulare sind besonders bekannte Vertreter der DB-Applikationen. Ihre primäre Aufgabe ist es eine Oberfläche (Datenmaske oder auch Eingabemaske) für die Dateneingabe, Datenausgabe und Datenbearbeitung (Modifizieren und Löschen) darzustellen. Solche Form der Dateneingabe wurde bereits vor dem Einsatz von DB-Technologie angewandt (z.B bei amtlichen Formularen), deswegen ist durch den Wiedererkennungsfaktor die Akzeptanz solcher Art der Oberfläche bei dem Benutzer groß. Die computerisierte Formularen bieten jedoch mehr Flexibilität als ihre Vorgänger (mehrere Eingabefelder-Typen, Intergritätsüberprüfung, Eingabe-Assistenten, Hilfe-Systeme). Die Formulare sind eine graphische Repräsentation des Datenschemas, das primär für Eingabe und Ausgabe der Daten benutzt werden kann.

Die Programmlogik ist bei solchen Applikationen weitgehend einheitlich. Es geht darum die Informationen aus der Eingabemaske richtig in die Datenbank abzulegen und umgekehrt die Datenmasken mit Dateninhalten zu füllen. Die datenspezifische (veränderliche) Komponente ist das Aussehen der Formulare selbst. Dieses Merkmal hat es ermöglicht, dass DB-Formulare als erste DB-Applikationen automatisch oder halb automatisch generiert wurden und ein Formular-Generator fast zu jedem DBMS gehört (dBase, Oracle, MS Access). Bei solchen Generatoren muss der Entwickler das Aussehen der Formulare und die Verknüpfungen zwischen den einzelnen Formularfeldern und Daten in der Datenbank spezifizieren. Die datenspezifische Logik kann oft als Makros in einer dazugehörigen Skriptsprache programmiert werden (Visaul Basic bei MS Access). Es wird auch die direkte Umsetzung der Tabellen zu Formularen unterstützt. Die Ergebnisse solcher automatischen Umsetzung sind oft unzureichend und bedürfen meistens einer manueller Anpassung. Hier ein Beispiel (Abb. 2.3), ein Personen-Formular wird automatisch aus einer Tabelle erzeugt und nachher angepasst.

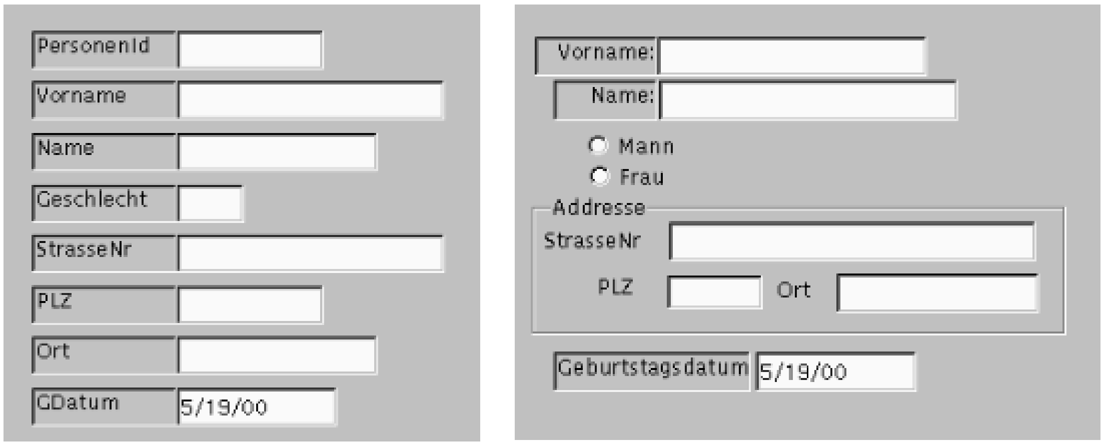

Abbildung 2.3: Automatische Formular-Generierung; links Original, rechts nach Anpassung (Generator StarOffice)

Die Anpassungen betreffen die Anordnung und Strukturierung der einzelnen Felder. Sie sind nötig, weil das Ausgangsformular nicht der Semantik und Struktur der Daten entsprach. Die Strukturierung ist aber nicht nur für dieses Formular spezifisch aber für Personendaten allgemein. Leider können solche Informationen mit relationalen Modell: wie Anordnung oder die Gruppierung der Attribute, nicht spezifiziert werden. Der Formular-Generator kann nicht entscheiden, dass die Attribute

Name und Vorname höhere Priorität haben. Die Gruppierung der Attribute: Straße, PLZ und Ort ist im relationalen Schema gleichfalls nicht festgeschrieben. Der Primärschlüssel der Tabelle wurde ausgeblendet, weil er für den Benutzer keine Aussage hat. Die Rolle des Objektidentifikator übernehmen Name und Vorname (Das könnte allerdings auch ein Photo sein). Der Formular-Generator kann nur so gute Formulare generieren wie das dem Datenschema (Datenstruktur) entspricht, das er zu Verfügung hat.

Der zweite Nachteil bei automatischer Generierung von DB-Formularen ist, dass die Formulare, die man erwartet hätte, Informationen aus vielen Tabellen involvieren. Zum Beispiel ein Formular einer Rechnung, das ein Teil eines Kundensystems sein könnte. Die direkte Umsetzung der Rechnung in eine Tabelle ist nicht möglich, weil die bestellten Waren eine Liste repräsentieren und nicht in der Struktur einer Tabelle unterzubringen sind. Oft wird nur eine spezifische Sicht auf die Informationen in einem Formulare erwartet. In der Rechnung z.B. wird nur die Adresse des Kunden wichtig, weitere Daten werden nicht weiter verwendet. Auch die Zwischensumme gehört zu den abgeleiteten Attributen der Rechnung und wenn sie von DBMS nicht berechnet werden kann, muss die Berechnung von der Applikationslogik durchgeführt werden. Es reicht nicht ein Formular als eine andere Darstellung der Tabelle (oder genauer Tabellen-Zeile) anzusehen, vielmehr ist ein Formular die Darstellung eines Objekts. Es verlangt eine Darstellung auf höherer Stufe der Abstraktion als das in relationalen DBMS möglich ist.

In der Diplomarbeit und in der Implementierung eines automatischen Formular-Generator, der ein praktisches Ergebnis der Diplomarbeit ist, werden zwei Ansätze verfolgt. Erstens, es wird eine maschinenlesbare Ergänzung zu relationalen Modell erwogen. Sie soll alle Informationen beinhalten, wie ein relationales Schema zu deuten ist, also alle die Informationen, die in dem Modellierungsprozess verloren gehen (bei Transformation des konzeptionellen Schema zum relationalen Schema). Diese Zusatzinformationen ermöglichen eine automatische Generierung von besseren Formularen und könnten auch als eine Basis für andere DB-Applikationen dienen. Zweitens, wird ein Formular als die Darstellung eines Objekts (oder einer Sicht auf Objekt) betrachtet. Es sollten alle relevanten Informationen, die einem Objekt (Formular) betreffen, durch Navigationshilfen aus dem Formular erreichbar sein.

3 Konzepte der Datenbankmodellierung

Das Aussehen der DB-Formulare wird durch die Struktur der Information bestimmt. Diese Struktur wird im Datenbankschema festgehalten. Die Syntax und die Semantik des Datenbankschemas, also wie die Struktur der Information beschrieben wird, ist durch die Konzepte eines Datenbankmodells festgesetzt.

Das Ziel der Diplomarbeit sind DB-Formulare, die mehr als eine einfache Abbildung der Relationen sind, trotzdem aber auf dem relationalen Modell basieren. Ein Weg zu guten DB-Formularen, die Informationen entsprechend ihrer Struktur darstellen, führt zu Strukturierungskonzepten der Datenbankmodellierung. Zuerst werden die klassischen Datenbankmodelle vorgestellt. Danach werden die Methoden beschreiben, wie höhere Abstraktion Konzepte in dem relationalen Modell realisiert werden können. Das letzte Unterkapitel behandelt die *Reverse Engineering* Techniken, die höhere Konzepte aus dem relationalen DB-Schema abzuleiten versuchen.

3.1 klassische Datenbankmodelle

Unter Datenbankmodell versteht man eine Menge von Konzepten, mit welchen sich die Struktur einer Datenbank beschreiben lässt. Die Struktur umfasst die Datentypen, Beziehungen und Bedienungen, welche von den Daten erfüllt werden sollen. Solche Beschreibung, in der Sprache des jeweiligen Datenbankmodells, nennt man Datenbankschema. Die meisten Datenbankmodelle definieren ferner eine Menge von Operationen auf den Datentypen. In Zusammenhang mit der Diplomarbeit ist die Beschreibung der statischen Struktur der Daten besonders interessant. Sie wird durch die Menge von atomaren Datentypen und der Möglichkeit der Typenbildung für ein Datenbankmodell bestimmt.

Die klassischen Datenbankmodelle kann man auf die abstrakten Modelle, für den konzeptionellen Entwurf und die konkreten Modelle, für den logischen Entwurf aufteilen. Zu den Modellen des konzeptionellen Entwurfs gehören die semantischen Modelle, unteranderem das ER-Modell. Sie unterstützen das Modellieren auf einem hohen Niveau der Abstraktion. Die Modelle des logischen Entwurfs werden direkt von DBMS unterstützt, sie bieten neben der Datenbeschreibungs-Sprache (DDL) auch die Datenmanipulations-Sprache (DML). Zu solchen Modellen gehören das relationale Modell, hierarchisches Modell und Netzwerk Modell. Die beiden letzten historischen Modelle werden nicht weiter betrachtet. Die logischen Modelle, wie das relationale Modell, unterstützen nur wenige und einfache Konzepte, was die Implementierung der DBMS sehr vereinfacht. Das objektorientierte Modell, oder besser die objektorientierten Modelle, gehören zu der Gruppe der konzeptionellen Modelle, können aber auch von einem DBMS direkt unterstützt werden. Das objektorientierte DBMS (OODBMS) könnte allerdings auch auf einem relationalen DBMS aufsetzen.

Bei dem Entwurf eines Datenbanksystems wird meist ein Schema mittels eines konzeptionellen Modells entworfen, das später in ein konkretes Schema abgebildet wird. Die Tranformationen der Schemta von einen Modell zum anderen sind nicht verlustfrei. Die Verluste betreffen die Informationen, wie eine konkrete Struktur zu interpretieren ist.

In nächsten Abschnitt werden die klassischen Datenbankmodelle kurz vorgestellt. Besonders wird auf die Konzepte der statischen Strukturierung eingegangen. Die Klassifizierung (Typenbildung) ist ein Konzept, das für alle klassischen Modelle gemeinsam ist. Die Möglichkeiten der Typenbildung und Modellierung der Beziehung zwischen Typen wird hier ein wichtiges Thema sein. Für die vollständige Einführung in die Konzepte der Datenbankmodelle werden zahlreiche Lehrbücher angeboten: [Voss99], [HeSa95], [KeEi96].

3.1.1 ER-Modell und seine Erweiterungen

Das ER-Modell entstand bereits Ende der 70-er Jahre und gehört zu dem populärsten Modellen, welche bei einem konzeptionellem Entwurf angesetzt werden. Das ER-Modell erlebte auch viele Erweiterungen, die oft unterschiedliche Notation ergaben. Die graphische Anschaulichkeit des Modells, das ER-Diagramm, wird für frühere Phasen der Datenbankentwicklung eingesetzt. Die zentralen Begriffe des ER-Modells sind Entity und Relationship. Ein Entity ist eine Informationseinheit, die mehrere atomare Attribute haben kann. Das Zusammenfassen von mehreren Attributen zu einem Entity ist die einzige Möglichkeit der Typenbildung in diesem Modell. Die Beziehungen werden durch Relationship ausgedrückt. Dabei werden auch spezielle Relationship-Arten unterschieden: IS-A-Relationship (für die Modellierung der Spezialisierung) und ID-Relationship (identification dependence, beschrieben im Buch [MaRa92]). Die Beziehungen können durch Eingabe der Kardinalitäten oder s.g. Min-Max-Notation noch weiter spezifiziert werden. Die Erweiterungen des ER-Modells betreffen die besondere Kennzeichnung von primär Schlüsseln, abgeleiteten oder optionalen Attributen, Attributgruppen, existenzabhängigen Entities (weak-Entity) und Aggregation von Entities und Beziehungen.

Der Erfolg des ER-Modells lässt sich dadurch Erklären, das ER-Diagramme relativ einfach in relationale Schemata überzuleiten sind. Die Transformation des Schemas kann nach einem einfachen Algorithmus erfolgen (siehe [KeEi96] Kapitel 3.2). Das entstandene relationale Schema bedarf je nach Erfahrungen des Designers keine weitere Normalisierung. Man kann auch die entstandenen Relationen sehr einfach zu den Entities und Relationship aus dem ER-Diagramm zuordnen. Dieses macht das ER-Diagramm für die Dokumentation eines relationalen DB-Schemas ideal. Die Abhängigkeiten zwischen ER-Modell und relationalem Modell werden formell gut in [MaRa92] Abschnitt 11 beschrieben. Der Autor führt auch eine „Entity-relationship normal form (ERNF)“ die nach der Abbildung des ER-Schemas auf ein relationales Schema entstanden ist. Die Verwandtschaft der beiden Modelle ist auf die einstufige Aggregation bei Typenbildung und atomare (nicht tupel- oder mengenwertige Attribute) zurückzuführen.

3.1.2 relationales Modell und NF2-Modell

Das relationale Modell gehört zu den am weitesten verbreiteten logischen Datenbankmodellen in der Praxis. Das ist auf die Einfachheit und Exaktheit dieses Modells zurückzuführen. Die gesamten Daten werden in Relationen (Tabellen) abgelegt. Die Relation (Tabelle) ist das einzige Strukturierungskonzept dieses Modells. Formal ausgedrückt, es existieren zwei Typenkonstruktoren, zuerst die Aggregation von mehreren atomaren Attributen zu einem Tupel (Spaltenbildung) und zweitens die Aggregation von gleichartigen Tupel zu einer Relation (Zeilenbildung), was allerdings auch einer Klassifizierung gleichzustellen ist.

Dieses Modell wurde auch theoretisch sehr gut erforscht. Die Manipulationen auf Relationen können durch eine relationale Algebra ausgedrückt werden, was eine Grundlage für SQL DML (Data Manipulation Language) ist. Die relationale Algebra ist mengenorientiert und in sich abgeschlossen. Die Abfragen mittels SQL sind deskriptiv und können intern sehr gut optimiert werden. Die Einfachheit des SQL ermöglicht, dass die Schnittstelle zu anderen Programmiersprachen, sich auf wenige Prozeduren beschränkt. Es existiert auch eine formale Entwurfstheorie für das relationale Modell. Dabei kann das relationale Schema in einem Normierungsprozess verbessert oder formal verifiziert werden.

Das relationale Modell ist streng wertorientiert. Es kennt keinen Beziehungstyp. Die Tupel besitzen keine eigene Identität und werden nur durch die Werte der Attribute unterschieden. Das einfache Strukturierungsprinzip dieses Modells führt zu komplexen Integritätsregeln. So kann das Konzept des Primärschlüssel und Fremdschlüssel als besondere Integritätsbedingungen aufgefasst werden. Ein getrenntes Kapitel 5 behandelt die Definition der Datenstruktur mittels SQL. Nicht alle RDBMS unterstützen die Angabe von Integritätsregeln (z.B. aus Effizienzgründen), so dass ihre Befolgung den DB-Applikationen oder dem Benutzer übelassen wird.

Das NF^2-Modell ist eine Erweiterung des relationalen Modells. In diesem Modell können strukturierte, nicht atomare, Attribute benutzt werden, die der ersten Normal-Form nicht entsprechen. Diese Eigenschaft erleichtert das Modellieren von tupel- und mengenwertigen Attributen. Der Typenkonstruktor des relationealen Modells kann so mehrmals rekursiv angewendet werden. Auf diese Weise können Tabellen andere Tabellen als Attribute haben, was zur Entstehung von geschachtelten Tabellen führt. Dieses Modell ist besonders für das Modellieren von hierarchischen Strukturen geeignet. Die Verwendung von geschachtelten Relationen, führt auch zur Erweiterung der DML.

3.1.3 semantische Modelle

Die semantischen Modelle erweitern das ER-Modell, das allerdings auch dem semantischen Modellen zugeordnet wird. Es gibt keine einheitliche Notation oder ein Diagramm für das semantische Modell, vielmehr kann man von semantischen Modellen sprechen. Die Bücher [Bekk92] [Rish92] und [Kron97] behandeln die semantische Modelle ausführlich. Gemeinsam für alle semantischen Modelle ist, das sie mehr abstrakte Konzepte für die Strukturierung der Daten als das ER-Modell anbieten. Vorallem kann man mit semantischen Modellen ein System stufenweise nach Top-Down oder Botton-Up Prinzip modellieren. Das wird durch erweiterte Typenkonstuktoren ermöglicht. Das semantischen Modell kennt zwei Strukturierugsprinzipien: Aggregation und Vererbung, diese können allerdings im Vergleich zum ER-Modell mehrmals und erweitert eingesetzt werden. Die Attribute können mengenwertigen oder objektwertigen Charakter haben. Es wird auch explizit zwischen Subjekt und Prädikat unterschieden. Die Abbildung 3.1 zeigt ein Beispiel eines semantischen Schemas, Notation angelehnt an Buch [Kron97]. Die Beziehungen werden als Aggregation der Objekte in ein Objekt modelliert, und gehören zu den Eigenschaften des Objekts. Eine Solche Lösung ist wahrscheinlich für einen Laien mehr verständlich als die ER-Diagramme. Besonders interessant ist die Tatsache, dass das semantische Schema sofort als eine gute Grundlage für den Entwurf eines DB-Formulars benutzt werden kann.

Der Begriff Entity wird durch den Begriff Objekt ersetzt. Ein Objekt beinhaltet alles, was es betrifft (Beziehungen und Eigenschaften). Die Transformation vom semantischen zu relationalen Schema ist schwierig aber auch maschinell machbar. Die Entsprechung zwischen den einzelnen Konstrukten eines semantischen und relationalen Schema sind aber nicht so offensichtlich wie beim ER-Schema.

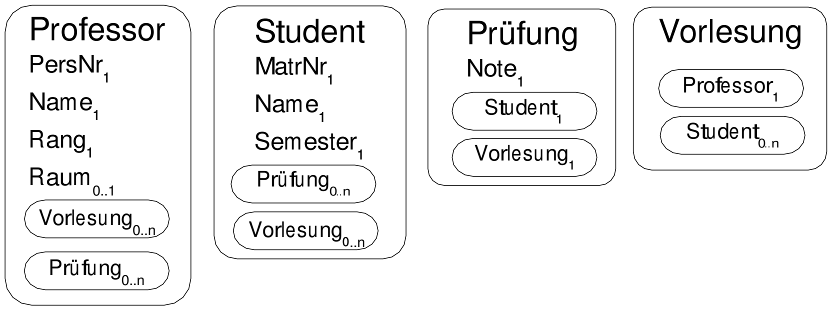

Abbildung 3.1: graphische Notatation eines semantischen Schemas

Manche semantische Modelle können auch die dynamischen Aspekte der Daten modellieren (Prozessmodellierung).

3.1.4 objektorientierte Modelle

Relativ spät, in den 80-er Jahren, haben die erfolgreichen Konzepte der objektorientierten Sprachen wie C++ oder Smalltalk den Eingang in die Welt der Datenbanksysteme gefunden. Die objektorientierten Konzepte der Modellierung kann man in strukturorientierte und verhaltensorientierte aufteilen. Die Konzepte zur Darstellung der Struktur beinhalten:

- komplexe Typenkonstruktoren wie Mengen, Tupel, Listen, Multilisten, Hashes, Arrayes.
- Vererbung von Attributen
- Objektidentität
- Direktes Modellieren der Assoziationen (Beziehungen)

Das Verhalten wird durch Objektmethoden modelliert. Dabei können die Methoden ähnlich wie bei C++ vererbt, überschrieben und virtuell sein (late binding). Die Definition einer Klasse beinhaltet dementsprechend einen Strukturteil und einen Verhaltensteil.

Die meisten objektorientierten DBMS unterstützen nur einen Teil dieser Konzepte. Es existiert ein Standard für objektorientierte DBMS der von der *Objekt Data Managment Group* erarbeitet wurde, der ODMG-93-Standart. Der Standard spezifiziert die Objekt-Definitionssprachen, Objekt-Anfragesprachen und Anbindungen an objektorientierte Programmiersprachen.

Aus der Sicht der Entwicklung von DB-Applikationen, bringen die objektorientierten Modelle viele Neuerungen. Das Verhalten der Objekte wird bereits von DBMS erledigt; die Implementierung der datenspezifischen Logik wird auf DBMS verschoben. Die DB-Applikationen müssen nur noch die applikationspezifische Logik implementieren, was meist auf die Programmierung der Benutzerschnittstelle zurückführt. Deswegen ist die nahtlose Integration der Programmiersprache und des Datenbanksystems besonders wichtig. Im Idealfall kann der Programmierer die Datenbankobjekte genau so wie die Objekte der Programmiersprache behandeln (Objektidentität).

Die objektrelationalen Datenbanken erweitern das relationale Modell um objektorientierte Eigenschaften. Die POSTGRES Datenbank fügt folgende objektorientierte Konzepte ein:

- mengenwertige Attribute (Array oder Listen)
- selbsdefinierte Typen
- direkte Unterstützung der Vererbung. Tabellen Vererben die Attribute von anderen Tabellen.
- Tupel besitzen eigene Identität

Für den konzeptionellen Entwurf der objektorientierten Systeme hat sich die UML (United Modeling Language) etabliert (Buch [BoEJ99]). Diese Sprache stellt eine Reihe von unterschiedlichen Diagrammen für das Modellieren von Struktur und Verhalten der Objekte bereit. Die Klassen-Diagramme sind sowohl für das Modellieren der Struktur der objektorientierten Programme als auch für Datenbanken geeignet. Besonders ausdruckkräftig ist das Modellieren von Objektbeziehungen im UML. Es werden folgende Beziehungstypen unterstützt:

- Abhängigkeiten
- Generalisierung
- Assoziation
- Aggregation (und ihre stärkere Variante Komposition)
- Realisierung

Die Klassen und Beziehungen können noch weiter mit den s.g. „Stereotypen“ oder „Standardeinschränkungen“ beschrieben werden.

3.2 Höhere Abstraktions-Konzepte im relationalen Modell

Das relationale Modell ist arm an Modellierungskonzepten, bittet jedoch genügend Flexibilität um höhere Abstraktionskonzepte in ihm zu simulieren. Hier werden die Möglichkeiten der Abbildung (mapping) solcher höheren Konzepte mit relationalen Modell vorgestellt. In der Literatur findet man gute Abbildungsvorschriften für Schemen Transformationen von ER (EER) Modell zu relationalen Modell ([MaRa92] Kapitel 11, [HeSa95] Kapitel 4.4.1, [KeEi96] Kapitel 3.3.2, [MaSh92]).

3.2.1 Generalisierung und Spezialisierung

A. Kemper und A. Eickler nennen die Generalisierung als Abstraktion auf Typenebene ([KeEi96] Seite 44). Ähnliche Entitytyps können als gleichartig betrachtet werden, wobei man von den Unterschieden abstrahiert. Spezialisierung stellt eine Umkehrung der Generalisierung dar, die Entitytyps werden in Untertypen verfeinert. Das Schlüsselkonzept der Generalisierung ist die Vererbung. Die Untertyps (childs) erben die Eigenschaften des Obertyps (parent). Die Beziehung zwischen den Untertyps und Obertyp wird IST-Beziehung (zu eng. IS-A Relationship) genannt. Man kann drei besondere Fälle der Generalisierung unterscheiden:

disjunkte Spezialisierung Jede Instanz gehört höchstens zu einem Untertyp (auch bekannt als Partitionierung).

vollständige Spezialisierung Es gibt keine Instanz ohne einen Untertyp. Bei den objektorientierten Modellen als abstrakte Klassen bekannt.

subset Hierarchie Eine Instanz kann zu mehreren Untertyps gehören (Gegensatz zu disjunkten Spezialisierung). Bei den objektorientierten Modellen als Mehrfachvererbung bekannt.

Die Berücksichtigung der zwei ersten Fälle verlangt bei der Handhabung der Instanzen eine Aufstellung von besonderen Integritätsregeln. Allgemein für Generalisierung gilt per Definition, dass es keine Instanz nur in Untertyp geben kann (ohne den Obertyp). Die Abbildung 3.2 zeigt eine gängige Notation für Generalisierung im ER-Modell.

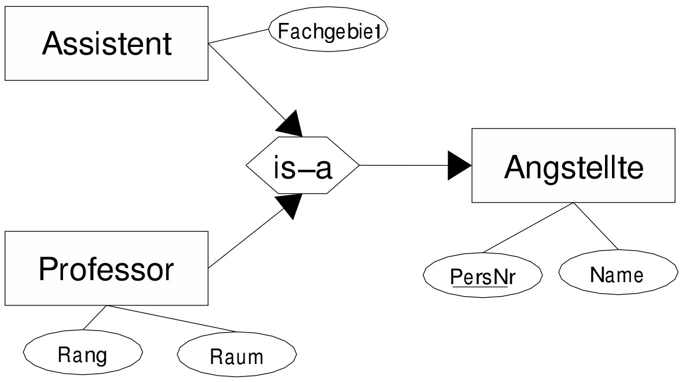

Abbildung 3.2: Modellierung der Generalisierung im EER-Modell

Das relationale Modell unterstützt das Modellieren der Generalisierung nicht. Es gibt zwei Möglichkeiten die Generalisierung in relationalen Modell abzubilden. In der einfachsten Methode fasst man alle Typen (Obertyp und alle Untertypen) zu einer Tabelle zusammen.

Angestellte : {[PersNr,Typ,Name,Rang,Raum,Fachgebiet]}

Die nicht benutzten Attribute werden nicht ausgewertet oder mit NULL belegt. Um eine Unterscheidung zu ermöglichen, zu welchen Typ (Untertyp) eine Instanz gehört, kann ein besonderes Attribut hinzugefügt werden (Typattribut). Die andere Methode basiert auf der Idee der Vererbung des Primärschlüssels. Der Primärschlüssel des

Untertyps wird gleichzeitig ein Fremdschlüssel. Die Benennung der Primärschlüssel wird oft beibehaltet.

Angestellte : {[PersNr,Name]}

Professor : {[PersNr,Rang,Raum]}

Assistent : {[PersNr,Fachgebiet]}

Es ist zu beachten, dass das so entstandene relationale Schema auch anders interpretiert werden kann. Die semantische Information, dass es sich um eine Vererbung handelt, geht bei der Transformation des Schemas verloren. Einen Hinweis auf eine Vererbung könnten die gleichnamigen Primärschlüssel geben, es ist aber aus dem relationalen Schema nicht ersichtlich welche Tabelle den Obertyp abbildet. Ein anderes Ergebnis der Abbildung sind verschiedene Integritätsbedingungen, die in das relationale Modell einfließen. In diesem Fall muss garantiert werden:

$$\Pi_{Angestellte}(PersNr) \supset \Pi_{Profesor}(PersNr)$$

$$\Pi_{Angestellte}(PersNr) \supset \Pi_{Assistent}(PersNr)$$

$$\Pi_{Profesor}(PersNr) \cap \Pi_{Assistent}(PersNr) = \emptyset$$

Sehr oft wird in der Praxis auf die spezielle Modellierung der Generalisierung verzichtet.

Professor : {[PersNr,Name,Rang,Raum]}

Assistent : {[PersNr,Name,Fachgebiet]}

3.2.2 Aggregation

Das relationale Modell unterstützt nur eine Art der einstufigen Aggregation. Es ist die Zusammenfassung der Attribute zu einer Relation (Spaltenbildung der Tabelle). Es fehlen andere Möglichkeiten der Typenbildung zB. die Aggregation der anderen atomischen Typen (strukturierte Attribute), Aggregation der Objekte oder mengenwertige Typen. Solche Arten der Aggregation fehlen auch dem ursprünglichen ER-Modell und wurden erst bei dem erweiterten ER-Modell (EER-Modell Extendet ER-Modell) vorgeschlagen. Nachteilig für beide Modelle ist, dass sie im Gegenteil zu semantischen Modellen das Objekt und die Eigenschaft nicht unterscheiden (Subjekt und Prädikat); (Vergleiche [Bekk92] Seite 81). Es ist aus dem ER-Modell nicht ersichtlich (siehe Abb. 3.3) , dass es sich bei der Tabelle „TelefonNr“ um die Modellierung eines Mengenattributs handelt. Es ist auch eine Aggregation auf einer höheren Ebene denkbar. Zusammenfassung mehrerer Objekte (Entities und Relationen) zu einem Objekt. Das graphische EER-Modell sieht dafür eine besondere Notation vor (Abgrenzung mit einem Rechteck), diese lässt sich aber nicht mit dem relationalen Modelle abbilden. (Vergleiche auch SQL VIEW Kapitel 5.3) Bei der

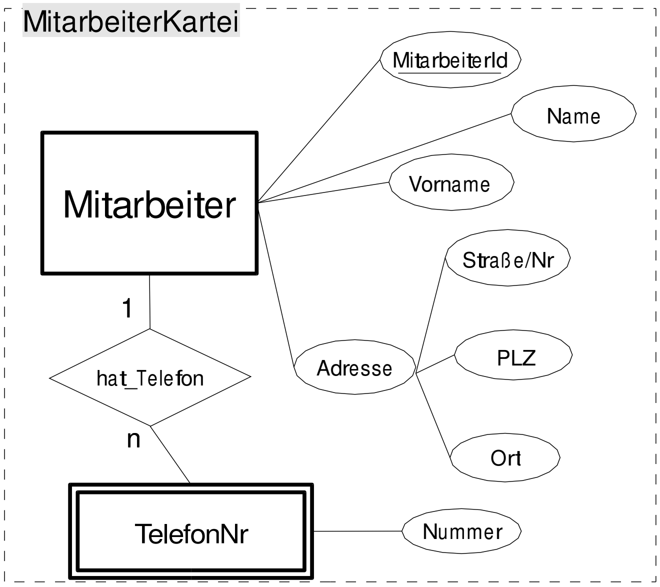

Abbildung 3.3: Modellierung der Aggregation im EER-Modell

Abbildung des konzeptionellen Modells auf relationales Modell gehen viele Informationen bzgl. der Aggregation verloren. Die mengenwertigen Eigenschaften werden als getrennte Tabellen interpretiert, was die Anzahl der Relationen, die keine Objekte repräsentieren, erheblich steigern kann.

Mitarbeiter : {[<u>MitarbeiterNr</u>,Name,Vorname,Strasse,Ort,PLZ]}

TelefonNr : {[<u>MitarbeiterNr,Nummer</u>]}

Eine andere Form der Aggregation ist das im [MaRa92] vorgestellte ID-Relationship. Die Abbildung 3.4 zeigt die graphische Notation der Beziehung. Sie wird allerdings auch als die Spezialisierung auf Objektebene betrachtet. Wichtig ist, dass das

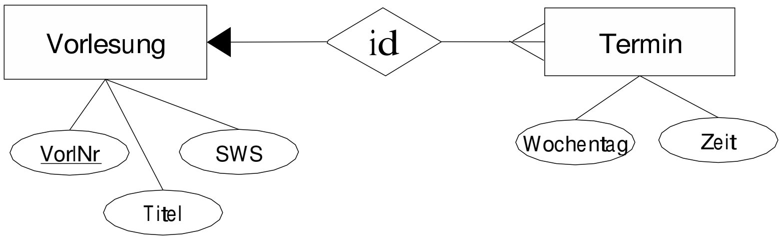

Abbildung 3.4: ID-Relationship

Tupel „Termin“ keine eigene Identität besitzt, also die Spezialisierung vom Objekt Vorlesung ist. Diese Art der Beziehung wird selten verwendet, kann allerding aus der Sicht der DB-Applikationen praktisch sein. Es bedeutet, dass die Identität des Objekts immer aus zwei Teilen besteht (geerbt wird). Eine konkrete Vorlesung wird also immer so identifiziert: Vorlesung C++, Montag 10:15.

Eine stärkere Form der Aggregation, ist die Komposition (auch part-of Beziehung). Zwischen den Komponenten und dem *container* besteht eine Existenzbedingung. Solche Existenzbedingungen werden als weak-Entities im ERR-Modell bezeichnet.

3.2.3 Assoziation

Der Begriff der Assoziation wird nicht einheitlich in der Literatur behandelt. Bei der semantischen Modellierung kommt er überhaupt nicht vor. ([Bekk92] und [KeEi96]. Auch im Kontext der ER-Modelle und relationalen Datenbank-Entwurf wird er oft nicht mal erwähnt [KaKl93] und [MaRa92]. Eine sehr gute Erklärung des Begriffs und nötige Abgrenzung bietet [GoSt99], Abschnitt 2.3. In der Arbeit wird Assoziation beschrieben als

> [...] form of abstraction in which a collection of members is considered as a higher-level (more abstact) set, called an entitiy set type. The details of the member objects are suppressed and properties of the set object (the assocation) emphasized.

In [HeSa95] S. 74 wird Assoziation als eine Sammlung definiert, im Gegensatz zu Mengenbildung auf Werten der Aggregation. Schwierigkeiten bereitet die nahe Verwandtschaft der Assoziation zur Aggregation (Vergleiche [GoSt99]).

Das semantische Modell kommt ohne Assoziation aus. Sie wird als eine Aggregation von Objekten (besser Referenzen auf Objekte) verstanden.

Die Assoziationen (s. Abb. 3.5) haben mehrere Eigenschaften, die jedoch nicht immer alle explizit genannt werden:

- Assoziation hat einen sinnvollen (interpretierbaren) Namen (im Beispiel „Prüfung“). Name der Sammlung
- Durch Assoziation bekommen die Objekte einen zusätzlichen sinnvollen Namen (Rollennamen).
- Granulität (Anzahl der Objekten in der Sammlung)
- Kardinalitäten
- Assoziation kann eigene Eigenschaften haben.
- Assoziation ist transitiv

Die Rollennamen können als ein Synonym für ein Objekt verwendet werden. Die Bedeutung von solchen Synonymen kann für die Benutzerschnittstelle wichtig sein, weil ihre Verwendung die Kenntnisse über Verknüpfungspfade (Fremdschüssel) unnötig macht. Man kann sich eine solche SQL-ähnliche Abfrage mit Verwendung der Synonyme vorstellen

```
SELECT * FROM Prüfling WHERE Professor.Name="Schmith"
```

Die Assoziation wird als eine bidirektionale, transitive Beziehung verstanden. Bei logischen Modellen wird sie oft als ein Verweis, ein Zeiger oder eine Referenz realisiert, so dass das Bestimmen der Assoziationen oft nur in einer Richtung möglich ist. UML (United Modeling Language) sieht deswegen auch gerichtete Assoziationen vor. Bei relationalen Modell wird die Assoziation durch Fremdschlüssel realisiert,

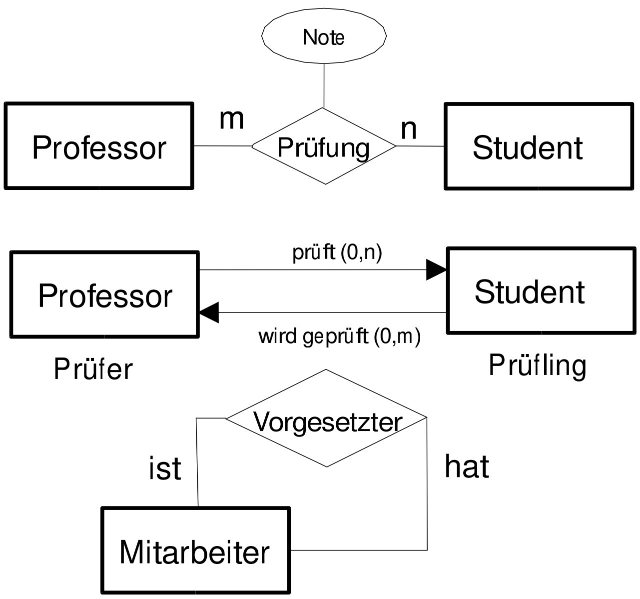

Abbildung 3.5: Semantik der Assoziation im ER-Modell

die auch einen Referenz-Charakter haben. Das Bestimmen der Assoziationen gegen die Richtung der Verweise wird durch Abfragen der Art ermöglicht: Suche alle, die auf bestimmtes Objekt verweisen. Bei relationalen Modell werden solche Abfragen durch Selektion erledigt. Nachteilig ist es, dass man für solche Abfragen das gesamte relationale Schema kennen muss.

Wie früher erwähnt, werden die Assoziationen im ER-Modell als Beziehungen (Relationship) realisiert. Dabei unterscheidet man drei Arten der Beziehungen: 1:n n:m oder 1:1. Bei n:m Beziehung ist die Verwendung einer zusätzlichen Tabelle bei der Abbildung auf ein relationales Schema nötig. Auch bei 1:n Beziehungen kann eine zusätzliche Tabelle (für die Kodierung der Assoziation) sinnvoll sein, wenn die Assoziationen zwischen den Objekten selten sind und man viele NULL Werte vermeiden will.[1] Die Abbildungsregeln sind in der Literatur sehr gut beschrieben. Es ist zu erwähnen, dass bei der Abbildung die Information über die Kardinalität der Beziehung verloren geht. Die Rollennamen gehen oft bei rekursiven Beziehungen in den Namen der Fremdschlüssel über. Bei Verwendung einer zusätzlichen Tabelle wird ihr Name nach den Namen der Sammlung ausgewählt. Eine mögliche Kodierung des Beispiels als ein relationales Schema wäre:

[1]Auch wenn man die Assoziation von den Tabellenbenutzern verstecken will

Professor {[PersNr,Name,Raum]}

Student {[MatrNr,Name,Semester]}

Prüfung {[MatrNr,PersNr,Note]}

3.3 Reverse Engineering: Ableitung der höheren Konzepte im relationalen Modell

Chikofsky and Cross ([ChCr90]) definieren Reverse Engineering als:

> the process of analysing a subject system to identify the system's componets and their relationships and create representations of the system in another form or at a higher level of abstraction

Reverse Engineering stellt eine Umkehrung des Datenbankentwurfs dar, wie in der Abbildung 3.6 gezeigt wird. Es wird versucht aus dem logischen Schema wieder

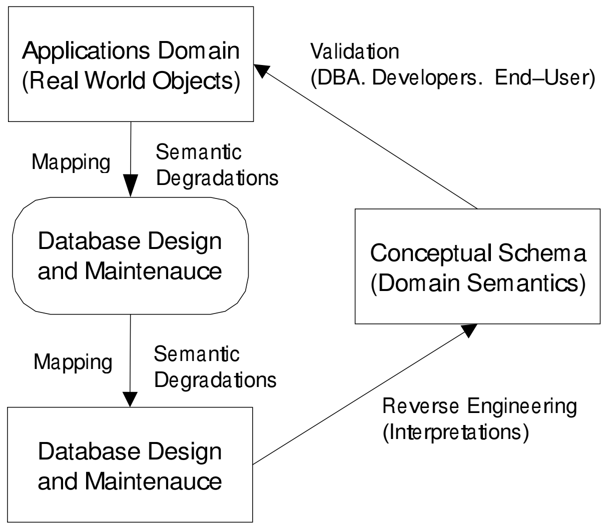

Abbildung 3.6: Datenbankentwurf und Reverse Engineering (Quelle [ChBS97])

ein konzeptionelles Schema zu gewinnen. Dabei werden die wenigen Abstraktion-Konzepte der logischen Schemata (hier relationales Modell) in ein semantisch ausdruckskräftigeres konzeptionelles Modell umgesetzt (meist ER-Modell).

Solches Vorgehen ist immer dann sinvoll, wenn eine Migration, Integration oder Änderung der Datenbank angestrebt wird und die Dokumentation des konzeptionellen Schema nicht vorhanden ist. Im Zusammenhang mit der Diplomarbeit ist die Reverse Engineering eine Technik um ein konzeptionelles Schema in maschinenlesbarer Form zu schaffen, gleichzeitig mit der Information wie es schließlich in das logische Schema abgebildet wurde.

Es existiert eine Reihe wissenschaftlicher Beiträge zu Reverse Engineering. Eine gute allgemeine Aufstellung und Klassifizierung bietet [ChBS97] und [FaVo95] . Die genauen Regeln und Algorithmen stellt die Arbeit [ChBS94] vor.

Problematisch bei der Reverse Engineering ist die Tatsache, dass man eine höhere Abstraktion aus einer einfachen ableiten muss. Für den Prozess der Reverse Engineering können drei Quellen der Informationen identifiziert werden.

1. logisches Schema selbst. relationales Modell gespeichert in Data Dictionaey der RDBMS (siehe Abschnitt 5.4)
2. Analyse der Daten, Integritätsregel, Sichten (Views) und SQL Abfragen.
3. Befragen der Benutzer.

Das relationale Modell ist streng Wertorientiert. Es kennt keine Verknüpfungstypen. Die Verknüpfungen werden als Fremdschlüssel repräsentiert. Die Tabellen können als Entities, Relationen, Teilobjekte (bei Generalisierung) oder Eigenschaften (z.B Listen der Attributen) verwendet werden. Bei Reverse Engineering wird die semantische Bedeutung, also die Information, wie das relationale Modell zu interpretieren ist, wiedergewonnen. Manche dieser Informationen werden nur durch Namensgebung der Attribute und Tabellen wiedergespiegelt. Es ist jedoch nicht immer voraussetztbar, dass die Namen nach bestimmten Konventionen ausgewählt wurden.

Die Informationen, wie die genaue Kardinalität der Beziehungen oder Interpretation der Nullwerte, sind aber nur durch Analyse der Daten oder Befragung der Benutzer zu gewinnen.

Der Prozess der Reverse Engineering wird durch viele Merkmale bestimmt (siehe [ChBS97] Kapitel 2). Besonders wichtig ist es, ob das logische Schema durch einen guten Entwurf entstanden ist. Auch das ausgewählte Modell für das konzeptionelle Schema ist wichtig. Meist wird dafür ein EER Modell benutzt aber auch die Benutzung der anderen semantischen Modelle wäre sinnvoll. Der Prozess selbst ist nicht trivial, er hängt davon ab, welche Informationen genutzt werden. Oft kann nur eine Auswahl der alternativen wahrscheinlichen Interpretierungen angeboten werden und die konkrete Semantik muss von dem Benutzer bestimmt werden.

4 Semantik der Daten im relationalen Modell

Das relationale Modell ist nur so weit vollständig, dass es eine korrekte Behandlung der Daten von Seiten der DBMS ermöglicht. Das relationale Schema beinhalten aber keine Aussagen über die mögliche Interpretation der einzelnen Relationen. Das relationale Modell ist streng wertorientiert, die Interpretation der Attribute nur als Werte ist aber oft ungenügend. In diesem Kapitel wird eine Systematisierung der Interpretation der Grundkonzepte des relationalen Modells vorgenommen. Die Einbeziehung der Semantik ist wichtig bei der Entwicklung der Datenbankapplikationen und Interpretation der Datenbank. Letztlich wird über die erweiterte Meta-Informationen der Daten berichtet, die zwar für ein RDBMS unwichtig sind aber eine große Hilfe bei der Verarbeitung der Daten darstellen.

4.1 Primärschlüssel und Schlüsselkandidaten

Das Konzept der Schlüssel ist im relationalen Modell sehr wichtig. Zuerst wird ein Schlüssel für die eindeutige Identifizierung des Objekts (besser Tupel) benutzt, zweitens wird über einen Schlüssel der Verknüpfungstyp realisiert. Formal ist der Primärschlüssel (ein ausgewählter Schlüsselkandidat) die minimale Menge an Attributen, die ein Tupel eindeutig identifiziert. Mathematische Definition:

$$\forall t, t' : t(K) = t'(K) \Rightarrow t = t' \quad \text{und K minimal}$$

Konkret wird der Primärschlüssel als eine Integritätsregel aufgefasst. Die Ausprägung der Prim-Attribute (Attribute, die zum Primärschlüssel gehören) kommt nur einmal in der Relation vor.

Betrachtet man die Semantik der Primärschlüssel kommen zwei Arten vor:

1. Der Primärschlüssel ist ein real existierendes Attribut des Objekts (z.B. Autokennzeichen) oder eine Menge seiner Eigenschaften (Name, Vorname, Geburtsdatum)

2. Der Primärschlüssel wird nur für die Zwecke der Datenbank erzeugt. Entweder als eine Kodierung von anderen Attributen a) oder wird er automatisch bei der Objekt-Erfassung erzeugt, mit der einziger Bedingung, dass er bereits nicht vorkommt b).

Solch eine Unterscheidung ist bei der Bildung von Benutzerschnittstellen sehr wichtig. Wird der Primärschlüssel nur für die internen Zwecke der Datenbank erzeugt, (Fall 2b) ist sein Wert für den Benutzer völlig unwichtig und kann ausgeblendet werden. Zu Bemerken ist die Tatsache, dass der Mensch auch gewisse Attribute für die Identifizierung der Objekte benutzt, die aber oft anders sind als der Primärschlüssel der Relationen. Für die Identifizierung der Personnen werden Name und Vorname benutzt. Für eine Datenbank sind Name und Vorname nicht ausreichend, weil Menschen mit den gleichen Namen und Vornamen vorkommen (z.B. Hans Schmith). Gibt es eine relativ kleine Gruppe von Personen können sich die Menschen leicht merken, dass zwei Personen den gleichen Namen und Vornamen haben (es ist eine Ausnahme). Für das relationale Modell wird aber so ein Fall Inkonsistenzen in der Datenbank hervorbringen, wenn Name und Vorname als Primärschlüssel ausgewählt wurden (keine Behandlung von Ausnahmen). Deswegen werden entweder die Identifizierungs-Merkmale erweitert (Geburtstag, Geburtsort) oder es wird ein synthetischer Schlüssel (PersonId) generiert. Als Schlüssel wird meist eine Nummer benutzt, die sich leicht indeksieren lässt und deren Benutzung effizienter ist als bei mehreren Zeichensatz-Attributen. Für den Benutzer bleiben aber die Identifizierungs-Merkmale immer noch Name und Vorname und im Grenzfall, wenn nötig Geburtsdatum. Problematisch wird es, wenn die Identifizierungs-Merkmale wie Foto oder

Stimmenaufnahme für Datenverarbeitung ungeeignet sind.

Bei der Implementierung der DB-Formulare in der Diplomarbeit wird zur Identifizierung der Objekte durch den Benutzer ein alternativer Schlüssel angeboten, er wird *Tabellen-Etikette* genannt.

4.2 Fremdschlüssel

Fremdschlüssel sind die einzige Möglichkeit den Verknüpfungstyp zu simulieren. Die Darstellung (Interpretation) der Fremdschlüssel in Benutzerschnittstellen bereitet besondere Schwierigkeiten. Ein Fremdschlüssel stellt eine Referenz (Verweis) auf ein anderes Tupel (meist auch ein Objekt) dar. Die Darstellung des Fremdschlüssels als einen Wert ist oft unzureichend, wenn der korrespondierende Primärschlüssel nicht von dem Benutzer zur Identifizierung des Objekts benutzt werden kann. (Siehe Fall 2b vorheriger Abschnitt 4.1). Von einer anspruchsvollen Benutzerschnittstelle erwartet man z.B. das statt der Ausgabe von abstrakten PersonenID der Name und Vorname (oder ein Photo) des Mitarbeiters dargestellt werden. Manche Formulargeneratoren wie MS Access unterstützen sogar direkt die Darstellung von alternativen Schlüsselkandidaten statt des Primärschlüssels.

Ein anderes Problem ist es, dass die Änderung der Verweise nicht einfach durch das Editieren von Fremdschlüsseln realisiert werden kann. Jeder Wert des Fremdschüssel muss eine existierende Entsprechung als Primärschlüssel haben (Der Wertebereich ist sehr eingeschränkt). Um die schlechten Eingaben der Fremdschüssel von Anfang an auszuschließen, wird die Eingabe standardmäßig als eine Auswahlliste dargestellt. Ist jedoch die Anzahl der Objekte sehr groß muss eine speziele Suchfunktion angeboten werden.

Eine besonders anspruchsvolle Aufgabe ist auch die Darstellung der n:m Beziehungen. Dieser Beziehungstyp wird durch eine zusätzliche Tabelle mit zwei Referenzen (Fremdschlüssel) simuliert. Die Abbildung der zusätzlichen Tabelle als ein Formular liefert schlechte Resultate. Das Formular kann dann nur zwei Verweise auf andere Objekte haben. Eine Bessere Möglichkeit bieten die Multiauswahllisten oder die Einbettung der Formulare. Näherliegend ist auch die Idee einen solchen Verweis als ein Link (im Sinne der Hyperdokumente) zu realisieren.

4.3 Spezielle Interpretation der Attribute

Das relationale Modell basiert auf dem mathematischen Konzept der Menge. In der Menge besitzen die Elemente keinerlei Anordnung. Um sich in großen Datenmengen zurecht zu finden erwarten die Benutzer einen Schlüssel der ihnen das schnelle Aussuchen des konkreten Objekts erlaubt. Dies garantiert oft eine Sortierung. Z.B wird die Mitarbeiterdatenbank vor der Ausgabe nach Namen und Vornamen alphabetisch sortiert. Auch wenn keine Sortierung sinnvoll ist, erwartet der Benutzer das Objekt-Ansammlungen immer in der gleichen Reihenfolge vorkommen.

Besitzen die Tupel eine natürliche Anordnung wird oft ein spezielles Attribut hinzugefügt, das die Anordnung repräsentiert. Die einfache Durchnummerierung (oder die Benutzung des Primärschlüssels für solche Zwecke) ist nicht immer die beste Lösung. Sollte sich die Anordnung nachträglich ändern, dann müsste man bei allen Tupel die Attribute neu durchnummerieren. Die Verwendung von Zeichenketten bietet mehr Flexibilität. Angenommen, die Anordnung der Tupel ist durch einen Buchstaben repräsentiert (a,b,c,d,e,f,g). Will man die zwei letzten Tupel zwischen dem Ersten und Zweiten anordnen, ändert man die „Anordnung“-Attribute auf (a,b,c,d,e,aa,ab), was lexikalisch sortiert (a,aa,ab,b,c,d,e) ergibt. Der Wert dieser Attribute ist für den Benutzer uninteressant.

Die s.g *temporale* Datenbank bildet nicht nur ein aktueller Abschnitt der Realität ab, sondern speichert auch historische Daten. Die relevanten Objekte werden in vielen Versionen, Zuständen eines Lebenszyklus, gespeichert. Die RDBMS unterstützen nicht direkt die Behandlung von temporalen Daten, so dass die korrekte Auswertung von DB-Applikationen vorgenommen werden muss. Die einfachste Möglichkeit besteht, indem man den Primärschlüssel jedes Objekts um ein Versions-Attribut (oder Zeitangabe TIMESTAMP) erweitert. Bei Modifizierung des Objekts wird durch kopieren ein neuer Tupel (andere Version des Objekts) erzeugt. Die DB-Applikationen müssen solche Versions-Attribute richtig interpretieren und eine Schnittstelle für den

Zugriff auf einzelne Versionen der Objekte anbieten.

Es sind aus der Praxis weitere Attribute in Relationen vorstellbar, die spezieller Interpretation, anderer als ein Wert, bedürfen.

- Autorisierung von Informationen (Zugriff Kontrolle für jedes Tupel)
- Weitere strukturelle Informationen
- Angaben über den Typ der anderen Attribute, die etwa als allgemeine Byte-Folge kodiert sind.

Solche Interpretationen gehen über die Möglichkeiten des relationalen Modells hinaus und müssen erst auf der Stufe der DB-Applikationen bewältigt werden.

4.4 Nullwert Interpretation

Die Nullwerte erweitern den Wertebereich eines Attributs um einen zusätzlichen speziellen Wert. Die Manipulation von Nullwerten bei Operationen (wie Durchschnitt berechnen, Sortieren, Vergleichen) ist problematisch. Die Richtlinien des guten Datenbankentwurfs sehen die Vermeidung von Nullwerten vor. Die relationalen Datenbanken sehen auch eine spezielle Klausel vor (*not null*), die die Benutzung von Nullwerten verbietet.

Andererseits sind Nullwerte sehr nützlich um unvollständige Information auszudrücken. Die Interpretation des Nullwertes kann unterschiedlich sein:

- Wert existiert, ist aber zur Zeit nicht bekannt
- Wert existiert nicht
- Wert ist für dieses Tupel nicht sinnvoll
- Es ist nicht bekannt, ob ein Wert für dieses Tupel sinnvoll ist.

Treten die Nullwerte bei Fremdschlüsselattributen (Modellierung der 1:N Beziehung), bedeutet es, dass dieses Objekt mit keinen anderen Objekt (N Seite des 1:N Relationships) in Beziehung steht.

Das Beispiel aus Abbildung 3.2 kann auch wie folgt modelliert werden:
Angestellte(PersNr,Name,Typ,Rang,Fachgebiet,Raum,Professor)
Das Attribut *Typ* spezifiziert, ob das Objekt vom Typ Professor oder Assistent ist. Davon abhängig, sind die typspezifischen Attribute, wie Rang und Professor für ein Objekt, nicht sinnvoll. Die benutzerfreundlichen DB-Applikationen sollen die Nullwerte genau nach dem Vorhaben des DB-Designers interpretieren und entsprechende Informationen an den Benutzer liefern.

4.5 Meta-Informationen der Attribute

Eine weitere Verfeinerung der relationalen Schemata erreicht man, indem man die Attribute der einzelnen Objekte noch weiter spezifiziert. Die einzigen Informationen über die Attribute im relationalen Schema liefern sein Name und der dazugehörige Datentyp. Der Datentyp sagt etwas über den Datenbereich (Domäne) und die gültigen Operationen auf diesem Typ aus. Die Anzahl von verfügbaren Datentypen (mit sinnvollen Operationen) ist beschränkt. Die Multimedia-Daten können von dem RDBMS nur als eine Folge von Bytes interpretiert werden (SQL-Typ LBLOB). Um eine Graphik Datei (z.B. Angestellten-Photo) sinnvoll zu interpretieren, braucht man dafür ein Graphik-Betrachter-Programm. Es existiert bereits ein Standard MIME (Multipurpose Internet Mail Extension, definiert im RFC 1521), das ein Format einer Datei spezifiziert. Die Angabe „text/html“ bedeutet, dass die Datei im HTML-Format gespeichert ist.

Es sind mehrere verschiedene Meta-Informationen zu einzelnen Attributen vorstellbar. Die Attribute können nach ihrer Semantik und Art systematisiert werden. Eine Möglichkeit von *data domain characterization* wird in [PaCh93] beschrieben. Der Autor schlägt in der Publikation eine komplexe Taxonomie der Daten vor. Dabei werden die Attribute auf mehre Kategorien und Subkategorien aufgeteilt (etwa: physische Eigenschaften, Namen und Synonyme, Messgrößen, Zeitangaben).

Aus der Problematik der Entwicklung der DB-Applikationen können folgende Informationen wichtig sein:

- Bei Zahlen. Welche statistische Informationen sind sinnvoll zu berechnen. Mittelwerte, Varianz. (Die Berechnung des durchschnittlichen PersID ist nicht sinnvoll). Statistische Zugehörigkeit zu Skalentypologie (Nominalskala, Ordinalskala, Intervallskala, ...), die eine sinngemäße automatische Auswertung erlaubt.

- Eingabe und Ausgabeformat (z.B. bei Datum, Währung)

- Sind die Attribute identifizierend (z.B. Name, Vornamme)

- Repräsentiert ein Attribut einen Satz der natürlicher Sprache. Ist die Rechtschreibüberprüfung sinnvoll? (z.B Attribut Bemerkung)

- kann das Attribut nachträglich geändert oder ergänzt werden? Die Primärschlüssel gehören zu dem s.g. *frozen* Attributen (UML Stereotyp).

Wie viele Meta-Informationen wirklich explizit benannt werden sollen hängt von der Art der späteren Datenbearbeitung ab. Die vollständigste Beschreibung der Daten wird in den Gebieten Wissens-Repräsentation (knowledge representation) und Experten-Systemen verfolgt.

5 SQL als Data Definition Language der RDBMS

Structured Query Language (SQL) ist eine weit etablierte Abfragesprache zur Datendefinition und Informationsgewinnung der relationalen Datenbanksysteme. Sie wird auch benutzt um Datenbanksysteme zu konfigurieren (Benutzerverwaltung) und das interne Schema zu manipulieren (Index-Verwaltung). Bei den meisten RDBMS ist SQL eine Schnittstelle zu allen Funktionalitäten des Datenbanksystems. Deswegen wird SQL oft als ein Synonym der RDBMS benutzt, man spricht von SQL-Datenbanken. In diesem Kapitel wird besonders auf die Eigenschaften der SQL als Datendefinitionsprache (Schemadefinition, DDL Data Definition Language) eingegangen. Auch die Aspekte der Abfrage des Datenbankschemas (Data Dictionary) werden berücksichtigt. Die Behandlung dieses Themas ist aus praktischen Gründen wichtig. Es beantwortet die Frage, wie ein relationales Modell praktisch umgesetzt wird und ist aus der Sicht der Implementierung unerlässlich. Es wird allerdings darauf verzichtet, eine grundlegende Einführung in SQL zu präsentieren. Hier wird auf die umfangreiche Literatur verwiesen wie [Misg98], [MaUn97] oder [KeEi96].

Obwohl SQL weitgehend standardisiert ist (ISO Normen SQL-86, SQL-92, neuerdings SQL3), weichen die verfügbaren Systeme von den Normen ab. Es werden manche von der Norm verlangten Funktionalitäten nicht angeboten oder neue Eigenschaften werden hinzugefügt, die später zu einer Quasi-Norm werden. Die Kenntnisse dieser Abweichungen ist für die Arbeit mit einem konkreten Datenbanksystem zwingend.

5.1 Mächtigkeit der SQL (DDL)

Zu den Aufgaben der Data Manipulations Language der SQL gehören:

- Anlage und Verwaltung von Schemata der Datenbank
- Definition von Domänen (Benutzer Datentypen oder Wertebereichen)
- Definition von Integritätsbedingungen
- Anlage von Datensichten (Views)
- Verwaltung von Benutzern und Benutzerrechten

Die wichtigste Anweisung für das Bestimmen des Datenbankschemas ist die CREATE TABLE. Sie ermöglicht auch gleich die Integritätsbedingungen und Triggermechanismen zu bestimmen. Die Syntax der Anweisung (nach SQL-92) sieht vereinfacht wie folgt aus.

```
CREATE TABLE tabelenname (
    { spalte type [ NULL | NOT NULL ] [ UNIQUE ]
                  [ DEFAULT value ] [ PRIMARY KEY] }
    [, PRIMARY KEY (spaltenlist) ]
    {, CHECK (condition) }
    {, UNIQUE (spaltenliste) }
    {, CONSTRAINT Regelname }
    {, FOREIGN KEY (spaltenlist)
         REFERENCES tabelenname [(splatenliste)]
  [ON DELETE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]]
  [ON UPDATE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]] }
    )
```

Die Domänen der einzelnen Attribute werden durch Datenbank-Typen definiert, die etwa den atomaren Typen der Programmiersprachen C oder Pascal entsprechen. Die Operationen auf diesen Typen sind vorgegeben. Der Benutzer kann allerdings mit CRAETE DOMAIN eigene Domänen durch Einschränkung des Werterbereichs des Ursprungtypen kreieren, die Operationen bleiben den Ursprungstypen immer gleich.

Der Rest der Klausel setzt entweder zahlreiche Integritätsregeln oder definiert bestimmte Aktionen an den Daten. Zur Sicherung der partiellen Integrität werden folgende Klausel benutzt: NOT NULL, UNIQUE, PRIMARY KEY. Sie bestimmen ob ein Tupel gültig für die Tabelle ist, weiterhin wird die PRIMARY KEY Klauseln im Unterschied zu UNIQUE oft als eine Anweisung zur Schaffung des internen Indexes verwendet. Die Klausel für referentielle Integrität ist FOREIGN KEY. Mit ihr wird dafür gesorgt, dass keine Tupel „Waisenkinder“ entstehen, also jeder Fremdschlüssel eine Entsprechung zu einem Primärschlüssel in der referenzierten Tabelle hat. In der gleichen Klausel können die Standardaktionen definiert werden, die bei Löschen oder Änderung des Primärschlüssels (ON DELETE, ON UPDATE) durchgeführt werden sollen. Solche vordefinierten Aktionen ersparen die Arbeit bei der Programmierung der DB-Applikationen, können aber oft zu ungewolltem Löschen von mehreren Tupel führen. Intelligente Datenbanksysteme benutzen auch die Informationen um bestimmte Tabellen, die oft verknüpft werden, in ihrer nähe zu speichern (in einem Cluster). Auch die DEFAULT Klausel stellt eine Vereinfachung für den Datenbankbenutzer dar und definiert den Standardwert eines Attributs.

Mit Hilfe von CHECK oder CONTRAINS Klauseln können allgemeinere Integritäts-Regeln bestimmt werden. Solche Regel können mit Hilfe von SQL Abfragen konstruiert werden z.B

```
CHECK
  (EXIST
     (SELECT * FROM Kurs WHERE Kurs.KursNr=Angebot.KursNr))
```

Allerding sind die Standards-Integritätsregeln wie PRIMARY KEY oder FOREIGN KEY nach Möglichkeit aus Effizienz- und Lesbarkeits-Gründen vorzuziehen.

5.2 Unterschiedliche SQL Standards und Dialekte

Es existieren zwei verabschiedete ISO SQL Standards genannt SQL-86 und SQL-92 oder SQL2. Bereits fertig ist der neue Standard SQL3 (genannt auch SQL-1999). Die angebotenen kommerziellen oder auch freien RDBMS weichen tatsächlich von diesen Standards ab und das in zwei Richtungen.

1. nicht alle Funktionen des Standards werden implementiert oder sind modifi-

ziert. Oft wird die referentielle Integrität mittels FOREIGN KEY nicht unterstützt.[1]. Auch VIEWS und CREATE DOMAIN werden nicht angeboten.

2. um sich von der Konkurrenz abzugrenzen werden weitere Funktionalitäten angeboten. Der Markt der Datenbanksysteme wird von relativ wenigen Firmen beherrscht, darum sind sie in der Lage durch ihre Produkte Quasi-Standards zu schaffen. Solche Quasi-Standards fließen je nach der Mächtigkeit der Firma in die nächste Standardisierung ein.

Hier eine Aufzählung von gängigen Erweiterungen im DDL Bereich.

- Triggermechanismen. Mit CREATE TRIGGER hat man die Möglichkeit selbsdefninierte Aktionen als SQL Anweisungen oder eigene Programme (in funktionalen Sprachen wie C oder PL2) bei bestimmten Vorkommnissen ausführen zu lassen.
- Definition von Regeln. Ein Mechanismus, das alternativ zu Triggern benutzt werden kann.
- Weitere Datentypen wie Enum, Set (mysql RDBMS), Arrays und geometrische Typen (postgresql RDBMS)
- Definieren der eigenen Typen, Funktionen und Operatoren CREATE TYPE, CREATE FUNTKION, CREATE OPERATOR (postgresql RDBMS)
- Kommentieren von Tabellen und Attributen mit COMMENT ON (postgresql, DBII)
- Automatisch generierte Primärschlüssel

Der SQL3 (SQL-1999) Standard definiert neue Eigenschaften, die man in relationale und objektorientierte aufteilen kann. Die genaue Einführung in den neuen Standard bietet [EiMe99], hier nur eine kurze Auflistung der strukturrelevanten Merkmale. Neue relationalorientierte Eigenschaften:

- Neue Datentypen: Large Object (LOB) und Array (Listentyp)

[1] Die bei Implementierung benutzte mysql RDBMS kennt nur PRIMARY KEY als Integritätsregel

- Struktuierte Attribute: ROW Typ
- Trigger Mechanismen. s.g. Aktive Datenbanken

Neue objektorientierte Eigenschaften:

- Strukturierte Typen. Gebaut aus mehreren Basis Typen oder anderen strukturierten Typen. Die neuen Typen können eigene Methoden und Funktionen besitzen.
- Spezielles Identifikationattribut OID, wann eine Tabelle auf der Basis eines strukturierten Typs definiert wurde.
- Referenztyp: Er ermöglicht Konstrukte wie: Vorlesung.Professor− >Name

5.3 Realisierung der höheren Abstraktions-Konzepte mittels SQL-VIEWS

Sichtenkonzept ist ein wichtiger Mittel zur Erhöhung der Datenunabhängigkeit von der Anwendungsprogrammierung. Mit Views können externe Sichten auf ein logisches Schema definiert werden, was der 3-Sichten Architektur entspricht. Man kann folgende Views-Typen klassifizieren, die von der Operation zur Erstellung der Sicht abgeleitet sind:

Projektionssicht Die Attribute einer Relation werden z.B. aus Datenschutzgründen ausgeblendet.

Selektionssicht Die Tupel einer Relation werden nach einer Bedingung ausgefiltert.

Verbundssicht Die Sicht beinhaltet Informationen aus mehreren Tabellen. Die Formulierung von komplexen JOIN Operationen wird versteckt.

Aggregierungssicht Die einzelnen Tupel der Sicht wurden aus mehren Tupel durch Gruppierung und Aggregation der Basisrelation berechnet.

Die Sichten sind auch ein gutes Konzept die Daten auf einer hohen Abstraktionsstufe betrachten zu können (Vergleiche [GoSt99] Abschnitt 3.2.3). Folgende zwei Relationen modellieren eine Spezialisierung durch die Vererbung des Primärschlüssels.

Angestellte : {[[PersNr,Name]}

Professor : {[[PersNr,Rang,Raum]}

Man kann eine Verbundssicht definieren, die die Handhabung der Spezialisierung vereinfacht.

```
CREATE VIEW Professor-Angestellte
AS SELECT PersNr,Name,Rang,Raum
FROM Angestellte,Professor
WHERE Angestellte.PersNr=Professor.PersNr
```

Es ist vorstellbar, die Sichten so zu definieren, dass die Daten auf immer höheren Abstraktionsniveau präsentiert werden. Das ist aber wenig sinnvoll, weil die Sichten immer noch die flache Tabellenform haben müssen, obwohl die Daten eine andere Struktur haben könnten. Durch die Benutzung von Verbundssichten werden die Probleme (vor allem Datenredundanz) wieder sichtbar, die man in dem Normalisierungsprozess zu beheben versuchte.

Im Idealfall sollte eine Sicht für den Benutzer genau wie eine normale Tabelle zu verwenden sein. Die zugrundeliegenden Operationen für die Erstellung der Sicht bleiben transparent. Die Umsetzung der Operationen wie UPDATE oder INSERT auf die Basistabellen sind technisch schwierig oder oft nicht machbar (bei Aggregierungssichten), so dass die Sichten oft nur zum Auslesen der Daten zu verwenden sind.

5.4 Abfragen der Datenstruktur mittels SQL

Die Informationen über das Schema der Datenbank werden im Data Dictionary gespeichert. Das Data Dictionary wird auch Data Katalog, System Katalog oder Repository genannt. Die Struktur des Data Dictionary wird Meta-Schema genannt. Die Informationen über den Aufbau der Relationen werden bei RDBMS auch in Form von Tabellen verwaltet. Leider ist das Schema des Data Dictionary (Meta-Schema) nicht standardisiert. Deswegen gibt es keine standardisierte Methode das Schema abzufragen und zu deuten. Nur die generellen Schnittstellen wie ODBC oder JDBC besitzen standardisierte Methoden für solche Aufgaben, die in spezifische SQL Anweisungen der RDBMS übersetzt werden. Manche RDMS wie MySql bieten spezielle Anweisungen wie SHOW COLUMNS FROM und DESCRIBE TABLE für Abfragen des Schemas an, bei anderen Systemen müssen die speziellen Systemtabellen mit normalen SELECT-Anweisungen ausgewertet werden. Hier zwei Beispiele der Abfrage des Data Dictionaries.

```
mysql>
SHOW COLUMS FROM Person;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| IDPersons  | int(11)     |      | PRI | 0       | auto_increment |
| Vorname    | varchar(20) | YES  |     | NULL    |                |
| Nachname   | varchar(20) | YES  |     | NULL    |                |
| Adresse    | varchar(30) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
Postgresql>
select attname,typname,attnotnull,atttypmod,attlen
  from pg_class,pg_attribute,pg_type
  where pg_class.relname='Person' and
        pg_class.oid=pg_attribute.attrelid and
        pg_attribute.atttypid=pg_type.oid and attnum>0;
attname          |typname|attnotnull|atttypmod|attlen
-----------------+-------+----------+---------+------
idpersons        |int4   |t         |        0|     4
vorname          |varchar|f         |       24|    -1
nachname         |varchar|f         |       24|    -1
adresse          |varchar|f         |       34|    -1
```

Viele Strukturinformationen aus der konzeptionellen Modellierung finden den Eingang in das relationale Schema nur in Form von unterschiedlichen Integritätsbedingungen und Namensgebungen für Tabellen und Attribute. Zwar bieten fast alle RDBMS mit Hilfe der COMMENT-Anweisung, die Möglichkeit das Schema genauer zu beschreiben, aber diese Beschreibung ist primär für den Menschen bestimmt. Die Integritätsbedingungen, die als PRIMARY KEY oder FOREIGN KEY definiert wurden, sind alledings auch maschinell gut zu deuten. Anders verhält es sich mit Bedingungen die mit Hilfe von SQL oder prozeduralen Sprachen als Trigger kodiert wurden. In solchen Fall ist das ursprüngliche konzeptionelle Schema maschinell nicht zu rekonstruieren, auch für einen Menschen ist das schwer ohne eine vollständiger Dokumentation.

6 Anforderungsanalyse und Spezifikation des Systems

Im praktischen Teil der Diplomarbeit werden die Ideen aus dem theoretischen Teil als Beispielimplementierung umgesetzt. Die primär Aufgabe der Implementierung ist es einen Prototypen zu schaffen, der die Realisierbarkeit neuer Ansätzte beweisen soll und eine Testumgebung liefert. Weil keine reale Benutzer des Systems existieren, wird bei der Anforderungsanalyse auf die Vergleiche mit ähnlichen Systemen, theoretische Überlegungen und die Erfahrung des Autors zurückgegriffen. Das Hauptaugenmerk soll dabei auf die neuen Ansätzte gerichtet werden. Gleichzeitig wird hier eine Reihe von implementierungsspezifischen Begriffen definiert. Bei der Implementierung wird ein bewertes Phasenmodell benutzt, mit Aufteilung: Analyse, Entwurf und Implementierung. Die Analyse betrifft ein allgemeines System der DB-Applikationen, das jedoch in weiteren Phasen auf die DB-Formulare beschränkt wird. Die Anforderungsanalyse beginnt mit der Klassifizierung der Benutzer. Als nächstes werden die neuen Ansätze vorgestellt. Zum Schluß wird eine Liste mit Funktionalitäten vorgestellt, die von dem System erfühlt wird.

6.1 Benutzer und Endbenutzer

Bei der Entwicklung der DB-Applikationen werden 3 Hauptgruppen der Benutzer involviert, die über unterschiedliche Kenntnisse verfügen (siehe 6.1). Oft kommt zusätzlich neben dem DB-Administrator auch der Datenbankdesigner vor, der haupt-

Benutzergruppe	Datenstruktur konzep. Modell	Programmierung DB-Werkzeuge	Daten Inhalte
DB-Administrator	X		
Anwendungs-programmierer		X	
Endbenutzer			X

Tabelle 6.1: Benutzerkenntnisse bei dem Datenbankentwurf

sächlich für das konzeptionelle Modell zuständig ist. Bei traditionellen Datenbankentwurf (siehe [HeSa95] Kapitel 4) ist die Datenbankstruktur (Schema) bei der Realisierung der DB-Applkationen bereits vorgegeben. Um so mehr trifft es zu, wenn DB-Applikationen nachträglich zu den existierenden Datenbanksystemen entwickelt werden. Das bedeutet jedoch nicht, das Datenbanken völlig losgelöst von Erfordernissen der Applikationen entworfen werden. Man versucht durch die Analyse der Geschäftsprozesse eine universelle integrierte Datenbankstruktur zu schaffen, die auch für zukünftige, noch nicht bekannte Anwendungen gültig wäre. In Abbildung 6.1 wird der Ablauf der Entwicklung der DB-Applikation gezeigt. Zuerst

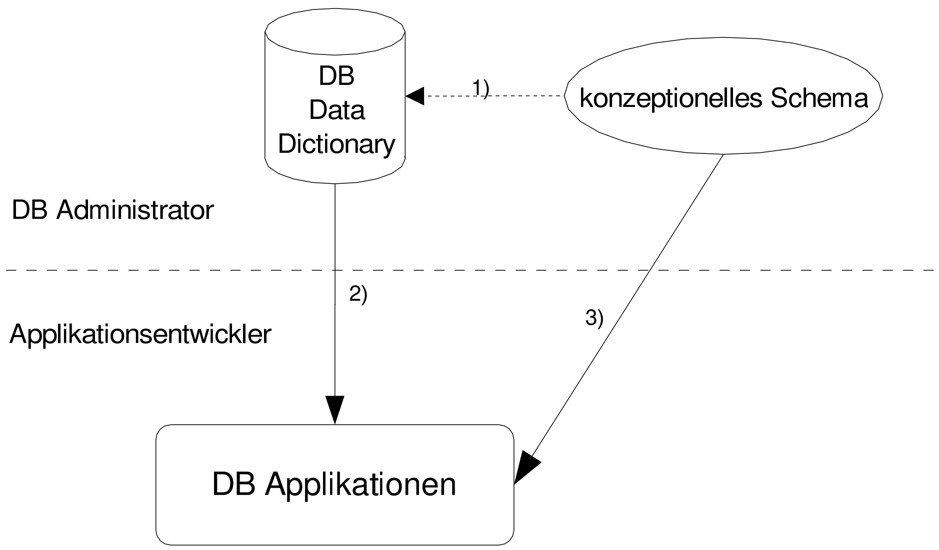

Abbildung 6.1: Informationsfluss über konzeptionelles Modell bei der Entwicklung der DB-Applikationen

wird ein konzeptionelles Schema der Daten von einem Datenbankdesigner erstellt. Es existiert in Form einer Dokumentation als ein Text oder auch als eine Graphik des ER-Modells. Danach wird das konzeptionelle Schema in ein relationales Schema transferiert (Pfeil 1), dadurch gehen viele semantische Informationen verloren (vergleiche Kapitel 3.2). Im nächsten Schritt findet der tatsächliche Entwurf der DB-Applikationen statt. Ein Applikations-Entwickler hat hier eine besonders schwere Aufgabe. Um einen korrekten Datentransfer zwischen der Applikation und dem Datenbanksystem zu gewährleisten, muss er die Data Dictionary der Datenbank auswerten (Pfeil 2) und um die Informationen gut zu verarbeiten und darstellen zu können, muss er auch auf das konzeptionelle Modell zurückgreifen (Pfeil 3). Zusätzlich müssen die Wünsche der Endbenutzer berücksichtigt werden. Die Endbenutzer haben oft genaue Vorstellungen über die Benutzerschnittstelle und die Funktionalitäten der Applikationen, die formale Datenstruktur ist für sie jedoch unbekannt.

Zu unterstreichen ist der Umstand, dass der Applikationsentwickler sich sehr intensiv mit dem konzeptionellen Modell auseinandersetzten muss. Das kann besonders schwierig sein, wenn die Daten einen spezifischen Charakter haben (chemische oder physische Daten, technische Messwerte). Die Applikationsentwicklung ist oft das teuerste Element der Informationsysteme, deswegen wächst die Bedeutung der Werkzeuge für automatische Generierung der DB-Applikationen unerheblich. Diese Werkzeuge können aber nur die Information benutzen, die in einer maschinenlesbaren Form vorliegen. Hier handelt es sich um Data Dictionary (relationales Schema der Daten). Das konzeptionelle Schema ist nur ein Halbprodukt des DB-Entwurfs und als Dokumentation nur von Menschen auswertbar. Das hat zur Folge, dass das Ergebnis von solchen Werkzeugen noch einer aufwendigen manuellen Anpassung bedarf.

6.2 Neue Ansätzte der Implementierung

Repository

Der Motor dieser Diplomarbeit ist die Idee einer „Vereinigung“ von dem konzeptionellen und dem logischen Modell zu einer maschinenlesbaren Form. Das heißt nicht, dass auf das konzeptionelle und das logische Modell verzichtet wird. Es soll eine Zwischenschicht genannt $d$as Repository entstehen, das die Informationen über relationale DB-Schema erhält und zusätzlich auch Hinwiese wie dieses Schema interpretiert werden soll. Dieses Repository bietet einen großen Vorteil, weil es durch verschiedene Werkzeuge maschinell ausgewertet werden kann. Dadurch kann die automatische Generierung der DB-Applikation viel bessere Resultate liefern. Es entlastet den Applikationsentwickler um die Aufgabe sich mit dem konzeptionellen Modell zu beschäftigen. Die Abbildung 6.2 zeigt die Entwicklung der DB-Applikationen bei Einbeziehung des Repositories. In disem Fall, wird ein Repository nachträglich zu

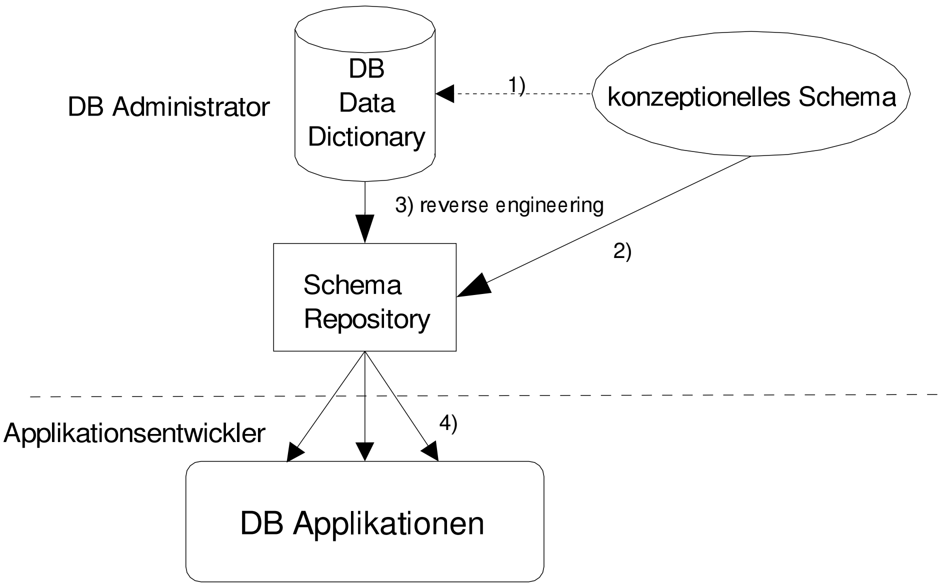

Abbildung 6.2: Informationsfluss bei Einbeziehung des Repositories

existierenden Datenbank erstellt. Dabei wird zuerst mit Hilfe von Reverse Engineering Techniken ein Grundgerüst des Repositories erstellt (Pfeil 3), das um weiter semantische Informationen ergänzt werden kann (Pfeil 2) . In dieser Phase können auch weitere Meta-Informationen in das Repository eingehen. Zu bemerken ist, dass das Aufstellen des Repositories zur Aufgaben des DB-Administrators gehört. Basierend auf dem Repository erstellen die Generatoren automatisch die gewünschten DB-Applikationen (Pfeile 4)

Ein anderes Bild ensteht (s. Abbildung 6.3), wenn man vom Anfang des DB-Entwurfs das Repository einbezieht.

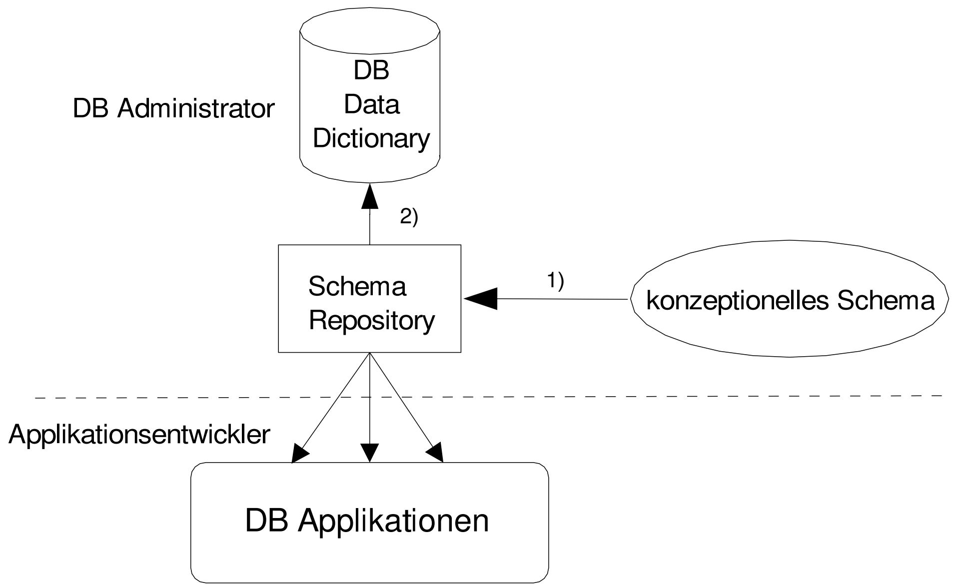

Abbildung 6.3: Repository als zentrale Stelle des DB-Entwurfs

Weil das Repository mehr strukturelle Informationen als das relationale Schema enthält, ist eine automatische Generierung des relationalen Schema aus dem Repository denkbar (Pfeil 2). In diesem Fall ist das Repository eine zentrale Stelle des Entwurfs der Datenbank (Pfeil 1).

Es werden folgende Anforderungen an das Repository gestellt:

1. maschinell lesbar
2. beinhaltet das logische Schema (Aufbau der Relationen)
3. beinhaltet die Konzepte der semantischen Modelle (höhere Abstraktionen wie Spezialisierung, Aggregation und Assoziation)
4. erweiterbar (es können leicht weitere Meta-Informationen hinzugefügt werden)
5. flexibel und universell (Es wird keine konkrete Benutzung vorausgesetzt)

Exkurs zum Begriff des Repositories

Der Gebrach des Begriffs Repository (oder zu deutsch Repositorium) ist noch nicht verfestigt. Es liegt nah oder wird auch als Synonym verwendet für: Data Dictionay, Entwiklungsdatenbank, Information Resource Dictionary System, Data Katalog, Metainformationssystem. In der Wirtschaftsinformatik wird das Repository sehr allgemein verstanden als, computerunterstütztes Informationssystem über die Informationsverarbeitung eines Unternehmens (s. [Ortn99]). Es sind Dokumentationssysteme, die die Daten und Prozesse des Informationssystem beschreiben. Als Repositories werden auch Verwaltungssysteme für Schematas bezeichnet (besonders XML-DTD Dokumente).

In der Diplomarbeit wird als Data Directory ein Teil des RDBMS verstanden, der das relationele Schema verwaltet, das auch dort abgefragt werden kann. Das Repository wird hier mehr als ein Dokument, die Beschreibung der Daten, verwendet. Es beschreibt die Daten auf einer höheren Abstraktionsstufe als relationales Modell, ohne das die Details über das relationale Schema verloren gehen. Konkret beinhaltet das Repository genügend Informationen um gültige SQL-Anweisungen daraus zu formulieren. Man kann auch solche Art des Repositories zu s.g. Schnittstellen-Repositories zählen. Sie beschreiben die Daten und auch wie auf diese Daten zugegriffen werden kann.

In [Bern99], das mit dem Produkt MS Repository der Firma Microsoft verbunden ist, wird das Repository als ein System für peresistente Haltung von Meta-Daten beschrieben. Dabei wird der Schwerpunkt auf die entwiklungsrelevanten Daten gelegt wie: DB-Schematas, Formularbeschreibungen, Quelltexten, Ausführbare Dateien und Hilfedateien. Das MS Repository bietet Funktionen für Versionifiezirung, Konfigurationmanagment und Workflowmanagment. Die Meta-Informationen werden ein einem objektorientierten Modell beschrieben. Der Zugriff auf die einzelnen Objekte im Repository erforlgt durch COM-Schnittstelle (Component Object Model).

Realisierung der Aggregation, Assoziation und Vererbung in den Formularen

Wie in Kapitel 2.4 beschrieben wurde, können die Formulare mehr strukturierte Informationen als Tabellen darstellen. Die Information muss nicht eine flache Tabellenform haben. Darum wäre es denkbar die Formulare als Darstellung der Informationen aus mehreren Tabellen zu benutzten. Ein Formular wäre dann mehr eine Entsprechung einer Sicht (View) aus dem konzeptionellen Modell als einer Tabelle aus dem logischen Modell. Allerdings ohne den Nachteil, dass die SQL-Views immer noch die Tabellenform haben.

Die Aggregation könnte durch eingebettete Formulare realisiert werden. Die Vererbung könnte als ein variables (veränderliches) Formular dargestellt werden. Zwar bieten die existierenden Systeme ähnliche Funktionalitäten, aber sie müssen immer noch manuell erstellt werden. Durch die Verwendung des Repositories könnten solche komplexen Formularen automatisch generiert werden, sofort mit der Entsprechung zum konzeptionellen Schema. Ein Folmular wäre dann als eine Darstellung eines Objekts und nicht einer Tabelle zu verstehen.

Navigieren durch Formulare, Formular-Links

Fast in allen verfügbaren Systemen wird ein Datebank-Formular als eine spezielle Repräsentation einer konkreten Tabelle verstanden, die für die Eingabe und das Auslesen der Daten benutzt wird. In der Implementierung wird ein anderer Weg eingeschlagen. Alle Formulare die zu einer Datenbank gehören werden zusammenhängend erzeugt. Jedes Formular wird mit einem anderen mit Hilfe von speziellen Links (Navigations Hilfen) entlang der Beziehungspfäden (Assoziationen) verknüpft. Dadurch wäre es möglich die Formulare zum Navigieren durch das ganze Datenbanksystem zu benutzen, ohne das das Datenbankschema bekannt sein muss. Durch die Navigationshilfen wäre jede assoziierte Information zu einem Objekt aus seinem Formular erreichbar.

6.3 Grundfunktionen des Systems

Dieses Kapitel spezifiziert alle Grundeigenschaften, die von dem System erfüllt sein müssen. Die Anforderungen bezüglich der Effizienz (Performance) und Ressourcenverbrauch werden nicht berücksichtigt, weil es sich um ein Prototypensystem handelt.

Das Hauptziel der Implementierung ist ein System zu entwickeln, das eine „einfache" Erstellung der Datenbank-Formulare zu eine existierenden relationalen Datenbank ermöglicht. Die Einfachheit zeigt sich dadurch, dass sowohl keine Programmierkenntnisse bei dem Benutzer vorausgesetzt werden, als auch die Erstellung der Formulare so weit wie möglich automatisch erfolgt. Im Idealfall müssen die automatisch erstellten Formulare von dem Benutzer verfeinert oder gestalterisch angepasst werden. Der Prozess der Erstellung der Formulare wird entsprechend der Abbildung 6.2 in zwei Stuffen geteilt. Zuerst wird das relationale Schema mit Hilfe von Reverse-Engineering Techniken um weitere semantische Informationen ergänzt. Dabei werden allgeimene Abstraktionen wie Assoziation, Aggregation und Vererbung unterstützt. Zusätzlich können Meta-Informationen zum Schema angegeben werden.

Die entstandenen DB-Formulare besitzen folgente Funktionalitäten:

1. Anlegen, Löschen und Modefizieren der Daten mit Hilfe von Formularen
2. Daten Ansehen und durch Daten Naviegieren (Formular-Links)
3. Implementierung von Aggregation, Assoziation und Vererbung in Formularen
4. Korrekte und benutzerfreundliche Anzeige von Fremdschlüsseln

7 Entwurf des Systems

Bei dem Entwurf des Systems werden zuerst die Grundziele der Entwicklung formuliert, danach werden die Systemkomponenten benannt und die Datenflüsse zwischen ihnen gezeichnet. Auch die Schnittstellen und Funktionen der einzelnen Systemkomponenten werden definiert. Auf die komplette Ausschreibung des Entwurfs wird aus Platzgründen verzichtet. Besonderheiten und neue Ansätzte werden beispielhaft gegeben.

7.1 Gesamtkonzept und Entwurfsgrundsätze

Das Ziel ist es, ein offenes System für die Entwicklung der Datenbank-Applikationen zu entwerfen. Dabei wird angenommen, dass nicht alle möglichen Datenbank-Applikationen bei dem Entwurf bekannt sein müssen. Die Entwicklung der DB-Applikationen zur Herstellung von Datenbankformularen, die auch vollständig implementiert wird, findet besondere Aufmerksamkeit. Prinzipiell wäre es möglich, durch das spätere Hinzufügen von weitere Systemkomponenten, leicht andere DB-Applikationen zu realisieren wie z.B Berichtsgenerator oder Abfrageeditor. Diese Offenheit wird durch folgende Entwurfsgrundsätze ermöglicht:

- Es wird angenommen, dass der Prozess der Entwicklung der DB-Applikationen stufenweise erfolgt. Für jede Stufe wird eine andere Komponente verwendet, die möglichst abstrakte und universelle Ergebnisse liefert. Es werden keine besonderen Annahmen über die Verwendbarkeit der Ergebnisse gemacht.

- Die Schnittstelle zwischen den Komponenten wird möglichst universell und offen gewählt. Zwischen den Komponenten werden lediglich verschiedene XML Dateien ausgetauscht. Das XML Format garantiert folgende nützliche Eigenschaften
  - von Menschen lesbar und editierbar
  - seine Syntax kann als DTD (Document Type Definition) oder seine Nachfolger (XML-Schema) definiert werden und von den zahlreichen Werkzeugen auf Korrektheit überprüft werden.
  - sehr flexibel in der Handhabung.
  - leicht erweiterbar.
  - speziele Programmierbibliotheken und Programmierwerkzuege beschleunigen und vereinfachen die Entwicklung.
  - Anerkennung und Aktualität der Technik
- Die einzelnen Komponenten sind getrennte Programme. Bei der Entwicklung der DB-Formulare (oder anderen DB-Applikationen) werden sie nacheinander von unterschiedlichen Personen ausgeführt. Das ermöglicht die Anpassung der einzelnen Benutzerschnittstellen auf konkrete Benutzergruppe und verhindert das Entstehen einer ressourceintensiven Riesenapplikation.
- Die einzelnen Komponenten sind leicht austauschbar. Eine Komponente genügt der Schnittstelle, wenn sie ein XML-Dokument richtig bearbeiten kann. Die Bearbeitung von XML-Dokumenten ist mit keiner besonderen Schnittstelle, Plattform oder Programmiersprache verbunden.

7.2 Systemkomponenten, Grobarchitektur

Das System besteht aus drei ausführbaren Programmen: *Schema-Editor, Formular-Editor und Formular-Server* und zwei XML-Dokumententypen (Repository und eigentliche Beschreibung der Formulare). Die Grobarchitektur zeigt die Abbildung 7.1. Die einzelnen Komponenten werden in den nachfolgenden Unterkapiteln beschrieben.

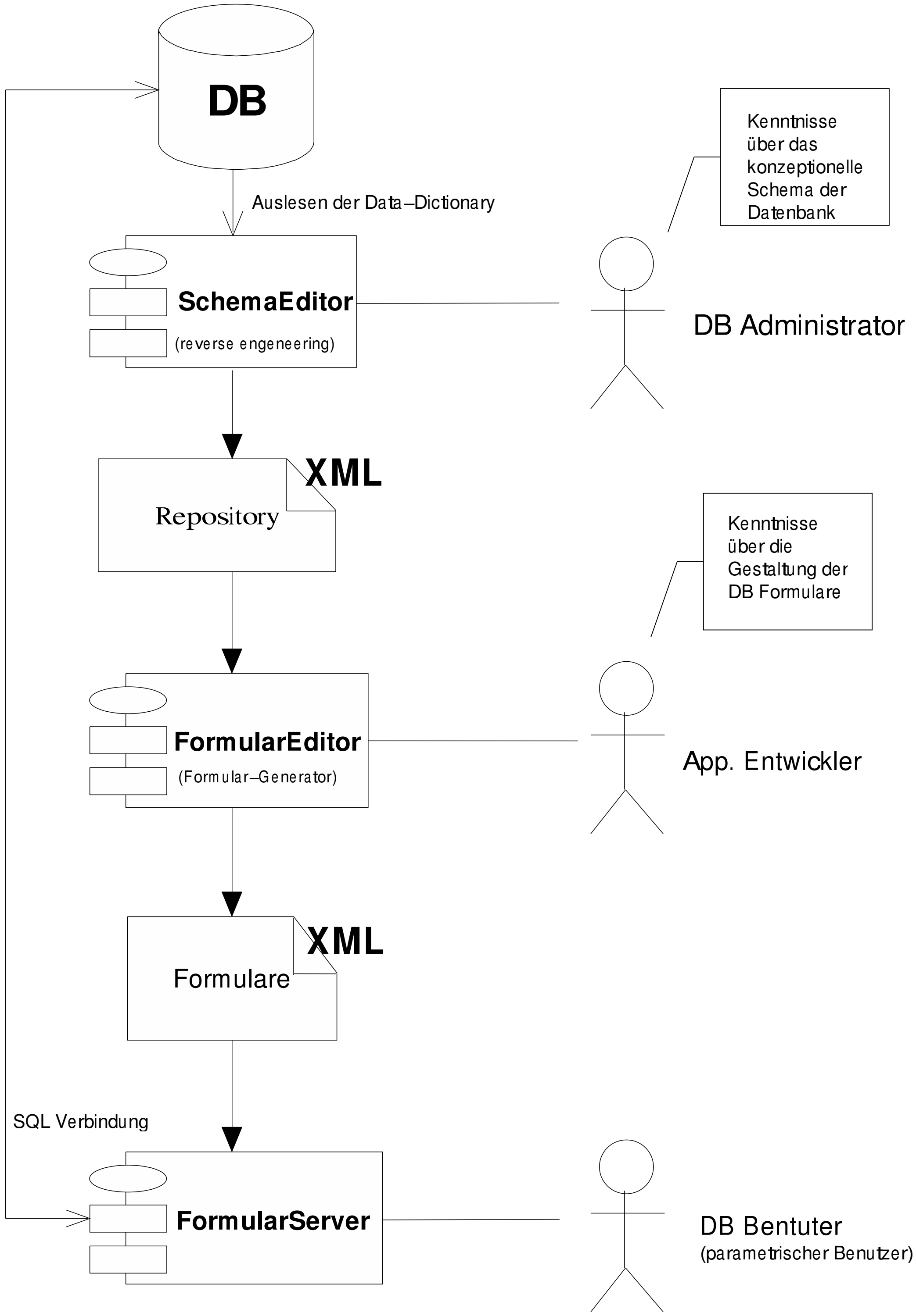

Abbildung 7.1: Systemarchitektur

7.2.1 XML-Repository

Die zentrale Rolle im System nimmt das Repository als eine XML-Datei ein. Das Repository kann als Grundlage für die anderen DB-Applikationen oder die Entwicklung von anderen DB-Applikationen dienen. Auf jeden Fall bieten die semantischen Informationen des Repositories einen Vorteil bei bei der automatischen Auswertung des Schemas der Datenbank.

Das Repository hat folgende Funktionen:

- Das Repository soll alle Informationen über das relationale Schema enthalten. Es wäre ein Ersatz für das Data Dictionary des RDBMS. Vor allen sollen die Namen der einzelnen Tabellen, die Namen der Attribute und ihre Typen kodiert werden.
- Die Informationen, wie das relationale Schema interpretiert werden soll. Folgende semantische Informationen werden unterstützt:
  - Information, ob eine Tabelle ein Objekt (Entity) oder Relationship modelliert
  - Kardinalitäten der Beziehungen (Relationship)
  - Weak-Entities. Aggregation von Tabellen
  - Attributgruppen (strukturierte Attribute)
  - Modellierung der Vererbung (Vererbung der Schlüssel). Art der Spezialisierung
  - *Etiketten* der Objekte sind formal ein Schlüsselkandidat und dienen als eine benutzerfreundliche Alternative zum Primärschlüssel. (z.B.. Name und Vorname statt PersonenID)
- Meta-Daten.
  - Beschreibung der Daten als Kommentar
  - Weitere Untertypenbildung, die von der Datenbank nicht angeboten wird und mit Hilfe von BLOB modelliert wird. (z.B HTML-Dateien, Graphik-Formate). Spezifizierung mit Hilfe von standardisierten MIME-Typen
  - Ausgabe und Eingabe Format (z.B Datum als 01-Jan-2000)
  - Gewichtung bei Ausgabe (z.B. immer Ausgeben (Name des Mitarbeiters) Ausgabe nur auf explizites Verlangen (Lebenslauf des Mitarbeiters)).

  – Bei Zahlen. Angaben, die die statistische Auswertung vereinfachen: Verhältnisszahl, ordinalskalierte Zahl, Aggregat

Der Entwurf des Repositories, reduziert sich hier zum Entwurf des Informationsmodells für die Speicherung des DB-Schemas und seiner Repräsentation als ein XML-Dokument. Als Grundlage für das Informationsmodell wird ein relationales Modell genommen, das um weitere semantische Informationen erweitert wird. Diese zusätzlichen Informationen stufen das Repository zum konzeptionellen Modellen ein, trotzdem bleibt die Verbindung zum relationalen Modell erhalten. Deswegen kann hier bereits von Objekten gesprochen werden. Das wird ermöglicht durch die Annahme, dass eine Tabelle (Relation) immer die Entsprechung eines Konzepts und nur eines Konzepts darstellt. Beim relationalen Schema, das wenigsten der 3. Normal-Form entsprechen, sollte es immer der Fall sein. Für das Repository können die Relationen folgende semantische Bedeutung haben:

- Eine Relation repräsentiert ein Objekt, das durch einen Primärschlüssel identifiziert werden kann.
- Eine Relation modelliert eine Assoziation. Die Tabelle besitzt wenigsten zwei Fremdschlüssel auf Objekte (Objekttabellen). Zusätzliche Attribute gehören zu den Eigenschaften der Assoziation.
- Eine Relation modelliert die Spezialisierung. Sie beinhaltet nur Teilinformationen über ein Objekt. Der Primärschlüssel wird von einer anderen Relation geerbt.
- Ein Objekt kann mehrere Objekte eines anderen Typs aggregieren. Das führt zu s.g. geschachtelten Relationen, ähnlich des NF^2-Modells. Die aggregierten Objekte besitzen einen Fremdschlüssel des Containers.

Zur Repräsentation des Repositories wird eine Baumstruktur gewählt, die auch die XML-Dokumente charakterisiert. Ein relationales Schema kann man leicht als einen Baum der Tiefe 2 darstellen. Jeder Knoten der Tiefe 1 repräsentiert eine Tabelle und die Blätter bilden die Attribute ab. Bei dem Repository wurden weitere Knotentypen hinzugefügt, die die Assoziation oder Meta-Informationen darstellen. Ein Knoten des

Typs Tabelle kann auch weitere Zweige des Typs Tabelle haben. Dadurch wird die Spezialisierung und Aggregation der Objekte (geschachtelte Relationen) modelliert. Jedes Attribut kann weitere Meta-Informationen haben, die durch einen Teilbaum der Tiefe 2 wiedergeben werden. Problematisch ist die Abbildung von Assoziationen. Eine Assoziation wird als Container (Sammlung) mit Verweisen auf Objekte aufgefasst. Die Sammlung kann entweder als eine getrennte Tabelle mit Verweisen oder als ein Verweis in einem Objekt (1:n) modelliert werden. Um eine Assoziation zu modellieren, werden zwei neue Knotentypen hinzugefügt: *Assozitaionscontainer* bei Verweisen und *Assoziationstarget* bei Objekten. Zu jeder Assoziation gehört ein Assoziationcontainer und mindestens zwei Assoziatontargets. Die Assoziationen (Assoziationcontainer) besitzen einen eindeutigen Namen. Auf diese Weise können auch komplexe Assoziationen modelliert werden, wie z.B.: rekursive 1:n und n:m Beziehungen und Beziehungen der höheren Granulität n:m:s:r. Wie ein relationales Schema in ein Repository umgewandelt wird, ist anhand eines Beispiels in Abbschnitt 8.4 beschrieben.

Die Syntax des Repositories wird in einem DTD-Dokument festgehalten, siehe Anhang Seite 88. Es muss darauf hingewiesen werden, dass nicht alle gültigen Dokumente nach XML-Definition gültige Repositories sind. Die DTD erlauben nur die Festlegung von kontextfreien Regel. Es kann z.B. im DTD nicht spezifiziert werden, dass ein Attributname in einer Tabelle eindeutig sein muss.

7.2.2 Schema-Editor

Die Komponente Schema-Editor wird als ein getrenntes Programm realisiert. Die Hauptaufgabe des Schema-Editors ist eine XML-Repository (s. Abschnitt 7.2.1) zu schaffen und zu verwalten. Es wird vorausgesetzt, dass ein Benutzer des Schama-Editors, ein Datenbank-Administrator oder auch ein Daten-Verwalter ist. Der Benutzer hat Kenntnisse über die Struktur der Daten und ihre Repräsentation im relationalen Modell. Der Schema-Editor besitzt folgende Funktionalitäten:

- Das Data Dictionary einer relationalen Datenbank auszulesen und in XML-Repository umzuwandeln.

- Hinzufügen von semantischen Informationen zum Repository. Dabei werden die Reverse Engineering Techniken (siehe. 3.3) benutzt, die entweder ganz automatisch oder durch Befragung des Benutzers verlaufen. Der Benutzer kann auch die semantischen Informationen selber hinzufügen.

- Hinzufügen und Editieren der Meta-Informationen von Attributen.

Die Art der Schema-Informationen, die von dem Benutzer verwaltet werden können, ist durch den Aufbau des XML-Repositories (siehe Abschnitt 7.2.1) vorgegeben. Die Angabe von semantischen Informationen sollte in Form von Befragung (Assistenten) durchgeführt werden.

Die Idee der graphischen Oberfläche basiert auf der Darstellung des Schemas als eine Baumstruktur, die von dem Benutzer manipuliert werden kann (s. Abb. 7.2). Die Darstellung des relationalen Schemas als einen Baum ist auch theoretisch begründet. P. Kandzia und H.J. Klein zeigen in ihrem Buch [KaKl93] Kapitel 11, dass das relationale Modell sich als ein Baum der Tiefe 2 darstellen lässt. Auch XML-Dateien und hier das Repository speziell haben eine Baumstruktur. Wegen der zusätzlichen Informationen in Repository sind weitere Knotentypen zu verzeichnen.

- Tabelle (Blatt und Knoten)

- Attribut (Blatt)

- Assoziation

- Tabellen-Etikette

- Attribut-Gruppen (Knoten, die nur Attribute als Blätter haben)

Weil das Repository die Aggregation in zwei Formen unterstützt, kann die Tiefe des Baumes mehr als 2 betragen. Tabellen können Tabellen oder Attribut-Gruppen enthalten. Die speziellen Tabellen die eine Spezialisierung oder Beziehungen (n:m) repräsentieren werden optisch besonders gekennzeichnet. Auch wichtige Attribute, die Primärschlüssel, werden durch modifizierte Symbole repräsentiert. Die Verwaltung des Repository wird als Operation auf dem Baum abgebildet. Die einzelnen

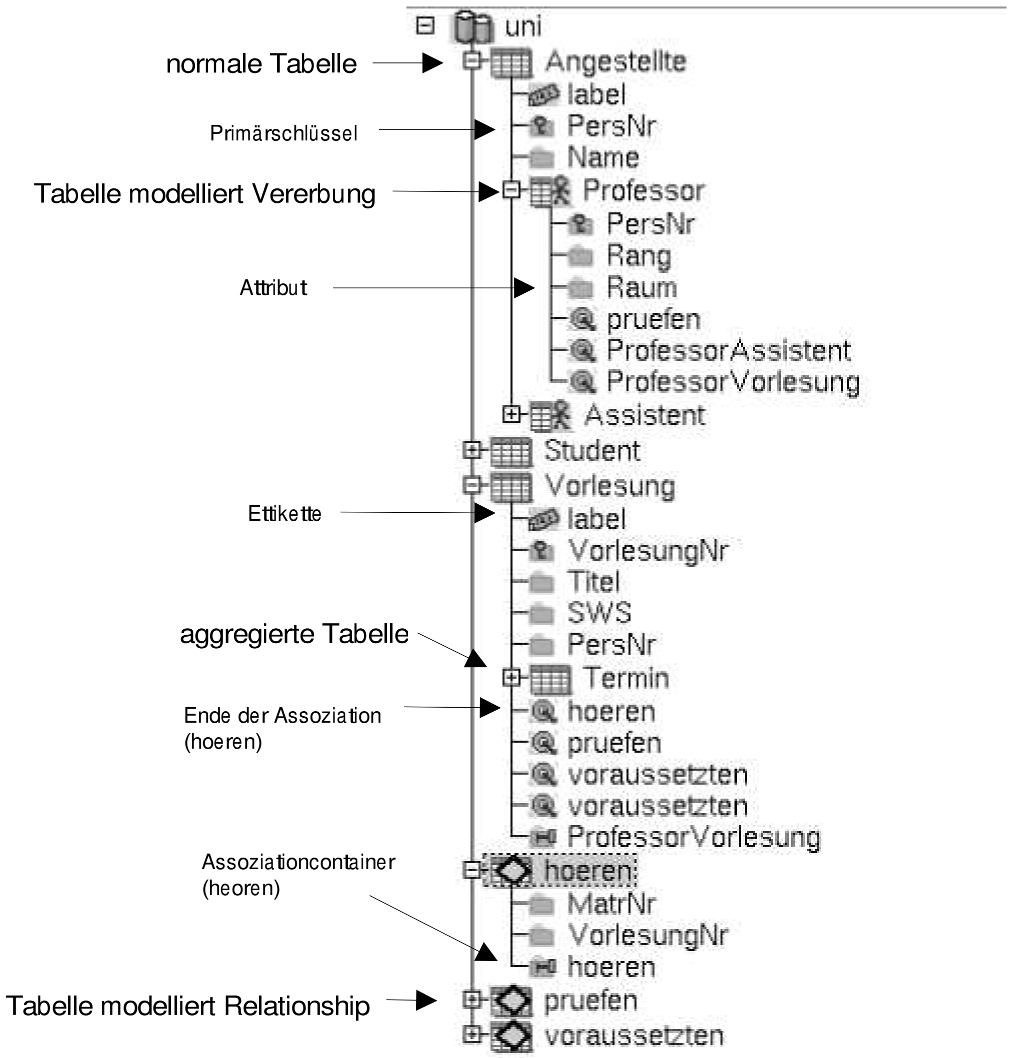

Abbildung 7.2: Darstellung des Schemas als Baum im Schema-Editor

Eigenschaften der Knoten des Baumes werden durch spezielle Dialoge vom Benutzer editiert.

7.2.3 Formular-Editor

Der Zweck der Komponente ist es zuerst, eine Sammlung der Formulare zu einer Datenbank zu generieren, und zweitens eine Benutzerschnittstelle für die Anpassung der Formulare zu liefern. Das Produkt des Formular-Editors ist eine Beschreibung der Formulare als ein XML-Dokument. Dabei wird im Gegenteil zu vorhandenen ähnlichen Systemen nicht einzeln zu jeder Tabelle ein Formular generiert, sondern wird eine ganze Sammlung von zusammenhängenden Formularen in einem Schritt erstellt.

Der Benutzer des Formulars-Editor (Applikations-Entwickler) muss das Schema der Daten nicht kennen. Seine Aufgabe ist es höchstens, die automatisch erzeugten Formulare unter den gestalterischen Gesichtpunkten anzupassen, oder die Eingabefelder des Formulars noch weiter zu spezifizieren.

Ein Formular wird als eine Sammlung von verschiedenen Eingabefeldern, ihrer Platzierung, und ihrer Entsprechung zu Objekten in der Datenbank verstanden. Folgende Typen von Eingabefeldern werden realisiert:

- einfache Textfelder (einzeilig)
- Eingabefelder für Ganzzahlen
- Listenfeldery
- Radiobuttons
- Checkbox Felder (Ja/Nein Schalter)
- mehrzeilige Textfelder
- Navigationslinks zwischen den Formularen
- Mehrspaltige Auswahllisten für die Darstellung der Fremdschlüssel (Referenzen)

Zu den speziellen Elementen der Formulare gehören die eingebetteten Formulare und die Formular-Links (s. Abb. 7.3). Beide sind verschiedene Visualisierungen der Formularverknüpfungen. Die Verknüpfungen entstehen entlang der Beziehungs-Pfäden (Assoziation oder auch Aggregation). Dabei kann man aus einem Formular ein Formular des Objekts erreichen, das einen Fremdschlüssel auf das erste Formular hat. Dabei wird bei dem verknüpften Formular der Fremdschlüssel ausgeblendet (der Wert wird von ersten Formular bestimmt) und die Daten (Tupel) werden nach dem Fremdschlüssel gefiltert. Es werden durch die Filterung nur die Objekte gezeigt, die mit dem gezeigten Objekt in dem ersten Formular in Verbindung stehen. Bei eingebetteten Formularen ist das verknüpfte Formular ein Teil des Elternformulars. Bei Formular-Links wird eine Schaltfläche eingebaut, die erst zum Öffnen des verknüpften Formulars dient.

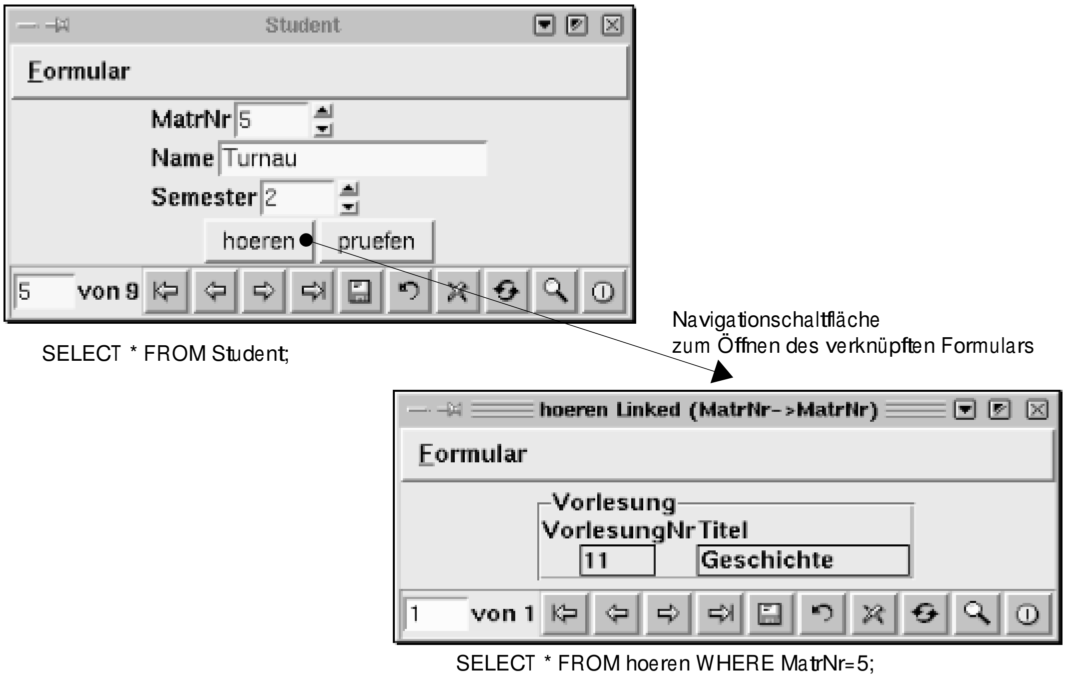

Abbildung 7.3: Verknüpfung der Formulare durch Formular-Links

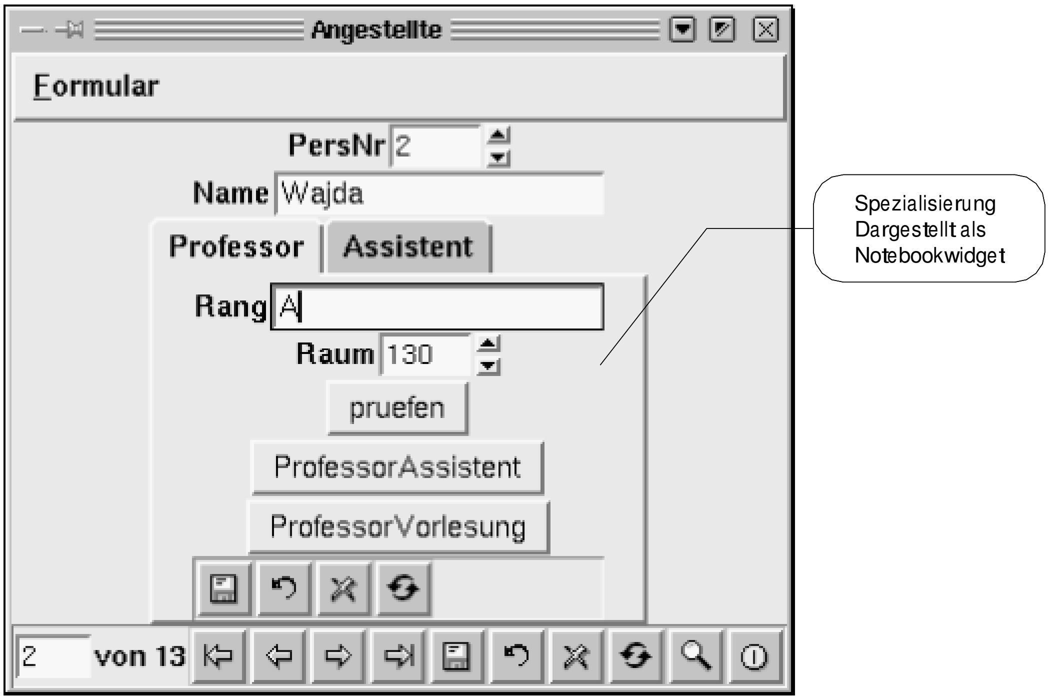

Abbildung 7.4: Anzeigen von Spezialisierung (Vererbung) in Formularen

Die Fremdschlüssel werden nicht als normale Attribute behandelt, sondern als spezielle graphische Elemente dargestellt (s. Abb. 7.5). Dabei sind zwei Tatsachen zu berücksichtigen. Zuerst ist der Wertebereich des Fremdschlüssels durch Integritäts-

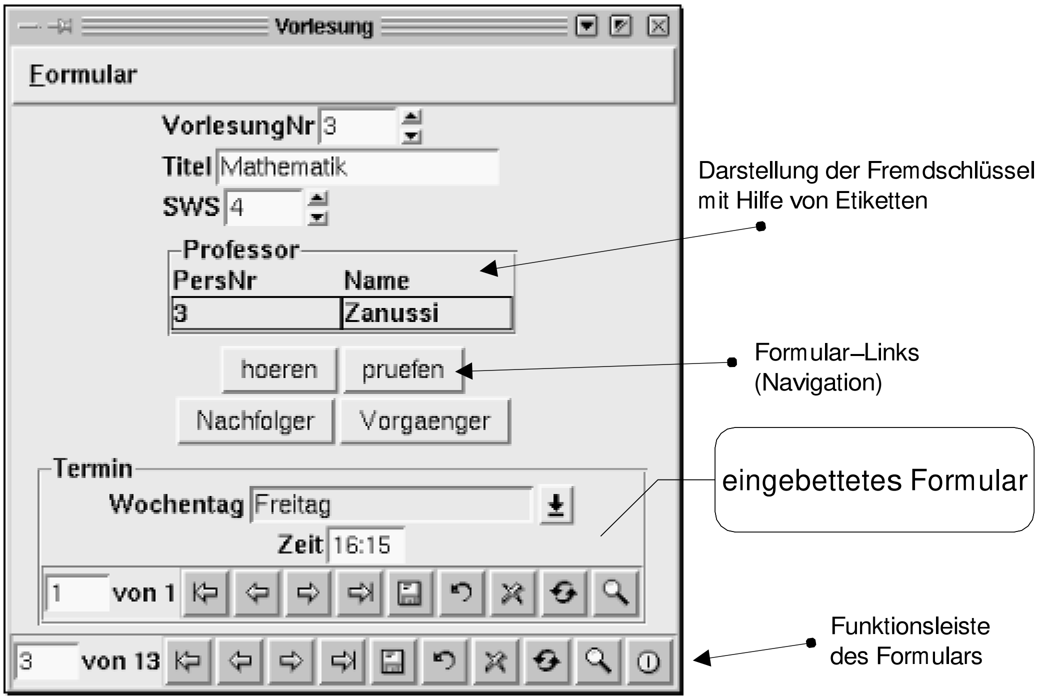

Abbildung 7.5: Realisierung der Aggregation durch eingebettete Formulare

bedingungen auf die Menge von Primärschlüsselwerten in der korrespondierenden Tabelle begrenzt. Das kann durch eine Auswahlliste repräsentiert werden. Der Benutzer weiß dann, dass er nur eine bestimmte Anzahl von Möglichkeiten hat, die er ansehen und auswählen kann. Zweitens sind die Werte des Schlüssels (s. Abschnitt 4.1) oft für Benutzer aussageleer oder ungeeignet für die Identifizierung des Objektes. Dafür erwartet der Benutzer weitere Attribute (z.B Name und Vorname außer des PersonalIDs).

Die Aggregation kann als die eingebetteten Formulare realisiert werden (s. Abb 7.5). Die Assoziation wird in Formular-Links umgesetzt. Die Spezialisierung kann auch als eine Verknüpfung von Formularen realisiert werden (s. Abb 7.4). Dabei wird ein Formular für ein Elternobjekt (Ober-Typ) und je ein Formular für den Kindobjekt (Unter-Typ) verwendet.

Die Beschreibung der Formulare sollte möglichst Implementierungs- und Plattformunabhängig sein, so dass sie flexibel und unabhängig von der Plattform eingesetzt werden können. Diese Eigenschaft ist sehr wichtig, weil in heutigen heterogenen Umgebungen verschiedene Schnittstellen zur gleichen Datenbanken benötigt werden. Vorteilhaft wäre es, wenn man nur einmal die Formulare spezifiziert, die nachher gleichartig ausgeführt werden, z.B. als HTML Formulare durch CGI-Schnittstelle, ein Java-Applet oder eine speziele Applikation auf dem Client-Rechner. Der Endbenutzer bekommt dann unabhängig von dem angebotenen Datenzugriff (per WEB-Browser oder eine Java/TCL Applikation) die gleichstrukturierten Formulare zu Gesicht. Das Ergebnis des Formular-Editors ist eine XML-Datei, die logische Struktur der Formulare spezifiziert. Die Platzierung der einzelnen Eingabefelder wird nicht durch absolute Masse eingegeben, sondern durch Eingaben für einem Geometry-Manger spezifiziert.

Die Syntax der Formularbeschreibungen wird in einem DTD-Dokument, siehe Anhang Seite 89, spezifiziert.

7.2.4 Formular-Server

Er ist ein Programm, mit dem der eigentliche Endbenutzer der Datenbank zu tun hat. Dieses Programm kann die Formularbeschreibungen, die vom Formular-Editor stammen, richtig auswerten und die Formulare darstellen. Dieses Programm muss die Manipulationen auf den Formularen richtig in die SQL-Anweisungen umsetzten und an die Datenbank quittieren. Dabei werden folgende Funktionen unterstützt:

- Daten Anzeigen (View)
- Daten Modifizieren (Update)
- Daten Löschen (Delete)
- Daten Anlegen (Create)
- Daten Filtern oder Durchsuchen
- durch Daten Navigieren (Browsing or Navigate)

Durch die Formular-Links ist auch eine Navigation durch Dateninhalte entlang der Verknüpfungspfäden möglich. Der Benutzer kann ähnlich wie in Hypertext-Dokumenten navigieren und jede Information auf verschiedenen Wegen erreichen.

Vor allem ist die Frage, was ist mit dem Objekt überhaupt verknüpft ist, schnell und ohne der Kenntnis des Schemas beantwortet.

Die Formulare selbst unterstützen direkt höhere Abstraktionskonzepte wie Assoziation, Aggregation und Spezialisierung. Die Kenntnisse über die Konzepte der relationalen Modellierung, wie Fremdschlüssel und Primärschlüssel müssen nicht vorhanden sein. Der Benutzer kann die verständlichen und allgemeineren Konzepte wie Aggregation und Assoziation wahrnehmen, ohne auf die besonderen Techniken des relationalen Modells eingehen zu müssen.

Es kann auch verschiedene Formular-Server geben, abhängig von Benutzer Anforderungen. Denkbar wäre die zusätzliche Benutzung von Format-Vorlagen für einzelne Formulare oder ganze Formulargruppen, die das Aussehen der Formulare noch weiter spezifiziert.[1] Es geht hier besonders um typische Layouteigenschaften, wie Schriftgröße, Schriftart oder Abstände.

Eine interessante Alternative zum Formular-Server wäre ein Formular-Applikations-Generator. Er wäre ein Werkzeug das aus Formular-Beschreibungen fertige Applikationen generiert. Die Information über das Aussehen der Formulare wäre dann in diesen Applikationen festkodiert. Solche Applikationen wären schlanker und schneller, als die Verbindung Formular-Server und Formular-Beschreibung (als XML-Datei). Vor allem könnte das aufwendige Parsen und Layouten der Formulare wegfallen. Liefert der Generator die Quelltexte der Applikationen (was bei Skriptsprachen wie TCL immer der Fall ist), können solche Applikationen nachträglich durch den Entwickler modifiziert werden. Zusätzliche Funktionalitäten (Autorisierung, Zugrifflisten, Versionifizierung) oder Modellierung der Prozesse wären denkbar.

[1] Ähnlich der CSS (Cascading Style Sheets) beim HTML-Browser

8 Implementierung

In diesem Kapitel werden zuerst die Entwicklungsumgebung und Entwicklungswerkzeuge vorgestellt. In einem besonderen Unterkapitel werden die aktuellen Entwicklungs-Techniken wie XML und DOM (Document Object Model) beschrieben. Es wird darauf verzichtet, das fast 10.000-Zeilen lange Programm zu dokumentieren. Beispielhaft wird das Klassendesign einer Komponente des Systems (Formular-Server) in Form eines UML Klassen-Diagramms präsentiert. Abschließend wird die Benutzung des Systems an einem bekannten Beispiel (Universität Datenbank) demonstriert.

8.1 Entwicklungswerkzeuge und Entwicklungsplattform

Die Basistechniken und Werkzeuge der Implementierung entstammen dem Umfeld des Betriebssystems Unix (Linux). Alle gehören zu den s.g. freien Programmen (Open Source), sind kostenlos zu verwenden, und erfreuen sich zum Teil großer Akzeptanz bei den Benutzern. Als Entwicklungs- und Testplattform diente das Linux Betriebsystem, allerdings sollte eine Portierung auf Betriebssysteme der Gruppe MS Windows keine Probleme bereiten und nur die Konfiguration betreffen.

Als Programmiersprache dient TCL (Tool Command Language) mit vielen seiner Erweiterungen. TCL ist eine interpretative und string-basierte Programmiersprache, und charakterisiert sich durch einfache Syntax und einfache Erweitbarkeit. Diese Eigenschaften machen sie ideal für das „Zusammenkleben“ von anderen spezialisierten Komponenten (meist Bibliotheken, die in anderen Sprachen (C, C++) geschrieben sind). Die Sprache wird oft für Zwecke des *Rapid Developments* oder *Prototypings* benutzt. Die Sprache TCL ist im Buch [Oute95] beschrieben. Es wurden folgende zusätzliche Komponenten des TCL benutzt.

Tk und Tix Eine Bibliothek für das Entwerfen von graphischen Benutzerschnittstellen (Widgetset).

TclX Erweitert TCL um zusätzliche Befehle (Listen, Mengenoperationen)

mysqltcl pgtcl Schnittstellen zu Mysql und Postgresql relationalen Datenbanken.

XOTcl Erweitert Tcl um objektorientierte Eigenschaften.

tDom DOM Implementierung für TCL

Die objektorientierte Implementierung wurde durch an der UNI Essen entwickelte XOTcl ermöglicht (Einstieg s. [NeZd00]).

8.2 XML und DOM

Ein aktueller Aspekt der Diplomarbeit ist die praktische Benutzung des XML (Extensible Markup Language) Standard bei der Implementierung des Systems (XML Standard s. [WWW1]). Zahlreiche wissenschaftliche Beiträge der letzten Jahre beschäftigen sich mit den Möglichkeiten des XML und der Rolle, die dieser Standard als allgemeines Austauschformat spielen wird (siehe [Tolk99] [Fars99]). XML bietet zuerst ein Format für die Darstellung der strukturierten Dokumente (Dokument Instanzen) und eine Meta-Beschreibungs-Sprache für kontextfreie Grammatiken, die die Syntax der Dokumente spezifiziert (s.g XML Dokument Klassen). Folgende Eigenschaften des XML waren ausschlaggebend für ihre Wahl:

- XML Klassen (Dokumententypen) sind leicht erweiterbar, ohne dass die Rückwärtskompatibilität darunter leidet (Nicht bekannte Elemente können ignoriert werden)

- menschen- und maschinnenlesbar
- XML bietet eine standardisierte Möglichkeit, die Syntax der Dokumente zu spezifizieren. Die s.g DTD (Document Type Definition).
- XML eignet sich sehr gut für die Darstellung von Baumstrukturen.
- Verfügbarkeit von freien XML-Parser für fast alle Programmiersprachen, was die Entwicklungzeit deutlich verkürzt.
- Zahlreiche freie Werkzeuge für das Erstellen von XML-Klassen und Validieren (überprüfen auf formale Gültigkeit) von XML Dokumenten.
- große Verbreitung des Standards

Im System dienen XML Dokumente als Schnittstellen zwischen Schema-Editor und Formular-Editor (XML Klasse dbschema) und weiter zwischen Formular-Editor und Formular-Sever (XML Klasse xmlforms). Dadurch wird die hohe Unabhängigkeit der Komponenten und ihre Austauschbarkeit garantiert. Jede Komponente kann durch eine andere ersetzt werden, die das entsprechende XML-Dokument manipulieren (interpretieren) kann. Es heißt, sie muss der Schnittstelle, die als eine XML-Kasse (DTD-Dokument) beschrieben ist, genügen. Für den Benutzer des Systems ist die interne Verwendung des XML-Formats unsichtbar. Er kann nach Bedarf die XML-Dokumente auch manuell erstellen, oder mit Hilfe von anderen allgemeinen XML-Werkzeugen verarbeiten. Durch die Benutzung von XML-Dokumenten steht ein Weg zur Integration mit anderen Systemen offen.

Den größten Vorteil bei der Implementierung von XML-orientierten Programmen bietet jedoch die Verwendung von speziellen Bibliotheken für die Bearbeitung der XML-Dokumente. Auf diese Weise kann man auf die eigene Programmierung von komplexen und fehleranfälligen Parsern verzichten, ganz im Sinne des Prinzips der Wiederverwendung der Software. Es existiert auch eine standardisierte Schnittstelle (API- Applikation Programm Interfaces) für solche Parser, DOM (Document Object Model, Standard siehe [WWW3]). DOM erlaubt nicht nur das Parsen von XML-Dokumenten, sondern bietet auch die Möglichkeit, die XML Dokumente zu generieren oder zu modifizieren. Dabei wird ein XML Dokument intern als eine

Baumstruktur abgebildet. Die DOM-API erlaubt den Zugriff auf verschiedene Knoten des Baumes, wie: XML-Tags, Tags-Attribute, Text, Kommentare, ProcessingInstructions. Manche der Parser bieten auch die Möglichkeit XML-Dokumente unter Verwendung von DTD-Dokumenten zu validieren. Die Programmierung der Fehlerbehandlung bei Parsen kann sich dadurch auf die kontextbezogenen Eigenschaften beschränken. Bei dem Entwurf von XML-bezogenen Applikationen hat man drei Möglichkeiten DOM zu benutzen:

1. Man transferiert mit DOM die Daten aus XML Dokumenten zu internen Strukturen. Z.B als Objekteigenschaften (Attribute). Nach der Bearbeitung wird wieder ein neuer DOM-Baum erzeugt und zu einem XML-Dokument umgewandelt.
2. Der DOM-Baum ist die einzige Repräsentation der Daten. Das Programm verarbeitet direkt die einzelnen Baumknoten.
3. Eine Mischform. Die Objekte verwenden DOM-Knoten als ein Platz für die Speicherung ihrer Daten.

Welche Strategie man wählt hängt von Art der Applikation ab. Die erste Strategie ist die effizienteste, weil der Programmierer eine spezielle Struktur für seine Daten entwickeln kann (Hash-Tabellen, Listen, Arrays). Verwendet man nur einen DOM-Baum, muss die interne Struktur nicht programmiert werden, dafür aber eine aufwendige Logik für das Interpretieren von einzelnen Baumelementen. Bei der Implementierung von Schema-Editor wurde die erste und beim Formular-Editor die zweite Strategie ausgewählt.

Es existiert auch eine standardisierte Methode die einzelnen DOM-Knoten abzufragen (Standard XPath [WWW4]). Hier ein Beispiel einer XPath-Abfrage
`child::table[@name=‘Student‘]`
Sie liefert alle Kind-Knoten des Typs „table“, wobei der Attribut Name gleich „Student“ ist (z.B. `<table name=''Student''/>`). Solche Abfragen sind eine Methode XML-Strukturen gezielt zu traversieren. Bei der Generierung von Formularen (Komponente Formular-Editor) wird von XPath-Abfragen intensiv gebrauch gemacht.

8.3 Klassen Design des Formular-Servers

Alle Komponenten wurden objektorientiert entworfen und implementiert. Bei dem Entwurf der Klassenstruktur wurden mehrere Entwurfsmuster erkannt und umgesetzt (Entwurfsmuster Design-Patterns [GHJV96]). Die Abbildung 8.1 zeigt ein UML-Klassendiagramm (Grobstruktur) der Komponente Formular-Server (UML United Modeling Language siehe [BoEJ99]). Viele der Klassen gehören auch zu anderen Komponenten (Formular-Editor und Schema-Editor) und wurden generell entworfen.

8.4 Anwendungsbeispiel

In diesem Abschnitt wird ein Beispiel einer Anwendung des Systems demonstriert. Es wird ein vollständiger Prozess der Erstellung der DB-Formulare, einer einfachen Universitäts-Datenbank, mit dem entwickelten System dokumentiert. Alle Zwischenschritte des Entwurfs werden als Dokumente präsentiert. Das Demonstrations-Beispiel (angelehnt an [KaKl93]) wurde so ausgewählt, dass es die ganze Palette der Modellierungskonzepte beinhaltet. Zuerst wird ein konzeptionelles Modell der Datenbank als ER-Diagramm erstellt (Abbildung 8.2). Die entsprechende Umsetzung des konzeptionellen Modells als relationales Modell produziert folgende Relationen. Es ist aber nur eine der möglichen Transformationen.

$$
\begin{aligned}
\text{Angestellte}: & \{[\underline{\text{PersNr}}, \text{Name}]\} \\
\text{Professor}: & \{[\underline{\text{PersNr}}, \text{Rang}, \text{Raum}]\} \\
\text{Assistent}: & \{[\underline{\text{PersNr}}, \text{Rang}, \text{Raum}, \text{Professor}]\} \\
\text{Student}: & \{[\underline{\text{MatrNr}}, \text{Name}, \text{Semester}]\} \\
\text{Vorlesung}: & \{[\underline{\text{VorlesungNr}}, \text{Titel}, \text{SWS}, \text{PersNr}]\} \\
\text{hoeren}: & \{[\underline{\text{MatrNr}}, \underline{\text{VorlesungNr}}]\} \\
\text{voraussetzten}: & \{[\underline{\text{VorgaengerNr}}, \underline{\text{NachfolgerNr}}]\} \\
\text{pruefen}: & \{[\underline{\text{MatrNr}}, \underline{\text{VorlesungNr}}, \underline{\text{PersNr}}, \text{Note}]\} \\
\text{Termin}: & \{[\underline{\text{VorlesungNr}}, \underline{\text{Wochentag}}, \underline{\text{Zeit}}]\}
\end{aligned}
$$

Dieses relationale Schema existiert physisch als eine Datenbank und wird von einem Datenbank-Manager verwaltet. Die Informationen über das Schema werden im Datenbank-Data-Dictionary abgelegt und können maschinell per SQL abgefragt werden. Die Erstellung der Formulare wird in zwei Stufen durchgeführt. Zuerst wird ein erweitertes Schema (genannt Repository) mit Hilfe der Komponente Schema-Editor erstellt. Dieses Repository beinhaltet alle Informationen aus dem Data-Dictionary der Datenbank (Relationen Schemata) und zusätzliche die Informationen wie sie interpretiert werden sollen. Die Erstellung des Repositories wird mit Reverse-Engineering-Techniken, besonders Befragung der Benutzer (DB-Administrator) ermöglicht. Das Repository (siehe Anhang S. 92) ist eine Grundlage für den nächsten Schritt.

Die Formulare werden mit Hilfe eines Generators aus den Informationen des Repositories erstellt. Sie werden als XML Dokumente (Dokument Klasse dbforms) spezifiziert. Im Anhang S. 94 wird ein Ausschnitt des Dokuments präsentiert, das das Aussehen von drei Formularen für Angestellte (Professor und Assistent), Vorlesung und Studenten beschreibt.

Das Aussehen der Formulare kann weiter mit dem Formular-Editor angepasst werden. Die eigentlichen Formulare werden durch den Formular-Server ausgeführt (quittiert), der die Formulare als eine graphische Benutzerschnittstelle zu der Datenbank darstellen kann. Die Screen-Shots der eigentlichen Formulare wurden bereits in der Diplomarbeit vorgestellt.

Angestellte siehe Abb. 7.4 Seite 65. Dieses Formular realisiert die Spezialisierung. Es werden 3 Formulare benutzt: Angestellte, Professor, Assistent.

Vorlesung siehe Abb. 7.5 Seite 66. Im Formular „Vorlesung“ wird das Formular „Termin“ eingebettet.

Student siehe Abb. 7.3 Seite 65.

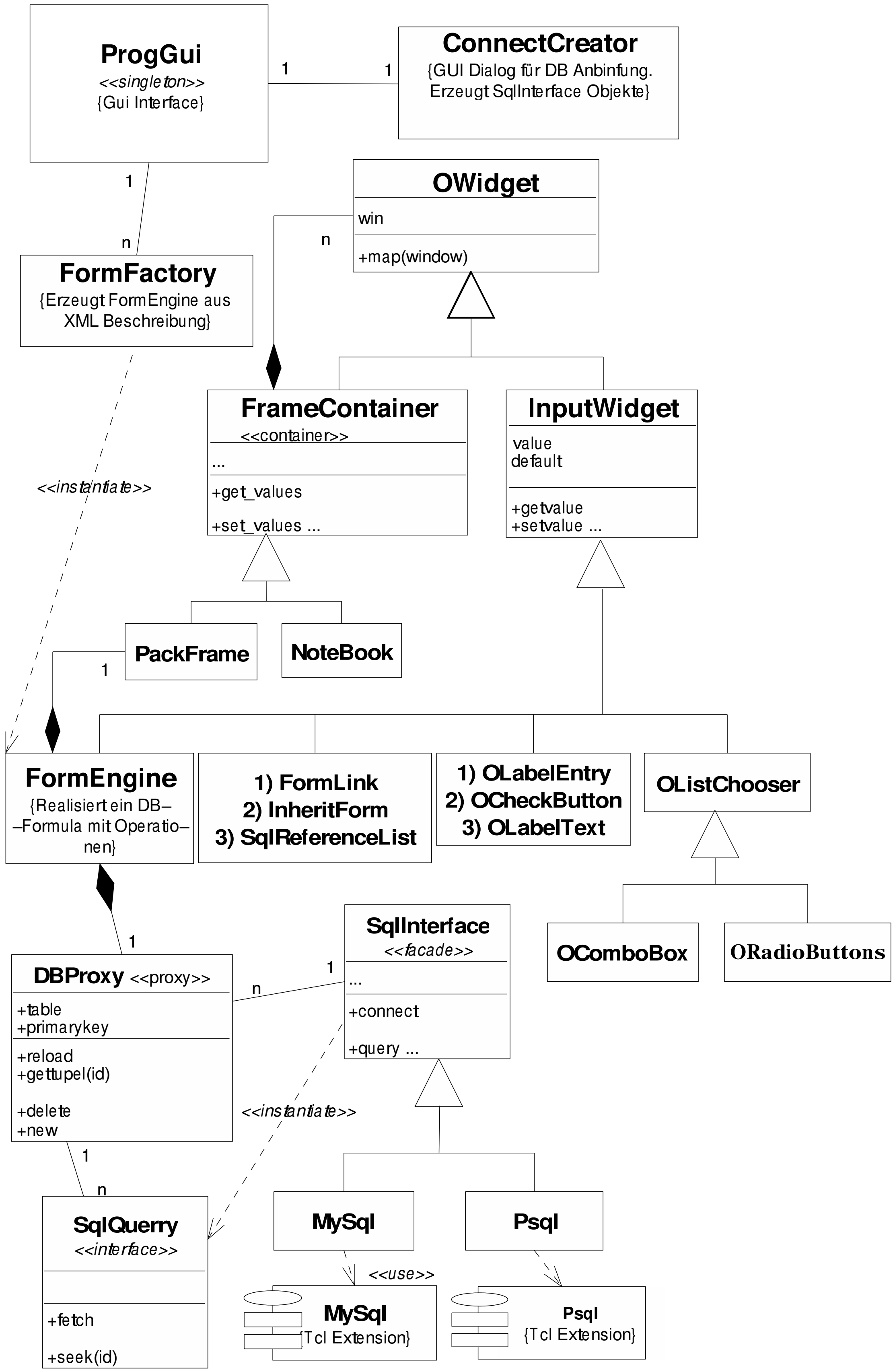

Abbildung 8.1: UML-Klassen Diagramm der Systemkomponente Formular-Server

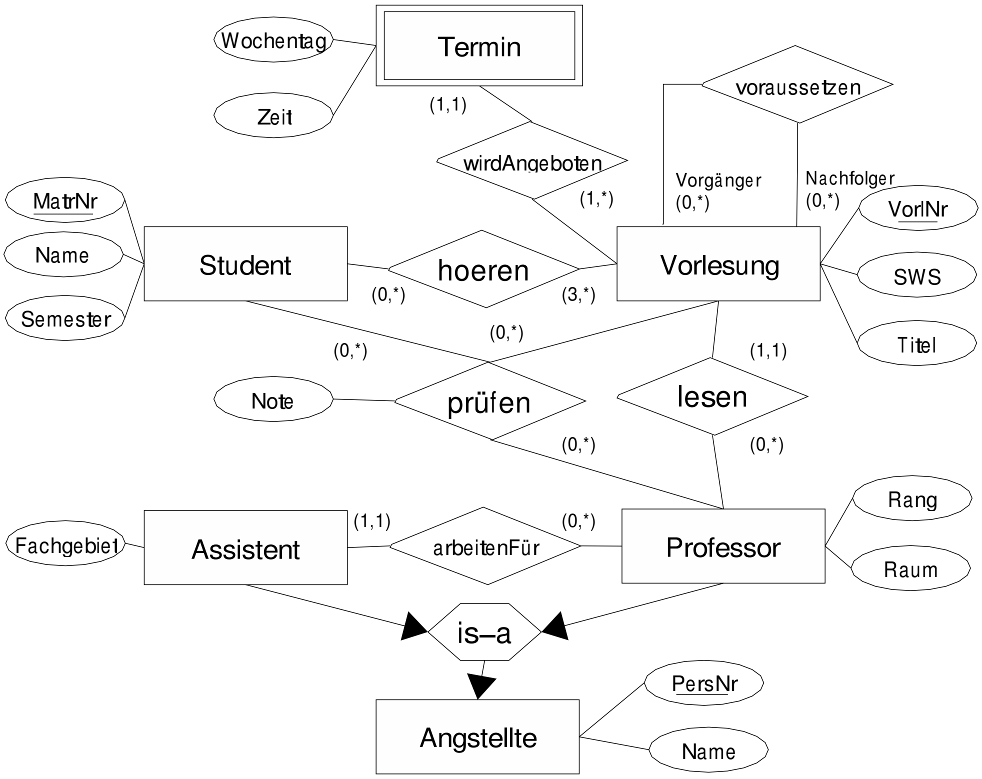

Abbildung 8.2: ER-Diagramm einer beispielhaften Universitäts-Datenbank

9 Relevante Arbeiten

Die Diplomarbeit berührt mehrere Felder der Forschung: Datenbank-Modelle und relationale Datenbanken speziell, Schema-Transformationen, Reverse Engeneerig, Entwicklung von graphischen Benutzterschnittstellen, CASE-Tools (Computer Aided Software Engineering) und Repositories. Bezeichnend ist, dass es nur wenige Arbeiten gibt, die die Entwicklung von DB-Applikationen behandeln. In der einschlägigen Literatur zu Datenbanken wie: [HeSa95],[KeEi96],[Voss99], wird die Entwicklung von DB-Applikationen höchstens erwähnt. Es werden nur die grundlegenden Schnittstellen zur Datenbank, wie SQL, und deren Einbettung in Programmiersprachen behandelt. Anders verhält es sich im Buch [PaCh93], wo DB-Applikationen zu einem wichtigen Teil der Datenbanksysteme werden. In diesem Buch werden die Datenbanksysteme aus der Sicht des Endbenutzers betrachtet. Relativ spät hat die Entwicklung von graphischen Benutzerschnittstellen (GUI) den Eingang in die Datenbanksysteme gefunden.

Einen direkten Bezug zur Diplomarbeit hat der Beitrag [CaSF92], der ein Teil der jährlichen Workshop Interfaces to Database System (IDS) ist. Darin wird das konzeptionelle Modell einer Datenbank als eine Grundlage für die automatische Entwicklung von DB-Applikationen (genauer DB-Formularen) gewählt. Als konzeptionelles Modell wird INFOLOG benutzt, das eine Erweiterung zum ER-Modell ist und auch dynamische Aspekte berücksichtigt. Die Entwicklung von DB-Applikationen verläuft in zwei Stufen. Zuerst wird aus dem konzeptionellen Modell eine Beschreibung der Benutzerschnittstellen in UIL (User Interface Language) erstellt. Danach werden aus dieser Beschreibung die DB-Applikationen generiert. In dieser Arbeit werden auch die dynamischen Aspekte des Systems berücksichtigt. Der Benutzter kann den Zustand der Objekte direkt durch die Auswahl eines Menüpunktes ändern (z.B Student-Stufe-Hauptdiplom, Student-Exmatrikulieren)

Der Journal-Beitrag [Ortn99] behandelt theoretisch aus der Sicht der Wirtschaftsinformatik die Repositories. Repositories werden hier als Systeme für die Dokumentation der Informationensverarbeitung eines Unternehmens verstanden. Der Autor beschreibt, wie die Repositories als eine Grundlage für die Entwicklung der Informationssysteme (durch CASE-Werkzeuge) dienen können. Für die Speicherung der Daten im Repository wird allerdings ein Ansatz aus Wissensrepräsentation und Sprachwissenschaft vorgeschlagen. Die bekannten Datenbankmodelle oder auch UML werden als für den Benutzer zu kompliziert abgelehnt.

Zahlreich sind dagegen die Arbeiten, die sich mit Datenbankmodellen ([GoSt99]) , Schema-Transformationen [FaVo95],[MaSh92],[PoBr98]) und Reverse Engineering ([ChBS97],[FaVo95],[ChBS94]) beschäftigen.

Großes Interese der letzten Jahren gilt dem XML-Standard, der von Anfang an mit Internet und Datenbanksystemen zusammengebracht wurde. Das XML-Format sollte das Internet zu einer Datenbank machen. Die Informationen könnten mehr struktuiert abgelegt werden, was eine automatische Auswertung möglich gemacht hätte ([Tolk99]). Eine wichtige Rolle spilet hier die Konventierung (mapping) der Daten vom XML-Format zu Datenbanken und umgekehr ([WWW5]) Die bestehenden großen relationalen Datenbanksysteme wurden um die Module erweitert, die solche Konventierung erlauben. Die bestehende Datenbank-Technologie wird immer häufiger verwendet um XML-Dokumente zu speichern (s.g XML-Server). XQL und XML-QL sind die Vorschläge einer Abfragesprache für XML-Dokumente. Zu erwähnen ist, dass die XML-Dokumente, im Vergleich zur Datenbank die s.g. semi-struktuirten Informationen speichern. Vor allen soll das XML-Format als ein Medium für die Integration der bestehenden Systeme dienen (als ein universelles Austauschformat).

10 Ausblick und Abschlussbetrachtung

Das Ergebnis der Diplomarbeit ist nicht nur die schriftliche Ausarbeitung, sonder auch ein fast 10.000 Programmzeilen starkes, vollfunktionsfähiges Programmpaket. Dieses Programmpaket wird auf den Namen *Xdobry* getauft und als Open Source Programm unter GPL (GNU Public Licence) gestellt. Die Quellen des Programms sind auf dem SUN und Metalab FTP-Server[1] oder zahlreichen Spiegelservern zu beziehen. Das Programm wird auch auf eigener Homepage (http://www.xdobry.de) präsentiert. Es ist beabsichtigt das System über die Dauer der Diplomarbeit weiter zu pflegen und zu entwickeln. Zwar ist das System kein vollständiges Produkt, aber als Prototypsystem für akademische Zwecke gut geeignet. Es könnte als eine Basis für die Entwicklung von unterschiedlichen DB-Applikationen dienen. Das wird durch den Entwurf als offenes System und die Benutzung von XML als Schnittstelle ermöglicht. Auf der Basis des Systems könnten weitere DB-Applikationen entwickelt werden wie: graphischer Abfrage-Editor, Berichts-Generator, Entwurf-Werkzeuge, ER-Diagramm Editor. Ohne großen Aufwand lässt sich der Schema-Editor in ein Werkzeug für die Administration der Data Dictionary umwandeln. Das komplette relationale Schema könnte dann, zuerst im Schema-Editor entwickelt und nachträglich als SQL-Anweisungen an den RDBMS geschickt werden. Für die praktische Arbeit mit dem System wäre die Möglichkeit der nachträglichen Anpassung der Schemata sehr wichtig. Im Idealfall sollte das System die Veränderungen an den Schemata

[1] ftp://sunsite.unc.edu/pub/Linux/apps/database

automatisch erkennen und an das Repository und andere DB-Applikationen weitergeben.

Die Entwicklung des Systems und die damit verbundene Recherche hat viele verfolgungswerte Ansätzte offengelegt, die jedoch in der Diplomarbeit nicht realisiert wurden. Man könnte die Entwicklung der DB-Applikationen in allgemeineren Bezug zu den Repository-Systemen stellen. Vielversprechend ist die Standardisierung der Objekt Management Group (OMG), bekannt auch von der Standardisierung der Objektorientierten-DBMS und CORBA (Common Object Request Broker). Es wurde ein Standard für den Zugriff auf Meta-Daten (MOF Meta-Objekt Facility) erarbeitet. Die Meta-Daten werden mit Hilfe der UML-Sprache beschrieben. Diese Beschreibungen können als XML-Dokumente (XMI XML Metadat Interchange) abgelegt werden. Den aktuellen Stand der Standardisierung von Repository-Systemen und Meta-Daten bietet [Thie00]. Die Formulare könnten nicht auf das relationale Schema zugreifen sondern auf CORBA-Objekte, was eine Entwicklung in die Richtung Objekt-Orientierte-Datenbanken wäre.

In der Diplomarbeit wurde der kleinste Nenner für relationale Datenbanken gewählt, was sich mit der Unterstützung des SQL-92 Standards gleichsetzt. Viele der Probleme, bzgl. der beschränkten Typbildungsmöglichkeiten, wurden in SQL99 Standard gelöst. Es bleibt vorerst abzuwarten bis die verfügbaren RDBMS diesen Standard unterstützen. Interessant könnte die Implementierung von objektrelationalen Eigenschaften des populären Datenbank Postgres sein. Viele von denen decken sich mit SQL3 Standard.

- Listen Attribute
- Tupel können per OID (Objectidentifier) identifiziert werden
- Vererbung von Tabellenschemata
- Versionifizierung
- Datenbank-Regel

Es existieren auch 2 interessante Projekte zur Beschreibung der graphischen Benutzerschnittstelle als XML-Dokumente. Eines davon entstammt als Nebenprodukt der Entwicklung von Open Source Netscape (Mozilla). Es ist XPToolkit[2]. Die Benutzerschnittstellen können plattformunabhängig als XML-Dokument (XUL-UserInterface Language) beschrieben werden. Ein anderes Projekt ist mit dem graphischen Editor für die Entwicklung von graphischen Benutzerschnittstellen GLADE [3] verbunden. Es ist ein Teil des GNOME Projekt, freies Desktop Enivronment für LINUX. Die Idee der beiden Projekte: das Aussehen der graphischen Benutzerschnittstelle ist nicht im Programm festkodiert, sondern immer liegt als ein XML-Dokument vor. Das könnte eine bessere und flexible Konfigurierbarkeit der Schnittstelle durch den Benutzter erlauben, ohne das Programm neukompilieren zu müssen. Die beiden Projekte sind um so wertvollel, als dass die kompletten Quelltexte frei sind und die verfügbaren Komponenten auch bei fremden Projekten verwendet werden können. Die Entwicklung des Formular-Editors und Formular-Servers könnte davon profitieren.

Es existieren auch weitere interessante Techniken, die mit dem XML-Standard verbunden sind. Durch die Extensible Stylesheet Language (XSL) läßt sich die Umwandlung von XML-Dokumenten in andere XML-Klassen oder andere Formate automatisieren. So könnte man die Umwandlung des Repository in konkrete SQL-Anweisungen (des Typs CREATE TABLE) leicht realisieren. Der Hauptnachteil der DTD (Document Type Defintion) ist, dass sie keine kontextbezogenen Regeln erlauben und zweitens nicht im XML selbst kodiert sind. Mit DTD kann nicht gewährleistet werden, dass ein wohlgeformtes Repository auch ein gültiges relationales Schema darstellt. Die neueren Entwicklungen wie das XML-Schema Standard erlauben eine genauere Definition der Dokument-Typen und Überwindung solcher Nachteile.

Eine andere Richtung für die Weiterentwicklung des Systems wäre die Unterstützung von weiteren Abstraktionen und Semantik im Schema-Editor und anschließend beim Formular-Server. Hier sind zwei Fälle vorstellbar:

1. das System bietet eine Schnittstelle auf die Zusatzfunktionen der DBMS wie Versioniefizierung, neue Datentypen, DB-Regel (Trigger)

2. das System implementiert selbst neue Funktionalitäten wie Versioniefizierung,

[2] http://www.mozilla.org/xpfe

[3] http://glade.pn.org

Authentiefizierung, Integritätsregel, kaskadierendes Löschen. Für das konzeptionelles Modell wäre die Unterstützung von weiteren Abstraktionen denkbar: abgeleitete Attributen, abgeleitete Beziehungen, abhängige Beziehungen.

Die Entwicklung hat einige interessante Aspekte offengelegt. Die XML-Parser-Bibliothek tDOM vereinfacht die Manipulation von XML-Dokumenten erheblich, viel Programmierarbeit kann dadurch gespart werden. Die Programmiersprache TCL erfüllt ihre Funktion als ein „Klebstoff" zwischen den spezialisierten Bibliotheken sehr gut. Allerdings macht erst die objektorientierte Erweiterung XOTcl die Entwicklung von großen TCL-Applikationen unproblematisch. Im Vergleich zu anderen objektorientierten Sprachen wie C++ oder Java, bietet XOTcl neue Eigenschaften, die eine flexible und dynamische Benutzung der Objekte erlauben. So können die Objekte zur Laufzeit ihre Methoden und Klassenzugehörigkeit ändern und nach ihren Eigenschaften abgefragt werden. XOTcl unterstützt weiterhin die dynamische Aggregation der Objekte und Klassen. (genaue Einführung bietet [NeZd00]). Die Benutzung der neuen Software-Techniken und neuen Komponenten hat es ermöglicht, dass sogar komplexe Projekte in kurzer Zeit entwickelt werden können. Die Rolle des Internets als Quelle von verschiedenen freien Software-Komponenten und Teil-Lösungen war bei der Diplomarbeit zentral. Die Qualität der freien Softwareprodukte ist erstaunlich hoch. Die Fehler, die während der Entwicklung in fremden Komponenten gefunden wurden, wurden nach wenigen Stunden von den Autoren der Komponenten per Email korrigiert. Mehrere Probleme könnten auch durch die Einsicht in die Quelltexten der Komponenten gelöst werden. Bemerkenswert ist, dass die großen Teilen der freien Software in akademischen Umfeld entwickelt wurden.

Literaturverzeichnis

[Bekk92] J. H. ter Bekke. Semantic Data Modeling. Prentice Hall . London. 1992.

[Bern99] Philip A. Bernstein. Repositories und Object Oriented Databases

[BoEJ99] Grady Booch; Jim Rumbaugh; Ivar Jacobson. Das UML - Benutzerhandbuch. Addison-Wesley. Deutschland. 1999.

[CaSF92] Rogério Carapuça; Artur Serrano, José Farinha. Automatic Derivation of Graphical Human-Machine Interfaces for Databases. Proceesings of the First International Workshop on Interfaces to Database Systems, Glasgow, 1-3 July 1992.

[ChBS94] Roger H.L. Chiang; Terence M. Barron; Veda C. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. Data & Knowledge Engineering 12. Elsevier. 1994. Seiten 107-142.

[ChBS97] Roger H.L. Chiang; Terence M. Barron; Vesa C. Storey. A framework for the design and evaluation of reverse engineering methods for relational databases. Data & Knowledge Engineering 21. Elsevier. 1997. Seiten 57-77.

[ChCr90] E.J. Chikofsky; J.H. Cross II. Reverse engineering and design recovery: A taxonomy. IEEE Software (January 19990). Seiten 13-17

[Eber94] Ray E. Eberts. User Interface Design. Prntice Hall. New Jersey USA. 1994

[EiMe99] Andrew Eisenberg; Jim Melton. SQL:1999, formerly known as SQL3. SIGMOD Record, Volume 28 Number 1, March 1999

[FaVo95] Christiona Fahrner; Gottfried Vossen; A survey of database design transformations based on the Entity-Relationship model. Data & Knowledge Engineering 15. Elsevier. 1995. Seiten 213-250.

[Fars99] Reza Farsi. XML. Informatik Spektrum. 22 Dez 1999. Springer-Verlag. Deutschland. 1999. Seiten 436-448

[GHJV96] E. Gamma; R. Helm; R. Johnson; J. Vlissides. Entwurfsmuster :Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley. Deutschland. 1996.

[GeEi98] Nahum Gershon; Stephen G. Eick. Information Visualisation. The Next Frontier. Journal of Intelligent Information Systems 11. Kluwe Academic Publishers. Netherlands. 1998. Seite 199-204.

[GoSt99] Robert C. Goldstein, Veda C. Storey. Data abstractions: Why and how?. Data & Knowledge Engineering 29. Elsevier. 1999. Seiten 293-311

[HeSa95] Andreas Heuer; Gunter Saake. Datenbanken: Konzepte und Sprachen. Internat. Thomson Publ. Bonn; Albany. 1995.

[KaKl93] Peter Kandzia; Hans-Joachim Klein. Theoretische Grundlagen relationaler Datenbanksysteme. BI-Wiss.-Verl. Mannheim; Leipzig; Wien; Zürich. 1993.

[KeEi96] Alfons Kemper; Andre Eickler. Datenbanksysteme. R. Oldenbourg. München; Wien. 1996.

[Kron97] David M. Kroenke. Database Processing: Fundamentals, Design, and Implementation. Prentice-Hall International. USA. 1997.

[MaRa92] Heikki Mannila; Kari-Jouko Räihä. The Design of Relational Databases. Addison-Weskey. UK. 1992.

[MaSh92] M. Markowitz; Arie Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. ACM Transactions on Database Systems, Vol. 17, No. 3, September 1992, Pages 423-464.

[MaUn97] Günter Matthiessen; Michael Unterstein. Relationale Datenbanken und SQL: Konzepte der Entwicklung und Anwendung. Addison-Wesley. Bonn. 1997.

[Misg98] Wolfgang D. Missgeld .SQL : Einstieg und Anwendung. Hanser. München; Wien. 1998.

[NeZd00] G. Neumann and U. Zdun. XOTCL, an object-oriented scripting language. In Proceedings of Tcl2k: 7th USENIX Tcl/Tk Conference, Austin, Texas, February 2000.

[Ortn99] Erich Ortner. Repository Systems. Teil 1: Mehrstuffigkeit der Entwicklungsumgebung. 22 Aug 1999. Springer-Verlag. Deutschland. 1999. Seiten 235-251

[Oute95] John K. Outerhout. Tck und Tk - Entwicklung graphischer Benutzerschnittstellen für das X Windows System Hanser. Addison-Wesley. Deutschland. 1995.

[PaCh93] Karman Parsaye; Mark Chingnell. Intelligent Database Tools & Application: Hyperinformation Access, Data Quality, Visualization, Automatic Discovery. John Wiley & Sons. 1993.

[PoBr98] Alexandra Poulovassikis; Peter M. Brien. A general formal framework for schema transformation. Data & Knowledge Engineering 28. Elsevier. 1999. Seiten 47-71.

[Rish92] Rishe Naphtali. Datebase design: the semantic modeling approach. McGraw-Hill. USA. 1992.

[Thie00] Uwe Thiemann. Quo Vadis, Meta Data? Unix-Open Zeitschrift August 8/2000. AWi UNIXopen Verlagsgesellschaft mbH. Deutschland. Seite 17-21.

[Tolk99] Robert Tolksdorf. XML und darauf basierende Standards: Die neuen Auszeichnungssprachen des Web. Informatik Spektrum. Springer-Verlag. Deutschland. 1999. Seiten 407-421

[Voss99] Gottfried Vossen. Datenbankmodelle, Datenbanksprachen und Datenbankmanagement-Systeme. R. Oldenbourg Verlag. München. 1999.

[ZhFe98] Michelle X. Zhou; Steven K. Feiner. Automated Visual Presentation: From Heterogeneous Information to Coherent Visual Discourse. Journal of Intelligent Information Systems 11. Kluwe Academic Publishers. Netherlands. 1998. Seite 205-234.

Internetquellen

[WWW1] Extensible Markup Language (XML) 1.0 W3C Recommendation, REC-xml-19980210, http://www.w3.org/TR/REC-xml, 10.02.1998

[WWW2] XML-Schema Working Draft, http://www.w3.org/TR/xmlschema-1, 7.04.2000

[WWW3] Document Object Model (DOM) Level 1 Specification Version 1.0, http://www.w3.org/TR/REC-DOM-Level-1, 1.10.1998

[WWW4] XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, 11.11.1999

[WWW5] Ronald Bourret, XML and Databeses, http://www.rpbouurret.com/xml/XMLAndDatabases.html, 27.09.2000

Anhang

DTD des Repositories

```
<!ELEMENT database (table+)>
<!ATTLIST database
 name CDATA #REQUIRED
>
<!ELEMENT table (attr+|assoziationtarget*|assoziationcollection*
                    |table*|attrgroup*|label?|description?)+>
<!ATTLIST table
 inheritkey CDATA #IMPLIED
 maxOccurs CDATA #IMPLIED
 type (child | part-of | relationship) #IMPLIED
 name ID #REQUIRED
 inherittype (unknown | subset | notsubset | parentabstract) "unknown"
 reference CDATA #IMPLIED
 minOccurs CDATA #IMPLIED
>
<!ELEMENT attr (valuelist?|metadata?)>
<!ATTLIST attr
 default CDATA #IMPLIED
 null (0 | 1) "0"
 type (int | varchar | enum | smallint | date | text | datetime
       | char | float | timestamp | longblob | tinyint | mediumint
       | bigint | double | time | decimal | tinyblob | tinytext
       | mediumblob | mediumtext | set) #REQUIRED
 name CDATA #REQUIRED
 auto_increment (0 | 1) "1"
 length CDATA #IMPLIED
 primary_key (0 | 1) "1"
 constrain CDATA #IMPLIED
 noteditable (1 | 0) "0"
 notvisible (0 | 1) "0"
>
<!ELEMENT assoziationcollection (participant+)>
```

```
<!ATTLIST assoziationcollection
 type (reftable | reference) #REQUIRED
 name CDATA #REQUIRED
 minOccurs CDATA #IMPLIED
>
<!ELEMENT participant EMPTY>
<!ATTLIST participant
 reference CDATA #REQUIRED
 rolename CDATA #IMPLIED
 table IDREF #IMPLIED
>
<!ELEMENT assoziationtarget EMPTY>
<!ATTLIST assoziationtarget
 table IDREF #IMPLIED
 maxOccurs CDATA #IMPLIED
 name CDATA #REQUIRED
 rolename CDATA #IMPLIED
 ondelete (nothing | delete | setnull) #IMPLIED
 minOccurs CDATA #IMPLIED
 onupdate (nothing | rename | setnull) #IMPLIED
>
<!ELEMENT valuelist (item+)>
<!ELEMENT attrgroup (attr+)>
<!ATTLIST attrgroup
 name CDATA #IMPLIED
>
<!ELEMENT item (#PCDATA)>
<!ELEMENT label (valuelist+)>
<!ATTLIST label
 id-relationship (0 | 1) #IMPLIED
>
<!ELEMENT metadata (importance?|mimetype?|description?)+>
<!ELEMENT mimetype (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT importance (#PCDATA)>
```

DTD der Formularbeschreibung

```
<!ELEMENT dbforms (form+)>
<!ATTLIST dbforms
 name CDATA #REQUIRED
>
<!ELEMENT form (frame)>
<!ATTLIST form
 primarykey CDATA #REQUIRED
 table CDATA #REQUIRED
 name ID #REQUIRED
```

```
>
<!ELEMENT frame (pack+)>
<!ATTLIST frame
 manager CDATA #REQUIRED
 label CDATA #IMPLIED
 border CDATA #IMPLIED
>
<!ELEMENT pack (boolean|frame|nestedform|formlink|inheritform
                |reference|numeric|list|text)*>
<!ATTLIST pack
 anchor (ne | n | e | se | sw | w | nw | center | s) "center"
 expand (1 | 0) "0"
 fill (x | y | both | none) "none"
 side (top | bottom | left | right) "top"
>
<!ELEMENT numeric EMPTY>
<!ATTLIST numeric
 default CDATA #IMPLIED
 label CDATA #IMPLIED
 textwidth CDATA #IMPLIED
 state (normal | disables) "normal"
 data CDATA #REQUIRED
 notnull (1 | 0) "0"
 dbdefault CDATA #IMPLIED
>
<!ELEMENT list (valuelist)>
<!ATTLIST list
 state (normal | disabled) "normal"
 data CDATA #REQUIRED
 subtype (radiobuttons | combobox) #IMPLIED
 default CDATA #IMPLIED
 label CDATA #IMPLIED
 notnull (0 | 1) "0"
 dbdefault CDATA #IMPLIED
>
<!ELEMENT valuelist (item+)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT formlink EMPTY>
<!ATTLIST formlink
 refattribute CDATA #REQUIRED
 form IDREF #REQUIRED
 label CDATA #IMPLIED
 data CDATA #REQUIRED
>
<!ELEMENT inheritform (child+)>
<!ATTLIST inheritform
 data CDATA #REQUIRED
>
<!ELEMENT child EMPTY>
<!ATTLIST child
 inheritkey CDATA #REQUIRED
```

```
 form IDREF #REQUIRED
 label CDATA #IMPLIED
>
<!ELEMENT nestedform EMPTY>
<!ATTLIST nestedform
 refattribute CDATA #REQUIRED
 form IDREF #REQUIRED
 label CDATA #IMPLIED
 data CDATA #REQUIRED
>
<!ELEMENT boolean EMPTY>
<!ATTLIST boolean
 data CDATA #REQUIRED
 label CDATA #IMPLIED
 onvalue CDATA "1"
 offvalue CDATA "0"
 default CDATA #IMPLIED
 state (normal | disabled) "disabled"
 notnull (1 | 0) "0"
 dbdefault CDATA #IMPLIED
>
<!ELEMENT reference (sqlquery)>
<!ATTLIST reference
 dropdown (0 | 1) "0"
 height CDATA #IMPLIED
 label CDATA #IMPLIED
 state (normal | disables) #IMPLIED
 showfirst (0 | 1) "1"
 data CDATA #REQUIRED
 columnslabel CDATA #IMPLIED
 columns CDATA #IMPLIED
 columnswidth CDATA #IMPLIED
 notnull (1 | 0) "0"
>
<!ELEMENT sqlquery (#PCDATA)>
<!ELEMENT text EMPTY>
<!ATTLIST text
 data CDATA #REQUIRED
 default CDATA #IMPLIED
 state (normal) #IMPLIED
 width CDATA #IMPLIED
 height CDATA #IMPLIED
 show (0 | 1) "0"
 textwidth CDATA #IMPLIED
 dbdefault CDATA #IMPLIED
 label CDATA #IMPLIED
 subtype (area | textfield) "textfield"
>
```

Repository einer beispielhaften Uni-Datenbank

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE database SYSTEM "dbschema.dtd">
<database name="uni">
    <?dbconnect {interface mysql} {hostname localhost} {user root}
                {dbank uni}?>
    <table name="Angestellte">
        <label>
            <valuelist>
                <item>Name</item>
            </valuelist>
        </label>
        <attr name="PersNr" type="int" length="11" primary_key="1"
              default="0" auto_increment="1"/>
        <attr name="Name" type="varchar" length="20" null="1"/>
        <table name="Professor" inherittype="unknown" type="child"
               inheritkey="PersNr">
            <attr name="PersNr" type="int" length="11" primary_key="1"
                  default="0"/>
            <attr name="Rang" type="varchar" length="20" null="1"/>
            <attr name="Raum" type="int" length="11" null="1"/>
            <assoziationtarget name="pruefen" table="pruefen"/>
            <assoziationtarget name="ProfessorAssistent"
                               table="Assistent"/>
            <assoziationtarget name="ProfessorVorlesung"
                               table="Vorlesung"/>
        </table>
        <table name="Assistent" inherittype="unknown"
               type="child" inheritkey="PersNr">
            <attr name="PersNr" type="int" length="11" primary_key="1"/>
            <attr name="Fachgebiet" type="varchar" length="20"
                  null="1"/>
            <attr name="Professor" type="int" length="11" null="1"/>
            <assoziationcollection name="ProfessorAssistent"
                                   type="reference" minOccurs="">
                <participant reference="Professor" table="Professor"/>
            </assoziationcollection>
        </table>
    </table>
    <table name="Student">
        <label>
            <valuelist>
                <item>Name</item>
            </valuelist>
        </label>
        <attr name="MatrNr" type="int" primary_key="1"
              auto_increment="1"/>
        <attr name="Name" type="varchar" length="20" null="1"/>
        <attr name="Semester" type="int" length="11" null="1"/>
```

```
    <assoziationtarget name="hoeren" table="hoeren"/>
    <assoziationtarget name="pruefen" table="pruefen"/>
</table>
<table name="Vorlesung">
    <label>
        <valuelist>
            <item>Titel</item>
        </valuelist>
    </label>
    <attr name="VorlesungNr" type="int" primary_key="1"
          auto_increment="1"/>
    <attr name="Titel" type="varchar" length="20" null="1"/>
    <attr name="SWS" type="int" length="11" null="1"/>
    <attr name="PersNr" type="int" length="11" null="1"/>
    <table name="Termin" reference="VorlesungNr">
        <attr name="VorlesungNr" type="int" default="0"/>
        <attr name="Wochentag" type="enum" null="1"
              default="Montag">
            <valuelist>
                <item>Montag</item>
                <item>Dienstag</item>
                <item>Mittwoch</item>
                <item>Donnerstag</item>
                <item>Freitag</item>
            </valuelist>
        </attr>
        <attr name="Zeit" type="varchar" length="5" null="1"/>
    </table>
    <assoziationtarget name="hoeren" ondelete="nothing"
                       table="hoeren"/>
    <assoziationtarget name="pruefen" table="pruefen"/>
    <assoziationtarget name="voraussetzten"
                       table="voraussetzten" rolename="Vorgaenger"/>
    <assoziationtarget name="voraussetzten"
                       table="voraussetzten" rolename="Nachfolger"/>
    <assoziationcollection name="ProfessorVorlesung"
                           type="reference">
        <participant reference="PersNr" table="Professor"/>
    </assoziationcollection>
</table>
<table name="hoeren" type="relationship">
    <attr name="MatrNr" type="int" length="11" default="0"/>
    <attr name="VorlesungNr" type="int" length="11" default="0"/>
    <assoziationcollection name="hoeren" type="reftable">
        <participant reference="MatrNr" table="Student"/>
        <participant reference="VorlesungNr" table="Vorlesung"/>
    </assoziationcollection>
</table>
<table name="pruefen" type="relationship">
    <attr name="MatrNr" type="int" length="11" default="0"/>
    <attr name="VorlesungNr" type="int" length="11" default="0"/>
```

```
        <attr name="PersNr" type="int" length="11" default="0"/>
        <attr name="Note" type="enum" null="1" default="sehr gut">
            <valuelist>
                <item>sehr gut</item>
                <item>gut</item>
                <item>befriedigend</item>
                <item>ausreichend</item>
                <item>mangelhaft</item>
            </valuelist>
        </attr>
        <assoziationcollection name="pruefen" type="reftable">
            <participant reference="MatrNr" table="Student"/>
            <participant reference="VorlesungNr" table="Vorlesung"/>
            <participant reference="PersNr" table="Professor"/>
        </assoziationcollection>
    </table>
    <table name="voraussetzten" type="relationship">
        <attr name="VorgaengerNr" type="int" length="11" default="0"/>
        <attr name="NachfolgerNr" type="int" length="11" default="0"/>
        <assoziationcollection name="voraussetzten" type="reftable">
            <participant reference="VorgaengerNr" table="Vorlesung"
                         rolename="Vorgaenger"/>
            <participant reference="NachfolgerNr" table="Vorlesung"
                         rolename="Nachfolger"/>
        </assoziationcollection>
    </table>
</database>
```


Beschreibungen der Formulare einer Uni DB-Anwendung (Ausschnitt)


```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE dbforms SYSTEM "xmlforms.dtd">
<dbforms name="uni">
    <?dbconnect {interface mysql} {hostname localhost} {user root}
                {dbank uni}?>
    <form name="Angestellte" table="Angestellte" primarykey="PersNr">
        <frame manager="pack">
            <pack>
                <numeric label="PersNr" data="PersNr"/>
                <text label="Name" textwidth="20" data="Name"/>
                <inheritform data="PersNr">
                    <child label="Professor" form="Professor"
                           inheritkey="PersNr"/>
                    <child label="Assistent" form="Assistent"
                           inheritkey="PersNr"/>
```

```
            </inheritform>
        </pack>
    </frame>
</form>
<form name="Professor" table="Professor" primarykey="PersNr">
    <frame manager="pack">
        <pack>
            <numeric label="PersNr" data="PersNr"/>
            <text label="Rang" textwidth="20" data="Rang" default="A"/>
            <numeric label="Raum" textwidth="11" data="Raum"/>
            <formlink label="pruefen" data="PersNr"
             refattribute="PersNr" form="pruefen"/>
            <formlink label="ProfessorAssistent" data="PersNr"
             refattribute="Professor" form="Assistent"/>
            <formlink label="ProfessorVorlesung" data="PersNr"
             refattribute="PersNr" form="Vorlesung"/>
        </pack>
    </frame>
</form>
<form name="Assistent" table="Assistent" primarykey="PersNr">
    <frame manager="pack">
        <pack>
            <numeric label="PersNr" data="PersNr"/>
            <text label="Fachgebiet" textwidth="20"
                  data="Fachgebiet"/>
            <reference data="Professor" label="Professor"
                       columns="2" columnslabel="PersNr Name">
                <sqlquery>SELECT Professor.PersNr,Name FROM
                           Professor,Angestellte
         WHERE Professor.PersNr=Angestellte.PersNr</sqlquery>
            </reference>
        </pack>
    </frame>
</form>
<form name="Student" table="Student" primarykey="MatrNr">
    <frame manager="pack">
        <pack>
            <frame manager="pack">
                <pack anchor="w">
                    <numeric label="MatrNr" data="MatrNr"/>
                </pack>
                <pack>
                    <text label="Name" textwidth="20" data="Name"/>
                </pack>
                <pack anchor="w">
                    <numeric label="Semester" data="Semester"/>
                </pack>
            </frame>
        </pack>
        <pack>
            <frame manager="pack">
```

```
                <pack side="left">
                    <formlink label="hoeren" form="hoeren"
                     refattribute="MatrNr" data="MatrNr"/>
                    <formlink label="pruefen" form="pruefen"
                     refattribute="MatrNr" data="MatrNr"/>
                </pack>
            </frame>
        </pack>
    </frame>
</form>
<form name="Vorlesung" table="Vorlesung" primarykey="VorlesungNr">
    <frame manager="pack">
        <pack>
            <frame manager="pack">
                <pack anchor="w">
                    <numeric label="VorlesungNr" data="VorlesungNr"
                             default="0"/>
                    <text label="Titel" textwidth="20" data="Titel"/>
                    <numeric label="SWS" data="SWS" default="2"/>
                </pack>
                <pack>
                    <reference label="Professor" columns="2"
  columnslabel="PersNr Name" height="12" columnswidth="12"
  showfirst="1" dropdown="1" data="PersNr">
                        <sqlquery>SELECT Professor.PersNr,Name
           FROM Professor,Angestellte
           WHERE Professor.PersNr=Angestellte.PersNr</sqlquery>
                    </reference>
                </pack>
            </frame>
        </pack>
        <pack>
            <frame manager="pack">
                <pack side="left">
                    <formlink label="hoeren" form="hoeren"
              refattribute="VorlesungNr" data="VorlesungNr"/>
                </pack>
                <pack side="left">
                    <formlink label="pruefen" form="pruefen"
               refattribute="VorlesungNr" data="VorlesungNr"/>
                </pack>
            </frame>
        </pack>
        <pack>
            <frame manager="pack">
                <pack side="left">
                    <formlink label="Nachfolger" form="voraussetzten"
                refattribute="NachfolgerNr" data="VorlesungNr"/>
                </pack>
                <pack side="left">
                    <formlink label="Vorgaenger" form="voraussetzten"
```

```
                refattribute="VorgaengerNr" data="VorlesungNr"/>
                    </pack>
                </frame>
            </pack>
            <pack>
                <nestedform label="Termin" form="Termin"
                 refattribute="VorlesungNr" data="VorlesungNr"/>
            </pack>
        </frame>
    </form>
    <form name="Termin" table="Termin"
          primarykey="VorlesungNr Wochentag Zeit">
        <frame manager="pack">
            <pack>
                <numeric label="VorlesungNr" textwidth="11"
                         data="VorlesungNr" default="0"/>
                <list label="Wochentag" data="Wochentag"
                      default="Montag">
                    <valuelist>
                        <item>Montag</item>
                        <item>Dienstag</item>
                        <item>Mittwoch</item>
                        <item>Donnerstag</item>
                        <item>Freitag</item>
                    </valuelist>
                </list>
                <text label="Zeit" textwidth="5" data="Zeit"/>
            </pack>
        </frame>
    </form>
</dbforms>
```

Eidesstattliche Versicherung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Wuppertal, den 29. Oktober 2000

Artur Trzewik