Persistenz von Objekten über eine relationale Datenbank

Lothar Wiedemer
Abteilung Berufliche Schulen
Landesinstitut für Erziehung und Unterricht
Rotebühlstr. 131
70197 Stuttgart
Lothar.Wiedemer@lowie.de

Abstract: Die objektorientierte Analyse, das Design und die Programmierung nehmen sich im Informatikunterricht kaum des Problems der Persistenz an. Objekte werden nur flüchtig gehalten oder höchstens in einem Stream aus dem Hauptspeicher auf einen Massenspeicher geschrieben bzw. von ihm gelesen. Im Unterricht werden die Ansätze der Objektorientierung und die der relationalen Datenbank nicht zusammengeführt und wenn, dann nach meiner Ansicht auf eine Art und Weise, welche die Objektorientierung zu einer GUI-Programmierung reduziert. Dass dies auch anders geht, soll dieser Vortrag zeigen.

1 Ausgangssituation

Der Objektorientierung und der Welt der relationalen Datenbanken liegen unterschiedliche Paradigmen zugrunde. Die zur objektorientierten Welt harmonisch passenden Datenbanken sind die objektorientierten Datenbanken, in denen die Klassen die Strukturen bilden und in denen sämtliche Konstruktionsmöglichkeiten der OOP abgedeckt werden. Diese Datenbanken setzen sich aber recht langsam durch, sind nicht ausreichend standardisiert und sehr resourcenintensiv. Ein weiterer Grund für die doch zähe Verbreitung ist die Zufriedenheit der Kunden relationaler Systeme. Diese Systeme (Oracle, DB2, SQL-Server etc.) erfuhren einen gewaltigen technischen Fortschritt, sind zuverlässig und sehr schnell. In die Entwicklung von Anwendungen, die auf den RDBMS basieren, wurden von Firmen, Forschung und öffentlicher Hand weltweit viele Milliarden Dollars investiert, was naturgemäß zu Beharrungsverhalten zugunsten der RDBMS führt.

Dennoch wird auf den technischen Fortschritt, welcher durch den Paradigmawechsel hin zur objektorientierten Welt zweifellos gegeben ist, in der Anwendungsentwicklung nicht verzichtet. Gemäß dem Grundsatz, wonach die Datenhaltung von der Programmierung unabhängig zu gestalten ist, findet man auch hier eine prinzipielle Lösung, ja mehrere prinzipielle Lösungen.

Bei den prozeduralen Sprachen wie COBOL oder PASCAL half man sich, indem man Embedded-SQL erfand. Ein Pre-Compiler übersetzte die Datenstruktur des Data-Dictionary in Datenstrukturen der jeweiligen Programmiersprache. Ein Result-Set eines SQL-Statements im Programm wurde dann in eine temporäre File in einer der Programmiersprache bekannten Struktur geschrieben und ggfls. sequenziell bearbeitet. Die Kommunikation war gegeben.

Heute verschaffen uns z.B. bei JAVA die JDBC[1]-Schnittstellen Zugriffe zur Datenbank. Wenn es sich um Microsoft-Produkte handelt, benötigen Sie zusätzlich auch die ODBC[2]-Schnittstellen (genauer JDBC:ODBC). Sie erlauben schier unbegrenzte Möglichkeiten der Manipulation von Daten (DML[3]) einer relationalen Datenbank und erschließen ebenso die Möglichkeiten der Veränderung von Datenstrukturen (DDL[4]).

JDBC sollte als Programmgerüst zur Anbindung relationaler Datenbanken an Java-Programme verstanden werden. JDBC funktioniert recht einfach, man gibt eine Zeichenkette als SQL-Anweisung an ein RDBMS und erhält die Antwort als Ergebnistabelle, die, wie schon in den prozeduralen Sprachen, sequentiell abgearbeitet wird. Es sei aber noch einmal darauf verwiesen, dass die relationalen Datenbanken nur flache Strukturen abbilden können. Aber man kann in der Programmiersprache die flach strukturierten Daten zu komplexen Objekten zusammenstellen (was aber hier nicht geschehen soll).

2 Gründe für die Verwendung einer Programmiersprache

Wie bereits ausgeführt, ist in die Entwicklung von relationalen Datenbanken viel Geld investiert worden. Die Zuverlässigkeit und Leistungsfähigkeit relationaler Datenbanken in einer Client-Server-Architektur lässt daher auch für die Zukunft vermuten, dass ein Druck, einen Wechsel hin zu objektorientierten Datenbanken zu vollziehen, kaum stattfinden wird. Die relationalen Datenbanken werden darüber hinaus in Richtung Objektorientierung (objektrelational) weiterentwickelt. Hierzu zählen auch komfortable Schnittstellen zu objektorientierten Anwendungen.

Im Unterricht werden die Funktionen eines RDBMS oft missbraucht und die wahre Aufgabe eines Datenservers ist vielen Informatik-Lehrern verschlossen. Die Beispieldaten, die normalerweise im Unterricht Verwendung finden, sind nicht umfangreich, sodass Verstöße gegen das Laufzeitverhalten kaum erkannt werden.

Dazu ein Beispiel:

---

[1] JDBC = Java DataBase Connectivity
[2] ODBC = Open DataBase Connectivity
[3] DML = Data Manipulation Language
[4] DDL = Data Description Language

Im Folgenden ist ein Ausschnitt der Struktur einer sehr vereinfachten Auftragsbearbeitung abgebildet:

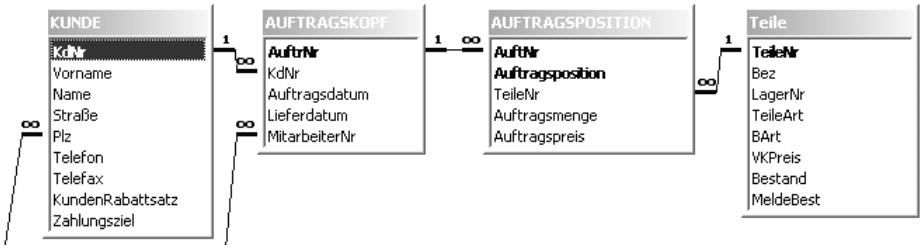

Abb. 1 Struktur Auftragsbearbeitung

Um Auftragspositionen für ein Auftragsformular zu erhalten, müssen die Tabellen Auftragsposition und Teile zusammengeführt werden.
Eine Auftragspositionszeile des Auftrags 0814 hat folgende Überschrift und Probedaten:

Positionsnummer	Teilenummer	Bezeichnung	Auftragsmenge	Preis	Gesamtpreis
1	AZ66	Kreuzschlüssel	700	5,00	3.500,00

Es wäre nun ein Verstoß gegen Performanceregeln, wenn diese Positionszeile durch folgendes SQL-Statement erzeugt würde, so elegant es auch klingen mag:

```
SELECT Auftragsposition, Teile.TeileNr, Bez, Auftragsmenge,
Auftragspreis, Auftragsmenge*Auftragspreis AS Gesamtpreis
FROM Teile, Auftragsposition
WHERE Teile.TeileNr = Auftragsposition.TeileNr
AND AuftrNr = 0814;
```

Die Begründung liefert die Client-Server-Architektur: Wenn immer möglich, sollte der Server zulasten des Client von nicht originären Datenbankaufgaben entlastet werden. Im Beispiel darf der Server die virtuelle Spalte Gesamtpreis nicht berechnen – dies ist Aufgabe des Client. Er verfügt nach der Abfrage über alle Daten und sollte mit moderner (auch objektorientierter Software) ausgestattet sein. Im Unterricht wird oft mit Software (z.B. MS-Access) gearbeitet, die in einer Client-Server-Architektur aber nur als Front-End einsetzbar ist. MS-Access ist eine Lösung für ein Einzelplatzsystem und nur hierfür auch von Microsoft vorgesehen.

Noch mehr in die falsche Richtung geht man, wenn auch die Auftragssumme, nun sogar mittels einer Gruppierungsfunktion in SQL, zusätzlich ermittelt wird, obwohl der Client alle Daten hat. Eine solch konzipierte Anwendung stößt bei großen Datenmengen und häufig aufgerufenen Funktionen sehr schnell an Grenzen und kann nur als minderwertige Lösung bezeichnet werden. Der Weg liegt in einer vernünftigen Aufteilung der Arbeit zwischen Client und Server. Im Beispiel hat der Server nur die gespeicherten Daten zu liefern, alle derivativen Datenkomponenten erzeugt der Client selbst.

Es gibt noch viele andere gute Gründe, objektorientierte Programmiersprachen und relationale Datenbanken zusammenzubringen. Dieses Thema soll aber hier nicht weiter verfolgt werden.

3 Ein kleines Beispiel

Das nun folgende Beispiel ist als Unterrichtseinstieg für den Datenbankzugriff gedacht:

Eine Bank verwaltet ihre Kunden in einer Datenbank. Die Struktur der Datenbank (nicht unser Thema) ist wie folgt gegeben, wobei wir im Beispiel die Konten und Buchungen aus Zeit- und Platzgründen nicht beachten:

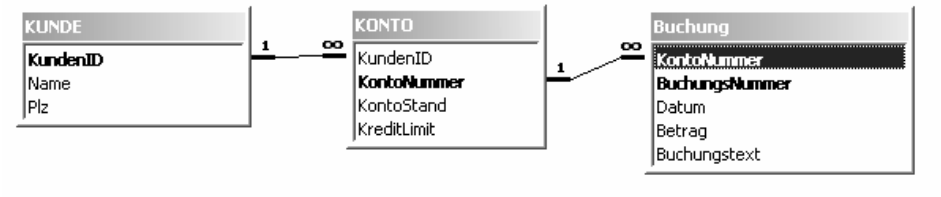

Abb. 2 Beispieldatenbank

Der Zugriff auf die Datenbank ist mit einer objektorientierten Programmiersprache zu realisieren.

3.1 Objektorientierte Analyse (OOA)

Aus der Aufgabenstellung ergibt sich das folgende Klassendiagramm für die Klasse Kunde:

Kunde	
-nachname:	String
-plz:	String
-id:	int
+Kunde	
+findeMitID:	void
+fuegeKundeHinzu	void
+aktualisiereKund	void
+loescheKunde	void
+setNachname	String
+getNachname	void
+setPlz	String
+getPlz	void
+setId	String
+getID	void

Die OOA ist für das gegebene Beispiel mit der definierten Einschränkung der Aufgabenstellung auf dieses Klassendiagramm reduzierbar.

3.2 Objektorientiertes Design (OOD)

Die Einfachheit der Aufgabenstellung erlaubt es, sich auf das Design der Benutzeroberflächen-Komponente (ohne Benutzerklassifizierung etc.) und der Datenzugriffskomponente[5]zu beschränken.

Der Zugriff der zu schreibenden Anwendung auf die Datenbank wird durch folgendes Drei-Schichten-Modell veranschaulicht:

Die Fachklassen erhalten alle Methoden, die zum Schreiben und Lesen auf die Datenbank benötigt werden. Damit wird auch der Datenbankzugriff gekapselt.

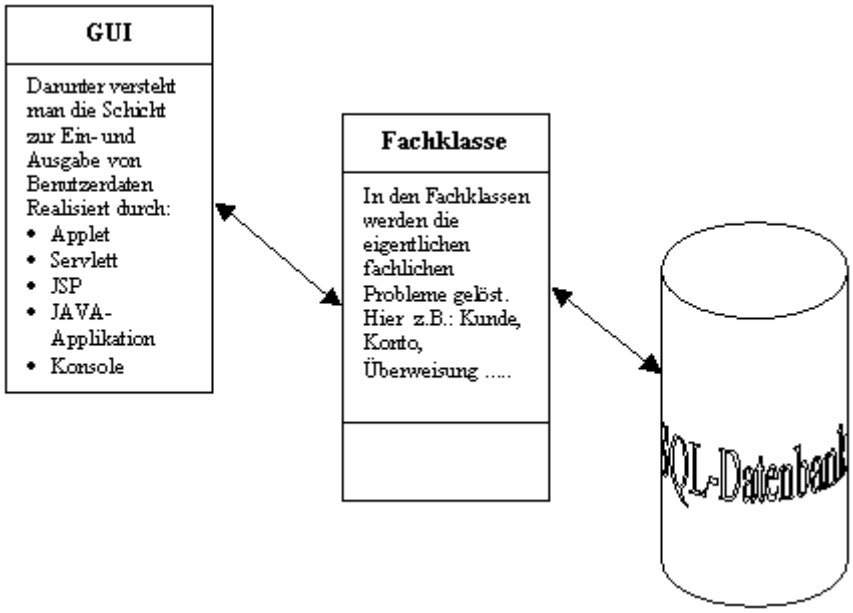

Abb. 3 Drei-Schichten-Modell des Datenbankzugfriffs

Der Datenzugriff ist so von der GUI völlig unabhängig.

Ein Klasse DBVerbindung lagert außerdem den Verbindungsaufbau aus. Diese Klasse hat nur Klassenvariablen und kann daher keine eigene Objekte erzeugen.

---

[5] [CY94]

Das Design der GUI ist in der Struktur einfach. Ein Hauptmenü (Windowsmenüzeile) verzweigt in einen Zweig mit INSERT-, UPDATE-, DELETE- und ANZEIGEN-Funktion, in einem Nachbarzweig kann man sich alle Tupel der Kundentabelle anschauen. Die GUI wurde mit einem Tool erzeugt:

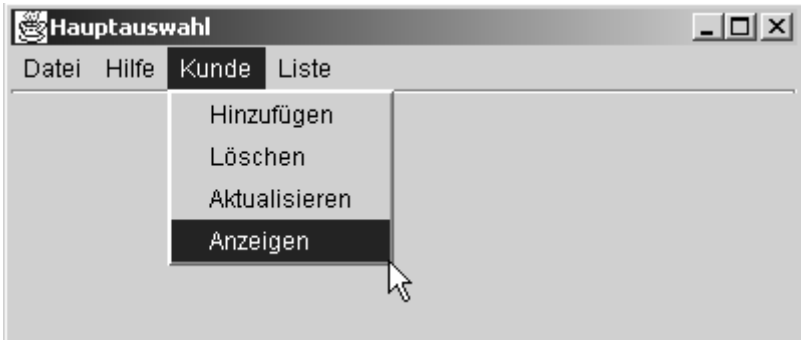

Abb. 4 Hauptauswahl des Prgrammbeispiels

Die einzelnen Fenster, die vom Menü aufgerufen werden, haben stets den gleichen Aufbau, auf ihre Darstellung (außer Kundendaten anzeigen) soll hier aus Platzgründen verzichtet werden.[6]

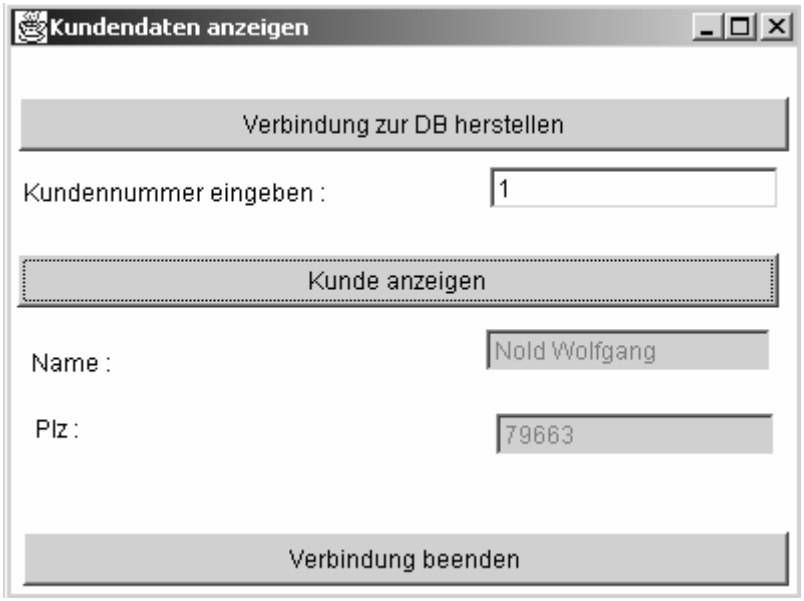

Abb. 5 Fenster Kundendaten anzeigen

[6] Der vollständigeQuellcode zusammen mit Erläuterungen kann unter http://www.lowie.de Menü Downloads und dort unter Verschiedenes, Stichwort GI, bezogen werden.

Das Programm ist in einer **Drei-Schichten-Architektur** konzipiert, d.h. alle Datenbankzugriffe erfolgen über die Fachklasse **Kunde**; die Verbindung zur Datenbank wird aus didaktischen Gründen in jedem Fenster mittels eines entsprechenden Buttons neu hergestellt.

Das Ergebnis des OOD, aus Gründen der Anschaulichkeit nur mit ausgewählten Klassen, dargestellt im UML-Design-Modus:

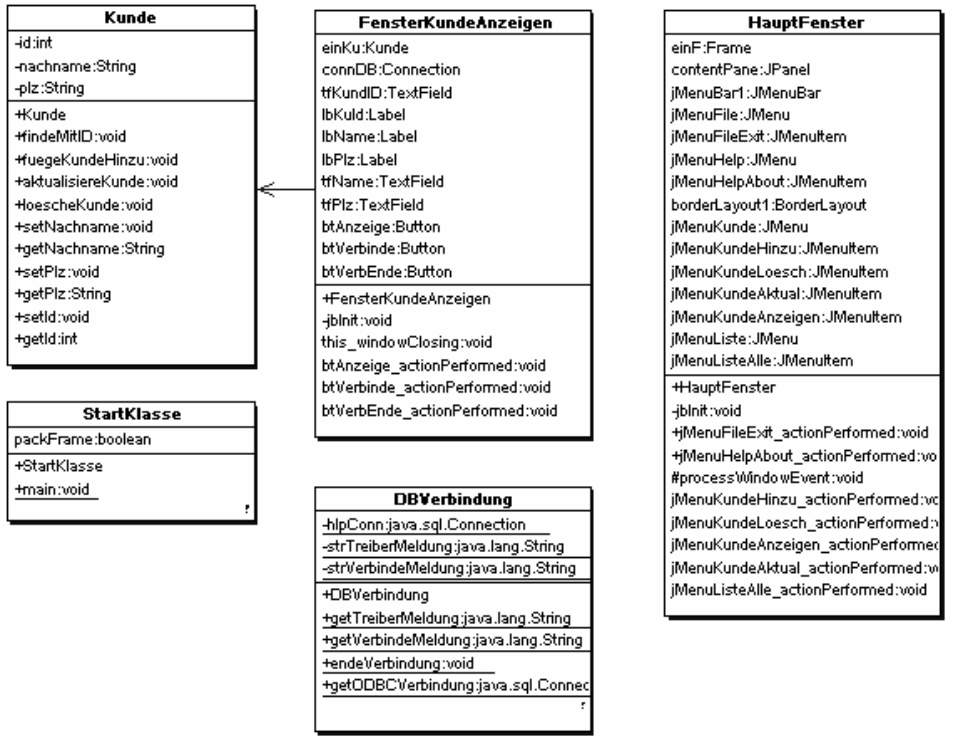

Abb. 6 UML-Diagramm des Beispiels

3.3 Implementierung der ODBC-Schnittstelle

Da im Beispiel MS-ACCESS als RDBMS einsetzt wird, muss vorher die ODBC-Schnittstelle definiert werden. Dies geschieht über das Menü Start/Einstellungen/Systemsteuerung/Verwaltung/Datenquelle(ODBC). Die Verbindung wird im Beispiel ODBCBank20 und die Datenbank AccBank2000.mdb genannt. In den zwei Fenstern müssen folgende Eintragungen vorgenommen werden:

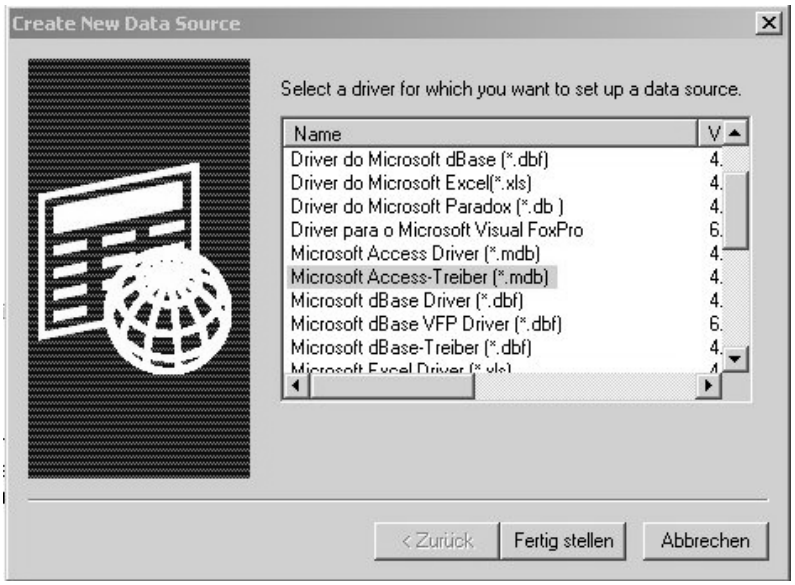

Abb. 7 Auswahl des ODBC-Treibers

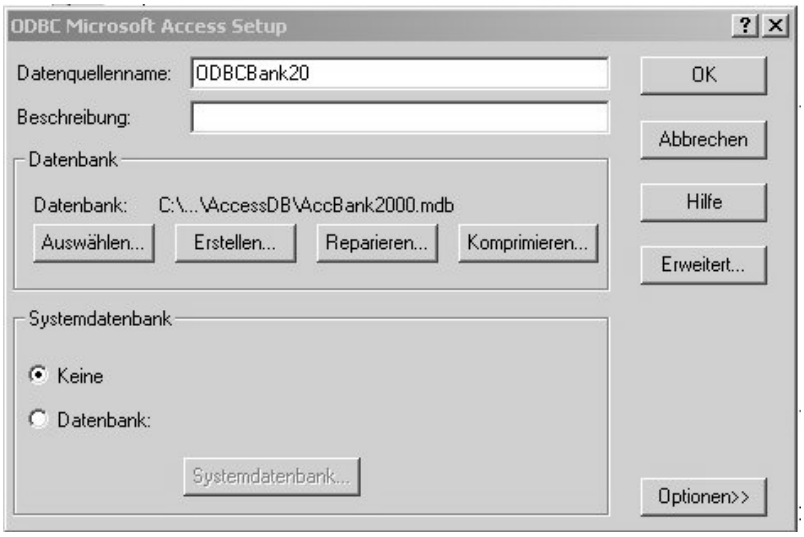

Abb. 8 Auswahl der Datenbank für ODBC-Treiber

Damit ist für alle Applikationen des benutzten Rechners die Schnittstelle zur Datenbank AccBank2000 definiert. Die Applikationen, die diese Schnittstelle kennen, können die Datenbank nutzen. Da hier via JDBC:ODBC zugegriffen wird, diese Schnittstelle ist im JDK enthalten, kann die Datenbank mit SQL über JAVA manipuliert werden.

3.4 Objektorientierte Programmierung (OOP)

Bevor ich die Verknüpfung mit einer Datenbank auf die bisher dargelegte Weise im Unterricht behandle, haben die Schüler Kenntnisse zu RDBMSs und sind in der Lage, eine GUI mit einem GUI-Generator zu erstellen. Der Unterricht kann sich somit auf die Verknüpfung des Programmes mit der Datenbank konzentrieren. Im vorliegenden Fall erhalten die Schüler die bis auf die Verbindung zur Datenbank fertige GUI. Auf Action-Listener und andere GUI-Komponenten wurde bereits an anderer Stelle eingegangen. In der URL: www.lowie.de , und dort unter downloads/GI finden finden Sie den Quellcode der Klassen des Beispiels und die Erläuterungen.

3.4 Schematische Darstellung des Programmablaufs

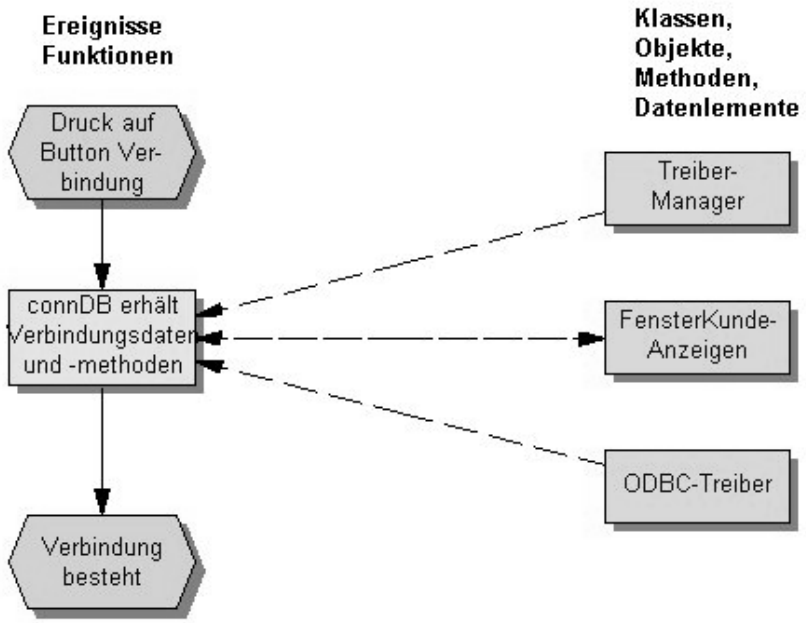

Abb.9 Ablauf der Verbindungserstellung zur Datenbank

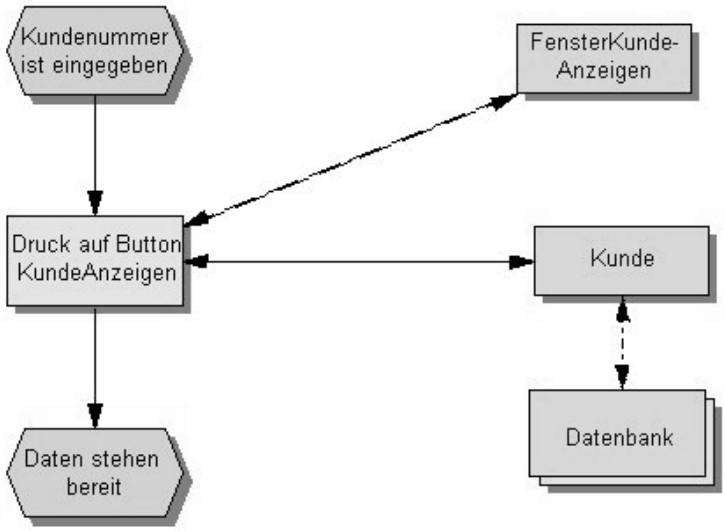

Abb. 10 Ablauf einer Kundensuche und -anzeige

4 Vorgehen im Unterricht

Um das Augenmerk der Schüler für die Verknüpfung der Datenbank zum Java-Programm frei zu machen, stelle ich alle notwendigen Programmkomponenten, die vom eigentlichen Problem ablenken könnten, zur Verfügung. Diese Bestandteile des Programmes wurden im vorausgehenden Unterricht in anderem Kontext behandelt. Die Schüler haben somit Kenntnis von

- dem Handling eines RDBMS
- Grundlagen der OOA, des OOD und OOP
- der Entwicklung einer GUI in einer objektorientierten Programmiersprache
- ODBC-Schnittstellen

Die Datenbank ist zunächst sehr einfach strukturiert, kann dann aber in guten Kursen Beziehungen enthalten, die in OOP über Assoziationen abgebildet werden.

Die unvollständig ausformulierten Klassen stehen dem Schüler zur Verfügung und werden nur durch die für die Verbindung relevanten Methoden ergänzt. Dabei kristallisiert sich bei mir folgendes Vorgehen heraus:

Jeweils eine Doppelstundestunde für

Step 1: Problemanalyse, Zielvorgabe

Step 2: Die Schüler bauen die Verbindung zur DB auf. Das bedeutet einmal die Einbindung der Klasse DBVerbindung zum anderen die Belegung der Schaltflächen „Verbindung zur DB herstellen“ mit Funktionalität (ODBC-Schnittstelle war bereits realisiert).

Step 3: Die Schaltfläche „Kunde anzeigen“ wird mit Funktionalität belegt.

Step 4: Weitere Problemstellungen/Vertiefung (z.B.: Anfügen und Löschen eines Kunden, Ausgabe einer Kundenliste etc).

Step 5: Ausarbeitung der Vorteile des Vorgehens (Nutzung einer modernen Programmiersprache, Kapselung des Datenbankzugriffs, Trennung der Aufgaben Client-Server), kritische Nachbetrachtung (insbesondere Paradigmenwechsel in einer Anwendung) , Vorstellung einer fertigen Applikation.

Bei der Umsetzung strebe ich schülerzentrierte Unterrichtsformen an. Dabei werden die Programmkomponenten modifiziert und nach der vorhergehenden Zieldefinition entworfen.

5 Begründung für das Vorgehen

Das beschriebene Vorgehen hat folgende Vorteile:

- Die Attribute der Klasse Kunde sind gekapselt, ebenso die Klassenattribute der Klasse DBVerbindung. Da alle Zugriffe auf die Datenbank über diese beiden Klassen erfolgen, ist somit jeder Datenzugriff gekapselt.

- Die Daten sind unabhängig von den Programmen, die auf sie zugreifen. Der hier gezeigte Weg mittels einer GUI in einer JAVA-Applikation könnte auch mit JSP, Applets, Servlets oder über die Konsole und zwar gleichzeitig realisiert werden.

- Programme sind unabhängig von dem zum Einsatz kommenden RDBMS. Wenn die Datenbank auf ein anderes System portiert werden würde, müssten im vorliegenden Fall nur der String für den Verbindungsnamen und der Treibernamen geändert werden.

- In einer objektorientierten Programmiersprache können Assoziationen, Vererbungen und andere die Beziehung zwischen Objekten darstellende Konstrukte beschrieben werden. Diese und weitere Mächtigkeiten einer Programmiersprache können genutzt und in relationalen Konzepten transformiert abgespeichert werden. So kann man viele Integritätsregeln auf Clientseite abdecken, und damit die Performance des Gesamtsystems erhöhen.

- Die Investitionen in die relationalen Datenbanken verlieren nicht an Wert, sie können, da die Informationsdichte hin zum Informationssystem wächst, sogar Wertsteigerungen erfahren.

- Die Schüler erfahren, dass in einer objektorientierten Entwicklungsumgebung die Regeln der Objektorientierung durchgängig sind. Sie erkennen Brücken zwischen Paradigmen der Informatik und lernen so, dass diese keine Gegensätze sein müssen.

- Der Informatikunterricht befreit sich von mathematischen Inhalten. Algorithmische Probleme treten etwas in den Hintergrund, sind aber durch die Wahl des Beispiels beliebig schwer zu gewichten (wenn z.B. alle Sätze einer Tabelle anzuzeigen sind, muss das in einer Schleife geschehen, Assoziationen werden über Vektoren realisiert).

Literaturverzeichnis

[Ba00] Balzert, H.: Lehrbuch der Softwaretechnik Band I, Softwareentwicklung, Spektrum-Verlag, 2. Auflage 2000.

[Ba99] Balzert, H.: Lehrbuch der Objektmodellierung, Analyse und Entwurf, Spektrum-Verlag Heidelberg, ISBN 3-8274-0285-9.

[CY94] Coad, P.; Yourdon, E.: Objektorientiertes Design, Prentice Hall Verlag, München, 1994.

[CY96] Coad, P.; Yourdon, E.: Objektorientierte Analyse, Prentice Hall Verlag, München, 1996.

[DE] Denne, N.: DB2 Theorie und Praxis, IBM-Fortbildungsunterlagen.

[HP98] Hohenstein, U.; Pleßler, V.: Oracle8 Effiziente Anwendungsentwicklung mit objektrelationalen Konzepten, dpunkt Verlag Heidelberg, 1998.

[Kr99] Krüger, G.: Go To Java 2, HTML-Version, Addison-Wesley, 1999.

[Oe98] Oestereich, B.: Objektorientierte Softwareentwicklung, Analyse Design mit der Unified Modeling Language, Oldenburg Verlag München, 1998.

[Sc97] Scheer, A.-W.: Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse, Springer Verlag Berlin, 1998.