# Praxisorientierte Aspekte der *Leu*-Datenmodellierung

Guido Dinkhoff, Volker Gruhn, Michael Zielonka
LION Gesellschaft für Systementwicklung mbH
UNI TECH CENTER
Universitätsstrasse 140
44799 Bochum
Deutschland

12. Dezember 1993

**Zusammenfassung**

In diesem Artikel wird über einen Ansatz berichtet, der Datenmodellierung mit Entity/Relationship-Diagrammen mit objektorientierten Modellierungsmöglichkeiten verbindet. Der vorgestellte Ansatz ist in der LION-Entwicklungs-Umgebung *Leu* realisiert.

Darüber hinaus wird berichtet, wie der Einsatz eines solchen kombinierten Ansatzes in einem großen industriellen Projekt zur Entwicklung eines wohnungswirtschaftlichen Systems aussieht.

## 1 Einleitung

Datenmodellierung in der industriellen Praxis ist weitestgehend durch die Verwendung relationaler Ansätze geprägt. Der industrielle Einsatz objektorientierter Techniken ist zur Zeit nur in geringem Maße zu beobachten. Die Verbindung von Entity/Relationship-Modellierung mit objektorientierter Datenmodellierung bietet eine Chance, objektorientiertes Denken in die industrielle Praxis zu transferieren.

Der in diesem Artikel skizzierte Datenmodellierungsansatz ist in das Umfeld der Entwicklung des bau- und wohnungswirtschaftlichen Informationssystems WIS (für **W**ohnungswirtschaftliches **I**nformations-**S**ystem) eingebettet. Zur Entwicklung von WIS wurde zunächst die Entwicklungsumgebung *Leu* [Gru93] erstellt. Grundidee von *Leu* ist es, die statischen Aspekte eines Informationssystems in Form von Datenmodellen zu beschreiben. Die auf diesen Datenmodellen basierenden Geschäftsprozesse werden mit Hilfe von abstrakten Petrinetzen (um genau zu sein mit FUNSOFT-Netzen [DG90, Gru91]) beschrieben. Die Geschäftsprozesse repräsentieren die dynamischen Aspekte eines Informationssystems, denn sie beschreiben in welcher Reihenfolge Daten geändert werden können. Die WIS-Entwicklungssituation sieht so aus, daß *Leu* von Wohnungswirten mit Großrechnererfahrung eingesetzt wird. In der gegebenen Situation konnten Kenntnisse der Entity/Relationship-Modellierung vorausgesetzt werden. Kenntnisse objektorientierter Modellierung waren nicht vorhanden. Darüberhinaus waren konkrete Wünsche an die zu benutzende Datenmodellierung vorhanden. Diese Wünsche entstanden in der Regel aus der Kenntnis von Problemen, die die Änderung von Datenmodellen in der Großrechnerwelt mit sich bringen.

Ziel dieses Papiers ist es, die konkreten Anforderungen an die in *Leu* zur Verfügung zu stellenden Datenmodellierungsmöglichkeiten zu erörtern und ihre Realisierung in der *Leu*-Datenmodellierung zu diskutieren. Zu diesem Zweck werden in Abschnitt 2 verschiedene Grade

von Objektorientiertheit dargestellt, danach in Abschnitt 3 Anforderungen an die Datenmodellierung diskutiert, in Abschnitt 4 der *Leu*-Ansatz zur Integration relational orientierter, objektorientierter und zusätzlicher Anforderungen diskutiert und abschließend die Erfahrungen beim Einsatz der *Leu*-Datenmodellierung im Entwurf von WIS zusammengefaßt (Abschnitt 5).

# 2 Eigenschaften objektorientierter Datenbanksysteme

Kaum ein Begriff ist in der Informatik in den letzten Jahren derart unterschiedlich verwendet worden, wie der der *Objektorientierung*. Im Datenbankbereich nannte sich jedes System, das in irgend einer Weise Objekte manipulierte, *objektorientiert*. Erst als 1989 *The Object-Oriented Database System Manifesto* [ABD+] veröffentlicht wurde, existierte so etwas wie eine Richtlinie, was ein DBMS (Datenbank-Management-System) erfüllen muß, um ein *objektorientiertes DBMS* (OODBMS) genannt werden zu können.

In diesem Abschnitt sollen zunächst diese Kriterien, die ein DBMS zu einem OODBMS machen, kurz beleuchtet werden. Anschließend werden Grade von Objektorientierung [Dit86] erläutert.

## 2.1 Objektorientierte Eigenschaften

Die im folgenden beschriebenen Eigenschaften sind notwendige Eigenschaften für ein objektorientiertes Datenbanksystem. Typische Datenbankfunktionalitäten, wie *Persistenz*, *Speichermanagement*, *Nebenläufigkeitskontrolle* und *Recovery* werden dabei vorausgesetzt. Nicht näher betrachtet werden die in [ABD+] noch erwähnten Punkte *Vollständige Berechenbarkeit* und *Ad-Hoc-Anfragemöglichkeit*, da diese bei den weiteren Betrachtungen nicht von Bedeutung sind. Zusätzliche Funktionalitäten wie Versionsverwaltung oder Typechecking sind vielfach wünschenswert, aber nicht unbedingt erforderlich.

**Komplexe Objekte** werden aus einfacheren Objekten mit Hilfe von Konstruktoren gebildet. Die minimale Anforderung an ein objektorientiertes System ist die Existenz eines Mengenkonstruktors (*set*), eines Listenkonstruktors (*list*) und eines Tupelkonstruktors (*tuple*). Diese Konstruktoren müssen *orthogonal* sein, d. h. jeder Konstruktor kann auf jedes Objekt angewendet werden. Das schließt ein, daß auch eine Schachtelung dieser Konstruktoren möglich ist.

Ein Datenbanksystem bietet eine Anzahl vordefinierter Grundtypen. Für objektorientierte Systeme muß die Möglichkeit der **Erweiterbarkeit** dieser Grundtypen existieren, d. h. es gibt eine Möglichkeit, neue Typen zu definieren. Dazu dienen die bereits erwähnten Konstruktoren. Ferner muß ebenso die Menge der Operationen erweiterbar sein, die auf neue und vorhandene Typen angewendet werden können. Nicht notwendig erweiterbar muß die Anzahl der Konstruktoren für neue Typen sein.

**Objektidentität** bedeutet, daß die Existenz von Objekten unabhängig ist von deren Werten. Somit ist es möglich, wertgleiche unterschiedliche Objekte zu verwalten. Es gibt demnach zwei Möglichkeiten, Objekte zu vergleichen:

- Zwei Objekte sind *identisch*, d. h. es handelt sich um dasselbe Objekt.
- Zwei Objekte sind *gleich*, d. h. sie haben die gleichen Werte.

In objektorientierten Systemen haben Objekte einen *Datenteil* und einen *Operationsteil.* Der Datenteil nimmt die Werte des Objekts auf. Der Operationsteil beinhaltet alle Operationen, die auf ein Objekt angewendet werden können. **Datenkapselung** bedeutet nun, daß dem Benutzer des Systems die Art der Datenhaltung und die Implementierung der Operationen verborgen bleibt. Notwendig für ein objektorientiertes System ist es, daß die Möglichkeit der strikten Datenkapselung existiert.

Eine Eigenschaft, die objektorientierte Systeme nach [ABD+] bieten müssen, ist **Vererbung.** Vererbung reduziert den Umfang eines Datenbankschemas und unterstützt das Konzept der *Wiederverwendbarkeit (reusability).* Gleiche Attribute und Operationen müssen bei Typen, die voneinander abhängen, nicht mehrfach aufgeführt oder implementiert werden. Dieses Konzept der Vererbung, in der Literatur oft durch das Wortpaar *Generalisierung/Spezialisierung* ausgedrückt, führt zu einer hierarchischen Anordnung der Typen eines Datenbankschemas.

In objektorientierten Systemen werden Objekte mit gleichen Eigenschaften zu **Typen** oder **Klassen** zusammengefaßt. Eine derartige Ordnung ermöglicht den Aufbau einer **Typhierarchie.**

Häufig ist es sinnvoll, eine Operation mit einer bestimmten Semantik, die für verschiedene Typen eine andere Realisierung hat, mit dem gleichen Namen zu belegen. Ein typisches Beispiel ist eine Ausgabeoperation für Objekte. Diese existiert für jeden Typ mit diesem Namen, wenngleich sie für unterschiedliche Typen unterschiedliche Realisierungen hat. Soll ein Objekt eines dieser Typen auf dem Bildschirm ausgegeben werden, wird die Operation angewandt und das System sucht sich unter den verschiedenen Realisierungen die heraus, die auf den Typ paßt. Die Redefinition einer Operation bei der Implementierung wird **Overriding** genannt. Die Anwendung einer neu definierten Methode heißt **Overloading.**

Unterstützt man den Erweiterbarkeitsgedanken objektorientierter Systeme, besteht die Möglichkeit, neue Typen und neue Operationen während der Laufzeit zu definieren. Diese neuen Operationen können demnach nicht in der Entwurfsphase an die entsprechenden Typen gebunden werden, sondern dies muß zur Ausführungszeit geschehen. Dieses verzögerte Anbinden von Operationen an Typen nennt man **late binding.**

## 2.2 Klassifizierung von Objektorientierung

Dittrich klassifiziert objektorientierte Datenbanksysteme aufgrund ihrer Eigenschaften [Dit86]. Typische Datenbankeigenschaften, wie Persistenz, Speichermanagement, Nebenläufigkeitskontrolle (*concurreny*) und Wiederherstellbarkeit (*recovery*) werden vorausgesetzt. Werden zusätzlich komplexe Objekte unterstützt, nennt Dittrich dieses *strukturelle Objektorientierung* und ein Datenbanksystem mit dieser Eigenschaft objektorientiertes Datenbanksystem (OODBS). Wird Erweiterbarkeit angeboten, nennt Dittrich dieses *verhaltensmäßig objektorientiert.* Sind beide Eigenschaften gegeben, so hat man nach Dittrich ein *voll objektorientiertes Datenbanksystem.*

# 3 Datenbank-Management-Systeme in der Praxis

Die Anforderungen, die in der Praxis an ein DBMS gestellt werden, werden von den meisten DBMSen nicht vollständig erfüllt. Die verschiedenen Produkte bieten unterschiedliche Eigenschaften, decken aber meist nur einen Teil der Anforderungen ab.

Im folgenden werden einige Anforderungen, wie sie sich aus der Praxis ergeben, formuliert und schließlich eine Integration verschiedener Ansätze erörtert.

## 3.1 Anforderungen an ein Datenbank-Management-System

Bei den Anforderungen an ein DBMS wird in diesem Abschnitt unterschieden zwischen grundsätzlichen Anforderungen (deren Erfüllung erwartet man jedem DBMS) und Eigenschaften, die bei relationalen oder objektorientierten Systemen zusätzlich erfüllt sein sollten. Schließlich werden Anforderungen beschrieben, die nicht typisch für eine bestimmte Klasse von DBMSen aber wünschenswert sind.

### 3.1.1 Grundsätzliche Anforderungen an ein Datenbank-Management-System

Die Hauptaufgabe eines DBMSs ist es, Daten zuverlässig zu speichern. Zuverlässigkeit bedeutet in diesem Zusammenhang, daß das System im Fehlerfall in der Lage ist, einen konsistenten Datenbankzustand mit möglichst wenig Datenverlust wieder herzustellen (*Recovery*).

In der Praxis müssen Datenbanken die Daten vieler Benutzer speichern, die ihrerseits teilweise parallel auf der Datenbank arbeiten. Diese parallele Arbeit muß synchronisiert werden (*Nebenläufigkeitskontrolle*). Dazu gehört auch ein Transaktionskonzept, so daß eine Benutzeraktion für andere erst sichtbar wird, wenn diese korrekt ausgeführt wurde, d. h. korrekt in der Datenbank gespeichert wurde (*Persistenz*).

Eine Anforderung, die für Endanwender wenig interessant ist, für Entwickler oder Administratoren jedoch unabdingbar, ist eine Ad-Hoc-Anfragemöglichkeit. Gängig ist das Vorgehen, neben dem reinen DBMS Werkzeuge anzubieten, die derartige Zugriffe auf die eine oder andere Art erlauben.

### 3.1.2 Relationale Anforderungen an ein Datenbank-Management-System

Relationale DBMSe stoßen in der Industrie auf eine immer größer werdende Akzeptanz. Das liegt zum einen daran, daß die Anbieter *relationaler Datenbank-Management-Systeme* sehr komfortable und vor allem sichere Produkte anbieten. Das liegt aber auch an der Konzeption, die sich hinter diesen Systemen verbirgt. Im Gegensatz zum hierarchischen Datenmodell [TL76] oder zum Netzwerkmodell [TF76], die lange Zeit den Standard bildeten, bietet das relationale Modell [Cod70, Cod79] mit SQL (**S**tructured **Q**uery **L**anguage) komfortable Zugriffsmöglichkeiten auf Daten, speziell wenn diese untereinander verknüpft werden sollen. Der wesentliche Vorteil von SQL ist, daß ein Standard hierfür existiert [ANS92]. Die Unterstützung dieses Standards ist eine wesentliche Anforderung an ein modernes DBMS.

Ein weiterer Aspekt ist der Verbreitungsgrad von SQL, der hinunter geht bis in den Anwenderbereich. Bei einer Neuentwicklung von Anwendungen ist es demnach erforderlich, schon allein aus Gründen der Zeit- und Kostenersparnis, auf dieses vorhandene Know-How aufzusetzen.

### 3.1.3 Objektorientierte Anforderungen an ein Datenbank-Management-System

Bis vor wenigen Jahren waren kommerzielle Anwendungen im Verwaltungssektor auf Großrechnern realisiert. Typische Eigenschaften dieser Systeme war die satzorientierte Verarbeitung

und Speicherung von Daten. Die Leistungsexplosion von Workstations speziell auf dem UNIX-Sektor und die ständig wachsende Verbreitung von graphikfähigen Terminals und PCs machte eine Verarbeitung u. a. von Bild- und Tondaten mit Hilfe des Computers möglich. Die enormen Datenmengen, die bei der Verarbeitung von sogenannten *Multimedia-Daten* anfallen, lassen sich nicht mehr satzorientiert verwalten. Der Begriff des *Objekts* wurde geprägt.

Bei einer Portierung oder Neuentwicklung eines existierenden Systems darf natürlich keine Funktionalität verloren gehen. Das bedeutet, daß so etwas wie satzorientierte Verarbeitung weiterhin möglich sein muß. Gleichzeitig müssen Objekte verwaltet werden, um den gewachsenen Anforderungen gerecht zu werden.

Meistens wird die alte Datenbankstruktur auf eine objektorientierte Struktur abgebildet. Das bedeutet, daß Konstruktoren zur Verfügung stehen müssen, um Felder zu Objekttypen zusammenzufassen. Mindestens erforderlich ist hier ein Tupelkonstruktor, hilfreich sind häufig auch Mengen- und Listenkonstruktoren.

Die Einführung von Objekten bedingt, daß diese eindeutig identifiziert werden können. Man muß sowohl die Identität von Objekten als auch die Wertgleichheit bestimmen können.

Für komplexe Objekte, speziell für Multimedia-Daten, ist es notwendig, Zugriffsoperation zur Verfügung zu stellen. Diese Operationen sollten für den Anwender im allgemeinen in einer Benutzeroberfläche verborgen sein (Datenkapselung).

Ebenso sinnvoll, aber nicht unbedingt notwendig, ist ein Vererbungskonzept. Dieses macht Datenmodelle übersichtlicher und reduziert den Ein-/Ausgabeaufwand von Daten. Da es sich hierbei allerdings um ein Konzept handelt, was in bestehenden Systemen nicht existierte, ist eine Integration dieser Eigenschaft nicht immer möglich.

### 3.1.4 Sonstige Anforderungen

Bei Verwaltungsanwendungen wird häufig ein Historisierungskonzept verwendet. Hierbei wird die *Vergangenheit* eines Objekts bzw. Datensatzes festgehalten, um z. B. tendenzielle Entwicklungen zu bestimmen oder Statistiken zu erstellen. Diese Eigenschaft bieten gängige relationale oder objektorientierte Datenbanken im allgemeinen nicht.

Zugangskontrollen sind in allen Systemen Standard. Allerdings werden die Berechtigungen meistens personen- bzw. kennungsgebunden vergeben. Das führt in der Praxis oft dazu, daß Personen unter Umständen Rechte zugebilligt werden, die nicht sinnvoll oder gar nicht erlaubt sind, oder aber eine Person erhält mehrere Benutzerkennungen, um gewisse Rechte zu haben. Ein Konzept, bei dem Berechtigungen an Rollen geknüpft sind, die den realen Personen zugeordnet werden, vermindert hier den Modellierungsaufwand und erlaubt ein differenzierteres Berechtigungssystem.

Bei größeren Anwendungen werden Datenmodelle in der Regel sehr komplex. In dem Fall ist es sinnvoll, Mittel und Methoden zur Strukturierung anzubieten. Denkbar wären hier hierarchische Aufbaustrukturen oder die Aufteilung des Datenmodells in Teilmodelle.

## 3.2 Integration der Anforderungen an ein Datenbank-Management-System

Bei der Neuentwicklung von Systemen im Verwaltungssektor oder bei der Portierung vorhandener Systeme stellt sich irgendwann die Frage nach der Datenhaltung. Einerseits bietet sich aus Gründen der Übersichtlichkeit eine objektorientierte Modellierung an. Ferner bietet die

Umsetzung in ein objektorientiertes DBMS die Möglichkeit der Verarbeitung von Multimedia-Daten. Andererseits ist die Anfragesprache SQL ein bekannter und akzeptierter Standard. Dieser basiert jedoch auf dem relationalen Ansatz und bietet in der derzeitigen Ausprägung die Verarbeitung von Objekten nicht. Hinzu kommt, daß die Berechtigungsprofile, die von Datenbankherstellern allgemein angeboten werden, vielfach nicht ausreichend sind. Ein Historisierungskonzept findet man ebenfalls selten.

Einen Schritt in die Richtung einer Integration beschreitet das American National Standards Institute (ANSI) zusammen mit kooperierenden Standardisierungsgremien, indem ein neuer SQL-Standard in Arbeit ist, der die Verarbeitung von Objekten beliebiger Struktur und Größe erlaubt. Zudem können Funktionalitäten und Konsistenzbedingungen dann direkt an die Objekte in der Datenbank gekoppelt werden.

Die großen Anbieter relationaler Datenbanken integrieren teilweise schon vorab diese im neuen SQL-Standard angedachten Eigenschaften in ihre Systeme. Da aber kein Standard verabschiedet ist, unterscheiden sich hier die Konezpte und Realisierung teilweise noch erheblich, was die Wahl nach der geeigneten Datenhaltung nicht einfacher macht.

Um bei der Entwicklung eines Systems unabhängig von einzelnen Produkten und Herstellern zu sein, beschränkt man sich auf Standards, die von vielen Produkten eingehalten und unterstützt werden. Ein solcher Standard ist SQL [ANS92]. Dieser reicht aber nicht aus, um die unterschiedlichen Anforderungen aus 3.1 abzudecken. Die notwendige Erweiterung des Standards ist noch nicht verfügbar, so daß hier ein Teil der Anforderungen von der Datenhaltung abgezogen werden muß, um in der Anwendung realisiert zu werden.

In den folgenden Abschnitten wird ein System beschrieben, daß viele Elemente dessen in sich vereinigt, was in der Praxis gefordert wird.

# 4 *Leu* als Beispiel für sinnvolle Integration

In diesem Abschnitt werden zunächst die unterschiedlichen Verwendungszwecke von Datenmodellen in *Leu* erörtert. Danach wird diskutiert, wie relationale und objektorientierte Modellierungsmöglichkeiten in *Leu* integriert werden und wie die Realisierung dieser Modellierungsmöglichkeiten aussieht.

## 4.1 Datenmodellierung in *Leu*

Die Datenmodellierung innerhalb von *Leu* ist Grundlage der Entwicklung kommunikationsorientierter Anwendungen mit *Leu*. Als erste Anwendung wird mit *Leu* das Wohnungswirtschafts-Informationssystem WIS realisiert. In WIS ist es erforderlich, sowohl strukturierte Daten (zum Beispiel Daten über Mietverträge, Wohnungen und Nutzungseinheiten) als auch Daten verschiedener Formate (zum Beispiel CAD-Daten) zu verwalten. Je nach Art des beschriebenen Objekts können unterschiedliche Operationen angewendet werden (Mietverträge können gekündigt und abgeschlossen werden, Nutzungseinheiten können vermietet, gebaut, renoviert, kalkuliert werden). Aufgrund dieser Überlegungen wird in *Leu* der Begriff des *Objekts* in den Mittelpunkt der Datenmodellierung gestellt. Datenmodelle in *Leu* beschreiben Objekttypen und deren Zusammenhänge. Abbildung 1 zeigt ein kleines *Leu*-Datenmodell, das aus vier Objekttypen besteht. Zu den Objekttypen werden Attribute definiert. Der Objekttyp *Person* beispielsweise besteht aus den Attributen *Name*, *Vorname*, *Familienstand*, *Geburtsdatum*

und *Telefonverbindung*. Der *Vorname* ist ein Attribut vom Typ *Menge von Strings*. Das bedeutet, daß eine *Person* mehrere *Vornamen* haben kann. Der Wertebereich des Attributs *Familienstand* ist durch den Aufzählungstyp *Fam.st* (ledig, verheiratet, geschieden, verwitwet) definiert. Die anderen Attribute sind durch elementare Typen typisiert. Alle Attribute bis auf die *Telefonverbindung* sind *muß*-Attribute, das heißt sie müssen für jedes Objekt des Typs *Person* definiert sein. Die *Telefonverbindung* ist ein *kann*-Attribut. Das bedeutet, daß nicht jede *Person* eine *Telefonverbindung* haben muß.

Das Beispiel aus Abbildung 1 verdeutlicht, daß sich der gewählte Modellierungsformalismus an traditionellen Entity/Relationship-Ansätzen [Bar90] orientiert. In diese Entity/Relationship-artige Beschreibung sind Sprachmerkmale integriert, die zur Erfüllung der objektorientierten und sonstigen Anforderungen an ein DBMS (vgl. Abschnitt 3.1) dienen. Details der graphischen Darstellung werden im Rest dieses Abschnitts erörtert.

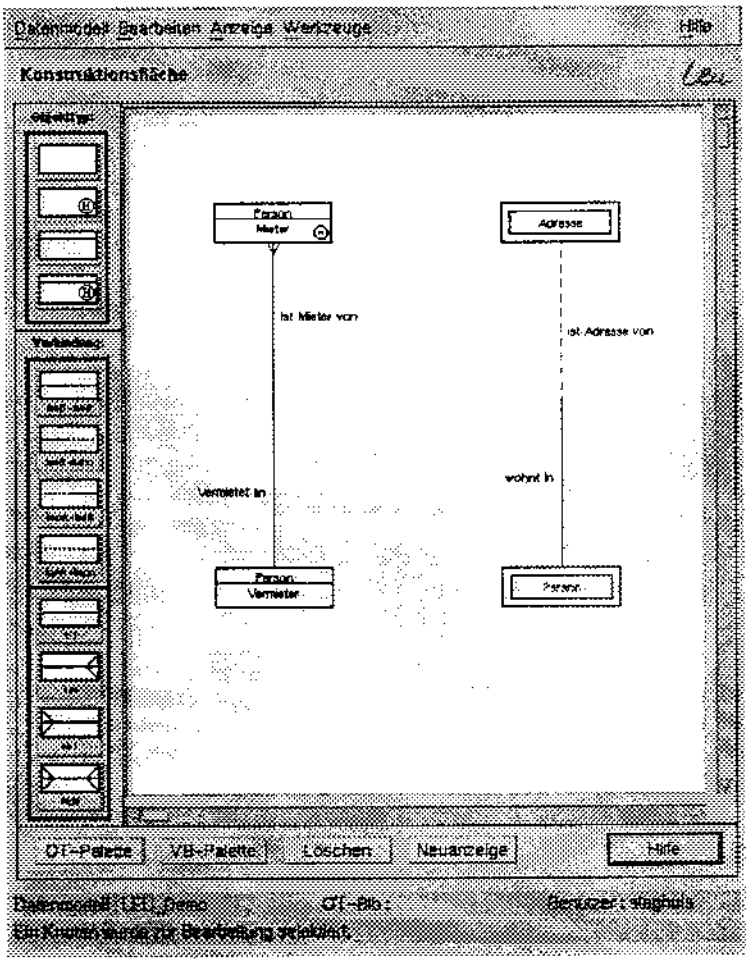


Abbildung 1: Beispiel eines *Leu*-Datenmodells

Datenmodelle sind das grundlegende Strukturierungsmittel in *Leu*. Komplexe Anwendungen bestehen in der Regel aus einer Vielzahl von Geschäftsprozessen. Die Menge der Geschäftsprozesse lassen sich nach der Menge der Objekttypen, auf denen die Prozesse basieren, zerlegen. Durch die Zusammenfassung von Geschäftsprozessen zu fachlich zusammenhängenden Gruppen und durch die Identifikation der Objekttypen, deren Objekte von diesen Prozessen erzeugt/manipuliert werden, kommt man in *Leu* zu *Teildatenmodellen* als Alternative zu unternehmensweiten Datenmodellen.

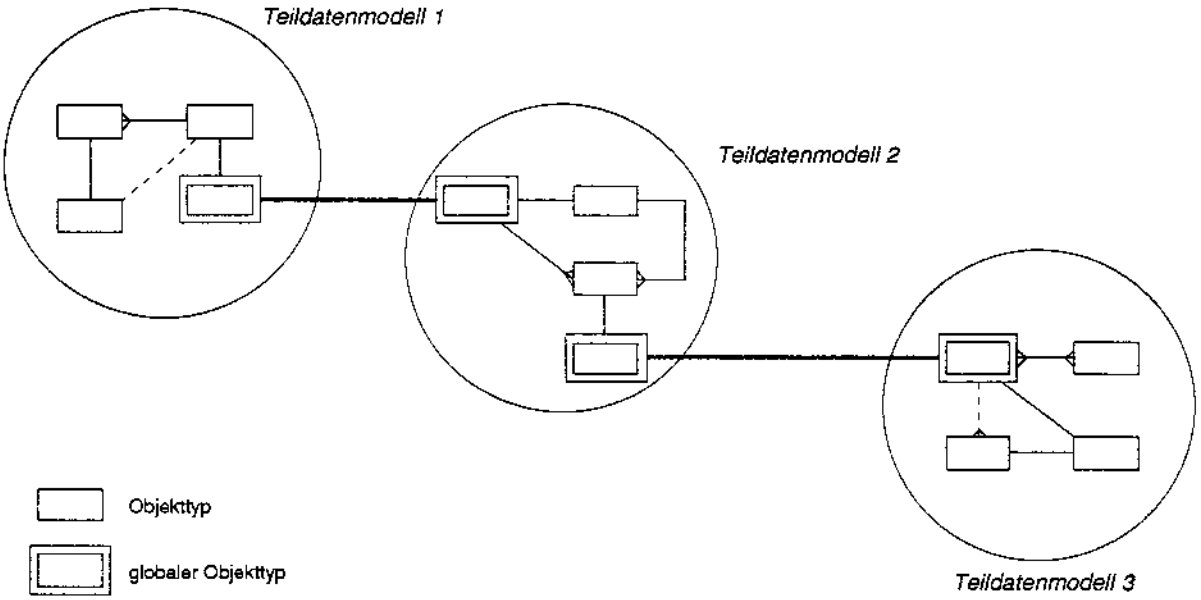


Abbildung 2: Zusammenhang von Teildatenmodellen

Abbildung 2 zeigt, wie logisch zusammengehörende Objekttypen zu Teildatenmodellen zusammengefaßt werden. Zwischen diesen Teildatenmodellen existieren Schnittstellen, die als *Klebestellen* zwischen den Teildatenmodellen dienen. Die Objekttypen, die als Klebestellen fungieren, werden in *Leu globale Objekttypen* genannt. Globale Objekttypen werden innerhalb eines Teildatenmodells definiert und können von anderen Teildatenmodellen importiert werden. In Abbildung 2 deuten die dicken Linien zwischen Objekttypen unterschiedlicher Teildatenmodelle an, daß die beiden verbundenen Objekttypen genau einen Objekttypen darstellen, der in einem Teildatenmodell definiert und in das andere importiert wurde. In dem in Abbildung 1 dargestellten Teildatenmodell ist der Objekttyp *Person* als ein globaler Objekttyp definiert. Dies wird graphisch durch die zweifache Umrandung dargestellt. Der Objekttyp *Person* ist als globaler Objekttyp angelegt, weil angenommen wird, daß Objekte des Typs *Person* auch in anderen Teildatenmodellen manipuliert werden müssen. Durch die Verwendung von globalen Objekttypen wird ein unternehmensweites Datenmodell in überschau- und wartbare Teile zerlegt.

Teildatenmodell dienen weiterhin dazu, Berechtigungen für Anwender zu definieren. Mit Hilfe des *Leu*-Berechtigungssystems ist es möglich, auf der Grundlage der Teildatenmodelle Berechtigungen für das Anlegen, Ändern, Löschen und Lesen einzelner Objekttypen oder ganzer Gruppen von Objekttypen zu definieren. Diese Berechtigungen können entweder direkt einzelnen Anwendern zugewiesen werden oder zunächst anwender-unabhängig an Rollen vergeben werden. Diese Rollen können zur Ausführung von Anwendern *eingenommen* werden.

Ein weiterer Zweck von Datenmodellen besteht darin, daß sie die Grundlage der Generierung von Standardfunktionalitäten sind. In diesem Zusammenhang ist die Generierung von Standarddialogen aus Objekttypbeschreibungen und die Bereitstellung von Standardoperationen für Objekttypen zu nennen. Ein ähnlicher Weg zur Generierung von Standarddialogen aus Objekttypbeschreibungen wird im JANUS-System verfolgt [Bal93]. Standarddialoge erlauben das Anlegen, das Manipulieren und das Löschen von Objekten des Objekttyps, zu dem der Standarddialog gehört. Darüberhinaus sind Standarddialoge die Grundlage von Individualdialogen. Individualdialoge entstehen durch die manuelle Nachbearbeitung von Standarddialogen (bei-

spielsweise um die Menge der angezeigten Attribute einzuschränken oder um sie um Attribute anderer Objekttypen zu erweitern). Individuelle Dialoge werden mit dem *Leu*-Dialogeditor erstellt.

Neben der Bereitstellung von Standardoperationen (Drucken, Anlegen, Ändern, Löschen) pro Objekttyp erlaubt es die *Leu*-Datenmodellierung, den einzelnen Objekttypen weitere Operationen zuzuordnen. Diese in der Modellierung zugeordneten Operationen werden sowohl in den Standarddialogen als auch in Individualdialogen berücksichtigt. Konkret bedeutet das, daß im Bearbeitungsmenü des Standarddialogs zu einem Objekttyp die diesem Objekttyp zugeordneten Operationen auftauchen. Der mit diesem Dialog arbeitende Anwender hat damit die Möglichkeit die angebotenen Operationen aufzurufen. Wenn beispielsweise dem Objekttyp *Person* die Operation *Namensänderung* zugeordnet ist, dann wird diese Operation im Berabeitungsmenü des Standarddialogs für *Personen* sowie in allen Individualdialogen, die für die Bearbeitung von *Personen* verwendet werden, angeboten. Ein Anwender, der mit einem dieser Dialoge arbeitet, hat die Möglichkeit diese Operation aufzurufen, soweit seine Berechtigungen dies zulassen.

In der Modellierung von Datenmodellen in *Leu* ist es möglich, Vererbungszusammenhänge zwischen Objekttypen zu definieren. In Abbildung 1 ererben die Objekttypen *Mieter* und *Vermieter* vom Objekttyp *Person*. Dieses wird bei der Darstellung dieser Objekttypen durch eine horizontale Linie dargestellt. Oberhalb dieser Linie steht der Objekttyp, von dem ererbt wird. Die Vererbungsbeziehung zwischen zwei Objekttypen bedeutet, daß der ererbende Objekttyp neben den eigenen Attributen auch die Attribute des vererbenden Objekttyps hat. Somit ist beispielsweise festgelegt, daß der Objekttyp *Mieter* alle Attribute des Objekttyps *Person* hat. Zusätzliche Attribute werden in der Definition des Objekttyps *Mieter* angegeben. Die Operationen des vererbenden Objekttyps werden in den Dialogen des ererbenden Objekttyps angeboten. Unter der Annahme, daß dem Objekttypen *Person* die Operation *Namensänderung* zugeordnet wurde, wird diese Operation auch in den Dialogen zur Manipulation von *Mieter*-Objekten angeboten. Das Konzept der Mehrfachvererbung wird in *Leu* nicht unterstützt. Jeder Objekttyp kann also nur von maximal einem Objekttypen erben.

Die *Leu*-Datenmodellierung erlaubt es, Objekttypen als zu historisierende Objekttypen zu definieren. In dem in Abbildung 1 dargestellten Datenmodell ist der Objekttyp *Mieter* als ein zu historisierender Objekttyp dargestellt. Wenn ein Objekttyp das Historisierungsattribut hat, dann werden Änderungen an Objekten dieses Typs protokolliert. Das bedeutet, daß für jede Änderung aufgezeichnet wird, wer welches Objekt wann geändert hat und wie die Wertänderung aussieht. In der Modellierung kann die Länge des Aufzeichnungszeitraums angegeben werden, so daß Aufzeichnungen eines bestimmten Alters gelöscht werden. Die Definition des Objekttyps *Mieter* als zu historisierender Objekttyp beruht auf der Annahme, daß es nötig ist, auch auf Informationen über ehemalige Mieter zuzugreifen.

## 4.2 Realisierung der *Leu*-Datenmodellierung

Datenmodellierung in *Leu* kombiniert Entity/Relationship-Diagramme mit objektorientierten Modellierungsansätzen. Zusätzliche Anforderungen haben zu einigen zusätzlichen Erweiterungen von Entity/Relationship-Diagrammen geführt.

Die Basis von *Leu* ist eine relationale Datenbank, wobei die Schnittstelle von *Leu* zur Datenbank unter strikter Einhaltung der SQL-ANSI-Norm [ANS92] verwirklicht ist. Das verfolgte Ziel ist die einfache Portierbarkeit.

Grundlage eines Informationssystems in *Leu* ist ein Datenmodell, welches mit dem Formalismus eines erweiterten Entity/Relationship-Modells dargestellt wird. Jedes Entity wird in *Leu* als Objekttyp und jede Relationship als Verbindung bezeichnet. Die Beschreibung wird durch eine automatische Generierung auf ein Datenbankschema abgebildet. Jeder Objekttyp und jede Verbindung werden auf mindestens eine Relation des Datenbankschemas umgesetzt. Die Attribute der Objekttypen spiegeln sich in den Attributen der Relationen wider.

### 4.2.1 Grundsätzliche Eigenschaften

Die mit *Leu* modellierten Informationssysteme müssen einige grundsätzliche Anforderungen aus der Praxis erfüllen:

**Recovery und Datenkonsistenz** wird von *Leu* sichergestellt, da alle in *Leu* anfallenden Objekte in der Datenbank gespeichert werden und somit die Eigenschaften der integrierten Datenbank genutzt werden. Nach einem Programmfehler, Stromausfall o. ä. ist *Leu* somit in der Lage auf einem konsistenten Zustand wieder aufzubauen.

**Ad-Hoc-Anfragen** lassen sich mit dem *Leu*-Anfrageeditor definieren und interpretieren. Die im Anfrageeditor verwendete Sprache wird LQL (*Leu* Query Language) genannt. Jede LQL-Anfrage wird auf eine ANSI-SQL-Anfrage [ANS92] auf der zugrundeliegenden, relationalen Datenbank abgebildet. Die Interpretation von LQL-Anfragen muß performant sein. Deshalb wird fast jede LQL-Anfrage auf genau eine ANSI-SQL-Anfrage abgebildet.

Das **Transaktionskonzept** der integrierten Datenbank ist auf die Ebene von Anfragen und Dialogen gebracht. Das heißt, daß jede Anfrage und jeder Dialog eine Transaktion ist. Der Anwendungsentwickler ist für die Definition von Transaktionen in Batch-Anwendungen verantwortlich.

### 4.2.2 Relationale Eigenschaften

**Speichermanagementaufgaben** wie die Indexierung, Speicherorganisationen, Pufferdimensionierungen etc. werden von der zugrundeliegenden Datenbank übernommen, um einen guten Durchsatz, eine gute Antwortzeit u. ä. zu gewährleisten. Die Indexierung ist ein Merkmal relationaler Datenbanken. In *Leu* kann man auf einer Gruppe von Attributen eines Objekttyps einen Index definieren, der bei der Generierung auf einen physikalischen Datenbankindex umgesetzt wird. Die Anzahl der Indizes ist begrenzt, da sonst negative Folgen auf die Antwortzeiten von Ad-Hoc-Anfragen oder die Ausführung von Standard- bzw. Individualdialogen auftreten könnten.

**Restriktionseigenschaften** können für Attribute von Objekttypen definiert werden. Damit wird zur Definitionszeit festgelegt, ob ein Attribut eines Objektes mit einem Wert versehen werden *muß* oder *kann*. Die Restriktionseigenschaft wird durch die NOT NULL Definition des entsprechenden Relationsattributs realisiert. Zusätzliche Unterstützung liefern die Standard- oder Individualdialoge, weil hier der Anwender bei der Manipulation eines Attributes mit *Muß*-Restriktion gezwungen wird, einen Wert anzugeben.
Verbindungen besitzen neben der Restriktionseigenschaft auch die Möglichkeit der Stelligkeitsdefinition. Die Restriktionseigenschaft stellt sicher, daß ein Objekt eines Objekttyps mit einem Objekt eines anderen Objekttyps in Beziehung stehen *muß* oder *kann*. Wie bei Objekttypen wird die Restriktion bei Verbindungen durch eine NOT NULL Definition des entsprechenden Relationsattributs sichergestellt. Die Stelligkeitsüberwachung wird durch ein algorithmische Lösung erzielt.

**Schlüssel** haben in relationalen Datenbanksystemen die Aufgabe, Sätze von Relationen eindeutig zu identifizieren. Dies impliziert, daß keine Sätze mit wertgleichem Schlüssel existieren können. In *Leu* dienen Schlüssel nicht zur eindeutigen Satzidentifikation, denn diese geschieht über interne Objektkennungen. Nichtsdestotrotz können in *Leu* Schlüssel auf Attributen von Objekttypen definiert werden, um Konsistenzbedingungen zu beschreiben. Die Eindeutigkeit der Objekte bezüglich der Schlüsselattribute wird algorithmisch sichergestellt.

### 4.2.3 Objektorientierte Eigenschaften

*Leu* bietet eine Anzahl objektorientierter bzw. teilweise objektorientierter Eigenschaften, die durch Anforderungen aus der Praxis entstanden sind.

**Komplexe Objekte** werden aus einfachen Objekten über Konstruktoren aufgebaut. Die *Leu*-Datenmodellierung erlaubt es, Attribute als Tupel zu einem Objekttyp zusammenzufassen. Der Mengenkonstruktor bezieht sich, wie der Listenkonstruktor, auf die Attributebene. Ein Attribut kann als Menge definiert werden. Die Elemente der Menge werden zur Definitionszeit in einer eigenen Relation gespeichert. Gleichermaßen erhalten Attribute mit Listenkonstruktor eine Relation, in der die Elemente der Liste numerisch sortiert zur Laufzeit abgelegt werden.

**Erweiterbarkeit** der vordefinierten Objekttypen wird durch die in *Leu* angebotenen Konstruktoren gewährleistet. Die Menge der Operationen kann durch anbindbare Operationen und LQL-Anfragen erweitert werden.

**Mulitmedia**-Verarbeitung ist ein typisches Merkmal von Objektorientierung. Innerhalb der *Leu*-Datenmodellierung können Informationstypen, die als binäre Daten zuzüglich sämtlicher Formatangaben und typbezogener Operationen in einem Schema abgelegt werden, definiert werden.

**Objektidentität** wird in *Leu* vollständig unterstützt, d. h. es können wertgleiche Objekte existieren. In der Realisierung wird dies unter Hinzunahme einer systemweit eindeutigen Objektkennung ermöglicht. Somit sind zwei Möglichkeiten des Vergleichs von Objekten erfüllt.

- Zwei Objekte sind *identisch*, wenn die systemweit eindeutigen Objektkennungen identisch sind.
- Zwei Objekte sind *gleich*, wenn die systemweit eindeutigen Objektkennungen unterschiedlich und die Daten der Objekte wertgleich sind.

**Datenkapselung** in objektorientierten Systemen bedeutet, daß der Zugriff auf Daten nur über die vorgesehenen Operationen möglich ist. Somit wird ein Datenteil und ein Operationsteil unterschieden. In *Leu* beschreibt der Datenteil die im Datenmodell definierten Objekttypen und Verbindungen. Die automatische Generierung eines Datenbankschemas verbirgt die konkrete Datenhaltung. In *Leu* wird der Operationsteil durch Ad-Hoc-Anfragen in LQL, durch erzeugte Standardfunktionen und durch Standard- und Individualdialoge beschrieben. Die Implementierung der Manipulationsoperationen ist für den Anwender nicht sichtbar.

**Vererbung** wird in *Leu* durch ein einfaches Konzept unterstützt. Ein Objekttyp kann die Attribute eines anderen Objekttyps erben. Sowohl für den vererbenden als auch für den ererbenden Objekttyp wird eine Relation erzeugt. Objekte des Vaters werden mit Objekten des Sohnes über eine verbindende Relation im Datenbankschema zueinander in Beziehung gesetzt. Der Einsatz einer verbindenden Relation für die Vererbung besitzt den Vorteil der Redundanzfreiheit.

### 4.2.4 Sonstige Eigenschaften

**Strukturierung** von Datenmodellen wird durch die Einführung von Teildatenmodellen unterstützt. Eine Verschmelzung von Teildatenmodellen wird über *globale Objekttypen*, die in den Teildatenmodellen vorkommen, erreicht. Die Umsetzung *globaler Objekttypen* auf ein Datenbankschema ist ähnlich zur Umsetzung anderer Objekttypen. Es ändert sich lediglich die Stelligkeit der Zuordnungsmöglichkeiten zu Teildatenmodellen. Ein lokaler Objekttyp kommt immer nur in einem Teildatenmodell vor, während ein globaler Objekttyp in mehreren Teildatenmodellen auftreten kann. Die Manipulation von Objekten eines nicht globalen Objekttyps ist somit nur innerhalb der Geschäftsprozesse genau eines Teildatenmodells möglich. Im Gegensatz dazu existieren bei globalen Objekttypen Einstiegsmöglichkeiten von unterschiedlichen Teildatenmodellen.
Da es zu jedem globalen Objekttypen nur genau eine Relation gibt (unabhängig davon, in wievielen Teildatenmodellen der globale Objekttyp vorkommt), dienen die Relationen, die zu globalen Objekttypen erzeugt werden, zur redundanzfreien Speicherung zentraler Objekte.

**Historisierung** ist in der industriellen Praxis wünschenswert und wird in *Leu* unterstützt. Der Anwendungsentwickler kann während der Definitionszeit entscheiden, ob die Werte der Attribute eines Objekttyps historisiert werden sollen. Hierbei kann definiert werden, daß jede Wertänderung historisiert werden soll oder daß nur die Änderungen bestimmter Attribute aufgezeichnet werden sollen. Jegliche Attribute, die von einem Grundtyp wie z. B. *BOOLEAN, INTEGER, REAL, STRING, TIME* oder *DATE* definiert sind, können historisiert werden. Wird die Historisierung für Attribute eines Objekttyps gefordert, so werden zusätzlich zu den generierten Relationen des Objekttyps noch weitere Relationen für die zu historisierenden Objektwerte erzeugt. In diesen zusätzlichen *Historisierungsrelationen* werden der Bezug zum Originalobjekt sowie der ändernde Benutzer und Datum und Zeit der Änderung verwaltet. Bei der Änderung eines Objekts, das einer Historisierung unterliegt, wird das ursprüngliche Objekt mit der Kennung des Benutzers und des Manipulationszeitpunktes aus den normalen Relationen in die Historisierungsrelationen übertragen.

**Berechtigungen** werden rollenbezogen vergeben. Nicht jeder Anwender sollte Zugang zu sämtlichen Daten eines modellierten Informationssystems haben. So bietet *Leu* die Möglichkeit, Anwendern Rollen mit manipulationsspezifischen Berechtigungen auf Objekttypen zuzuordnen. Realisiert wird die Berechtigungsbeziehung zwischen den Rollen und Objekttypen durch eine attributierte Verbindung. Das Attribut der Verbindung definiert die Art der Berechtigung, zum Beispiel, ob ein Anwender Objekte eines Objekttyps löschen oder lesen darf.

Im folgenden Abschnitt wird über Erfahrungen des Einsatzes von *Leu* in der Praxis berichtet.

# 5 Erfahrungen mit *Leu*

*Leu* ist zur Entwicklung von WIS im Einsatz. Wie bereits erwähnt, unterstützt *Leu* die Modellierung statischer Aspekte von Informationssystemen mit Hilfe von Datenmodellen und die Modellierung dynamischer Aspekte von Informationssystemen mit Hilfe von Geschäftsprozeßmodellen. In der gegebenen Situation sind die Datenmodellierungswerkzeuge von *Leu* unmittelbar als Erleichterung aufgefaßt worden. Diese Akzeptanz basiert auf verschiedenen Gründen: Zum einen sind Probleme der Datenmodellierung (und insbesondere der Änderungen von Datenmodellen) in der Großrechnerwelt bekannt. Zum zweiten waren Kenntnisse der Entity/Relationship-Modellierung vorhanden. Zum dritten sind die in *Leu* realisierten objektorientierten Erweiterungen von Entity/Relationship-Diagrammen immer nur in Absprache und

nach vorheriger Motivation eingeführt worden. Zum vierten sind individuelle Wünsche (wie zum Beispiel der Wunsch die Historisierung von Objekttypen direkt im Datenmodell angeben zu können) weitgehend berücksichtigt worden.

Aufgrund dieses abgesprochenen und *vorsichtigen* Vorgehens sind die Datenmodellierungsmöglichkeiten relativ schnell verwendet worden. Demgegenüber sieht die Situation in der Modellierung von Geschäftsprozessen anders aus. Dies dürfte vor allem daran liegen, daß es erheblich aufwendiger war, den Sinn der Modellierung dynamischer Aspekte mit Hilfe von Geschäftsprozeßmodellen zu motivieren. Inzwischen ist jedoch auch hier ein Stand erreicht, in dem die Geschäftsprozesse, die es in WIS zu unterstützen gilt, in einheitlicher Art und Weise beschrieben sind.

## Literatur


[ABD+] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. pages 40–57.

[ANS92] *American National Standard for Information Systems – Database Language - SQL.* American National Standards Institute, New York, 1992. (ANSI Standard X3.135-1992).

[Bal93] Helmut Balzert. Der JANUS-Dialogexperte: Vom Fachkonzept zur Dialogstruktur. In E. Doberkat, editor, *Proceedings der GI-Fachtagung Software-Technik 93*, pages 62–72, Dortmund, November 1993.

[Bar90] Richard Barker. *Entity Relationship Modelling.* Case*Method (Computer aided systems engineering). Workingham, England, 1990.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. In P. Baxendale, editor, *Communications of the ACM*, volume 13, pages 377–387, June 1970.

[Cod79] E. F. Codd. Extending the Database Relational Model to Capture More Meaning. In *ACM Transactions on Database Systems*, volume 4, 1979.

[DG90] Wolfgang Deiters and Volker Gruhn. Managing Software Processes in MELMAC. In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, pages 193–205, Irvine, California, USA, December 1990. Appeared as ACM Software Engineering Notes, 15(6), December 1990.

[Dit86] Klaus R. Dittrich. Object-oriented database systems: The notation and the issues. In Klaus R. Dittrich and U. Dayal, editors, *Proceedings of the* 2nd *International Workshop on Object-Oriented Database Systems*, number 334 in LNCS, pages 2–4, Pacific Grove, California, 1986. IEEE Computer Science Press.

[Gru91] Volker Gruhn. *Validation and Verification of Software Process Models.* PhD thesis, University Dortmund, June 1991. Appeared as Technical Report No. 394/91.

[Gru93] Volker Gruhn. Entwicklung von Informationssystemen in der LION-Entwicklungsumgebung. In G. Scheschonk and W. Reisig, editors, *Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*, pages 31–45, Berlin, FRG, September 1993. Springer-Verlag.

[TF76] R. Taylor and R. Frank. Codasyl data base management systems. *ACM Computing Surveys*, 8(1):67–104, March 1976.

[TL76] D. C. Tsichritzis and F. H. Lochovsky. Hierachichal data base management: A survey. *ACM Computing Surveys*, 8(1), March 1976.