Informatik – Forschung & Entwicklung
Vol. 14, No. 4, pp. 218–226, 1999 Springer-Verlag, Berlin/Heidelberg

Restrukturierung objektorientierter Datenbankschemata mittels formaler Begriffsanalyse

Ingo Schmitt, Stefan Conrad*

Universität Magdeburg, Institut für Technische und Betriebliche Informationssysteme,
Universitätsplatz 2, D-39106 Magdeburg, Deutschland
Tel.: +49-391-67-12994 bzw. -18066, Fax: +49-391-67-12020
e-mail: {schmitt,conrad}@iti.cs.uni-magdeburg.de

Manuskript vom 15. Juli 1999

Zusammenfassung Eine Restrukturierung eines gegebenen Datenbankschemas kann aus unterschiedlichen Gründen notwendig werden. So kann dadurch z.B. die Verständlichkeit des Schemas verbessert werden. Im Bereich der relationalen Datenbanken steht die Normalisierungstheorie zur Verfügung, um z.B. Redundanz zu vermeiden. Der Normalisierung liegt dabei ein Begriff der Äquivalenz von Datenbankschemata zugrunde. Im Bereich der objektorientierten Datenbanken gibt es bislang kaum vergleichbare Ansätze. In diesem Beitrag präsentieren wir daher einen Ansatz, mit dem Klassenhierarchien in objektorientierten Datenbankschemata in eine „normalisierte“ Form transformiert werden können. Dazu muß zunächst eine extensionale Analyse durchgeführt werden, die die notwendigen Informationen über extensionale Beziehungen zwischen den gegebenen Klassen liefert. Anschließend wenden wir Konzepte der formalen Begriffsanalyse an, um eine „normalisierte“ Klassenhierarchie abzuleiten.

Schlüsselwörter: Klassenhierarchie, Restrukturierung, Begriffsanalyse, Datenbankentwurf, Objektorientierung, Reverse Engineering

Abstract Different reasons may require to restructure an existing database schema. For instance, restructuring can improve the comprehensibility of a schema. For relational databases there is the relational normalization theory which provides a means of avoiding redundant storage of data. Normalization is based on a notion of equivalence for database schemata. At the time being, there are only very few comparable approaches to normalizing object-oriented databases. In this article we present an approach to transforming class hierarchies in object-oriented database schemata into a “normal form”. For this transformation we first have to perform an extensional analysis which provides the necessary information about extensional relationships among the classes in a given schema. Then, we apply the framework of formal concept analysis in order to derive a “normalized” class hierarchy.

* Diese Arbeit wurde teilweise vom Land Sachsen-Anhalt unter FKZ 1987A/2527R unterstützt.

Key words: class hierarchy, restructuring, concept analysis, database design, object-orientation, reverse engineering

CR Subject Classification: D.2.2, D.2.7, D.2.10, D.3.3, E.2, H.2.1, H.2.4, H.2.5

1 Einleitung

In der Praxis wird der Entwurf eines objektorientierten Datenbankschemas oft von spezifischen Charakteristika des objektorientierten Datenbanksystems beeinflußt, das später als Implementierungsplattform genutzt werden soll. Eine charakteristische Eigenschaft ist beispielsweise, ob ein Rollenkonzept von dem System unterstützt wird. Ein Rollenkonzept erlaubt einem Objekt, gleichzeitig Instanz mehrerer Klassen zu sein, ohne daß eine gemeinsame Subklasse existieren muß [22, 8]. Betrachtet man etwa objektorientierte Datenbanksysteme, die in ihrer Entwicklung im wesentlichen von objektorientierten Programmiersprachen abstammen, so wird ein solches Rollenkonzept von diesen nicht unterstützt.

Außerdem gibt es in manchen objektorientierten Datenbanksystemen die Einschränkung, daß mehrfache Vererbung (also eine Klasse, die gleichzeitig mehrere Superklassen hat) nicht erlaubt wird. Zum Beispiel kennen SMALLTALK-basierte Systeme mehrfache Vererbung prinzipiell nicht. In solchen Fällen werden typischerweise konzeptionell zusammengehörige Objekte auf mehrere Klassen aufgeteilt. Damit steigt die Anzahl der Klassen im objektorientierten Datenbankschema, und die Struktur des Schemas gibt keinen Anhaltspunkt mehr für die Zusammengehörigkeit der Objekte. Die Verständlichkeit, die eigentlich eines der Hauptziele der objektorientierten Modellierung ist, leidet dabei erheblich. Ferner ist auch die Benutzung eines solchen Schemas oft aufwendig und unintuitiv für Benutzer und Anwendungsprogrammierer.

Neben solchen systemabhängigen Beschränkungen gibt es noch andere Ursachen, warum objektorientierte Datenbankschemata (und auch Datenbankschemata im allgemeinen) umständlich und unintuitiv sein können. Wenn zum Beispiel ein Datenbankentwerfer nicht mit allen Modellierungskonzepten des Datenmodells und ihrer Verwendung vertraut ist, wird er versuchen, mit den ihm vertrauteren Konzepten den Diskursbereich zu modellieren und dabei andere Konzepte zu simulieren, die das Datenmodell zwar direkt anbietet, er selbst aber nicht verwenden kann oder will. Das Ergebnis ist dann oft ein Datenbankschema, das für Benutzer und Anwendungsprogrammierer schwer nachzuvollziehende Entwurfsentscheidungen hinsichtlich der Modellierung enthält.

Ein konzeptionelles Datenbankschema sollte jedoch unabhängig vom verwendeten Datenbanksystem sein und die Modellierungskonzepte des Datenmodells in adäquater Weise anwenden. Dies ist für die Verständlichkeit von Datenbankschemata eine unverzichtbare Forderung. Die Restrukturierung bestehender Datenbankschemata kann hier helfen bzw. wird oft sogar notwendig, um die Wartbarkeit des Systems sowie die Portierbarkeit für einen eventuellen, zukünftigen Wechsel des Datenbankmanagementsystems erheblich zu verbessern.

Eine solche Restrukturierung erfordert in der Regel ein Reverse Engineering des bestehenden Datenbankschemas. So muß häufig erst ein konzeptionelles Datenbankschema in einem geeigneten, nicht implementierungsnahen Datenmodell aus dem vorgefundenen Schema gebildet werden.

Eine Ursache hierfür ist die Tatsache, daß häufig aus vordergründigen Zeit- und Kostenargumenten auf einen methodisch sauberen Datenbankentwurf verzichtet wurde und gleich ein Ad-Hoc-Schema implementiert wird. Die darin getroffenen Entwurfs- und vor allem Implementierungsentscheidungen sind später kaum noch nachvollziehbar, und das Schema ist nur noch mit sehr hohem Aufwand wartbar.

Nicht nur in der Forschung, sondern auch in der Praxis hat man sich in letzter Zeit viel mit dem Reverse Engineering von relationalen Datenbankschemata in objektorientierte konzeptionelle Schemata beschäftigt. Hierzu ist in der Regel eine semantische Anreicherung notwendig [10, 6], um die wenigen Konzepte des relationalen Datenmodells in adäquater Weise auf objektorientierte Modellierungskonzepte abzubilden, da im relationalen Modell viele unterschiedliche Dinge mit den gleichen Konzepten modelliert werden müssen. Zum Beispiel gibt es in objektorientierten Modellen das Konzept der Spezialisierung, das im relationalen Modell typischerweise mit Hilfe von Fremdschlüsselbeziehungen nachgebildet wird.

Im Prozeß des Reverse Engineering bestehender Datenbankschemata konzentrieren wir uns auf den speziellen Aspekt der Restrukturierung von Klassenhierarchien. Ob diese Klassenhierarchien direkt aus einem bereits ursprünglich objektorientierten Schema stammen oder im Reverse-Engineering–Prozeß, z.B. aus einem relationalen Schema, durch Transformation entstanden sind, ist für die weitere Betrachtung unerheblich. Wir wollen beliebige Klassenhierarchien derart restrukturieren, daß wir als Ergebnis eine Hierarchie erhalten, die den Anforderungen genügt, die man von einer guten konzeptionellen Modellierung erwartet.

Diese Restrukturierung von Klassenhierarchien in eine eindeutig bestimmte, „normalisierte“ Form könnte den Kern einer Normalisierungstheorie für objektorientierte Datenbankschemata darstellen. In Abschnitt 6 werden wir daher die hier vorgestellte Restrukturierung mit der relationalen Normalisierungstheorie vergleichen.

Die Restrukturierung eines Schemas sollte gewissen Qualitätskriterien genügen. Insbesondere darf sich die Informationskapazität nicht ändern. Dafür wird z.B. in [11] ein Äquivalenzbegriff für Schemata definiert, der auf der Idee der gegenseitigen Dominanz beruht. Sinngemäß sind zwei Schemata äquivalent, wenn eine bijektive Abbildung zwischen den zugehörigen möglichen Datenbankzuständen angegeben werden kann.

Im folgenden präsentieren wir unseren Ansatz zur „Normalisierung“ von Klassenhierarchien. Dabei werden nur strukturelle Aspekte berücksichtigt. Weiterhin gehen wir von atomaren Attributen aus, erlauben also keine Typkonstruktionen.

Im nächsten Abschnitt führen wir zwei einfache Beispielschemata ein, an denen wir im weiteren Verlauf unseren Ansatz exemplarisch vorstellen. Anschließend stellen wir eine Zwischenrepräsentation vor, die ursprünglich für die Schemaintegration entwickelt wurde [16, 15, 18, 14]. In dieser Zwischenrepräsentation werden die extensionalen Beziehungen zwischen den einzelnen Klassen bzw. Objektmengen dargestellt.

Basierend auf dieser Darstellung von Klassenhierarchien wenden wir dann Konzepte der formalen Begriffsanalyse an [23, 7]. Die formale Begriffsanalyse basiert auf der Verbandstheorie. In der Begriffsanalyse wird ein Konzeptverband aufgebaut. Ein solcher Konzeptverband kann in unserem Fall als Klassenhierarchie verstanden werden. Anhand der zuvor eingeführten Beispielschemata zeigen wir dann die Anwendung dieses Ansatzes. Abschließend vergleichen wir unseren Ansatz mit anderen Arbeiten.

2 Beispielschemata

Um unseren Ansatz zu demonstrieren, führen wir zunächst zwei umständliche und unintuitive Beispielschemata ein, die denselben Diskursbereich einmal relational und einmal objektorientiert beschreiben und jeweils restrukturiert werden sollen. Die Beispiele wurden so konstruiert, daß ihr schlechter Entwurf sofort ersichtlich wird. Dies ist bei größeren Schemata nicht immer so leicht möglich. Wir verzichten hier auf die Behandlung von Integritätsbedingungen und verweisen statt dessen auf die Arbeit [4].

Beispiel: In unserer exemplarischen Diskurswelt haben wir Personen (`Person`) mit den Attributen Sozialversicherungsnummer (`SVNr`) und Name (`Name`). Studenten

(`Student`) und Mitarbeiter (`Mitarbeiter`) sind spezielle Personen, die zusätzliche Attribute haben: Studenten haben ein Fach (`Fach`) und Mitarbeiter ein Gehalt (`Gehalt`). Es gibt Personen, die gleichzeitig Student und Mitarbeiter sind: die studentischen Mitarbeiter (`HiWi`). Einige Personen (`BibBenutzer`) sind in dem Informationssystem einer Bibliothek registriert.

Das erste Schema `S1` ist im relationalen Datenmodell formuliert:

```
Person(SVNr,Name)
BibBenutzer(SVNr)
Student(SVNr,Fach)
Mitarbeiter(SVNr,Gehalt)
HiWi(SVNr,Fach,Gehalt)
```

Abb. 1 zeigt das zweite Schema `S2`, das zwar ein objektorientiertes Schema ist, aber denselben Diskursbereich etwas umständlich in einem Objektmodell ohne Mehrfachvererbung beschreibt.

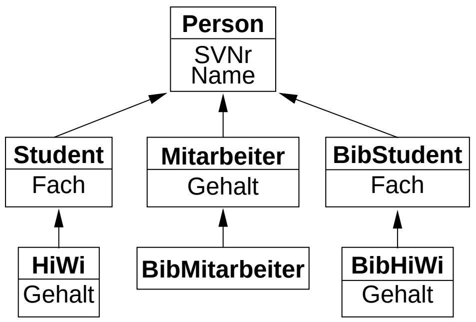

Abbildung 1 Objektorientiertes Schema S2 ohne Mehrfachvererbung

Das objektorientierte Schema ist so aufgebaut, daß jedes Objekt nie gleichzeitig zu zwei Klassen in verschiedenen Zweigen der Hierarchie gehören kann, da das zugrundeliegende Datenmodell weder Mehrfachvererbung noch das Rollenkonzept unterstützt. Zum Beispiel kann es kein Objekt geben, daß gleichzeitig zu den Klassen `Student` und `Mitarbeiter` gehört. In unserem Beispiel sind alle Objekte, die konzeptionell zu beiden Klassen `Student` und `Mitarbeiter` gehören, Instanzen in `HiWi`, einer Subklasse von `Student`. Damit sind sie zwar immer auch Instanzen der Klasse `Student`, aber nicht von `Mitarbeiter`. Hier wurde also eine Entwurfs- bzw. Implementierungsentscheidung zur Aufteilung von Klassen getroffen, die vielleicht in diesem einfachen Beispiel leicht zu erkennen und nachzuvollziehen ist, aber in größeren Datenbankschemata oft schwierig zu entdecken ist.

3 Zwischenrepräsentation für Klassenhierarchien

Um eine gegebene Klassenhierarchie restrukturieren zu können, führen wir eine Zwischenrepräsentation ein, die es uns anschließend erlaubt, Verfahren aus dem Bereich der formalen Begriffsanalyse anzuwenden. Für diese Zwischenrepräsentation benötigen wir Informationen über die extensionalen Beziehungen zwischen den Klassen (Relationen) des zu restrukturierenden Schemas. Die *Extension* einer Klasse (Relation) ist dabei die Menge ihrer Instanzen, genauer, die Menge der durch die Instanzen eines Datenbankzustandes repräsentierten Objekte der realen Welt. Bei den extensionalen Beziehungen geht es hier darum, Mengenbeziehungen zwischen Extensionen potentiell möglicher Datenbankzustände zu erfassen.

Ein Teil der extensionalen Beziehungen zwischen Klassen kann direkt aus dem gegebenen Schema abgelesen werden. So bedeutet beispielsweise eine Spezialisierungsbeziehung (Klasse-Subklasse-Beziehung) zwischen zwei Klassen, daß die Extension der Subklasse immer eine Teilmenge der Extension der anderen Klasse ist. Weitere extensionale Beziehungen lassen sich direkt daraus ableiten, in welchem Datenmodell das Schema ausgedrückt ist. Wenn das Datenmodell z.B. kein Rollenkonzept unterstützt, dann wissen wir für alle Paare von Klassen, die keine gemeinsame Subklasse haben, daß ihre Extensionen jeweils disjunkt sind. Auf diese Weise lassen sich viele, aber meistens nicht alle benötigten extensionalen Beziehungen aus dem Schema ableiten.

Um die Informationen über die extensionalen Beziehungen zu vervollständigen, muß der Datenbankentwerfer (oder die Anwender) in der Regel anwendungsspezifisches Wissen ergänzen. Soweit dieses Wissen nicht zur Verfügung steht, kann auch mit Default-Annahmen gearbeitet werden. Ein Beispiel dafür wäre die Annahme, daß, wenn wir über die Beziehung zwischen den Extensionen zweier Klassen keine abgeleiteten oder sonstigen Informationen haben, diese dann als disjunkt anzunehmen sind. In [19] wird ein Vorgehen beschrieben, wie extensionale Beziehungen semiautomatisch bestimmt werden können. Im folgenden gehen wir davon aus, daß das Wissen über die extensionalen Beziehungen vorliegt.

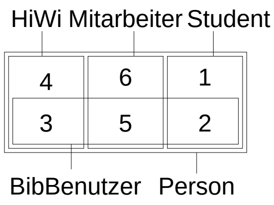

Abbildung 2 Extensionale Beziehungen von S1 im Euler/Venn-Diagramm

Die extensionalen Beziehungen des eingeführten Beispielschemas S1 sind in der Abb. 2 als Euler/Venn-Diagramm dargestellt. Die Überlappungen der möglichen Extensionen zeigen eine Zerlegung in insgesamt sechs disjunkte Extensionsbereiche, die im folgenden als *Basisextensionen* bezeichnet werden. Eine äquivalente Darstellung ist die Beschreibung der extensionalen Zusammenhänge in Tabellenform (Tabelle 1 und 2). Jede Zeile stellt die potentielle Extension einer Klasse (Relation) dar. Aufgrund der extensionalen Be-

Relation	Basisextension 1	2	3	4	5	6
`S1.Person`	√	√	√	√	√	√
`S1.BibBenutzer`		√	√		√	
`S1.Student`	√	√				
`S1.Mitarbeiter`					√	√
`S1.HiWi`			√	√		

Tabelle 1 Schema S1: Extensionale Beziehungen

Klasse	Basisextension 1	2	3	4	5	6
`S2.Person`	√	√	√	√	√	√
`S2.Student`	√			√		
`S2.HiWi`				√		
`S2.Mitarbeiter`					√	√
`S2.BibMitarbeiter`					√	
`S2.BibStudent`		√	√			
`S2.BibHiWi`			√			

Tabelle 2 Schema S2: Extensionale Beziehungen

ziehungen zwischen den Klassen teilen wir deren Extensionen in disjunkte Basisextensionen auf (die analog zur Abb. 2 von `1` bis `6` durchnumeriert sind). Jedes potentielle Objekt kann immer genau einer Basisextension aber dafür mehreren Klassen zugeordnet werden.

Aus dem Euler/Venn-Diagramm (Abb. 2) ist beispielsweise ersichtlich, daß es eine extensionale Inklusion zwischen `S1.BibBenutzer` und `S1.Person` gibt. Diese wird auf der Ebene der Basisextensionen durch eine Teilmengenbeziehung der zu `S1.BibBenutzer` und `S1.Person` gehörenden Basisextensionen widergespiegelt. Aus der Tabelle 1 können wir entnehmen, daß $\{2, 3, 5\}$ die Menge der Basisextensionen zu `S1.BibBenutzer` und $\{1, 2, 3, 4, 5, 6\}$ zu `S1.Person` ist. Die Extensionen von `S1.Student`, `S1.Mitarbeiter` und `S1.HiWi` sind hingegen immer paarweise disjunkt.

Die Darstellung extensionaler Beziehungen in Form einer Tabelle ist semantisch äquivalent zu einem korrespondierenden Euler/Venn-Diagramm. Aufgrund häufig auftretender graphischer Beschränkungen von Euler/Venn-Diagrammen für extensionale Beziehungen zwischen mehr als drei Mengen bevorzugen wir hier die Tabellendarstellung.

Die extensionalen Beziehungen müssen für jeden zulässigen Datenbankzustand gelten. Wenn zum Beispiel die Extensionen zweier Klassen als disjunkt beschrieben sind, dann darf es in keinem Datenbankzustand eine gemeinsame Instanz geben. Eine weitere wichtige Eigenschaft ist, daß jedes mögliche Objekt (oder Tupel) in einem Datenbankzustand immer zu genau einer Basisextension gehört. Eine Migration von Objekten von einer Basisextension zu einer anderen wird damit jedoch nicht ausgeschlossen.

Aus den ursprünglichen Schemata sowie den Informationen über extensionale Beziehungen können wir die Zuordnung von Attributen zu Basisextensionen ableiten. Die Relation `S2.Student` hat zum Beispiel die Attribute `SVNr`, `Name` sowie `Fach` und die Basisextensionen `1` und `4`. Daher sind diese Attribute für die Basisextensionen `1` und `4` definiert. Insgesamt entsteht so eine binäre Relation zwischen der Menge der Basisextensionen und der Menge aller Attribute. Diese Relation kann wiederum als Tabelle dargestellt werden, die die angestrebte Zwischenrepräsentation für Klassenhierarchien darstellt.

Attribute	Basisextension 1	2	3	4	5	6
`SVNr`	√	√	√	√	√	√
`Name`	√	√	√	√	√	√
`Fach`	√	√	√	√		
`Gehalt`			√	√	√	√

Tabelle 3 Beziehungen zwischen Basisextensionen und Attributen

In diesem Beitrag betrachten wir vereinfachend keine Datentypen und keine Integritätsbedingungen. Diese lassen sich analog zu dem in [4] beschriebenen Vorgehen zur Behandlung von Integritätsbedingungen bei der Schemaintegration berücksichtigen. Für die Darstellung der Grundprinzipien unseres Ansatzes verzichten wir auf deren Behandlung.

Für beide Beispielschemata und ihre korrespondierenden extensionalen Beziehungen erhalten wir dieselbe Relation zwischen den Basisextensionen und Attributen, die in Tabelle 3 dargestellt ist. Auf diese Relation werden wir im folgenden die Mechanismen der formalen Begriffsanalyse anwenden.

Diese Zwischenrepräsentation kann als ein Schema aufgefaßt werden, bei dem die Basisextensionen disjunkte Klassen darstellen. Sie ist auf der Grundlage einer extensionalen Zerlegung entstanden. Wie man sich leicht klarmachen kann, ist diese Zwischenrepräsentation äquivalent zum Ausgangsschema, da eine bijektive Funktion über die jeweiligen Datenbankzustände angegeben werden kann. Ein Euler/Venn-Diagramm impliziert eine eindeutige Abbildung von Klassenextensionen auf Basisextensionen durch Mengenausdrücke und eine inverse Abbildung, bei der die ursprünglichen Klassenextensionen durch Vereinigung der entsprechenden Basisextensionen ermittelt werden können.

4 Begriffsverbände der formalen Begriffsanalyse

Um eine normalisierte Klassenhierarchie zu erhalten, verwenden wir die Theorie der *formalen Begriffsanalyse* [23,7].

Die Theorie der Begriffsanalyse basiert auf folgender Formalisierung [5]: Ein *Kontext* (G, M, I) besteht aus einer Menge G von Objekten, einer Menge M von Eigenschaften und einer binären Relation $I \subseteq G \times M$ zwischen diesen beiden (endlichen) Mengen. Die binäre Relation I beschreibt, welches Objekt aus G welche Eigenschaft aus M besitzt. Für ein Objekt $g \in G$ und eine Eigenschaft $m \in M$ schreiben wir $(g, m) \in I$, wenn g die Eigenschaft m hat.

Der *intent* einer Objektmenge $A \subseteq G$ ist definiert als:

$$intent(A) := \{m \in M \mid \forall g \in A : (g, m) \in I\}$$

Analog ist der *extent* einer Attributmenge $B \subseteq M$ definiert als:

$$extent(B) := \{g \in G \mid \forall m \in B : (g, m) \in I\}$$

Beispiel: Die Menge G umfaßt in unserem Beispiel die in der extensionalen Analyse bestimmten Basisextensionen: $G = \{1, 2, 3, 4, 5, 6\}$. Entsprechend wird die Menge M von Eigenschaften durch die Attribute bestimmt, die in dem vorgefundenen Schema (`S1` bzw. `S2`) enthalten sind: $M = \{\texttt{SVNr}, \texttt{Name}, \texttt{Fach}, \texttt{Gehalt}\}$. Die Relation I ist für das Beispiel durch die Tabelle 3 beschrieben. Der *intent* von $\{2, 5\}$ ist beispielsweise $intent(\{2, 5\}) = \{\texttt{SVNr}, \texttt{Name}\}$; und der *extent* von $\{\texttt{Fach}, \texttt{Gehalt}\}$ ist $extent(\{\texttt{Fach}, \texttt{Gehalt}\}) = \{3, 4\}$.

Ein *Begriff* (oder auch *Konzept*) in (G, M, I) ist ein Paar $(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M)$, für das $A = extent(B)$ und $B = intent(A)$ gilt. Folglich repräsentiert ein Begriff ein „maximales Rechteck" in der binären Relation I. Ein solches „maximales Rechteck" läßt sich dabei anschaulich durch die zwei folgenden Forderungen charakterisieren: (i) jedes Objekt in A besitzt jede Eigenschaft in B, und (ii) es kann kein Objekt zu A und keine Eigenschaft zu B hinzugenommen werden, ohne daß die Forderung (i) verletzt wird.

Beispiel: $(\{3, 4, 5, 6\}, \{\texttt{SVNr}, \texttt{Name}, \texttt{Gehalt}\})$ ist in unserem Beispiel ein *Begriff*. Wie in Tabelle 3 direkt zu erkennen ist, läßt sich dies nicht mehr erweitern, da die Basisextensionen 1 und 2 die Eigenschaft `Gehalt` nicht besitzen und die Eigenschaft `Fach` für die Basisextensionen 5 und 6 nicht gegeben ist.

Sei nun

$$L := \{(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M) \mid A = extent(B) \wedge B = intent(A)\}$$

die Menge aller Begriffe (maximaler Rechtecke) in (G, M, I), und sei $\leq$ eine partielle Ordnung auf L, die wie folgt definiert ist:

$$(A_1, B_1) \leq (A_2, B_2) \Longleftrightarrow A_1 \subseteq A_2$$

Als Ergebnis erhalten wir einen vollständigen Verband, beschrieben durch:

$$\mathcal{L} := (L, \leq, \wedge, \vee, (extent(M), M), (G, intent(G)))$$

Die Verbandsoperationen sind durch folgende Definitionen bestimmt:

$$(A_1, B_1) \wedge (A_2, B_2) = (A_1 \cap A_2, intent(A_1 \cap A_2))$$

und

$$(A_1, B_1) \vee (A_2, B_2) = (extent(B_1 \cap B_2), B_1 \cap B_2).$$

Der in dieser Weise über Begriffen aufgebaute Verband wird auch als *Begriffsverband* bezeichnet [5]. [21] beschreibt die Implementierung eines Algorithmus, der den Begriffsverband zu einem gegebenen Kontext (G, M, I) berechnet.

Beispiel: Auf den in unserem Beispiel entstehenden Begriffsverband gehen wir im folgenden Abschnitt ausführlicher ein. Die Menge aller Begriffe, die sich aus Tabelle 3 ableiten lassen, wird in Tabelle 4 angegeben, wobei dort jedem Begriff ein Klassenname zugeordnet ist (das in Tabelle 4 neu auftauchende Attribut `BibNutzer` sollte zunächst ignoriert werden; es wird dann in Abschnitt 5 erläutert).

Die Theorie der Begriffsanalyse kann nun genutzt werden, um ein Schema in eine objektorientierte „Normalform" zu transformieren. Die binäre Relation zwischen Basisextensionen und Attributen (siehe Tabelle 3) wird — wie bereits angedeutet — dazu als binäre Relation I eines Kontextes aufgefaßt, wobei G die Menge der Basisextensionen und M die Menge der Attribute aus dem Schema ist. Jetzt wird auch klar, warum die Extensionen in disjunkte Basisextensionen überführt wurden. Basisextensionen können als disjunkte Mengen frei zu größeren Extensionen zusammengefaßt werden.

Ein Begriffsverband, der von einer Relation zwischen Basisextensionen und Attributen abgeleitet wurde, kann dann in der folgenden Weise als Klassenhierarchie interpretiert werden:

- Ein Begriff $(A, B) \in L$ ist eine Klasse mit der Extension A (genauer, die Vereinigung aller in A enthaltenen Basisextensionen), und B repräsentiert die Menge der Attribute dieser Klasse;
- $\leq$ ist die Spezialisierungsbeziehung zwischen Klassen. Für die Repräsentation als Datenbankschema muß diese Beziehung um alle transitiv und reflexiv ableitbaren Paare von Begriffen bzw. Klassen reduziert werden;
- $\wedge$ ist die Operation, die zu zwei Klassen (Begriffen) die größte gemeinsame Subklasse bildet. Deren Extension ist die Schnittmenge der Extensionen der zwei Klassen (Spezialisierung);
- $\vee$ ist die Operation, die zu zwei Klassen (Begriffen) die kleinste gemeinsame Superklasse bestimmt. Deren Attributmenge ist die Schnittmenge der Attributmengen der beiden Klassen (Generalisierung);
- $(extent(M), M)$ ist das Infimum des Verbandes und repräsentiert die Klasse, die alle Attribute besitzt (und deren Extension leer sein kann); diese Klasse ist Subklasse aller anderen Klassen;
- $(G, intent(G))$ ist das Supremum des Verbandes und repräsentiert die Klasse, die alle Basisextensionen umfaßt (und deren Attributmenge leer sein kann); diese Klasse ist direkte oder indirekte Superklasse aller anderen Klassen.

Die sich so ergebende Klassenhierarchie hat einige wichtige Eigenschaften:

- Sie erfüllt alle Verbandseigenschaften, d.h.
  - zu jedem Paar von Klassen gibt es eine eindeutig bestimmte (größte) gemeinsame Subklasse (Abgeschlossenheit bezüglich der Schnittbildung von Extensionen). Dual dazu gibt es zu jedem Paar von Klassen auch eine eindeutig bestimmte (kleinste) gemein-

same Superklasse (Abgeschlossenheit bezüglich der Schnittbildung von Attributmengen).
  - zu jedem Attribut gibt es genau eine höchste Klasse in der Hierarchie, in der das Attribut definiert ist. Wiederum dual dazu gibt es auch zu jeder Basisextension eine eindeutig bestimmte unterste Klasse, die diese Basisextension umfaßt.
- Die normalisierte Klassenhierarchie kann maximal 2^n Klassen besitzen ($n = \min(|G|, |M|)$). Um alle Klassen bzw. Begriffe aus einem gegebenen Kontext abzuleiten, müssen alle Teilmengen von G oder von M betrachtet werden. Daher entsteht im schlechtesten Fall ein exponentieller Aufwand für die Berechnung des vollständigen Begriffsverbands (bzw. der entsprechenden Klassenhierachie).
- Jede resultierende Klasse entspricht einem „maximalen Rechteck", d.h., keine Klasse kann um weitere Attribute oder Basisextensionen erweitert werden, ohne die Klasseneigenschaft, daß alle Objekte der Klasse zu allen definierten Attribute Werte besitzen müssen, zu verletzen.
  Daraus folgt unmittelbar, daß es keine zwei Klassen geben kann, die die selbe Extension besitzen oder die selben Attribute[1] haben. Wenn solche Klassen auftreten würden, könnte mindestens eine Klasse extensional oder um Attribute erweitert werden.

Für die praktische Anwendung zur Restrukturierung von Schemata ist die exponentielle Komplexität für die Berechnung des Begriffsverbandes augenscheinlich zu hoch, zumal die Anzahl der Basisextensionen oft größer als die Anzahl der im ursprünglich vorgefundenen Schema vorhandenen Klassen sein kann.

Allerdings läßt sich leicht zeigen, daß nicht alle berechneten Begriffe des Begriffsverbands sinnvolle Klassen eines Datenbankschemas darstellen (dies wird im Zusammenhang mit Schemaintegration ausführlich in [18] dargelegt, wobei die Argumentation sich unverändert übertragen läßt). Oft entstehen Begriffe bzw. Klassen, die keine eigene Extension in dem Sinne besitzen, daß ihre Extension sich durch Vereinigung der Extensionen ihrer Subklassen ergibt, und die keine eigenen Attribute in dem Sinne besitzen, daß alle ihre Attribute von ihren Superklassen vererbt werden. Solche Klassen ergeben für den Datenbankanwender keinen Sinn. Wir können die Erzeugung solcher überflüssiger Begriffe bzw. Klassen vermeiden, indem wir nicht den vollständigen Verband bestimmen. Wenn wir statt L die Menge L^{red} berechnen:

$$\begin{aligned} L^{red} := {} & \{(extent(intent(\{a\})), intent(\{a\})) \mid a \in G\} \\ & \cup\{(extent(\{b\}), intent(extent(\{b\}))) \mid b \in M\}, \end{aligned}$$

dann enthält L^{red} nur sinnvolle Klassen, erzeugt aber im allgemeinen keinen vollständigen Verband. Für Klassenhierarchien in Datenbankschemata ist dies nicht nur akzeptabel, sondern auch wünschenswert.

Für die Berechnung von L^{red} läßt sich eine Laufzeitkomplexität von $O(n^3)$ im Gegensatz zur exponentiellen für die Berechnung des vollständigen Verbandes feststellen (der Nachweis dafür wird auch in [18] gegeben).

Beispiel: In unserem Beispiel gibt es keine überflüssigen Begriffe bzw. Klassen, so daß L^{red} hier die selben Begriffe enthält wie L.

Die Vorteile der Restrukturierung objektorientierter Schemata durch den hier vorgestellten Ansatz können folgendermaßen zusammengefaßt werden:

- konzeptionelle Unabhängigkeit von der Implementation: Das restrukturierte Schema kann beispielsweise Mehrfachvererbung enthalten. Dies ist deswegen vorteilhaft, da das Verbot von Mehrfachvererbung häufig zu schwer verständlichen Klassenhierarchien führt.
- Verständlichkeit: Dieser oft recht subjektive Aspekt wird hier dadurch begründet, daß durch die Restrukturierung keine zusätzlichen Klassen entstehen, sondern oft sogar die Anzahl der Klassen reduziert wird. Intuitiv läßt sich dies durch die Ermittlung maximaler Rechtecke verdeutlichen.
- eindeutige Darstellung: Das restrukturierte Schema ist durch den vorgestellten Algorithmus eindeutig festgelegt. Die Restrukturierung äquivalenter Schemata führt immer zu dem selben Schema.
  Ein eindeutiges Schema mit den genannten garantierten Eigenschaften kann, wenn gewünscht, als Ausgangsschema für weitere Schematransformation dienen, um zum Beispiel eine Sicht auszudrücken oder um ein bestimmtes Datenmodell zu unterstützen. So sind, wie in z.B. [17] beschrieben, bestimmte Zusammenfassungen und Zerlegungen von Klassen möglich, die dann aber natürlich wieder zu schlecht entworfenen Schemata führen können. Als Beispiel sei hier die Beseitigung von Mehrfachvererbung genannt.

5 Restrukturierung am Beispiel und Diskriminante

In diesem Abschnitt zeigen wir das Ergebnis der Restrukturierung unseres Beispiels. Tabelle 4 enthält die abgeleiteten Klassen, ergänzt um ein Attribut `BibNutzer` (dessen Name frei, aber möglichst sinnvoll gewählt werden kann). Dieses künstliche Boole'sche Attribut (in der Literatur oft auch *Diskriminante* genannt) zeigt an, ob eine Person in der Bibliothek angemeldet (registriert) ist oder nicht. Ein solches Attribut muß an dieser Stelle eingeführt werden, um einen Informationsverlust zu verhindern. In Tabelle 3 finden wir Basisextensionen mit den gleichen Attributmengen (und zwar die Basisextensionen `1` und `2`, `3` und `4` sowie `5` und `6`). Der Algorithmus zur Berechnung des Begriffsverbandes faßt diese Basisextensionen jeweils zu einer Einheit zusammen. Daher würden wir in der resultierenden Klassenhierarchie ohne Diskriminante die Information verlieren, welche Instanzen etwa zur Basisextension `1` oder `2` gehören.

Diese Unterscheidung, die zur Gewährleistung äquivalenter Schemata im Sinne gleicher Informationskapazität notwendig ist, bleibt nur durch das künstliche Attribut erhalten.

[1] Hier sind alle Attribute, also auch die vererbten Attribute gemeint.

Klassenname	Extensionen	Attributmenge
`Person`	`1,2,3,4,5,6`	`SVNr, Name, BibNutzer`
`Student`	`1,2,3,4`	`SVNr, Name, BibNutzer, Fach`
`Mitarbeiter`	`3,4,5,6`	`SVNr, Name, BibNutzer, Gehalt`
`HiWi`	`3,4`	`SVNr, Name, BibNutzer, Fach, Gehalt`

Tabelle 4 Abgeleitete Klassen

	Basisextension					
Attribute	1	2	3	4	5	6
`SVNr`	√	√	√	√	√	√
`Name`	√	√	√	√	√	√
`BibNutzer` (Diskr.)	√	√	√	√	√	√
`Fach`	√	√	√	√		
`Gehalt`			√	√	√	√

Tabelle 5 Beispielkontext mit Diskriminante

Es ermöglicht eine injektive Abbildung von Basisextensionen auf Klassen des restrukturierten Schemas.

Die Einführung eines künstlichen Attributs ist nur dann notwendig, wenn Basisextensionen gleiche Attributmengen haben. Das läßt sich automatisch feststellen. In diesem Fall wird ein künstliches Attribut in den Kontext für diejenigen Basisextensionen eingefügt, für die eine andere Basisextension mit den selben Attributen existiert. Der erweiterte Kontext für unser Beispiel ist in Tabelle 5 dargestellt. Dem neuen Attribut müssen ein möglichst verständlicher Name und ein geeigneter Wertebereich geben werden. In unserem Beispiel reicht ein Boole'sches Attribut aus, da nur zwischen jeweils zwei Basisextensionen unterschieden werden muß. Wenn mehrere Basisextensionen durch das künstliche Attribut unterschieden werden sollen, muß der Wertebereich entsprechend umfangreicher sein.

Die erneute Ableitung auf der Grundlage eines erweiterten Kontextes liefert dann das gewünschte restrukturierte Schema.

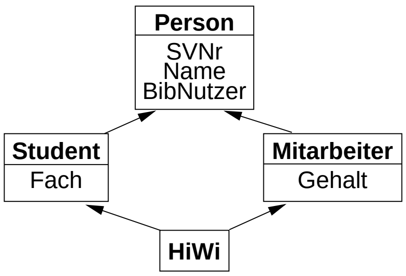

Abbildung 3 Normalisiertes Beispielschema

Das resultierende, restrukturierte Schema ist in Abb. 3 angegeben (ohne explizite Darstellung der jeweils von Superklassen vererbten Attribute). In unserem Beispiel ergibt sich ein Schema, daß einem vollständigen Verband entspricht, d.h., es ist hinsichtlich Schnittbildung von Extensionen und Schnittbildung von Attributmengen abgeschlossen. Jedes Attribut wird genau in einer Klasse deklariert (und dann an alle Subklassen vererbt). Entsprechend gibt es für jede Basisextension eine eindeutig bestimmte kleinste Klasse, die diese Basisextension umfaßt.

Aufgrund der Tatsache, daß die abgeleiteten Klassen maximale Rechtecke repräsentieren, haben wir im Ergebnis nur vier Klassen. In Vergleich zum Ausgangsschema (siehe Abb. 1) ist dieses Schema übersichtlicher und leichter zu verstehen. Die Namen der Klassen müssen prinzipiell vom Entwerfer bzw. von der Person, die die Restrukturierung durchführt, bestimmt werden. In einigen Fällen lassen sich Klassennamen aber auch automatisch aus dem Ursprungsschema bestimmen, und zwar immer dann, wenn im Ursprungsschema eine Klasse mit genau derselben Extension vorhanden ist. Dann kann deren Klassenname einfach übernommen werden. In unserem Beispiel trifft dies für die Klasse `Person` zu. Im Gegensatz dazu umfaßt die Klasse `Student` in Abb. 3 die Basisextensionen `1`, `2`, `3` und `4`, während die Relation `Student` im Schema `S1` die Basisextensionen `1` und `2` und die Klasse `Student` in Schema `S2` die Basisextensionen `1` und `4` umfassen (vgl. Tabelle 1 und 2). Wenn wir uns allerdings die Ausgangsschemata genauer anschauen, können wir feststellen, daß Studenten jeweils auf mehrere Klassen verteilt worden sind, die zusammen genau die Basisextensionen `1`, `2`, `3` und `4` ergeben. Daher haben wir für diese Klasse im Ergebnis den Namen `Student` gewählt.

6 Vergleich mit der relationalen Entwurfstheorie

Die Restrukturierung von Klassenhierarchien unter Anwendung der Theorie der formalen Begriffsanalyse weist gegenüber der relationalen Normalisierungtheorie Gemeinsamkeiten und Unterschiede auf. In beiden Verfahren werden Schematransformationen unter Erhalt der Informationskapazität durchgeführt. Die Schematransformationen folgen bestimmten Regeln, die die Überführung in eine bestimmte Zielstruktur mit definierten Eigenschaften festlegt. Als Ziel steht in beiden Fällen eine Erhöhung der Entwurfsqualität.

Die Unterschiede beider Verfahren werden in der Tabelle 6 dargestellt. Bei der relationalen Normalisierung, oder genauer bei der Transformation in die 2NF, 3NF und BCNF, wird auf der Grundlage funktionaler Abhängigkeiten Redundanz auf der Datenebene beseitigt. Dabei erhöht sich die Anzahl der Relationen. Die Schemagröße wächst also.

Im Gegensatz dazu wird bei der Restrukturierung Redundanz auf der Schemaebene verringert, was zu einem minimalen Schema führt. Die Minimalität bezieht sich hier auf die Anzahl der Klassen. Da jede Klasse, wie in Abschnitt 4 beschrieben, einem maximalen Rechteck entspricht und das Spezialisierungskonzept voll genutzt wird, folgt daraus eine

	Redundanz-beseitigung auf	**Schema-größe**
Normalisierung	Datenebene	steigt
Restrukturierung	Schemaebene	fällt

Tabelle 6 Unterschiede zwischen relationaler Normalisierung und Restrukturierung

minimale Klassenanzahl. Jedes Attribut wird weiterhin in genau einer Klasse eingeführt.

Die Minimalität der Schemagröße gilt hier unter der Vorbedingung, daß keine künstlichen Nullwerte eingeführt und keine abstrakten Superklassen und Subklassen ohne eigene Attribute entfernt wurden. Eine abstrakte Klasse ist eine Klasse, deren Extension der Vereinigung der Extensionen seiner Subklassen entspricht. Wenn der Entwerfer auf abstrakte Klassen bei gleichzeitiger Verschlechterung der Verständlichkeit verzichten möchte, kann er die Schemagröße entsprechend dem in [18] beschriebenen Verfahren unter Beibehalt der Informationskapazität weiter reduzieren. Die Reduzierung um Klassen ohne eigene Attribute entsprechend [18] verringert die Klassenanzahl weiter, setzt aber das Rollenkonzept voraus.

Minimalität auf der Schemaebene ist auch eine Forderung beim relationalen Entwurf. Im Unterschied zum hier vorgestellten Verfahren wird jedoch dort damit die Vermeidung von Redundanz auf der Datenebene angestrebt.

Eine methodische Analogie läßt sich zwischen dem hier vorgestellten Verfahren und dem Syntheseverfahren der relationalen Entwurfstheorie erkennen. Die funktionalen Abhängigkeiten dort entsprechen hier den Informationen, für welche Extensionen welche Attributwerte prinzipiell existieren. In einem ersten Schritt werden funktionale Abhängigkeiten vereinfacht bzw. Extensionen in einfachere, d.h. disjunkte, Basisextensionen überführt. Im zweiten Schritt werden aus dieser „atomarisierten“ Darstellung funktionale Abhängigkeiten bzw. Basisextensionen zu maximalen Relationen bzw. Klassen zusammengefaßt, so daß jeweils minimale Schemata entstehen.

7 Andere Ansätze

Der in [6] beschriebene Ansatz zur Transformation relationaler Schemata in objektorientierte erkennt Vererbungsbeziehungen nur aufgrund intensionaler Aspekte. So werden zum Beispiel Inklusionsbeziehungen zwischen Schlüsselattributen als Vererbungsbeziehungen erkannt. Jedoch werden in diesem Ansatz die extensionalen Aspekte nicht ausreichend berücksichtigt.

In [10] wird ein manuelles Verfahren zur Ableitung objektorientierter Schemata aus relationalen Schemata durch semantische Anreicherung vorgestellt. Ein Schritt dabei betrifft die Ableitung von Vererbungsbeziehungen. In diesem Ansatz müssen diese Beziehungen manuell anhand von vorgegebenen Regeln gefunden werden. Im Gegensatz zu diesem Ansatz stellen wir den Bezug zu einer mathematischen Theorie her, die die automatische Ableitung einer gesamten Klassenhierarchie erlaubt.

Die Idee der Interpretation von Begriffsverbänden als Klassenhierarchien ist nicht neu. Zum Beispiel wird in [24] diese Idee für den objektorientierten Entwurf genutzt. Dort werden allerdings im Gegensatz zu unserem Ansatz nur Untermengenbeziehungen zwischen Attributen (intensionale Beziehungen) betrachtet. Diese Untermengenbeziehungen legen die Vererbungsbeziehungen fest, ohne die extensionalen Beziehungen zu berücksichtigen. Dadurch wird die Vererbungssemantik nicht immer vollständig erfaßt. In unserem Ansatz betrachten wir explizit sowohl die extensionalen als auch die intensionalen Beziehungen.

[9] betrachtet Begriffsverbände als Klassenhierarchien und Objekthierarchien zum Beschreiben des Effekts einer Schemaevolution bzgl. existierender Objekte einer Objektdatenbank. Eine Schemaevolution wird dort als Transformation eines Begriffsverbandes in einen anderen Verband angesehen. Eine solche Transformation kann eine Migration von Objekten in eine neue Klasse verursachen. Dieser Ansatz geht jedoch von einer Klassenhierarchie in Form eines Klassenverbandes als Startpunkt aus. In vielen Fällen, wo diese Annahme nicht erfüllt ist (und unsere Erfahrung zeigt, daß diese Annahme in den meisten praktisch auftretenden Fällen verletzt wird), kommt unser Ansatz zum Einsatz.

Um Objekte als Instanzen eines beliebigen Datenbankschemas in eine neue Objektdatenbank zu migrieren, die die vorteilhaften Verbandseigenschaften besitzt, kann unser Ansatz eingesetzt werden. Dadurch, daß wir explizit extensionale Beziehungen berücksichtigen, haben wir gleichzeitig auch die für das Migrieren von Objekten benötigten Informationen gegeben. Jedes Objekt ist Mitglied genau einer Basisextension (was aufgrund der disjunkten Zerlegung der überlappenden Klassenextensionen garantiert wird). Dadurch kann jedes Objekt zu einer eindeutig definierten Klasse des Zielschemas migriert werden.

Abschließend seien noch einmal die bisher entwickelten Ansätze zur Normalisierung in objektorientierten oder anderen semantischen Datenmodellen erwähnt. Ansätze wie etwa [1, 12, 13, 20] haben bereits verschiedene Vorarbeiten geleistet, um beispielsweise spezielle Formen der Datenredundanz oder von Änderungsanomalien zu identifizieren und geeignete Normalformen zu finden, um diese Probleme zu beseitigen. In der Regel werden dazu die aus dem relationalen Bereich bekannten funktionalen und mehrwertigen Abhängigkeiten geeignet erweitert und darauf basierend Normalformen definiert.

In [2] wird speziell auf die Zerlegung mehrstelliger Beziehungen bei der Transformation in objektorientierte Datenbankschemata eingegangen und dafür eine Transformation (*property pivoting*) basierend auf funktionalen Abhängigkeiten vorgestellt.

[20] kann unter den zuvor genannten als der wohl weitreichendste Ansatz angesehen werden. Allerdings werden dabei Klassenhierarchien nicht explizit betrachtet. Der von uns vorgestellte Ansatz könnte daher genau diese Lücke schließen,

wobei allerdings noch im Detail geklärt werden muß, wie die beiden Ansätze, die auf sehr unterschiedlichen formalen Fundamenten basieren, miteinander verknüpft werden können.

8 Ausblick

Wie diese Vergleiche mit anderen Arbeiten zeigen, kann unser Ansatz nicht nur für das Restrukturieren von Datenbankschemata sondern auch für den objektorientierten Datenbankentwurf und zur Unterstützung von Migration zwischen unterschiedlich strukturierten Datenbeständen verwendet werden bzw. Ansätze aus diesen Gebieten sinnvoll ergänzen. Die Vereinheitlichung unterschiedlicher Strukturierungen von Datenbeständen ist insbesondere auch bei der Integration von bestehenden Datenbanken bzw. existierenden Informationssystemen ein zentrales Problem (vgl. auch [3, 14]). Folglich bestehen für unseren Ansatz eine Reihe von nutzbringenden Anwendungsgebieten.

Für eine weitreichendere „Normalisierung" von objektorientierten Datenbankschemata muß jedoch das Verfahren um weitere Aspekte erweitert werden. Eine Erweiterung betrifft Referenzattribute. Die Referenzattribute selbst können wie normale Attribute behandelt werden. Problematisch ist jedoch die inhärente Integritätsbedingung, daß Objekte nur einer vorgegebenen Klasse referenziert werden dürfen. Diese Zielklasse kann sich durch die Restrukturierung extensional ändern. Ein Ausweg besteht darin, dafür zu sorgen, daß dies nicht geschehen kann. Die Einführung eines künstlichen Attributs für die Basisextensionen im Kontext, die Bestandteil der ursprünglichen Zielklasse sind, erzeugt eine Klasse mit der selben Extension.

In dem vorgestellten Verfahren wurden keine Methoden behandelt. Methoden sind in der Regel auf konkrete Datenstrukturen abgestimmt, so daß Anpassungen bei Strukturänderungen nötig sind. Dafür ist eine Analyse der Methoden nötig, um den Zusammenhang von Methoden zu bestimmten Attributen festzustellen.

Typkonstruktionen wurden in dieser Arbeit ebenfalls nicht berücksichtigt. Prinzipiell können natürlich komplex strukturierte Attribute analog zu den atomaren Attributen behandelt werden. Leider können dann aber beispielsweise ähnliche Datentypkomponenten nicht isoliert in einer Superklasse definiert werden. Um dieses Ziel zu erreichen, sind Transformationen auf Typkonstruktionen notwendig. Ein Beispiel für eine recht einfache Transformation ist die Transformation eines tupelwertigen Attributs in mehrere eigenständige Attribute.

Danksagung Unsere Kollegen Kerstin Schwarz, Gunter Saake und Can Türker haben durch Kommentare und Fragen zu Vorversionen dieser Arbeit sowie durch ihre Beteiligung an gemeinsamen Arbeiten im Bereich der Datenbankintegration, die uns zu der hier diskutierten Problematik geführt haben, zur Entwicklung und zum Gelingen dieses Artikels beigetragen. Darüber hinaus möchten wir auch ausdrücklich den Gutachtern danken, die durch ihre detaillierten und fundierten Anmerkungen uns weitere Anregungen zur Verbesserung dieses Beitrages gegeben haben.

Literatur

1. Andonoff, E.: Normalization of Object-Oriented Conceptual Schemes. In: Rolland, C., Bodart, F., Cauvet, C. (eds.), *Advanced Information Systems Engineering (Proc. CAiSE'93)*. Lecture Notes in Computer Science, Vol. 685, pp. 449–462. Berlin: Springer 1993
2. Biskup, J., Menzel, R.,Polle, T., Sagiv, Y.: Decomposition of Relationships Through Pivoting. In: Thalheim, B. (ed.), *Conceptual Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996*. Lecture Notes in Computer Science, Vol. 1157, pp. 28–41. Berlin: Springer 1996
3. Conrad, S.: *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Berlin/Heidelberg: Springer 1997
4. S. Conrad, S., Schmitt, I., Türker, C.: Considering Integrity Constraints During Federated Database Design. In: Embury, S. M., Fiddian, N. J., Gray, A. W., Jones, A. C. (eds.), *Advances in Databases, 16th British National Conf. on Databases, BNCOD 16, Cardiff, Wales, July 1998*. Lecture Notes in Computer Science, Vol. 1405, pp. 119–133. Berlin: Springer 1998
5. Duquenne, V.: Contextual implications between attributes and some properties of finite lattices. In: Ganter, B., Wille, R. (eds.), *Beiträge zur Begriffsanalyse*. Kapitel 10, pp. 213–239. Mannheim: B. I.-Wissenschaftsverlag 1987
6. Fahrner, C. Vossen, G.: Transformation relationaler Datenbank-Schemas in objekt-orientierte Schemas gemäß ODMG-93. In: Lausen, G. (ed.), *Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'95), Dresden, März 1995*. Informatik aktuell, pp. 111–129. Berlin: Springer 1995
7. Ganter, B., Wille, R.: *Formale Begriffsanalyse*. Berlin/Heidelberg: Springer 1996
8. Gottlob, G., Schrefl, M., Röck, B.: Extending Object-Oriented Systems with Roles. ACM Transactions on Information Systems **14**(3), 268–296 (1996)
9. Haav, H.-M.: An Object Classifier Based on Galois Approach. In: Kangassalo, H., Nilsson, J. F., Jaakkola, H., Ohsuga, S. (eds.), *Information Modelling and Knowledge Bases VIII*. pp. 309–321. IOS Press 1997
10. Hohenstein, U., Körner, C.: Semantische Anreicherung relationaler Datenbanken. In: Lausen, G. (ed.), *Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'95), Dresden, März 1995*. Informatik aktuell, pp. 130–149. Berlin: Springer 1995
11. Hull, R.: Relative Information Capacity of Simple Relational Database Schemata. SIAM Journal on Computing **15**(3), 856–886 (1986)
12. Lee, B. S.: Normalization in OODB Design. ACM SIGMOD Record **24**(3), 23–27 (1995)
13. Rauh, O., Stickel, E.: Standard Transformations for the Normalization of ER Schemata. Information Systems **21**(2), 187–208 (1996)
14. Schmitt, I.: *Schemaintegration für den Entwurf Föderierter Datenbanken*. Dissertationen zu Datenbanken und Informationssystemen, Band 43. Sankt Augustin: infix 1998
15. Schmitt, I., Conrad, S.: Restructuring Class Hierarchies for Schema Integration. In: Topor, R., Tanaka, K. (eds.), *Database Systems for Advanced Applications '97, Proc. of the 5th Int. Conf., DASFAA'97, Melbourne, Australia, April 1–4, 1997*. pp. 411–420. Singapore: World Scientific Publishing 1997
16. Schmitt, I., Saake, G.: Integration of Inheritance Trees as Part of View Generation for Database Federations. In: Thalheim, B. (ed.), *Conceptual Modelling — ER'96, Proc. of the 15th Int.*

Conf., Cottbus, Germany, October 1996. Lecture Notes in Computer Science, Vol. 1157, pp. 195–210. Berlin: Springer 1996

17. Schmitt, I., Saake, G.: Merging Inheritance Hierarchies for Schema Integration based on Concept Lattices. Preprint 2, Fakultät für Informatik, Universität Magdeburg 1997 (auch verfügbar als http://wwwiti.cs.uni-magdeburg.de/publikationen/97/SS97.ps.gz)
18. Schmitt, I., Saake, G.: Merging Inheritance Hierarchies for Database Integration. In: Halper, M. (ed.), *Proc. of the 3rd IFCIS Int. Conf. on Cooperative Information Systems, CoopIS'98, August 20–22, 1998, New York, USA*. pp. 322–331. Los Alamitos (CA): IEEE Computer Society Press 1998
19. Schmitt, I., Türker, C.: Refining Extensional Relationships and Existence Requirements for Incremental Schema Integration. In: Gardarin, G., French, J., Pissinou, N., Makki, K., Bougamin, L., (eds.), *Proc. of the 7th ACM CIKM Int. Conf. on Information and Knowledge Management, November 3–7, 1998, Bethesda, Maryland, USA*. pp. 322–330. New York: ACM Press 1998
20. Tari, Z., Stokes, J., Spaccapietra, S.: Object Normal Forms and Dependency Constraints for Object-Oriented Schemata. ACM Transactions on Database Systems **22**(4), 513–569 (1997)
21. Waiyamai, K., Taouil, R., Lakhal, L.: Towards an Object Database Approach for Managing Concept Lattices. In: D. W. Embley, D. W., Goldstein, R. C. (eds.), *Conceptual Modelling — ER'97, Proc. of the 16th Int. Conf., Los Angeles, CA, USA, November 1997*. Lecture Notes in Computer Science, Vol. 1331, pp. 299–312. Berlin: Springer 1997
22. Wieringa, R. J., De Jonge, W., Spruit, P. A.: Using Dynamic Classes and Role Classes to Model Object Migration. Theory and Practice of Object Systems **1**(1), 61–83 (1995)
23. Wille, R.: Concept lattices and conceptual knowledge systems. Computer & Mathematics with Applications **23**(6-9), 493–515 (1992)
24. Yahia, A., Lakhal, L., Cicchetti, R., Bordat, J. P.: iO_2: An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In: Thalheim, B. (ed.), *Conceptual Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996*. Lecture Notes in Computer Science, Vol. 1157, pp. 422–437. Berlin: Springer 1996