Ferstl, Otto K.; Sinz, Elmar J.

Objekmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Erstveröffentlichung

Ferstl, Otto K.; Sinz, Elmar J. (1990): Objekmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM), in: Wirtschaftsinformatik : WI, Wiesbaden: Springer Gabler, Jg. 32, Nr. 6, S. 566–581.

Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Otto K. Ferstl, Elmar J. Sinz*

Stichworte: Semantisches Objektmodell (SOM), Konzeptionelle Objektmodellierung, Vorgangsmodellierung, Spezifikation betrieblicher Informationssysteme, objektorientierte Spezifikation

Zusammenfassung: Die konzeptionelle Datenmodellierung markiert einen entscheidenden Fortschritt in Richtung einer ganzheitlichen Modellierung betrieblicher Informationssysteme. Ein Indiz hierfür sind die aktuellen Bemühungen bei der Aufstellung „unternehmensweiter Datenmodelle". Keine vergleichbaren Fortschritte konnten dagegen bisher im Bereich der Funktionsmodellierung erzielt werden. Sichtbares Zeichen hierfür sind erhebliche Funktionsredundanzen und die daraus resultierenden Fehler und Wartungsprobleme.

Im vorliegenden Beitrag wird ein methodischer Ansatz für eine ganzheitliche konzeptionelle Modellierung betrieblicher Informationssysteme in objektorientierter Form vorgeschlagen. Der Ansatz besteht aus einer Methode zur konzeptionellen Objektmodellierung sowie einer Methode zur Vorgangsmodellierung. Er integriert die Daten-, Funktions- und Interaktionssicht eines betrieblichen Informationssystems. Das verwendete objektorientierte Systemparadigma harmoniert mit der zunehmenden Dezentrlaisierung von Systemen.

Den Kern der Arbeit bildet die Vorstellung eines hierfür geeigneten Semantischen Objektmodells (SOM), das auf dem Strukturierten Entity-Relationship-Modell und dem Objektmodell von Smalltalk beruht.

Object-oriented modelling of business information systems using a Semantic Object Model (SOM)

Keywords: Semantic Object Model (SOM), conceptual object modelling, transaction modelling, specification of business information systems, object-oriented design

Abstract: Conceptual data design provides an important step toward conceptual modelling of business information systems. This is indicated by the current efforts on building "enterprise-wide data models". Conceptual modelling still lacks in the field of functional modelling, where functional redundancies and resulting errors and maintainance problems are widespread.

* Prof. Dr. Otto K. Ferstl, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Rheinau 3–4, D-5400 Koblenz.
Prof. Dr. Elmar J. Sinz, Lehrstuhl für Wirtschaftsinformatik, insbesondere Systementwicklung und Datenbankanwendung, Universität Bamberg, Feldkirchenstraße 21, D-8600 Bamberg

The paper proposes an approach for conceptual modelling of business information systems in an object-oriented way. The approach consists of method for conceptual object modelling and a method for transaction modelling. It integrates the views of data, functions and interactions within a business information system. This object-oriented system paradigma fits nicely together with the increasing decentralization of computer systems.

This paper presents a Semantic Object Model (SOM), which is based on the Structured Entity-Relationship Model and the Smalltalk Object Model.

1 Einführung

Der vorliegende Beitrag beschreibt den methodischen Rahmen für eine ganzheitliche Modellierung betrieblicher Informationssysteme in objektorientierter Form. Das dazu notwendige Modellierungskonzept wird durch ein semantisches Objektmodell (SOM) bereitgestellt, dessen Vorstellung den Kern der Arbeit bildet. Die Modellierung im SOM wird in zwei Ebenen durchgeführt:

a) Gegenstand der konzeptionellen Objektmodellierung sind die betrieblichen Objekttypen und deren Beziehungsstruktur. Der Aufbau der Objekttypen wird im Objektentwurf, die Beziehungsstruktur im Objektsystementwurf modelliert. Das Ergebnis wird in einem konzeptionellen Objektschema beschrieben.
b) Gegenstand der Vorgangsmodellierung ist die Definition betrieblicher Abläufe in Form von Vorgangsobjekttypen. Dabei werden die an einem Vorgang beteiligten betrieblichen Objekttypen und deren Beziehungsstruktur anhand des konzeptionellen Objektschemas ermittelt und die zulässigen Abläufe festgelegt.

Konzeptionelle Objektmodellierung wird im vorliegenden Zusammenhang als objektorientierte Erweiterung der konzeptionellen Datenmodellierung verstanden. Ziel ist die gemeinsame Modellierung von Daten und Funktionen auf der Grundlage eines objektorientierten Paradigmas. Funktionen werden dabei durch exklusive Zuordnung von Methoden zu (Daten-) Objekttypen sowie durch den Nachrichtenaustausch zwischen Objekten beschrieben.

Die konzeptionelle Datenmodellierung hat als integraler Bestandteil der Systementwicklung in den letzten Jahren eine breite Akzeptanz gefunden. Dies wird u.a. deutlich an den Anstrengungen vieler Unternehmen, sogenannte „unternehmensweite Datenmodelle" aufzustellen. Ziel der konzeptionellen Datenmodellierung ist es, im Rahmen einer Informationsanalyse einen gegebenen Ausschnitt der Realität in Form eines konzeptionellen Datenschemas zu beschreiben, wobei die Art der Beschreibung durch das verwendete semantische Datenmodell bestimmt wird. Im Bereich betrieblicher Informationssysteme dominieren semantische Datenmodelle auf der Grundlage des Entity-Relationship-Modells (ERM). Ein konzeptionelles Datenschema gemäß ERM besteht aus Datenobjekttypen mit zugeordneten (Daten-) Attributen, Schlüsselreferenzen zwischen Datenobjekttypen und Integritätsbedingungen. Letztere beschreiben in Form von statischen Integritätsbedingungen zulässige Ausprägungen, in Form von dynamischen Integritätsbedingungen zulässige Transformationen der Extensionen (Datenbasen) eines Datenschemas.

Abgesehen von dynamischen Integritätsbedingungen wird mit Hilfe eines Datenmodells ausschließlich die Datensicht eines Systems beschrieben (Bild 1). Im

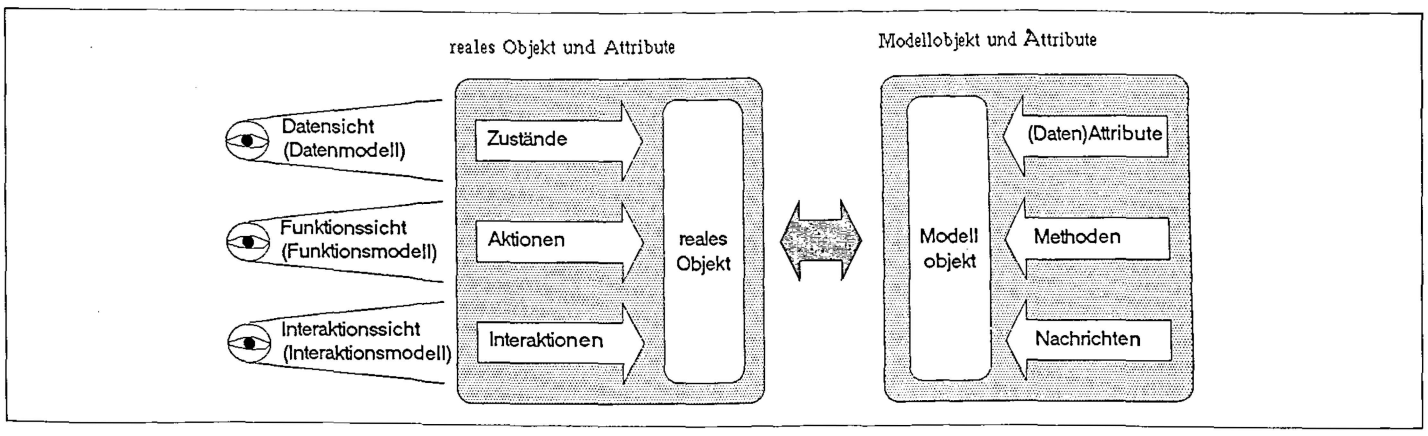

Bild 1 Objekt und Objektsichten

Gegensatz dazu beschreibt ein Objektmodell zusätzlich die Funktionssicht und die Interaktionssicht eines Systems. Ein Objektmodell unterstützt somit eine ganzheitliche, objektorientierte Systembeschreibung.

Objektorientierte Ansätze befinden sich seit einigen Jahren in einer stürmischen Entwicklung. Beispiele für einzelne Entwicklungsrichtungen sind objektorientierte Programmiersprachen, objektorientierte Analyse und Spezifikation sowie objektorientierte Datenbanksysteme und Benutzeroberflächen. Den unterschiedlichen Ansätzen liegen Objektmodelle mit einer weitgehend gemeinsamen Objektsicht zugrunde. Dabei wird jedes Objekt als Instanz einer Klasse (Objekttyp) aufgefaßt. Eine Klasse wird als abstrakter Datentyp verstanden, der durch Variablen (Attribute) und Methoden (Operatoren) beschrieben wird. Jede Klasse kann als Spezialisierung (Subklasse) einer oder mehrerer Superklassen vereinbart werden. Dabei vererbt eine Superklasse Variablen und Methoden an ihre Subklassen. Die Instanzen der einzelnen Klassen interagieren mit Hilfe von Nachrichten. Eine Nachricht löst beim Empfängerobjekt die Durchführung einer Methode aus.

Ziel der konzeptionellen Objektmodellierung ist, einen gegebenen, als Miniwelt bezeichneten Ausschnitt der Realität in Form eines konzeptionellen Objektschemas zu beschreiben. Die Art der Beschreibung wird durch das verwendete semantische Objektmodell bestimmt. Mit dem Adjektiv „semantisch" wird dabei in Analogie zu den semantischen Datenmodellen die „Nähe" des Begriffsystems zur abzubildenen Realität betont. So wie ein semantisches Datenmodell im Vergleich zum relationalen Daten(bank)modell „näher an der Realität" sein soll, gilt dies analog für ein semantisches Objektmodell z.B. im Vergleich zum Objektmodell von Smalltalk.

Die konzeptionelle Objektmodellierung im SOM erfolgt in zwei Stufen, dem Objektsystementwurf und dem Objektentwurf. Der Objektsystementwurf grenzt das Objektsystem von seiner Umwelt ab und legt die Klassen des Objektsystems sowie die Beziehungsstruktur der Klassen fest. Die Schlüsselreferenzen zwischen den Objekttypen (Datensicht) bilden gleichzeitig Interaktionskanäle für den Austausch von Nachrichten (Interaktionssicht) und erhalten auf diese Weise eine zusätzliche, funktional auswertbare Semantik (Funktionssicht). Neben dem Austausch von Nachrichten (*"interacts_with"*) verwendet SOM die in obektorientierten Ansätzen üblichen Abstraktionen Generalisierung (*"is_a"*) und Aggregation (*"is_part_of"*).

Neben einer engen Kopplung der einzelnen Sichten wird damit gleichzeitig erreicht, daß die in vielen Unternehmen existierenden konzeptionellen Datenschemata als Grundlage für konzeptionelle Objektschemata verwendbar sind. Diese durch Informationsanalyse gewonnenen Schemata sind nach wie vor gültig, sie beschreiben lediglich einen Teilaspekt, die Datensicht des Systems. Das konzeptionelle Datenschema stellt somit eine Projektion des konzeptionellen Objektschemas dar. Die Beziehung zwischen Objektschema und Datenschema ist durch Konsistenzbedingungen geregelt. Die in die konzeptionelle Datenmodellierung getätigten Investitionen werden damit geschützt.

Der Objektentwurf definiert die Struktur und das Verhalten der im Objektsystementwurf festgelegten Klassen. Dabei werden für jede einzelne Klasse die Attribute, die von ihr interpretierbaren Nachrichten sowie die zugehörigen Methoden beschrieben.

Die zweite Modellierungsebene besteht aus dem Vorgangsentwurf. Dieser modelliert betriebliche Vorgänge in Form von Vorgangsobjekttypen. Die Grundlage hierfür bilden der Objektsystementwurf und der Objektentwurf.

Die drei Entwurfsschritte sind wegen ihrer gegenseitigen Abhängigkeit nicht isoliert und nicht streng sequentiell durchführbar.

2 Die methodische Einbettung des SOM

Entsprechend dem Verständnis der konzeptionellen Objektmodellierung als objektorientierte Erweiterung der konzeptionellen Datenmodellierung sind die methodischen Wurzeln des SOM ein semantisches Datenmodell und ein Objektmodell. Diese beiden Modelle werden im folgenden als Basis-Datenmodell bzw. Basis-Objektmodell von SOM bezeichnet.

Das gewählte Basis-Objektmodell orientiert sich weitgehend an Smalltalk, dessen Objektmodell sich in vielen anderen Ansätzen widerspiegelt. Als Basis-Datenmodell für SOM wird das Strukturierte Entity-Reationship-Modell (SERM) gewählt. Ausschlaggebend für die Wahl des SERM ist dessen Hierarchisierung von Datenobjekttypen auf der Grundlage von Existenzabhängigkeiten. Diese Hierarchisierung ist strukturkompatibel mit der Bildung von Generalisierungs- und Aggregationshierarchien im Objektmodell.

2.1 Das Strukturierte Entity-Relationship-Modell (SERM) als Basis-Datenmodell des SOM

Das Strukturierte Entity-Relationship-Modell (SERM) [12], [13] ist eine Weiterentwicklung des klassischen Entity-Relationship-Modells (ERM) [1]. Der wesentliche Unterschied zum ERM besteht darin, daß Schlüsselreferenzen zwischen Datenobjekttypen primär unter dem Gesichtspunkt der darin enthaltenen Existenzabhängigkeiten analysiert werden. Dabei

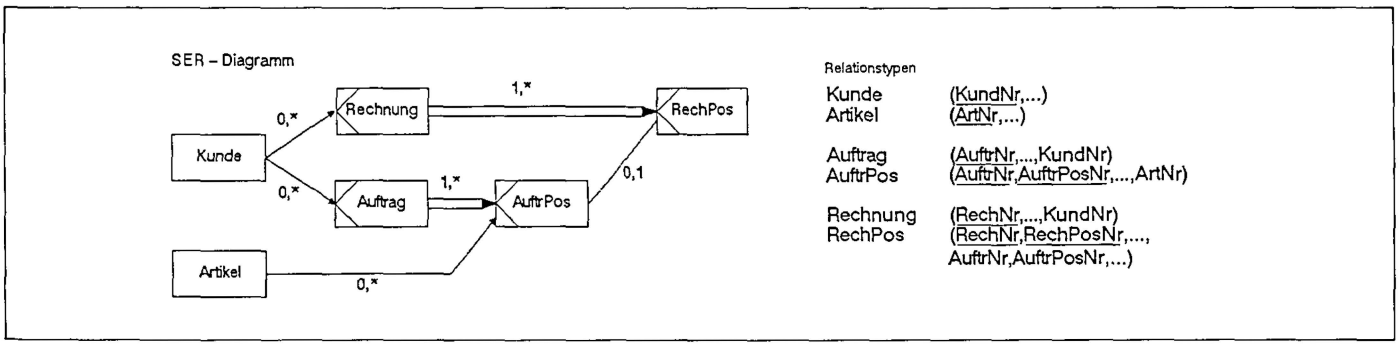

Bild 2 Konzeptionelles Datenschema des Handelsbetriebs (Ausschnitt)

werden alle Paare von in Beziehung stehenden Datenobjekttypen nach dem Schema originär/abhängig geordnet. Bei der graphischen Darstellung im SER-Diagramm wird der abhängige Datenobjekttyp rechts vom originären angeordnet. Dadurch entsteht ein quasi-hierarchischer Graph (gerichtet und azyklisch), der Existenzabhängigkeiten zwischen Datenobjekttypen sowie Folgen von Existenzabhängigkeiten klar visualisiert. Bild 2 zeigt einen Ausschnitt aus dem konzeptionellen Datenschema eines Handelsbetriebes. Hier sind die Datenobjekttypen *Auftrag* und *Rechnung* von Kunde abhängig, *AuftrPos* hängt von *Auftrag* und *Artikel* ab usw.

Im ERM werden Existenzabhängigkeiten zwischen Datenobjekttypen lediglich im Zusammenhang mit schwachen Entity-Typen betrachtet. Sie führen zu keiner speziellen Darstellungsrichtung im ER-Diagramm. Die Grundstruktur eines ER-Diagramms ist somit die eines allgemeinen (nicht-hierarchischen) Graphen.

Definition von Existenzabhängigkeiten im SERM

Die Beziehung zwischen einem originären Datenobjekttyp A und einem abhängigen Datenobjekttyp B wird durch einen Komplexitätsgrad comp(A, B) in (min, max)-Notation beschrieben ($0 \leq \min \leq 1 \leq \max \leq *$; „*" bedeutet „beliebig viele"). Dieser gibt an, mit wievielen Datenobjekten des Typs B ein Datenobjekt des Typs A minimal in Beziehung stehen muß und maximal in Beziehung stehen kann. Jeder Komplexitätsgrad drückt damit eine referentielle Integritätsbedingung (Referenzbedingung) zwischen den in Beziehung stehenden Datenobjekttypen aus. Aus den Eckwerten (0, 1), (0, *), (1, 1) und (1, *) sind zwei Formen von Existenzabhängigkeiten ableitbar:

- Einseitige Existenzabhängigkeit (B hängt von A ab): comp(A, B) = (0, 1) oder comp(A, B) = (0, *).
- Wechselseitige Existenzabhängigkeit (B und A sind wechselseitig abhängig): comp(A, B) = (1, 1) oder comp(A, B) = (1, *).

In Bild 2 stellt z.B. die Beziehung zwischen *Kunde* und *Auftrag* eine einseitige Existenzabhängigkeit, die Beziehung zwischen *Auftrag* und *AuftrPos* eine wechselseitige Existenzabhängigkeit dar.

Elemente des SERM

Wie das ERM unterscheidet auch das SERM zwischen Gegenstands-Objekttyp (Entity-Typ, E-Typ) und Beziehungs-Objekttyp (Relationship-Typ, R-Typ) (Bild 3a). Als dritter Datenobjekttyp kommt der Gegenstands-Beziehungs-Objekttyp (Entity-Relationship-Typ, ER-Typ) hinzu. Dieser entsteht durch Zusammenfassung eines E-Typs mit denjenigen R-Typen, die mit ihm durch (1, 1)-Beziehungen verbunden sind. (1, 1)-Beziehungen werden daher in expliziter Form nur für den Spezialfall der Generalisierung benötigt.

Beziehungen zwischen Datenobjekttypen werden im SERM gerichtet interpretiert. Jede Beziehung verläuft von einem Gegenstands-Objekttyp (E-Typ, ER-Typ) zu einem Beziehungs-Objekttyp (ER-Typ, R-Typ) und wird durch einen Komplexitätsgrad beschrieben (Bild 3b). Aus der Beziehungsrichtung folgt, daß im SER-Diagramm ein E-Typ nicht Zielknoten, ein R-Typ nicht Startknoten einer Kante sein kann. Zur Visualisierung der Existenzabhängigkeit wird der Startknoten einer Kante links vom Zielknoten angeordnet. Z.B. bedeutet die (0, *)-Beziehung zwischen *Kunde* und *Auftrag* in Bild 2, daß jeder Kunde null bis beliebig viele Aufträge erteilen kann. Umgekehrt bezieht sich jeder Auftrag auf genau einen Kunden. Zu einem Auftrag gehört wenigstens eine Auftragsposition (*AuftrPos*), die sich umgekehrt auf genau einen Auftrag und genau einen Artikel bezieht.

Die quasi-hierarchische Struktur des SERM bleibt auch auf der Attributebene erhalten (Bild 3c). Zur Realisierung von Schlüsselreferenzen gibt jeder Relationstyp seinen Primärschlüssel als (Teil-) Primärschlüssel oder als Fremdschlüssel an seine Nachfolger weiter.

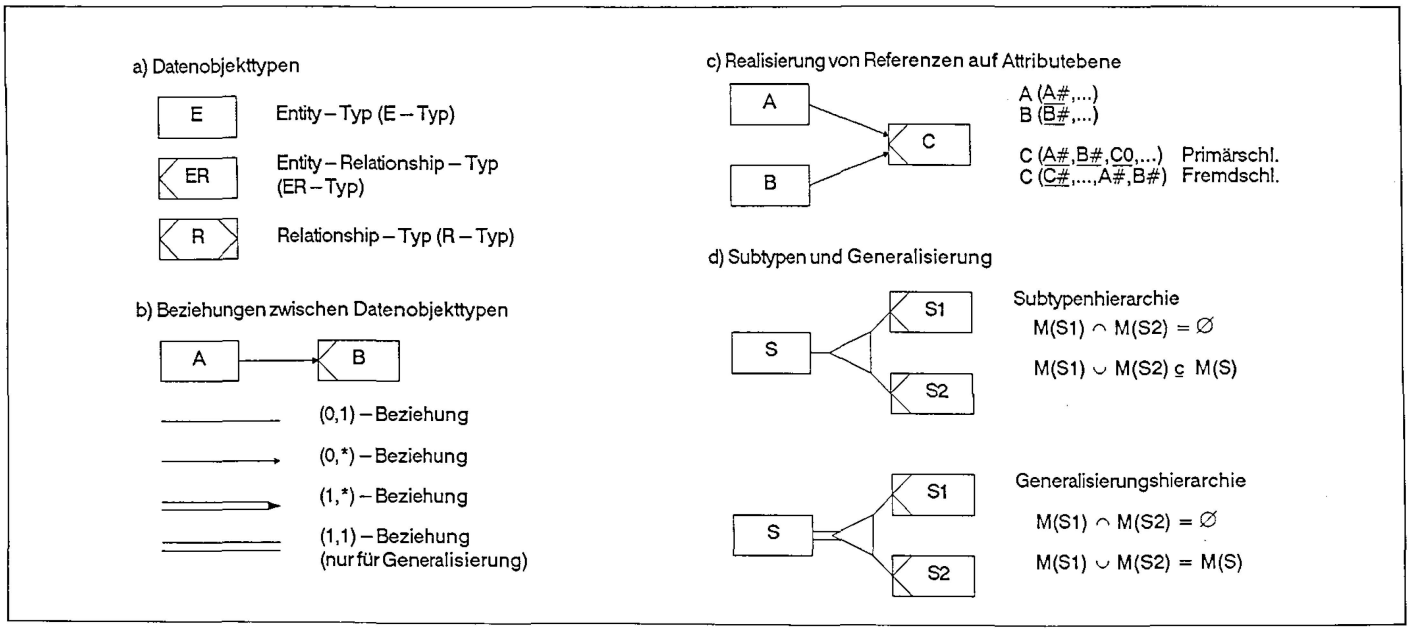

Bild 3 Elemente des Strukturierten Entity-Relationship-Modells (SERM)

Dabei gilt:

- Der Primärschlüssel eines R-Typs besteht aus den Primärschlüsseln seiner Vorgänger.
- Ein ER-Typ enthält in der Regel die Primärschlüssel seiner Vorgänger als Fremdschlüssel (z.B. *Auftrag*).
- Ein ER-Typ kann auch den Primärschlüssel eines oder mehrerer Vorgänger als (Teil-)Primärschlüssel enthalten (z.B. *AuftrPos*). Dies entspricht einem schwachen Entity-Typ im Sinne des ERM.

Bild 3d zeigt schlie·ßich das Generalisierungskonzept von SERM in Form von Subtypen- und Generalisierungshierarchien. Eine Subtypenhierarchie liegt vor, wenn jedes Entity des Typs $S_1, S_2 \ldots S_n$ auch Entity des Typs S ist. Eine Generalisierungshierarchie liegt vor, wenn jedes Entity von S auch Entity von genau einem der Entity-Typen $S_1, S_2 \ldots S_n$ ist. Das Dreieck-Symbol bedeutet, daß die Datenobjektmengen $M(S_1)$ und $M(S_2)$ disjunkt sind ($M(S_1) \cap M(S_2) = \emptyset$). Dies stellt eine zusätzliche Integritätsbedingung dar.

2.2 Das Objektmodell von Smalltalk als Basis-Objektmodell des SOM

Das für SOM verwendete Basis-Objektmodell orientiert sich weitgehend am Objektmodell von Smalltalk [6], [7]. Geringfügige Abweichungen sind durch die beabsichtigte Integration mit dem Basis-Datenmodell bedingt.

Im Mittelpunkt eines objektorientierten Ansatzes steht das Konzept der Klasse (Objekttyp) (Bild 4). Die Bildung von Klassen dient der Typisierung von Objekten. Jedes Objekt ist genau einer Klasse als Instanz zugeordnet. Alle Instanzen einer Klasse besitzen den gleichen strukturellen Aufbau, der durch Instanzvariablen (Attribute) beschrieben wird. Zu jeder Klasse gehört eine Menge von Methoden (Operatoren), die auf den Instanzen ausführbar sind.

Objekte kommunizieren untereinander mit Hilfe von Nachrichten (Messages). Jeder interpretierbaren Nachricht ist über ein Method Directory (Index) eine Methode zugeordnet. Empfängt ein Objekt eine Nachricht, so wird anhand des Method Directory eine Methode zur Behandlung der Nachricht ausgewählt und auf dem Objekt ausgeführt. Dadurch wird es möglich, den Inhalt („was") einer Nachricht von ihrer Behandlung („wie") zu trennen. Eine Methode kann während ihrer Ausführung Nachrichten an andere Objekte senden. Nachrichten sind mit Argumenten parametrisierbar. Jede Nachricht wird durch eine Antwort quittiert. Dies führt zu dem in Bild 5 dargestellten Stufenkonzept der Methodendurchführung.

Die Klasseninstanz repräsentiert die Klasse selbst. Sie dient dazu, Nachrichten an eine Klasse senden zu können, um dort Klassenmethoden auszulösen (z.B. *Erzeuge_Instanz*). Klasseninstanzen besitzen keine Instanzvariablen und damit keinen nach außen sichtbaren, permanenten Zustand.

Klassen werden als Spezialisierungen (Subklassen) einer allgemeineren Klasse (Superklasse) beschrieben. Auf diese Weise entsteht eine Klassenhierarchie.

Jede Superklasse vererbt alle Attribute und Methoden an ihre Subklassen. Bei einfacher Vererbung übernimmt eine Klasse die Attribute und Methoden von einer Superklasse, bei multipler Vererbung von mehreren Superklassen, ggf. unter Berücksichtigung spezieller Vererbungsregeln.

Durch das Prinzip der Vererbung wird es möglich, Methoden für mehrere Klassen gemeinsam zu beschreiben. Eine Subklasse kann die ererbte Attributmenge erweitern und die ererbten Methoden modifizieren. Empfängt ein Objekt eine Nachricht, die im Method Directory seiner Klasse nicht vor-

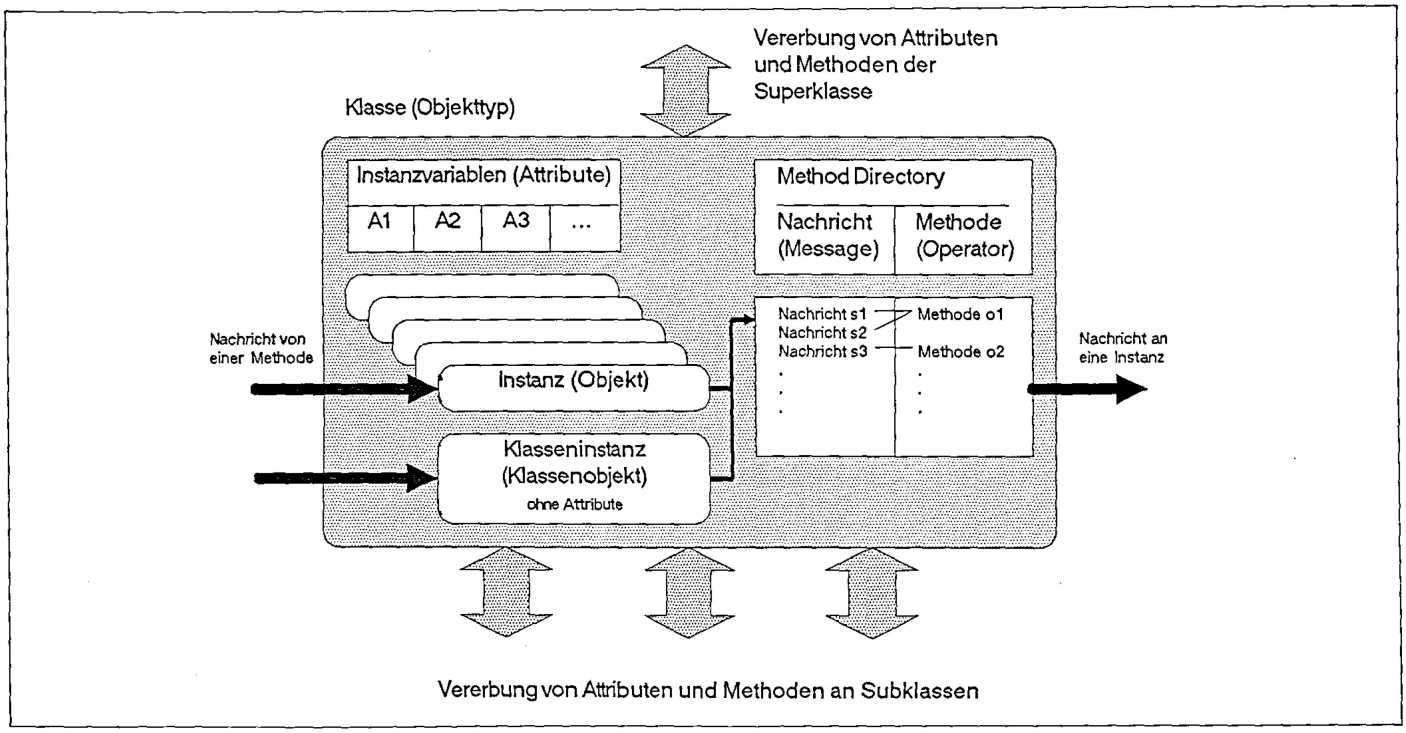

Bild 4 Basis-Objektmodell des SOM

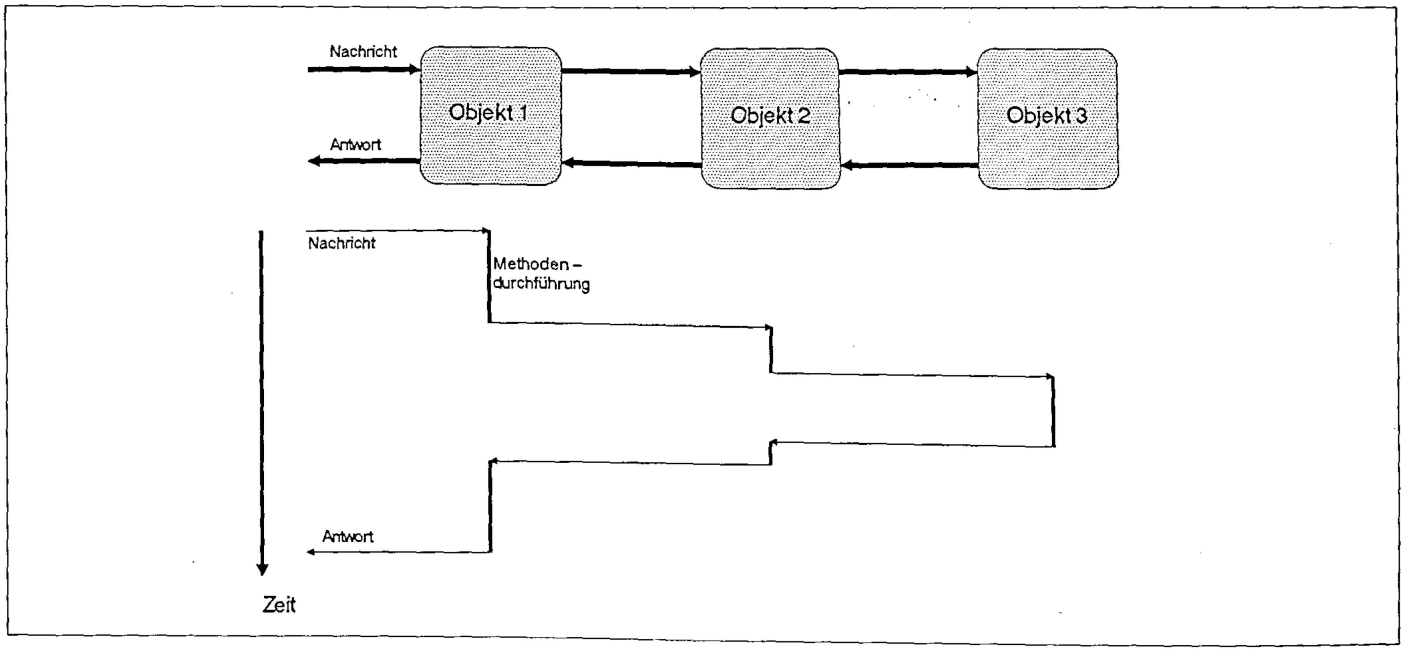

Bild 5 Stufenkonzept der Methodendurchführung

kommt, so wird in der Superklasse weitergesucht. Erst wenn die Nachricht auch im Method Directory der obersten Superklasse nicht gefunden wird, wird sie als nicht interpretierbar zurückgemeldet.

Eine bestimmte Nachricht, die an Objekte verschiedener Klassen gesandt wird, kann dort unterschiedliche Wirkungen erzeugen. Diese Eigenschaft wird als Polymorphie bezeichnet und ist ein wesentliches Merkmal objektorientierter Systeme.

3 Die konzeptionelle Modellierungsebene des SOM

Die konzeptionelle Modellierungsebene des SOM beruht auf der Integration von Basis-Datenmodell und Basis-Objektmodell. Mit Hilfe des Basis-Datenmodells wird die Datensicht eines Informationssystems in Form eines konzeptionellen Datenschemas beschrieben. Dieses

- legt die Instanzvariablen der Klassen fest,
- bestimmt die Grundstruktur für die Hierarchisierung der Klassen und
- bildet das "Verkehrsnetz" für den Nachrichtenaustausch zwischen den Instanzen.

Mit Hilfe des Basis-Objektmodells wird zusätzlich die Funktionssicht und die Interaktionssicht eines Informationssystems in Form eines konzeptionellen Objektschemas beschrieben.

Das konzeptionelle Datenschema stellt eine Projektion des zugehörigen konzeptionellen Objektschemas in Bezug auf seine Attribute dar. Dadurch wird die Kompatibilität zwischen Objektmodell und Datenmodell erreicht, die durch nachstehende begriffliche Zuordnung beschrieben wird. Die Begriffe *Objekt* und *Objekttyp* werden im folgenden für SOM und SERM gemeinsam verwendet.

Datenmodell	*Objektmodell*
Datenobjekttyp, Objekttyp	Klasse, Objekttyp
Datenobjekt, Objekt	Instanz, Objekt
Attribut	Instanzvariable, Attribut
Beziehung	Kommunikationsweg, Interaktionskanal

3.1 Objekte und Interaktionen im SOM

Bei der Modellierung eines konzeptionellen Objektschemas im SOM wird im Objektsystementwurf die Beziehungsstruktur der Klassen beschrieben. Gegenstand des Objektentwurfs ist dagegen das Verhalten und die Struktur der einzelnen Klassen. Hierbei werden für jede Klasse ihre Attribute, die von ihr interpretierbaren Nachrichten und die zugehörigen Methoden festgelegt. Die Zuordnung der Attribute erfolgt unter Beachtung der dritten bzw. vierten Normalform.

Objektorientierte Ansätze gehen in der Regel davon aus, daß jedes Objekt in beliebiger Weise Nachrichten an andere Objekte bzw. Systemschnittstellen senden und umgekehrt Nachrichten empfangen kann. Diese Möglichkeit der uneingeschränkten Kommunikation führt dazu, daß Abläufe in objektorientierten Systemen nur schwer nachvollziehbar sind (Notwendigkeit eines *System Browser*) und damit ein erhebliches Fehlerpotential darstellen. SOM verwendet dagegen die im konzeptionellen Datenschema festgelegten Schlüsselreferenzen zwischen Datenobjekttypen als „Verkehrsnetz" für die Kommunikaion der Objekte mit Hilfe von Nachrichten (Bild 3c). Eine Interaktion zwischen einem Sender-Objekt a und einem Empfänger-Objekt b umfaßt

- das Versenden der Nachricht durch a,
- den Empfang der Nachricht durch b und die Methodendurchführung bei b sowie
- das Versenden einer Quittung durch b und den Empfang der Quittung durch a.

Interaktionen finden entweder direkt zwischen Gegenstands-Objekten (des Typs E oder ER) oder indirekt unter Zwischenschaltung von Beziehungs-Objekten (des Typs R) statt. Die zugehörigen Schlüsselreferenzen werden gemäß SERM durch Primär- und Fremdschlüsselwerte hergestellt. Dadurch wird die Adressierbarkeit eines oder mehrerer Empfängerobjekte durch ein Senderobjekt garantiert. Mit anderen Worten, eine Methode des Objekttyps B kann während ihrer Ausführung auf einem Objekt b des Typs B (b:B) nur dann eine Message an ein Objekt a:A senden, wenn zwischen A und B eine SER-Beziehung besteht.

In Bild 6 sind für die beiden Fälle die (statische) SER-Beziehung und die (dynamische) SOM-Interaktion gegenübergestellt.

a) Direkte Interaktion (Bild 6a): Jedem Objekt a:A sind null bis beliebig viele Objekte b:B zugeordnet. Jedem b:B ist genau ein a:A zugeordnet. Sender einer Nachricht können Objekte des Typs A oder B sein. Objekte des Typs B senden stets an das zugehörige a:A. Ein Objekt a:A sendet entweder an ein ausgewähltes b:B oder an alle zugeordneten b:B (broadcast message).

b) Indirekte Interaktion (Bild 6b): Jedem a:A sind mehrere b:B, jedem b:B sind mehrere a:A über Beziehungsobjekte des Typs R zugeordnet. Sender einer Nachricht können Objekte des Typs A oder B sein. Empfänger der Nachricht ist entweder ein ausgewähltes Objekt des Typs B bzw. A oder alle zugeordneten b:B (a:A).

Die Fälle (a) und (b) sind für SER-Kanten des Typs (0, 1) oder (1, *) in analoger Weise interpretierbar.

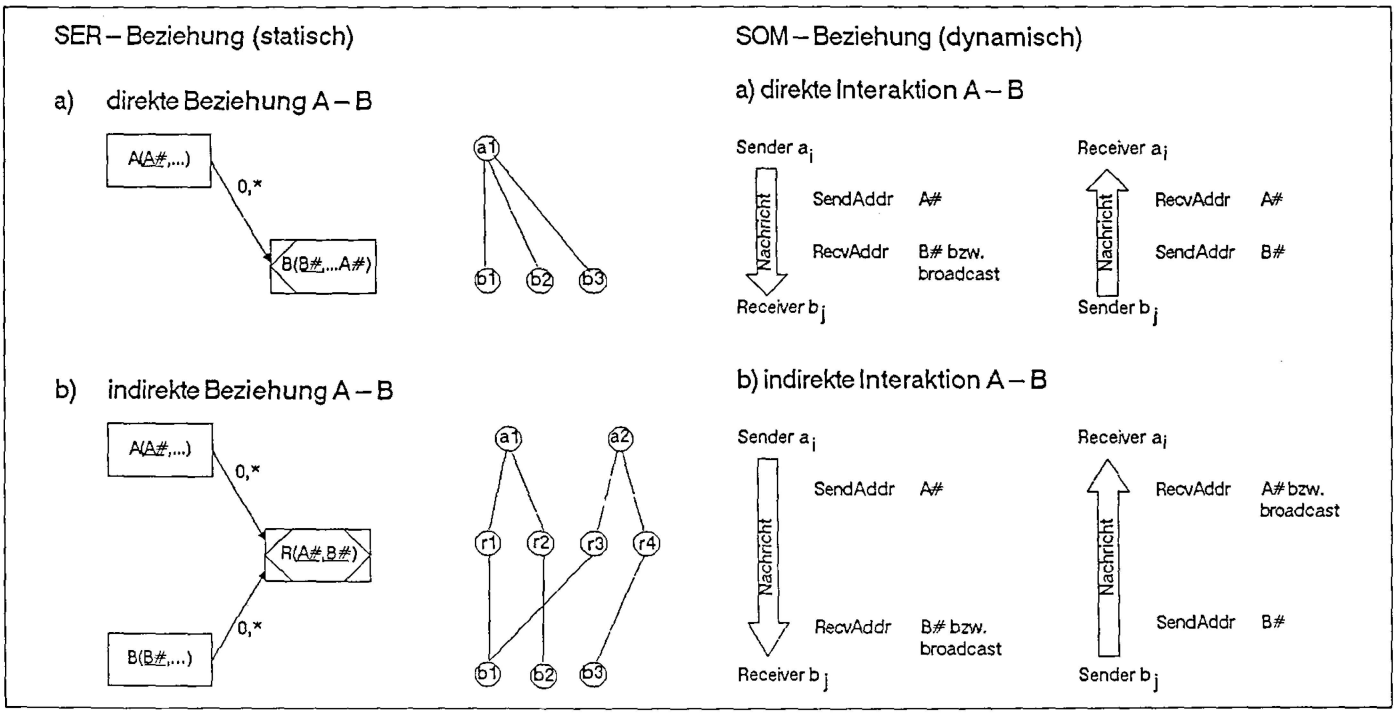

Bild 6 Interaktionen im SOM

3.2 Beziehungsarten im SOM

Im SOM werden die Objekttypen eines konzeptionellen Objektschemas durch Beziehungen des Typs *interacts_with, is_a* und *is_part_of* verknüpft. Aus Datensicht stellt jede dieser Beziehungen eine SER-Beziehung dar. Dadurch erhalten die SER-Beziehungen eine zusätzliche Semantik. Graphisch wird dies durch eine Kombination aus Kantensymbol (SER-Beziehung) und Linienart (SOM-Beziehung) dargestellt (Bild 7).

Eine SOM-Beziehung enthält Aussagen sowohl über die Verknüpfung der Objekttypen als auch über die Verknüpfung der zugehörigen Instanzen. So beschreibt z.B. eine *interacts_with*-Beziehung einen Interaktionskanal zwischen zwei Objekttypen sowie den Nachrichtenaustausch zwischen den zugehörigen Instanzen. Aus Datensicht werden diese Beziehungen auf Schlüsselreferenzen zwischen Objekttypen bzw. Schlüsselwertreferenzen zwischen den zugehörigen Objekten projiziert.

Die Beziehungsarten des SOM werden im folgenden näher beschrieben. Die elementare Beziehungsart ist *interacts_with*. Sie dient zur Abbildung von Interaktionsbeziehungen der realen Welt in Form eines Nachrichtenaustausches zwischen Objekten. Zur Generalisierung (bzw. Spezialisierung) von Objekttypen wird die Beziehungsart *is_a* verwendet. Aggregationen von Objekten, bzw. Zerlegungen von Objekten wer-

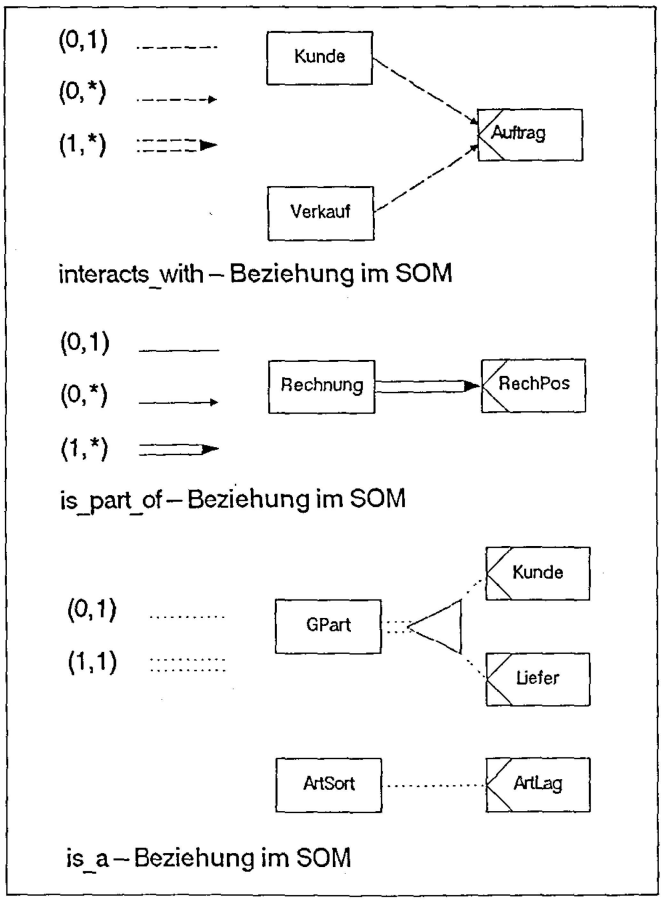

Bild 7 SOM-Beziehungsarten

den mit Hilfe der Beziehungsart *is_part_of* beschrieben.

Die Beziehungsarten *is_a* und *is_part_of* sind in den meisten Objektmodellen verfügbar und stellen Hilfsmittel zur Reduzierung der Komplexität sowie der Funktionsredundanz dar [2], [10]. Die Beziehungsarten *is_a* und *is_part_of* bilden hierarchische bzw. quasi-hierarchische Strukturen. Die dadurch entstehenden Hierarchieebenen sind zueinander orthogonal

interacts_with

Eine *interacts_with*-Beziehung zwischen zwei Objekttypen spezifiziert einen Interaktionskanal für den Nachrichtenaustausch zwischen zugehörigen Objekten. Objekte sind dabei die Instanzen bzw. die Klasseninstanzen der zugehörigen Objekttypen. Gemäß Abschnitt 3.1 ist eine direkte oder eine indirekte Interaktion möglich.

is_a

Die *is_a*-Beziehung des SOM dient der Modellierung von Generalisierungen. Sie wird üblicherweise bei Objektmodellen und Datenmodellen unterschiedlich interpretiert:

a) Bei der *is_a*-Beziehung von Objektmodellen vererbt eine Superklasse S alle Attribute und Methoden an ihre Subklassen $S_1 ... S_n$. Die Spezialisierung einer Subklasse gegenüber den zugehörigen Superklasse(n) drückt sich in einer Erweiterung der Attributmenge und/oder in einer Ergänzung bzw. Abänderung der Methoden aus. Jede Instanz ist genau einer Klasse zugeordnet und besitzt Attribute und Methoden dieser Klasse und der zugehörigen Superklassen.

b) Bei Datenmodellen dient die Generalisierung der Zerlegung der Attributmenge eines Objekttyps E in einen Supertyp S und ggf. mehrere Subtypen $S_1 ... S_n$. S umfaßt die Attribute, die alle Objekte von E besitzen.$S_1 ... S_n$ enthalten Attribute, die nur zu bestimmten Spezialisierungen von E gehören. Eine Instanz von E wird daher durch eine Instanz von S und ggf. zusätzlich durch eine Instanz von einem der Subtypen $S_1 ... S_n$ repräsentiert. Ein Beispiel hierfür ist die Spezialisierung des Supertyps Geschäftspartner (*GPart*) in die Subtypen *Kunde* und *Liefer*. Dabei ist jeder Geschäftspartner exklusiv Kunde oder Lieferant. Kann ein Geschäftspartner sowohl Kunde als auch Lieferant sein, wird dies unter Weglassung des Dreiecksymbols durch (0, 1)-Beziehungen zwischen *GPart* und *Kunde* sowie *GPart* und *Liefer* beschrieben.

Im SOM wird eine Synthese der Interpretationen (a) und (b) verwendet. Unter dem Blickwinkel der Nachrichtenbehandlung durch Objekte gilt Interpretation (a). Dagegen wird unter dem Blickwinkel der Vermeidung von Datenredundanz Interpretation (b) verwendet. Aus dieser Interpretation folgt, daß mehrere Objekte unterschiedlicher SOM-Objekttypen eine gemeinsame Objektidentität [4] besitzen. Z.B. besitzen das einen bestimmten Kunden repräsentierende Objekt von *GPart* und das zugehörige Objekt von *Kunde* eine gemeinsame Objektidentität. Für den Fall, daß ein *GPart* gleichzeitig *Kunde* und *Liefer* ist, erstreckt sich die Objektidentität auf drei Objekte. Dieser Fall wird allerdings in herkömmlichen Objektmodellen nicht berücksichtigt.

Die *is_a*-Beziehung bildet im Fall von einfacher Vererbung hierarchische, im Fall von multipler Vererbung quasi-hierarchische Strukturen.

is_part_of

Die *is_part_of*-Beziehung des SOM dient der Modellierung komplexer Objekttypen (complex objects) in Form von Aggregationen. Sie beschreibt die Beziehung zwischen einem Objekt und seinen (mehrfach auftretenden) Teilobjekten.

Aus Datensicht entsteht die *is_part_of*-Beziehung durch Normalisierung (3NF bzw. 4NF) der Attributmenge eines Objekttyps. Für die Beziehung zwischen einem Objekt und seinen Teilobjekten gilt in der Regel eine semantische Integritätsbedingung. Z.B. muß der Gesamtbetrag einer Rechnung (Attribut des Objekttyps *Rechnung*) gleich der Summe der Beträge der zugehörigen Rechnungspositionen (Attribut des Objekttyps *RechPos*) sein.

Ein Objekt und seine Teilobjekte interagieren durch Austausch von Nachrichten. Durch diese Interaktion wird u.a. die Erfüllung der jeweiligen semantischen Integritätsbedingung ermöglicht. Wird z.B. eine Rechnungsposition mit Wert *x* storniert, so muß eine Nachricht *VerändereRechnungsbetrag*(*-x*) an die zugehörige Rechnung gesandt werden.

Bei einem physischen Objektverständnis bildet die *is_part_of*-Beziehung hierarchische Strukturen (z.B. kann ein konkreter Motor nur in ein konkretes Fahrzeug eingebaut werden). Objektbeschreibungen können dagegen auch quasi-hierarchische Strukturen bilden (z.B. kann ein bestimmter Motortyp in mehreren Fahrzeugtypen Verwendung finden). Quasi-hierarchische *is_part_of*-Strukturen führen zu shared subobjects [4].

3.3 Konsistenzbedingungen zwischen SERM und SOM

Für die Beziehung zwischen SERM und SOM gelten die in Bild 8 dargestellten Konsistenzbedingungen. Diese werden im folgenden näher erläutert:

- Der R-Typ des SERM dient aus der Sicht von SOM ausschließlich der Realisierung von indi-

SOM – Beziehung	SER – Beziehung					
	(0,1)	(0,*)	(1,*)	Genera – lisierung	Sub – typen	Kanten zu einem R – Typ
interacts_with	selten	häufig	selten	nie	nie	stets
is_a	häufig	nie	nie	stets	stets	nie
is_part_of	selten	häufig	häufig	nie	nie	nie

Bild 8 Konsistenzbedingungen zwischen SOM-SERM-Beziehungen

rekten Interaktionen. Alle zu einem R-Typ führenden SER-Beziehungen werden daher im SOM als *interacts_with*-Beziehungen modelliert.

- Subtypen- und Generalisierungshierarchien des SERM korrespondieren stets mit *is_a*-Beziehungen im SOM. Daneben korrespondieren auch (0,1)-Beziehungen im SERM mit *is_a*-Beziehungen im SOM, liefern aber im Gegensatz zu Subtypen- und Generalisierungshierarchien keine zusätzlichen Integritätsbedingungen. Aus Datensicht wäre die im Zusammenhang mit der Generalisierung auftretende Objektzerlegung auch Hilfe der *is_part_of*-Beziehung beschreibbar. Dies widerspricht jedoch dem Objektverständnis von SOM.
- (1, *)-Beziehungen des SERM korrespondieren in der Regel mit *is_part_of*-Beziehungen im SOM. Sie beschreiben eine wechselseitige Existenzabhängigkeit zwischen einem Objekt und seinen Teilobjekten. Dagegen stellen *interacts_with*-Beziehungen zwischen wechselseitig abhängigen Objekten die Ausnahme dar.
- (0, *)-Beziehungen des SERM korrespondieren sowohl mit *is_part_of*-Beziehungen als auch mit *interacts_with*-Beziehungen des SOM.
- (0,1)-Beziehungen des SERM korrespondieren, wie bereits erwähnt, in der Regel mit *is_a*-Beziehungen des SOM. Objekte mit maximal einem Teilobjekt bzw. Interaktionspartner sind dagegen selten.

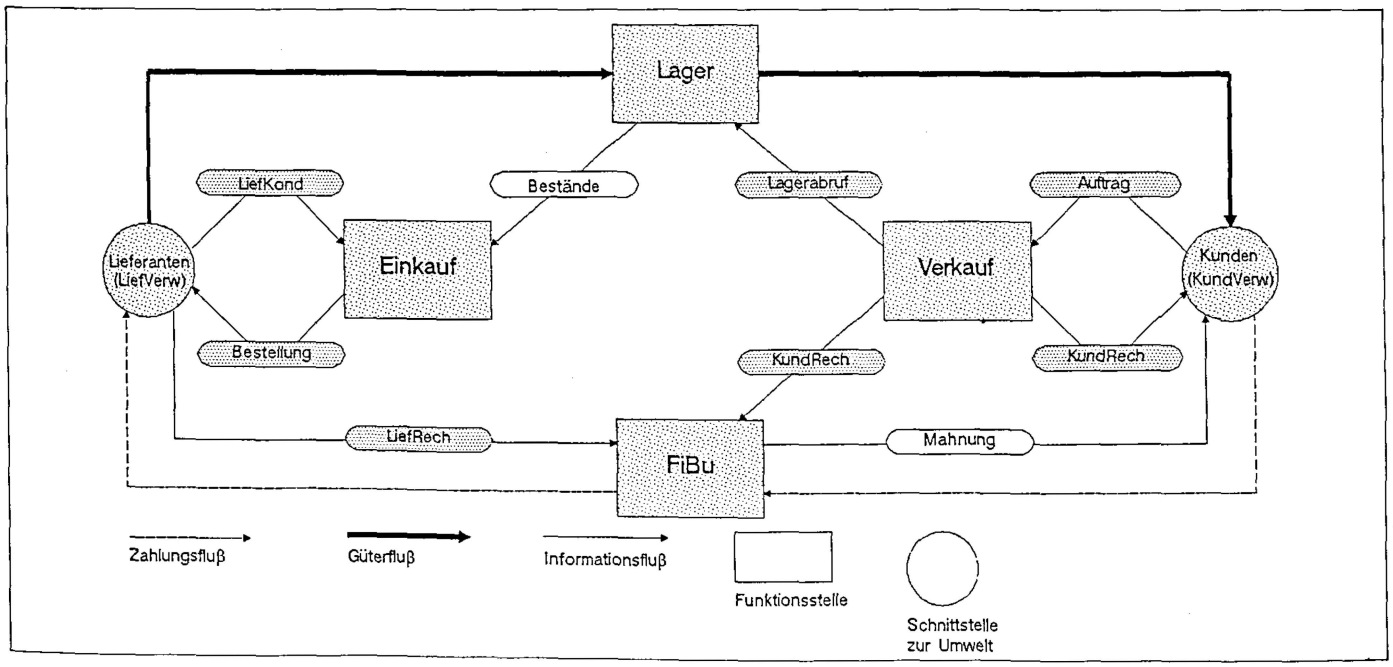

Bild 9 Interaktionsmodell des Handelsbetriebes

Bei der Zuordnung von SER- zu SOM-Beziehungen besteht eine Reihe von Freiheitsgraden (Bild 8). Die jeweils sinnvolle Zuordnung ist im Rahmen der Objektanalyse zu entscheiden. Zu Modellierungshinweisen siehe Kapitel 3.5.

3.4 Modellierung konzeptioneller Objektschemata im SOM

Die Modellierung im SOM wird nun anhand eines größeren Beispiels erläutert. Ausgangspunkt ist das Interaktionsmodell eines Handelsbetriebes (Bild 9) mit den betrieblichen Funktionsstellen *Lager, Verkauf, Einkauf* und Finanzbuchhaltung (*FiBu*) [5]. Kunden und Lieferanten bilden die Schnittstellen zur Umwelt; sie werden zu einer Kundenverwaltung (*KundVerw*) und einer Lieferantenverwaltung (*LiefVerw*) zusammengefaßt. Die Interaktionen zwischen den Funktionsstellen und/oder der Umwelt bestehen aus Zahlungs-, Güter- und Informationsflüssen.

Dieser Handelsbetrieb wird als konzeptionelles Objektschema im SOM modelliert (Bild 10), das die Grundlage für das objektorientierte Informationssystems dieses Handelsbetriebs bildet. Die Modellierung wird in zwei Schritten, dem Objektsystementwurf und dem Objektentwurf durchgeführt.

Objektsystementwurf

Beim Objektsystementwurf wird wie folgt vorgegangen:

- Funktionsstellen und Umweltschnittstellen des Interaktionsmodells werden auf E-Typen des konzeptionellen Objektschemas abgebildet und stellen dort die originären Klassen dar. Der Detaillierungsgrad des Interaktionsmodells determiniert somit den initialen Detaillierungsgrad des konzeptionellen Objektschemas. Die E-Typen sind unter Verwendung der *is_part_of*-Beziehung weiter verfeinerbar.
- Die Informationsflüsse des Interaktionsmodells werden im konzeptionellen Objektschema mit Hilfe der *interacts_with*-Beziehung in Verbindung mit ER- bzw. R-Typen modelliert. Informationsflüsse sind ebenfalls mit Hilfe der *is_part_of*-Beziehung weiter verfeinerbar.
- Jeder Gegenstands-Objekttyp (E- oder ER-Typ) kann unter Verwendung der *is_a*-Beziehung spe-

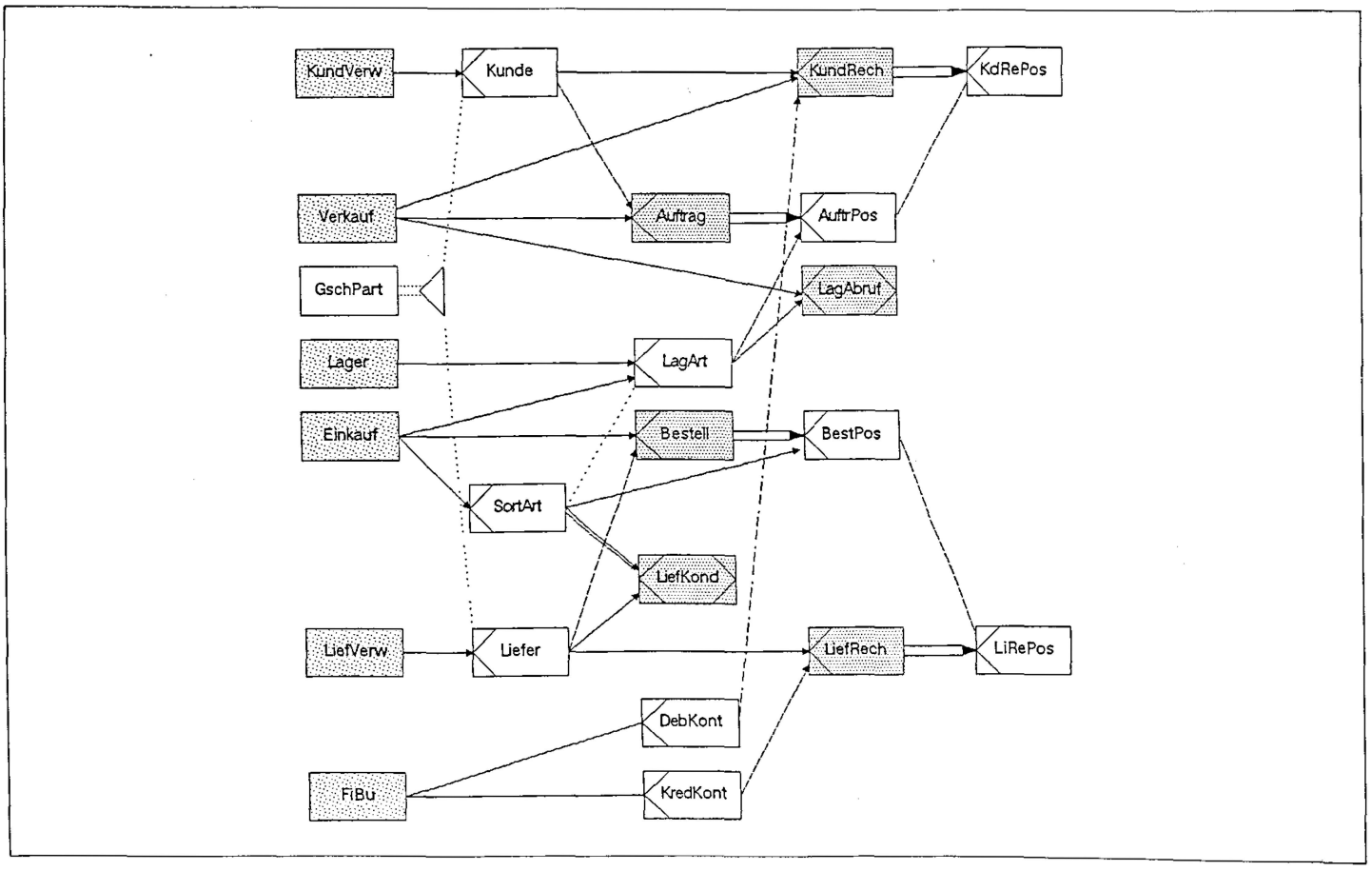

Bild 10 Konzeptionelles Objektschema (Objektsystementwurf) des Handelsbetriebes

zialisiert werden. Umgekehrt kann er bei Erkennen von Gemeinsamkeiten mit anderen Gegenstands-Objekttypen generalisiert werden.

Die Modellierung von Informationsflüssen wird nun anhand von *Auftrag* erläutert. Ein Auftrag ist ein Informationsfluß zwischen einem individuellen Kunden und der Funktionsstelle *Verkauf. KundVerw* wird daher unter Verwendung der *is_part_of*-Beziehung verfeinert. Dabei wird *Kunde* als Teilobjekttyp von *KundVerw* herausgelöst. Aus Datensicht ist diese Beziehung vom Typ (0, *). Der Informationsfluß selbst wird durch den Objekttyp *Auftrag* und die beiden *interacts_with*-Beziehungen zu *Kunde* und *Verkauf* modelliert. Aus Datensicht ist *Auftrag* ein ER-Typ, die beiden Beziehungen sind vom Typ (0, *).

Um den Objekttyp *Auftrag* vollständig beschreiben zu können, wird dieser mit Hilfe der *is_part_of*-Beziehung in *Auftrag* und *AuftrPos* zerlegt. Aus Datensicht entspricht dies einem Normalisierungsschritt. Da jeder Auftrag wenigstens eine Auftragsposition haben muß, ist die Beziehung vom Typ (1, *). Jede Auftragsposition bezieht sich außerdem auf genau einen Lagerartikel (*LagArt*). Diese Abhängigkeit wird durch eine weitere *interacts_with*-Beziehung modelliert, die im Interaktionsmodell (Bild 9) aufgrund des dort gewählten Detaillierungsgrades nicht sichtbar ist.

In den Bildern 9 und 10 sind korrespondierende Funktionsstellen bzw. Informationsflüsse entsprechend gekennzeichnet. Der Informationsfluß *Bestände* umfaßt Bestandsmeldungen an den Einkauf. Dazu hat *Einkauf* über eine *interacts_with*-Beziehung Zugang zu *LagArt*. Der Informationsfluß *Mahnung* führt zu keinem weiteren Objekttyp, sondern wird durch die *interacts_with*-Beziehung zwischen *DebKont, KundRech* und und *Kunde* modelliert.

is_a-Beziehungsstrukturen finden sich in Form der Generalisierung von *Kunde* und *Liefer* zu Geschäftspartner (*GschPart*) sowie der Spezialisierung von Sortimentsartikel (*SortArt*) in Lagerartikel (*LagArt*).

Objektentwurf

Im Objektentwurf wird die Struktur und das Verhalten eines jeden Objekttyps des konzeptionellen Objektschemas beschrieben. Gemäß dem gewählten Objektmodell (Abschnitt 2.2) werden dabei Attribute, Nachrichten und zugehörige Methoden festgelegt.

Die Zuordnung der Attribute erfolgt nach den Grundsätzen der konzeptionellen Datenmodellierung im SERM und wird daher nicht weiter erläutert. Bei den Methoden wird zwischen Konstruktoren und objektspezifischen (anwendungsspezifischen) Methoden unterschieden. Konstruktoren sind Klassenmethoden und dienen zur Instantiierung (*Create*) und Deinstantiierung (*Destroy*) von Objekten und werden für alle Objekttypen benötigt. Objektspezifische Methoden werden zum Teil erst während der Vorgangsmodellierung erkannt (Abschnitt 4).

Nachrichten enthalten in der Regel Argumente zur Parametrisierung der zugehörigen Methoden. Nachrichten an Konstruktoren enthalten als zusätzliches Argument die Anzahl der zu instantiierenden bzw. zu deinstantiierenden Objekte in Form eines Komplexitätsgrades in (min,max)-Notation (Abschnitt 2.1).

Nachrichten werden entlang von *is_part_of-* und *interacts_with*-Beziehungen ausgetauscht. *is_part_of*-Beziehungen beschreiben in Form von Aggregationen die Beziehung zwischen einem Objekt und seinen (mehrfach auftretenden) Teilobjekten. Beim Nachrichtenaustausch entlang einer *is_part_of*-Beziehung ist daher der Adressat der Nachricht bekannt und kann bei der Spezifikation einer Nachricht in der Methodendefinition explizit mitangegeben werden.

Beim Nachrichtenaustausch über *interacts_with*-Beziehungen wird dagegen das Konzept des Software-IC [3] konsequent verfolgt. Danach werden die Methoden eines Objekttyps unabhängig von dessen Einbettung in ein konzeptionelles Objektschema definiert. Innerhalb einer Methode werden nur Nachrichten, nicht aber die zugehörigen Adressaten spezifiziert. Diese werden erst beim „Schaltungsentwurf der Leiterplatte", d.h. beim Vorgangsentwurf festgelegt.

In Bild 11 ist ein Teil des Objektentwurfs für den Objekttyp *Auftrag* des Handelsbetriebs dargestellt. Es werden die Konstruktoren *Create* und *Destroy* sowie die Methode *Terminiere* spezifiziert. Aus Gründen der Vereinfachung wird nicht zwischen Nachrichten- und Methodenname unterschieden.

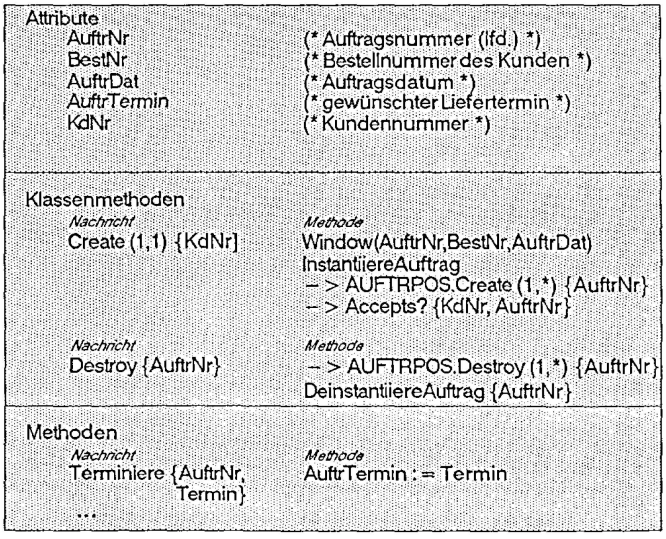

Bild 11 Objektentwurf für den konzeptionellen Objekttyp-Auftrag des Handelsbetriebes

3.5 Die Beziehung zwischen konzeptionellem Datenschema und konzeptionellem Objektschema

Unter syntaktischen Gesichtspunkten stellt ein konzeptionelles Datenschema eine Projektion des zugehörigen konzeptionellen Objektschemas in Bezug auf die Attribute dar. Aus diesem Grund gelten die Normalformen für Datenschemata unverändert als Kriterien für die Zerlegung von Objekttypen in Objektschemata.

Weitere Aussagen sind bezüglich der Semantik der Schemata möglich. Aus dem Datenschema ist eine funktionale (oder operationale) Semantik nur insoweit ableitbar, wie sie sich aus der (Wieder-) Erfüllung (verletzter) referentieller Integritätsbedingungen ergibt. In einem Objektschema erhalten die Schlüsselreferenzen eine zusätzliche Interpretation als *interacts_with-*, *is_a-* oder *is_part_of*-Beziehung. Diese zusätzliche Interpretation liegt dem Nachrichtenaustausch und der Methodendurchführung der Objekte zugrunde.

Subtypen- und Generalisierungshierarchien des Datenschemas werden im Objektschema stets mit *is_a*-Beziehungen gekoppelt. Während im Datenschema der Gesichtspunkt zusätzlicher Integritätsbedingungen im Vordergrund steht, dient die *is_a*-Hierarchie im Objektschema zusätzlich der Vermeidung von Funktionsredundanz. Im Objektschema sind daher auch (0, 1)-Beziehungen als *is_a*-Beziehungen interpretierbar.

Aus Sicht der Modellierungspraxis ist es interessant, die Ergebnisse einer Objektmodellierung mit denen einer entsprechenden Datenmodellierung zu vergleichen. Hier zeigt sich, daß das Objektschema in der Regel komplexer als das zugehörige Datenschema ist und letzteres, korrekte Modellierung vorausgesetzt, als Teilschema enthält. Hierfür gibt es zwei Gründe:

1. Die herkömmliche Informationsanalyse analysiert (statische) Beziehungen zwischen Datenobjekttypen und hat in der Regel kein Interaktionsmodell als Grundlage.
2. In dem durch Objektmodellierung gewonnenen Datenschema treten "Single Instance Classes" auf. Das sind in der Regel Funktionsstellen und Umweltschnittstellen, zu denen nur eine einzige Instanz existiert. Da in der Informationsanalyse ein Datenobjekttyp in der Regel durch Abstraktion einer Menge als gleichartig klassifizierter Objekte gefunden wird, werden Datenobjekttypen mit nur einer Instanz möglicherweise nicht entdeckt. Hier besitzt die Informationsanalyse einen „blinden Fleck".

Insgesamt zeigen die bisherigen Erfahrungen, daß die Objektmodellierung im SOM zu einer Stabilisierung und Erweiterung der Datenmodellierung im SERM führt.

4 Die Vorgangsebene des SOM

Wie in Abschnitt 3.4 behandelt, beschreibt das konzeptionelle Objektschema betriebliche Funktionsstellen und Informationsflüsse zwischen diesen. Auf der Vorgangsebene wird nun der betriebliche Ablauf untersucht, der aus dem Zusammenwirken betrieblicher Vorgänge resultiert.

4.1 Die Durchführung betrieblicher Aufgaben in Form von Vorgängen

Die Durchführung betrieblicher Aufgaben geschieht in Vorgängen und Vorgangsketten. Aus Sicht des Betriebsablaufes stellen Vorgänge die Elementarbausteine dar, aus denen Vorgangsketten, bzw. der gesamte Betriebsablauf gebildet werden. Vorgänge steuern und kontrollieren unteilbare betriebliche Geschäftsvorfälle und werden aus dieser Sicht nicht weiter zerlegt.

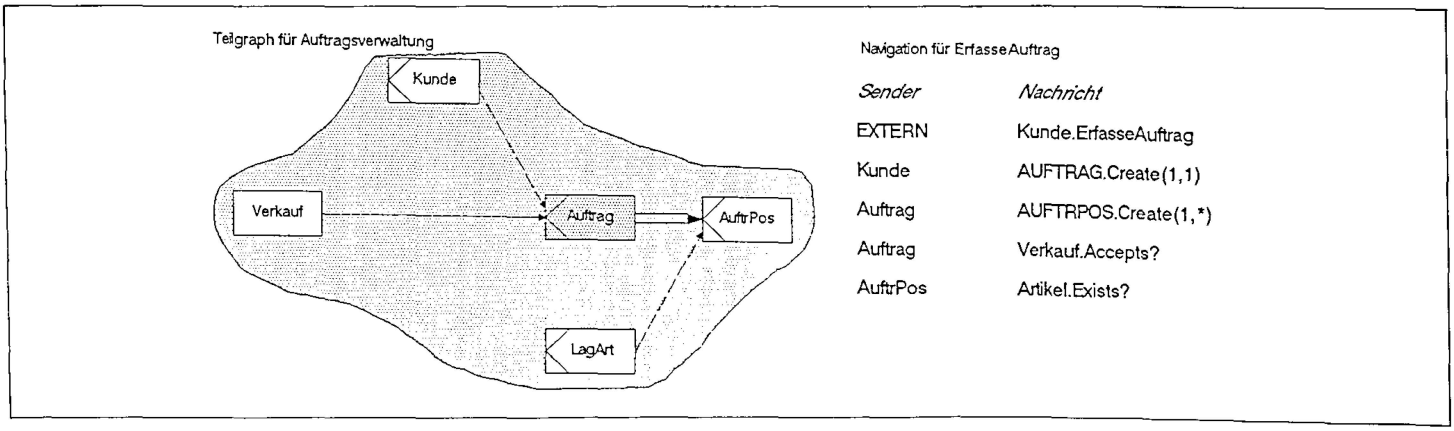

Bild 12 Vorgangsobjekttyp Auftragsverwaltung des Handelsbetriebs

Aus der Sicht der Vorgangsmodellierung ist aber eine sukzessive weitere Zerlegung betrieblicher Vorgänge erforderlich, bis jeder erzeugte Teilvorgang der Durchführung einer Methode eines konzeptionellen Objekts entspricht. Kriterium für die notwendige Zerlegungstiefe ist daher die Granularität des konzeptionellen Objektschemas.

Zur Klassifikation betrieblicher Vorgänge werden Ähnlichkeiten im Aufgabentyp und der Aufgabendurchführung herangezogen:

a) Beispiele für ähnliche Aufgabentypen sind *ErfasseAuftrag* und *StorniereAuftrag*, die beide dem Aufgabenkomplex *Auftragsverwaltung* zuzuordnen sind.
b) Beispiele für ähnliche Aufgabendurchführungen sind Varianten von *ErfasseAuftrag* in Abhängigkeit vom Kundentyp, von der Auftragsart, vom Beauftragungsweg und von den bestellten Artikeln.

Gemeinsame Grundlage von (a) und (b) ist ein bestimmter Teilgraph des konzeptionellen Objektschemas. Dieser grenzt eine Menge konzeptioneller Objekttypen sowie die zugehörigen Interaktionskanäle ab. Die unter (a) und (b) genannten Varianten können jedoch in unterschiedlichen Folgen von Methodendurchführungen resultieren. Bild 12 zeigt den Teilgraphen für den Aufgabenkomplex *Auftragsverwaltung* des Handelsbetriebs.

4.2 Modellierung von Vorgängen durch Vorgangsobjekttypen

Alle zu einem Aufgabenkomplex gehörigen Vorgänge werden im SOM durch einen Vorgangsobjekttyp modeliert. Ein Vorgangsobjekttyp steuert das Zusammenwirken mehrerer konzeptioneller Objekttypen zur Durchführung betrieblicher Aufgaben in Form von Vorgängen. Ein Vorgang stellt somit eine Instanz eines Vorgangsobjekttyps dar.

In Analogie zu den konzeptionellen Objekttypen werden auch Vorgangsobjekttypen durch Attribute, Nachrichten und Methoden definiert:

- Wesentliche Attribute eines Vorgangsobjekttyps sind der Teilgraph des konzeptionellen Objektschemas sowie Variablen zur Erfassung des Vorgangszustands.
- Nachrichten eines Vorgangsobjekttyps repräsentieren die einzelnen Aufgaben eines Aufgabenkomplexes. Im Beispiel der *Auftragsverwaltung* sind dies u.a. *ErfasseAuftrag, StorniereAuftrag* sowie deren Varianten.
- Jede Methode eines Vorgangsobjektyps beschreibt eine bestimmte Navigation durch den Teilgraph sowie die zugehörigen Operatoren und Prädikate. Die Navigation besteht in der Generierung von Nachrichten an konzeptionelle Objekte und in der Festlegung der Empfänger für die von konzeptionellen Objekten ausgehenden Nachrichten. Unter dem Blickwinkel „Software-IC-Konzepts"(Abschnitt 3.4) verbindet ein Vorgangsobjekttyp mehrere unabhängige konzeptionelle Objekttypen (Software-ICs) und realisiert somit die Schaltung der zugehörigen Leiterplatte. Bild 12 zeigt die Navigation für die Nachricht *ErfasseAuftrag* des Vorgangsobjekttyps *Auftragsverwaltung*.

Die Bildung von Vorgangsobjekttypen kann mehrstufig durchgeführt werden. Auf diese Weise können hierarchische Strukturen von Vorgangsketten modelliert werden. Für die Attribute, Nachrichten und Methoden des Vorgangsobjekttyps einer Vorgangskette gelten die gleichen Regeln wie für einzelne Vorgänge.

4.3 Realisierung von Vorgangsobjekttypen

Die Aufgabe eines Vorgangsobjekttyps besteht in der Generierung von Nachrichten an konzeptionelle Objekte und in der Festlegung der Empfänger für die von konzeptionellen Objekten ausgehenden Nachrichten. Für die Realisierung eines Vorgangsobjekttyps bestehen grundsätzlich die folgenden Möglichkeiten.

Feste Verdrahtung konzeptioneller Objekte

Der Vorgangsobjekttyp ist bereits im konzeptionellen Objektschema realisiert. Nachrichten an den Vorgangsobjekttyp entsprechen Nachrichten an einen Einstiegspunkt im Teilgraph des konzeptionellen Objektschemas. Alle Empfänger von Nachrichten sind in den Methoden der konzeptionellen Objekttypen festgelegt. Die Methodendurchführung erfolgt nach dem in Abschnitt 2.2 dargestellten Stufenkonzept.

Flexible Zuordnung konzeptioneller Objekte

Der Vorgangsobjekttyp wird getrennt vom konzeptionellen Objektschema realisiert. Jeder Vorgang wird durch eine Instanz des Vorgangsobjekttyps (Vorgangsobjekt) gesteuert. Diese Instanz empfängt die vorgangsauslösende Nachricht und wickelt den gesamten Nachrichtenaustausch zwischen den unabhängigen konzeptionellen Objekten (Software-ICs) ab. Eine Nachricht zwischen zwei konzeptionellen Objekten läuft dabei stets über das Vorgangsobjekt. Dadurch eröffnet sich die Möglichkeit einer flexiblen Bestimmung des Nachrichtenweges durch das Vorgangsobjekt.

5 Implementierungskonzept für konzeptionelle Objektschemata

Im Gegensatz zu herkömmlichen Modellen der Systementwicklung ermöglichen objektorientierte Ansätze strukturgleiche Systemkonzepte über alle Phasen des Software-Life-Cycle. Ein in der Definitionsphase erstelltes konzeptionelles Objektschema ist ohne Strukturbruch in einen Systementwurf und eine programmtechnische Realisierung umsetzbar. Im folgenden werden einige Aspekte eines solchen Implementierungskonzepts für konzeptionelle Objektschemata skizziert. Dabei sind vor allem Fragen

- der Software-Architektur objektorientierter Systeme,
- der dafür geeigneten Hardware- und Betriebssystem-Plattformen und
- der zugehörigen Software-Entwicklungsumgebungen

von Interesse. Diese Fragen werden derzeit parallel zur Erprobung der SOM-Methodik untersucht.

5.1 Struktur eines SOM-Software-Systems, Objektaußensicht

Entsprechend der Struktur konzeptioneller Objektschemata bildet ein gemäß SOM erstelltes Software-System eine quasi-hierarchische Struktur von Objekttypen. Die Nachrichten, mit denen die zugehörigen Objekte kommunizieren, werden entlang der *interacts_with-* und *is_part_of*-Kanten des Objektschemas transportiert. In der Objektaußensicht wird daher von einem Objekt nur gefordert, vereinbarte Nachrichtenformate empfangen, bearbeiten und versenden zu können.

Zur Reduzierung der Komplexität eines Software-Systems können unter Implementierungsgesichtspunkten Cluster von Objekttypen gebildet werden. Ein derartiger Cluster deckt einen zusammenhängenden Teilgraphen des Objektschemas ab. Natürliche Kriterien zur Clusterbildung sind Teilhierarchien von *is_a-* bzw. *is_part_of*-Beziehungen oder die Häufigkeit des Nachrichtenaustausches.

5.2 Struktur eines SOM-Objektes, Objektinnensicht

Im SOM sind die Objekte eines Software-Systems ausschließlich über ihren Nachrichtenaustausch gekoppelt. Dies erlaubt hohe Freiheitsgrade bei der Gestaltung der Objektinnensicht, d.h. der Implementierung der einzelnen Objektklassen. Zur Realisierung des geforderten Objektverhaltens kommen prinzipiell folgende Software-Paradigmen in Frage:

- Herkömmliche prozedurale Programmiersprachen, die Datenkapselung zur Bildung abstrakter Datentypen unterstützen. Beispiele sind Modula-2, mehrere Pascal-Dialekte sowie Ada. Sie sind hierfür wegen ihrer mangelnden Objektorientierung nur bedingt geeignet.
- Objektorientierte Erweiterungen herkömmlicher prozeduraler Programmiersprachen wie C++ oder objektorientierte Pascal-Dialekte. Sie bieten zusätzlich den Vorteil der Kompatibilität mit bestehenden Software-Bibliotheken und Systemschnittstellen.
- Objektorientierte Systeme wie Smalltalk und Eiffel. Von Nachteil ist derzeit die begrenzte Verfügbarkeit dieser Systeme für gängige Hardware-Plattformen sowie der Mangel an kommerziellen Software-Entwicklungsumgebungen.
- Objektorientierte Datenbanksysteme (ooDBS), die im Gegensatz zu herkömmlichen DBS Datenobjekte zusammen mit ihren Methoden verwalten. Diese Systeme befinden sich derzeit noch im Forschungs- und Prototypenstand [8], [4], [9].
- Wissensbasierte Systeme. Dabei unterstützt die Trennung von Wissensbasis und Inferenzmaschine eine deklarative und damit äußerst flexible Beschreibung von Methoden.

5.3 Hardware- und Betriebssystem-Plattformen

Aus Durchführungssicht bildet ein Software-System gemäß SOM eine Menge gekoppelter, asynchron ablaufender Objekte. Die Objekte sind grundsätzlich parallel ausführbar; die zugehörigen Synchronisationsbedingungen ergeben sich aus dem Nachrichtenaustausch.

Wegen dieser Architektureigenschaften ist ein Software-System gemäß SOM auf unterschiedliche Hardware-Plattformen abbildbar. Neben einer Einrechnerlösung mit streng sequentieller Objektausführung sind eng gekoppelte Mehrprozessorlösungen oder lose gekoppelte Rechnerverbundlösungen geeignet.

5.4 Geplante Werkzeugunterstützung der Modellierung

Der Objektsystementwurf, der Objektentwurf und der Vorgangsentwurf ist ohne Werkzeugunterstützung nicht sinnvoll durchführbar. Parallel zur Erprobung der SOM-Methodik werden derzeit prototypische Werkzeuge entwickelt. Basis hierfür ist ein existierendes, wissensbasiertes Werkzeug zur Unterstützung der konzeptionellen und externen Datenmodellierung [11].

Literatur

[1] *Chen P. P.-S.:* The Entity-Relationship Model - Toward a Unified View of Data. In: ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, 9 - 36

[2] *Coad P., Yourdon E.:* Object-Oriented Analysis. Prentice Hall, New Jersey 1990

[3] *Cox B.:* Object-Oriented Programming. An Evolutionary Approach. Addison-Wesley, Reading, Massachusetts 1986

[4] *Dittrich K. R.:* Object-Oriented Database Systems: The next Miles of the Marathon. In: Information Systems, Vol. 15, No. 1, 161 - 167, 1990

[5] *Ferstl O. K., Sinz E. J.:* Software-Konzepte der Wirtschaftsinformatik. DeGruyter, Berlin 1984

[6] *Goldberg A.:* Smalltalk-80: The Interactive Programming Environment. Addison-Wesley, Reading, Massachusetts 1985

[7] *Goldberg A., Robson D.:* Smalltalk-80: The Language and its Implementation. Addison-Wesley, Reading, Massachusetts 1983

[8] IEEE Transactions on Knowledge and Data Engineering. Special Issue on Database Prototype Systems. March 1990

[9] *Kim W., Lochovsky F. H.* (ed.): Object-Oriented Concepts, Databases, and Applications. Addison-Wesley, Reading, Massachusetts 1989

[10] *Meyer B.:* Object-Oriented Software Construction. Prentice Hall, New Jersey 1988

[11] *Sinz E. J.:* Datenmodellierung betrieblicher Probleme und ihre Unterstützung durch ein wissensbasiertes Entwicklungssystem. Habilitationsschrift, Regensburg 1987

[12] *Sinz E. J.:* Das Strukturierte Entity-Relationship-Modell (SER-Modell). In: Angewandte Informatik 5/1988, 191 - 202

[13] *Sinz E. J.:* Das Entity-Relationship-Modell (ERM) und seine Erweiterungen. In: Handbuch der modernen Datenverarbeitung (HMD) 152, Verlag Forkel, Wiesbaden 1990