Hierarchische Zerlegung in objektorientierten Spezifikationsmodellen

Stefan Joos, Stefan Berner, Martin Arnold, Martin Glinz
Institut für Informatik der Universität Zürich
CH-8057 Zürich
{sjoos, berner, arnold, glinz}@ifi.unizh.ch

Zusammenfassung

Dieser Beitrag zeigt Schwachstellen von bestehenden objektorientierten Methoden bei der Dekomposition von Spezifikationen auf. Ausgehend von einer Präzisierung des bisher verwendeten Klassifikationskonzeptes wird ein Ansatz vorgestellt, welcher frei von diesen Schwachstellen ist. Dieser Ansatz baut auf ein Modellierungskonzept auf, welches Anforderungen auf der Ebene von Objekten hierarchisch modelliert und so die Schwächen der bisherigen, eher klassenorientierten Methoden überwindet.

1 Einleitung

Objektorientierte Methoden für die Spezifikation von Software haben in den letzten Jahren zunehmend an Bedeutung gewonnen. Objektorientierte Konzepte stehen im Ruf, Anforderungen wirklichkeitsnah und verständlich zu beschreiben. Zudem erlauben sie, Anforderungsmodelle einfach in objektorientierte Systemarchitekturen und Programme umzusetzen. Objektorientierte Konzepte besitzen zudem das Potential, auch komplexe und umfangreiche Spezifikationen geeignet zu beschreiben. Die existierenden objektorientierten Methoden schöpfen jedoch dieses Potential bis heute nicht aus. Anhand von Lehrbuchbeispielen wurde zwar die Verständlichkeit und die Realisierbarkeit aufgezeigt. Bei umfangreicheren Problemen jedoch reichen diese Methoden nicht aus, da deren Systemmodelle mit zunehmender Größe unüberschaubar und dadurch unverständlich werden.

Dieses Defizit wird in erster Linie durch den Mangel an brauchbaren Konzepten zur Dekomposition von Systemmodellen verursacht. Mächtige Dekompositionskonzepte aus anderen Ansätzen, wie etwa die Aktivitätenzerlegung in der Strukturierten Analyse [DeMar78], gibt es derzeit für objektorientierte Methoden nicht. Die einzelnen Methoden bieten zwar diverse Strukturierungskonstrukte an, allein jedoch die Generalisierung wird konsequent in Form von typbezogenen Hierarchien umgesetzt. Alle weiteren Konstrukte, im speziellen diejenigen zur Dekomposition von Systemen, sind wenig durchdacht und meist nur Stückwerk.

Damit auch große Spezifikationen beherrscht werden können, ist ein Zerlegungsmechanismus notwendig, der folgende Forderungen erfüllt:

(i) Ein System ist so in Teile zu gliedern, daß jedes Teil lokal mit möglichst wenig Bezug zu anderen Teilen verständlich ist.

(ii) Aufgrund einer Gliederung in Teile ist eine vergröberte Darstellung abzuleiten, so daß ein grobes Verständnis des Gesamtsystems oder größerer Teilsysteme möglich wird, ohne daß die Detailstrukturen bekannt sein müssen.

(iii) Den Teilen des Systems ist ein bestimmtes Leistungsangebot zuzuordnen, ohne offenzulegen, wie diese Teile dieses Leistungsangebot realisieren. Es sollte explizit möglich sein, zu verbergen, ob und welche anderen Teile zur Erfüllung dieses Leistungsangebots herangezogen werden.

Die Begriffe Dekomposition und Zerlegung werden hierbei und im folgenden als Synonyme verwendet. Die Forderungen (i) - (iii) erinnern an das Geheimnisprinzip von Parnas [Parna72] und die daraus hervorgehenden Forderungen an Modulkonzepte für Programmiersprachen. In objektorientierten Spezifikationen sind analoge Dekompositionskonzepte wünschenswert. Voraussetzung dafür ist das Vorhandensein einer geeigneten Abstraktion und die durchgängige Umsetzung dieser Abstraktion in entsprechende Dekompositionsmechanismen.

In diesem Beitrag wird gezeigt, daß eine hierarchische Dekomposition mit den klassenorientierten Modellen der heutigen objektorientierten Methoden prinzipielle Schwierigkeiten bereitet. Wir präsentieren daher ein neues Modellkonzept, das Klassen als reine Typen auffaßt und alle nicht typbezogenen Aspekte mittels abstrakter Objekte (wir nennen sie *prototypische Objekte* bzw. *prototypische Objektmengen*) beschreibt. Die nicht typbezogenen Aspekte sind auf den Objektkontext bezogene Aspekte, insbesondere alle Beziehungen außer der Generalisierung, die Objektkommunikation sowie Teile des Systemverhaltens. Für solche Objektmodelle kann nun mit Hilfe einer Teil/Ganzes-Hierarchie eine brauchbare hierarchische Dekomposition definiert werden, welche die bei der Klassendekomposition auftretenden Modellierungsprobleme löst. Unser Ansatz benötigt kein spezielles Behälter- oder Teilsystemkonstrukt, sondern interpretiert eine Komposition von Objekten wieder als Objekt. Dadurch können Objekte über mehrere Stufen in Gruppen von Komponentenobjekten zerlegt werden, was eine einfache und leicht zu verstehende Dekompositionshierarchie ergibt. Die Modellierung von Verhalten und Funktionalität wird in diesem Beitrag nur am Rande erwähnt. Über diesen Beitrag hinaus beschäftigen wir uns mit einem Ansatz, der sich in Anlehnung an [Glinz93] mit der hierarchischen Beschreibung des Verhaltens und der Funktionalität von Systemen beschäftigt, indem Objekte als eine Art Makrozustände in einer Hierarchie von Statecharts interpretiert werden.

Der Beitrag ist wie folgt gegliedert: Im zweiten Kapitel werden Defizite objektorientierter Spezifikationsmethoden anhand eines Beispiels demonstriert. Im dritten Kapitel wird gezeigt, an welchen Stellen objektorientierte Grundlagen und Konzepte ergänzt und präzisiert werden müssen, um hierarchische Objektmodelle zu ermöglichen. Im letzten Kapitel schließlich wird ein Ansatz für eine objektorientierte hierarchische Anforderungsmodellierung auf der Grundlage einer Teil/Ganzes-Abstraktion vorgestellt.

2 Dekomposition in objektorientierten Spezifikationen

Trotz ihrer großen Bedeutung haben Dekompositionskonstrukte in den bestehenden objektorientierten Spezifikationsmethoden einen sehr unterschiedlichen Stellenwert. Die Spannbreite geht von keinerlei (z.B. OOA96 [Schlae96]) über wenig (z.B. OOA&D [Coad91]) bis zu seriöser Beschäftigung mit der Problematik (z.B. RDD [Wirfs90]).

Alle Methoden, in denen es Dekomposition gibt, verwenden prinzipiell den gleichen Ansatz: Die Klassen des Klassenmodells werden in Teilsysteme gegliedert. Die Klassen werden fallweise eher intensional (als Typen) oder extensional (als Mengen von Objekten) interpretiert, wobei die intensionale Interpretation überwiegt (siehe Kapitel 3, Präzisierung der objektorientierten Klassifikation). Die Bedeutung der Teilsysteme wird nicht oder nur ansatzweise geklärt.

Probleme bei der Dekomposition

Bei der Dekomposition auf der Basis von Klassenmodellen entsteht eine Reihe von Problemen, die sich drei Bereichen zuordnen lassen:

- *Problem der Überlappung*
  Eine Klasse tritt in mehr als einem Teilsystem auf. Bei intensionaler Betrachtung der Klassen erfordert das überlappende Teilsysteme, bei extensionaler Betrachtung ergibt sich eine ungeklärte Bedeutung: Eine eindeutige Zuordnung der Objekte einer Klasse zu den verschiedenen Teilsystemen ist hier nicht mehr möglich. Beides verträgt sich schlecht mit *Forderung (i)* (aus Kapitel 1) nach lokaler Verständlichkeit der Teile.
- *Problem des Zusammenhangs von Teilsystemen*
  Die Zusammenhänge (z. B. Beziehungen, Nachrichtenflüsse) zwischen Klassen verschiedener Teilsysteme müssen geeignet auf Zusammenhänge zwischen den Teilsystemen abgebildet werden. Andernfalls sind die *Forderungen (ii)* und *(iii)* nicht erfüllbar.
- *Problem der Ausdrucksschwäche von Klassenmodellen*
  Die Tatsache, daß mit einem Klassenmodell (d.h. einem Typen- oder Objektmengenmodell) gearbeitet wird, hat eine grundsätzliche Ausdrucksschwäche zur Folge: Sie verhindert überall dort, wo auf unterscheidbare (d.h. in unterschiedlichen Rollen auftretende) Objekte einer Klasse Bezug genommen werden muß, eine präzise Modellierung.

Im folgenden erläutern wir zunächst an einem Beispiel, wo diese Probleme auftreten und zeigen dann, wie existierende Methoden damit umgehen.

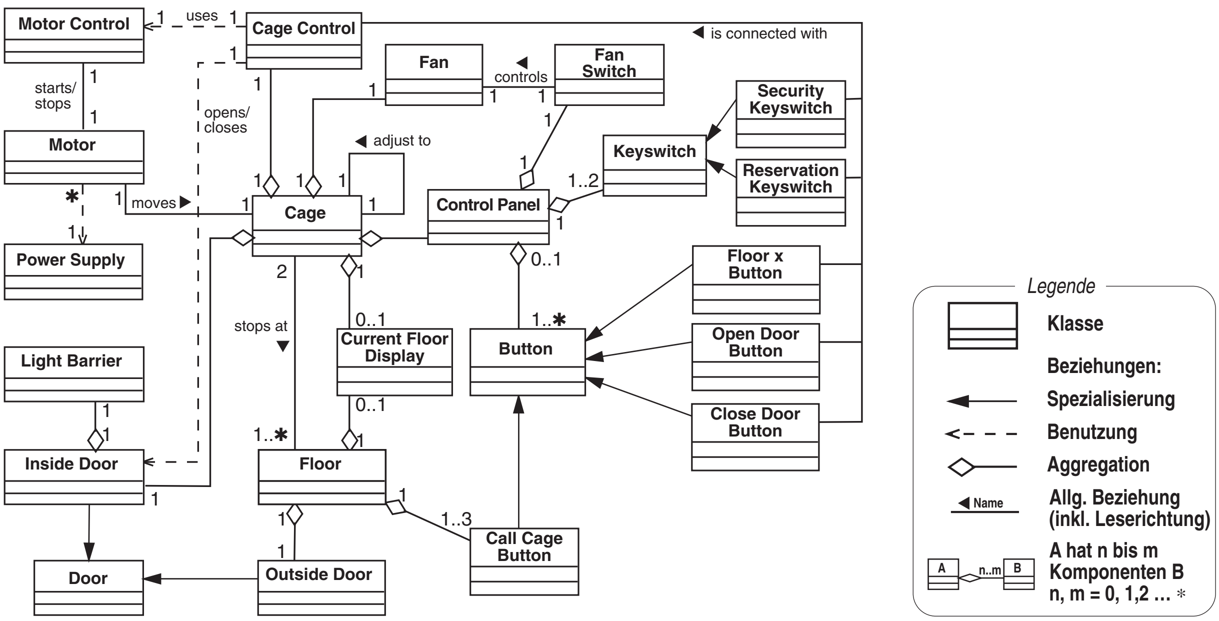

Abb. 1: Klassenmodell eines Doppelfahrstuhls (*UML091*-Notation)

Beispiel Szenario «Doppelfahrstuhl»

Ein Doppelfahrstuhl ist ein Fahrstuhl mit zwei Schächten und je einer Kabine in jedem Schacht. Der Fahrstuhl erstreckt sich über mehrere Stockwerke. Auf jedem Stockwerk besteht die Möglichkeit, eine Kabine mittels eines Rufknopfes anzufordern. Das Beförderungsverhalten beider Kabinen soll in geeigneter Art und Weise aufeinander abgestimmt sein. Neben einer Stockwerksanzeige in den Kabinen und auf jedem Stockwerk befinden sich in jeder Kabine ein Bedienfeld und ein Ventilator.

Ein risikobehafteter Teil der Spezifikation ist der Zustand der Kabinen- und Außentür während des Betriebes. Ein weiterer zentraler Bestandteil einer Spezifikation ist die Synchronisation der beiden Kabinen.

Das Klassenmodell dazu wird in Abb. 1 beschrieben (dargestellt in der Notation der *Unified Modelling Language v0.91*, kurz *UML091*, [Booch96]). Da Attribute und Operationen für das Beispiel nicht relevant sind, werden diese nicht weiter aufgeführt. Das Modell in Abb. 1 wird nun in Teilsysteme zerlegt. Dies wird mit den in der *UML091* vorgeschlagenen Categories[1] durchgeführt. Als Teilsysteme werden gewählt: Kabine, Stockwerk, Innentür und Antrieb (siehe Abb. 2).

[1] Eine Category ist nach [Booch96, UML Semantics Glossary] “eine Teil/Ganze-Struktur mit Kopplung der Teile-Lebensdauer an die Category“ ohne Angabe einer zugrundeliegenden Bedeutung

Problem der Überlappung

Die Klassen Current Floor Display, Button und Door (Abb. 2) werden für die Beschreibung zweier Teilsysteme benötigt. Bei Button und Door handelt es sich um Typzusammenhänge (in Form von Spezialisierungen), bei Current Floor Display werden Objekte derselben Klasse an mehreren Stellen des Modells verwendet. Dieses Problem ließe sich durch Einführung zusätzlicher Klassendefinitionen scheinbar lösen, führt aber gleichzeitig bei häufiger Verwendung zu einer drastischen Verkomplizierung des Klassenmodells .

Problematisch im ersten Fall ist, ob und wie sich Typzusammenhänge auf der Ebene von Teilsystemen ausdrücken lassen. Im zweiten Fall stellt sich das Problem, daß nur Aussagen über sämtliche Objekte einer Klasse getroffen werden können. Der Sachverhalt, daß verschiedene Objekte der gleichen Klasse unterschiedliche Rollen in verschiedenen Kontexten annehmen, kann nicht berücksichtigt werden. Es ist daher auch nicht möglich, darzustellen, daß Objekte der Klasse Current Floor Display einerseits die Rolle der Anzeige auf Stockwerken, andererseits die Anzeige in Kabinen übernehmen können.

Problem des Zusammenhangs von Teilsystemen

Will man erreichen, daß aus den gefundenen Teilsystemen eine vergröberte Darstellung abgeleitet werden kann, müssen statische und dynamische Zusammenhänge zwischen den Teilsystemen modelliert werden. Ein statischer Zusammenhang beschreibt eine strukturelle Beziehung (Relation), ein dynamischer Zusammenhang ist ein Methodenaufruf, Ereignis-, Nachrichtenfluß o. ä. Grundsätzliche Forderung dabei ist, daß

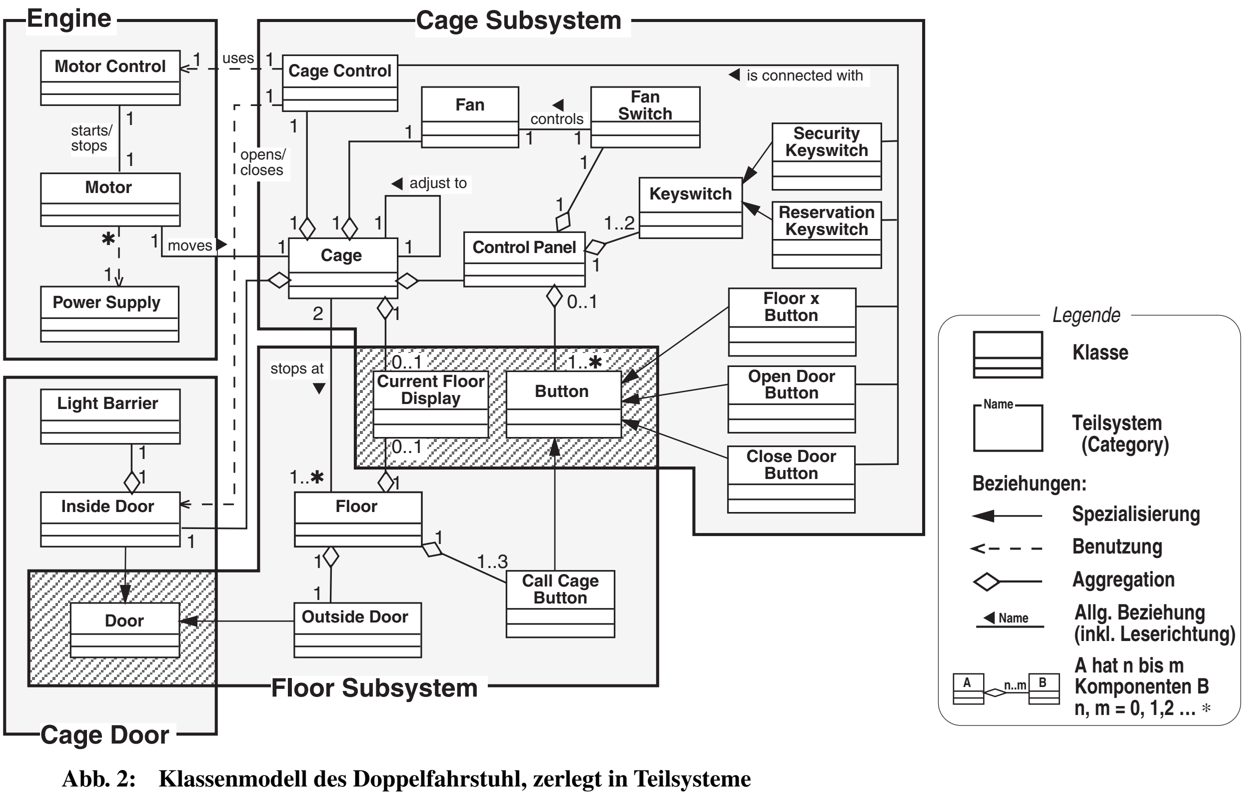

Abb. 2: Klassenmodell des Doppelfahrstuhl, zerlegt in Teilsysteme

- sich ein Zusammenhang auf Detailebene (hier Klassenebene) auf Teilsystemebene wiederfinden muß
- Zusammenhänge zwischen den abstrakten Teilsystemen Sinn machen; daß also nicht nur die Teilsysteme selber, sondern auch die Zusammenhänge zwischen ihnen eine Abstraktion der Detailebene sind.

Eine objektorientierte Methode müßte nun beschreiben können, welche Beziehung zwischen den abstrakten Teilsystem-Zusammenhängen und den vorhandenen Zusammenhängen auf Detailebene besteht. Zusätzlich muß geklärt sein, ob und welche Integritätsbedingungen für die Kardinalitäten der beteiligten Beziehungen gelten.

Für das Fahrstuhl-Beispiel wäre folgende Teilsystem-Darstellung vorstellbar:

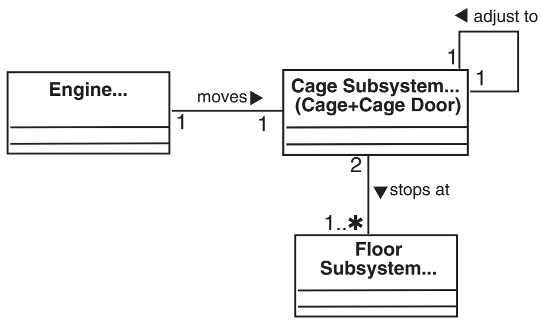

Abb. 3: Teilsysteme als grobe Umschreibung des Doppelfahrstuhlsystems

Im Fahrstuhl-Beispiel (Abb. 2 und 3) etwa müßte beschrieben werden, wie die Teilsystem-Beziehung moves mit der Beziehung uses zwischen Motor Control und Cage Control und der Beziehung moves zwischen Motor und Cage zusammenhängt. Weiterhin dürfen die Aggregationsbeziehungen zwischen Cage und Current Floor Display, zwischen Floor und Current Floor Display sowie zwischen Control Panel und Button nicht in die abstraktere Beschreibung mit einfließen, da dadurch kein semantischer Zusammenhang zwischen den Teilsystemen Cage Subsystem und Floor Subsystem ausgedrückt wird.

Eine systematische Behandlung solcher Teilsystem-Beziehungen und ihrer Zusammenhänge zu den Beziehungen zwischen den Klassen in den Teilsystemen wird durch die Überlappung von Teilsystemen massiv erschwert.

Problem der Ausdrucksschwäche von Klassenmodellen

Die oben vorgestellten objektorientierten Spezifikationsmethoden wie auch der Entity-Relationship-Ansatz [Chen76] bauen auf der Grundidee auf, einen Problembereich mit Hilfe von Typen (z. B.: Gegenstandstypen, Objekttypen, Klassen) zu beschreiben. Die alleinige Modellierung durch Typen ist jedoch zu abstrakt und führt zu unpräzisen Modellen. In Abb. 2 beispielsweise kann die Aggregationsbeziehung zwischen Current Floor Display und Cage durch folgende Aussage beschrieben werden:

- Eine Stockwerksanzeige kann in 0..1 Kabinen bzw. in 0..1 Stockwerken aufgeführt sein.

Die bedingte 0..1 Aussage ist unscharf, weil sie nur für die gesamte Klasse Current Floor Display gemacht wird, nicht aber für einzelne Objekte. Zudem ist das Modell dadurch unterspezifiziert: Es ist nicht klar, ob ein Objekt gleichzeitig in einer Kabine *und* in einem Stockwerk verwendet werden kann oder ob dazu zwei Objekte unterschiedlicher Identität nötig sind. Ähnliche Probleme ergeben sich bei der Klasse Cage, bei der nicht ausgeschlossen werden kann, daß ein Objekt der Klasse mit sich selber in adjust to-Beziehung steht. Abb. 4 zeigt, wie im Fahrstuhlbeispiel eine präzisere Modellierung möglich ist, wenn mit *Objekten* anstelle von Klassen modelliert wird:

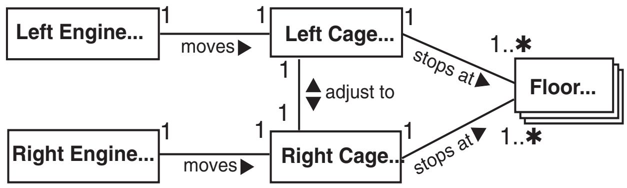

Abb. 4: Aussagekräftigere Modelle durch Modellierung auf Objektebene

Auf Ebene der Teilsysteme wird der Doppelaufzug durch zwei explizit aufgeführte Kabinen beschrieben, nämlich einer linken und einer rechten Kabine. Diese wiederum werden jeweils durch einen Antrieb bewegt, ebenfalls durch einen linken und einen rechten. Die Stockwerke werden durch eine Menge von Objekten beschrieben. Aus dieser Beschreibung ist zum einen die präzise Kooperationsbeziehung zwischen den beiden Kabinen ersichtlich und zum anderen, welche Objekte in welcher Kardinalität verwendet werden.

Behandlung der Probleme in ausgewählten Spezifikationsmethoden

a) Überlappung und Zusammenhang von Teilsystemen

Verschiedene existierende objektorientierte Spezifikationsmethoden ignorieren die Dekomposition bzw. die damit verbundenen Probleme. Zu dieser Gruppe gehören zum Beispiel OOA96 [Shlae96], OOA&D [Coad91], OOSE [Jacob94] und OMT [Rumba91]. Die drei letztgenannten Methoden ermöglichen immerhin die Zusammenfassung von Objekten zu Sachgebieten oder Teilsystemen, kümmern sich aber nicht weiter um deren Bedeutung. Sie versagen daher insbesondere bei *Forderung (ii)* und *(iii) (siehe Kapitel 1).* Die Geringschätzung der Dekomposition dürfte zu einem erheblichen Teil darauf zurückzuführen sein, daß viele objektorientierte Methoden aus der Entity-Relationship-Modellierung hervorgegangen sind ([Chen76] u. a.), welche ebenfalls keine Dekomposition kennt.

Die Teil/Ganzes-Beziehung, welche unter diesem Namen oder in Form eines Aggregationskonstrukts in fast allen Methoden verfügbar ist, ist kein für unsere Zwecke brauchbares Dekompositionskonstrukt, weil die Teile auf der gleichen Abstraktionsebene modelliert werden wie das Ganze. Dies zeigt sich deutlich daran, daß keine Methode Sichten anbietet, die es gestatten, ein einzelnes Teil einer solchen Ganzes-Teil-Struktur für sich zu betrachten oder in einer vergröberten Sicht alle Teile wegzulassen und nur die Aggregate mit ihren Zusammenhängen untereinander zu betrachten.

Lediglich die *UML091*, *OOAD* [Booch94], *OSA* [Emble92] und *RDD* [Wirfs90] gehen wirklich auf die Dekompositionsproblematik ein. Keine der genannten Methoden löst das Problem jedoch befriedigend:

➢ Die *UML091* bietet zwei Dekompositionskonstrukte für Objektmodelle an: Die bereits vorgestellten *Categories* [Booch96, p. 5 und p. 10] und *Composites* [Booch96, p. 5 und p. 11]. Um das Problem mit der Mehrfachverwendung von Klassen zu lösen, schlagen Booch und Rumbaugh vor, mehrfachverwendete Klassen einer *Owner-Category* zuzuordnen und in allen weiteren *Categories* diese Klasse mit einem Verweis auf die Owner-Category aufzuführen.
Dabei wird offen gelassen, welche *Categories* geeignete *Owner-Categories* sind und wie widersprüchliche oder überschneidende Aussagen über ein und dieselbe Klasse in verschiedenen *Categories* aufzulösen sind. Auch wird nicht erläutert, wie Klassenbeziehungen auf Beziehungen zwischen *Categories* abgebildet werden. *Composites* unterscheiden sich von *Categories* dahingehend, daß sie durch Klassen und durch Objekte beschrieben werden. Unklar dabei ist, welches Klassen- und Objektverständnis diesem Ansatz zugrundeliegt, wenn Klassen und Objekte auf einer logischen Ebene aufgeführt werden. Das in der *UML10* [2][Unified Modeling Language v1.0, Booch97] vorgestellte Konstrukt *Package* eignet sich ebenfalls nicht zur Systemdekomposition, da weder Aussagen über die Bedeutung des Konstruktes selber gemacht, noch der Zusammenhang zu den enthaltenen Komponenten geklärt werden.

➢ *OSA* beschäftigt sich vor allem mit der Frage, wie Klassenbeziehungen im *High-Level Model* [Emble92, p. 78 ff.] dargestellt werden. Dabei wird angestrebt, vorhandene Beziehungen zwischen Klassen verschiedener Teilsysteme automatisch auf diese Teilsysteme zu übertragen. Erfolgreich ist dies aber nur, wenn auf Detailebene bereits eine Klasse vorhanden ist, die auch das gesamte Teilsystem umschreibt (z. B. die Klasse `Cage` in den Abb. 3 und 4) und es auf Detailebene nur Beziehungen zu dieser Klasse gibt, nicht aber zu Komponenten anderer Teilsysteme.

➢ *OOAD´*s Ansatz der *Class Categories* [Booch94, p. 181 ff.] ähnelt schließlich dem der *Composites* in der *UML091*. Im Gegensatz zu den *Composites* werden Beziehungen zwischen Klassen verschiedener *Class Categories* in Benutzungszusammenhänge umgeformt: So werden sowohl Klassen-Klassen-Benutzungen als auch Spezialisierungen und Aggregationen in Benutzungsbeziehungen transformiert. Die Bedeutung der *Class Category*-Benutzungsbeziehung ist daher nicht präzise faßbar und somit wenig aussagekräftig.

➢ *RDD* stellt ein Teilsystem-Konstrukt namens *Subsystem* vor, dem mit Hilfe von *Verträgen* [Wirfs90, S. 63] ein bestimmtes Leistungsangebot zugeordnet wird. Das Problem des Zusammenhangs von Teilsystemen wird dadurch umgangen, daß in Anlehnung an einen Client-Server-Ansatz ausschließlich Benutzungsbeziehungen modelliert werden, die sich ohne Probleme auf gröbere Teilsysteme übertragen lassen. Die Problematik der Überlappung von Teilsystemen wird dadurch umgangen, daß eine Mehrfachaufführung einer Klasse schlichtweg verboten wird [Wirfs90, p. 137]. Dies hat zur Folge, daß mehrfach aufzuführende Klassen durch umständliche Hilfskonstruktionen modelliert werden müssen: Zu jeder mehrfach aufzuführenden Klasse *K* müssen zusätzliche mit *K* typgleiche Klassen als Spezialisierungen definiert werden. Dies führt zu einer unnötigen Verkomplizierung der Modelle.

Zusammenfassend kann man feststellen: Die Probleme der Überlappung und des Zusammenhangs von Teilsystemen werden zwar in den oben aufgeführten Methoden erkannt und angegangen. Die gewählten Lösungen sind jedoch unbefriedigend.

b) Ausdrucksschwäche von Klassenmodellen

Dieses Problem wird in allen uns bekannten Methoden ignoriert.

Anzumerken ist noch, daß die von einigen Autoren (z. B. *[Booch94]*, *[Rumba91]*, *[Booch96]*) verwendeten Objektdiagramme nicht geeignet sind, um obige Probleme zu beheben. Ein Objektdiagramm beschreibt für jeweils einen speziellen Handlungsablauf, welche Objekte beteiligt sind und welche Objektinteraktionen gelten. Objektdiagramme erlauben nur punktuelle Aussagen und eignen sich daher nicht für die Dekomposition ganzer Systeme.

3 Ansatz zur hierarchischen Modellierung von Anforderungen

Um die im letzten Kapitel angesprochenen Probleme zu beheben, ist zunächst eine Präzisierung des Klassifikationskonzeptes nötig. In diesem Kapitel wird diese Präzisierung vorgenommen. Anschließend wird darauf aufbauend ein Ansatz vorgestellt, der Lösungen für die angesprochenen Probleme liefert und damit eine hierarchische objektorientierte Modellierung von Spezifikationen erlaubt.

Präzisierung der objektorientierten Klassifikation

Die bisher aufgeführten objektorientierten Methoden basieren auf dem Grundsatz, Daten- oder Objektmodelle durch *Klassen* und Beziehungen zwischen *Klassen* zu beschreiben. Die eigentliche Problematik dabei ist, daß

[2] der Nachfolger der *UML091*

eine dualistische Bedeutung zugrundegelegt wird, nämlich eine *intensionale* und eine *extensionale*, und daß diese innerhalb eines Modells *gleichzeitig* verwendet werden:

- Die *intensionale* Bedeutung sieht eine *Klasse* als einen Typ einer unbestimmten Menge von Objekten. Intensionsbezogene Aussagen sind unter anderem Spezialisierungs-, Generalisierungs- und Subtypenzusammenhänge oder die Beschreibung der internen Struktur der zugeordneten Objekte (z.B. Attribute oder Objekte anderer Klassen). Eine ebenfalls intensionale Aussage aus funktionaler Sicht sind Angaben über das Leistungsangebot, welche von Objekten einer *Klasse* zur Verfügung stellen.
- Die *extensionale* Bedeutung sieht eine Klasse als Menge von Objekten desselben Typs. Extensionale Aussagen über Objekte sind kontextbezogen und beziehen sich auf einzelne Objekte oder auf Teilmengen von Objekten eines Typs. Beispiele hierfür sind allgemeine Beziehungen oder aus funktionaler Sicht Benutzungsbeziehungen.

Die Konsequenzen dieser Bedeutungsdualität sind die im letzten Kapitel aufgeführten Probleme: Überschneidungen von Teilsystemen treten nur deshalb auf, weil einerseits Klassen als Typen deklariert werden, andererseits die Dekomposition in Teilsysteme nicht die Typen, sondern Objekte der Typen, also deren Extension berücksichtigt. Das Problem der Ausdrucksschwäche von typbasierten Modellen tritt auf, weil auf intensionaler Ebene nur eingeschränkt extensionale Aussagen über Objekte gemacht werden können.

Ziel einer geeigneten Klassifikation ist es nun, die Dualität des Klassenbegriffes aufzulösen und extensionale und intensionale Aussagen zu trennen. Ein erster Schritt ist die Festschreibung des Klassenbegriffs auf die intensionale Bedeutung: *Eine Klasse beschreibt allein den Typ einer Menge von Objekten.*

Dies bedeutet, daß alle extensionalen Aussagen wie Dekompositionen oder Objektzusammenhänge auf *Ebene der Objekte* modelliert werden müssen. Für diese Art der Modellierung sind grundsätzlich zwei Arten von Objekten vorstellbar, die wir *ausgeprägte* und *prototypische Objekte* nennen:

> Ein *ausgeprägtes Objekt* entspricht der aus der gängigen OO-Terminologie bekannten *Instanz*: Eine konkrete Ausprägung mit vorhandener Identität, einem bekanntem Typ und mit ausgeprägten Attributen. (Beispiel: Die Person «Eva Müller», der Laserdrucker mit der Seriennummer «PQ-380 4789»).
>
> Beim *prototypischen Objekt* hingegen ist zwar der Typ, aber weder Objektidentität noch die Attributsausprägungen bekannt. Ein *prototypisches Objekt* kann man sich als Platzhalter oder Repräsentant für potentielle Klasseninstanzen vorstellen. (Beispiel: Eine Person, ein Laserdrucker).

Ausgeprägte Objekte eignen sich nicht zum Modellieren von Spezifikationen, da sie zuwenig allgemein sind und außerdem die meisten konkreten Objektausprägungen beim Spezifizieren noch nicht bekannt sind. Im folgenden werden daher nur noch prototypische Objekte betrachtet.

Es ist explizit möglich, verschiedene prototypische Objekte des gleichen Typs zu modellieren. Diese haben die Eigenschaft, für verschiedene Instanzen dieses Typs zu stehen (Beispiel: Ein Studentendrucker und ein Sekreteriatsdrucker derselben Bauart in einem Universitätsinstitut). Prototypische Objekte modellieren nicht Typen, sonden Rollen, in denen sich konkrete Instanzen befinden können. Dies ermöglicht zusätzlich zur Festlegung der Objektcharakteristiken eine präzise Beschreibung des Kontextes eines Objektes. Anzumerken ist noch, daß die Modellierung von Rollen in bestimmten objektorientierten Methoden (*UML091*, *OMT*) möglich ist. Die Rollen werden dort jedoch mit auf der Klassenebene modelliert, die nach wie vor die in Kapitel 2 aufgeführten Schwächen aufweist.

Für eine quantitative Modellierung sind neben *prototypischen Objekten* auch *prototypische Objektmengen* sinnvoll:

> Eine *prototypische Objektmenge* ist eine Menge von *prototypischen Objekt*en desselben Typs. Es ist also explizit möglich, Aussagen über Teilmengen und nicht nur über die Gesamtmenge von Objekten desselben Typs zu machen (Beispiel: Eine Personengruppe einer Arbeitsgruppe, vier gleichbenutzte Studentendrucker).

Die Eigenschaft, über einzelne Objekte bzw. über Mengen von Objekten desselben Typs Aussagen machen zu können, ermöglicht eine präzisere Modellierung und eine hierarchische Modellzerlegung.

Abb. 5 zeigt ein Beispiel für die Modellierung mit *prototypischen Objekten* anhand einer Grobbeschreibung des Doppelfahrstuhls. Die Rechtecke stellen *prototypische Objekte* dar, der Rechteckstapel eine prototypische Objektmenge. Die Leserichtung der Kardinalitäten ist dieselbe wie in der *UML091*.

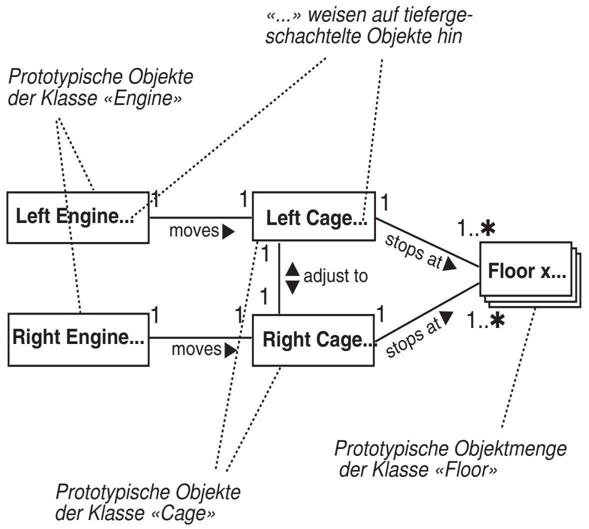

Abb. 5: Dekomposition mit prototypischen Objekten für das Doppelfahrstuhl-Beispiel (aus Abb. 4)

Die Modellierung erfolgt auf zwei Ebenen, auf Ebene der Klassen und auf Ebene der prototypischen Objekte (siehe Abb. 6). Klassen werden hier ausdrücklich nur in ihrer intensionalen Bedeutung verwendet und dienen der Deklaration von Objekttypen und der Spezifikation von Typzusammenhängen (z. B. Subtypen, Spezialisierungen). Die zentrale Ebene ist aber die der prototypischen Objekte. Sie dient zur Beschreibung von prototypischen Objekten und prototypischen Objektmengen und deren extensionaler Zusammenhänge und kann geeignet hierarchisch zerlegt werden.

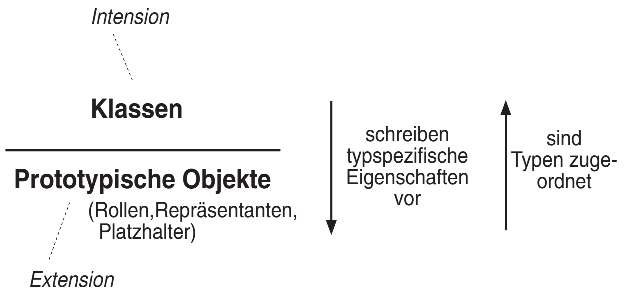

Abb. 6: Modellierung auf zwei Ebenen: Intensional durch Typen und extensional durch Objekte

Auf den Zusatz «prototypisch» werden wir in den folgenden Ausführungen der Einfachheit halber meistens verzichten. Überall dort, wo ohne nähere Qualifikation von Objekten die Rede ist, sind daher prototypische Objekte gemeint.

Ein für die Klassenebene geeigneter Zerlegungsmechanismus ist die Generalisierung und die damit verbundene Generalisierungshierarchie. Die Generalisierung von Typen und Klassen behandeln wir an dieser Stelle nicht weiter, sondern verweisen auf die Arbeit von [Ecker94].

Teil/Ganzes-Hierarchien zur Dekomposition von Systemen

In diesem Abschnitt wird die Teil/Ganzes-Hierarchie als Zerlegungsmechanismus vorgestellt, um speziell auf Objektebene Dekompositionen zu beschreiben. Die Teil/Ganzes-Hierarchie ist wie folgt charakterisiert:

- Eine *Komposition* ist eine Zusammenfassung von *Komponenten* (prototypische Objekte und/oder prototypische Objektmengen) zu einem neuen übergeordneten prototypische Objekt bzw. einer übergeordneten prototypischen Objektmenge.
- Da eine *Komposition* wieder ein prototypisches Objekt bzw. eine prototypische Objektmengen ist, geht sie über die Bedeutung eines reinen Behälters hinaus und hat sämtliche Eigenschaften (wie etwa eigene Attribute und eigenes Verhalten), die auch ein Objekt hat.
- *Kompositionen* sind überschneidungsfrei, d.h. ein Objekt kann nur Komponente *einer* Komposition sein (Bemerkung: Die Komposition-Komponenten-Relation ist irreflexiv und asymmetrisch).

Das Verhältnis einer *Komposition* zu ihren *Komponenten* wird wie folgt beschrieben:

- Jedes Objekt und somit auch jede Komposition hat einen eigenen Zustandsraum.
- Die Zustandsräume der Komponenten liegen vollständig im Zustandsraum ihrer Komposition.

Die Teil/Ganzes-Hierarchie ist die Hierarchie, die sich aus den Kompositionen und deren Komponenten bildet. Die Teil/Ganzes-Hierarchie besitzt eine Baumstruktur, d.h., ein Objekt einer Klasse kann nur an einer Stelle der Hierarchie aufgeführt werden. Die Teil/Ganzes-Hierarchie unterscheidet sich fundamental von der gewöhnlichen Teil/Ganzes-Beziehung (vgl. Kapitel 2, Behandlung der Probleme in ausgewählten Spezifikationsmethoden), indem die Teil/Ganzes-Hierarchie mit einer *Abstraktion* verknüpft ist: Einerseits kann jede Komposition mit ihren Komponenten getrennt von den übrigen Kompositionen betrachtet werden, andererseits können die Kompositionen ohne ihre Komponenten betrachtet werden und bilden so ein vergröbertes Modell. Die Teil/Ganzes-Hierarchie ermöglicht somit eine vom Groben zum Detaillierten ver-

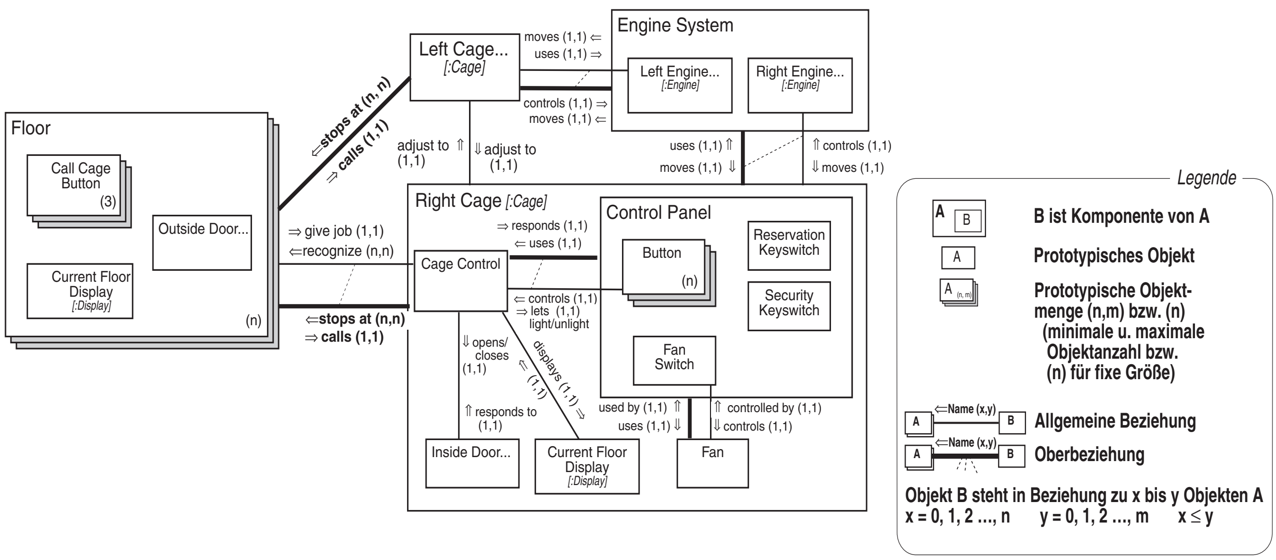

Abb. 7: Hierarchisches Objektmodell, beschrieben durch prototypische Objekte als Kompositionen

laufende einfache und anschauliche Zerlegung eines Systems.

Beispiel «Objektmodell des Doppelfahrstuhls»

In Abb. 7 wird ein hierarchisches Objektmodell zum Doppelfahrstuhl-Problem aus Kapitel 2 beschrieben. Es gliedert sich in acht teilweise aufeinander aufbauende Kompositionen, wobei Engine System, Floor, Right Cage, Control Panel detailliert beschrieben und Left Cage, Door, Left Engine, Right Engine vergröbert beschrieben sind. Die in den prototypischen Objektmengen aufgeführten Zahlentupel beschreiben die minimale und maximale Anzahl der Menge.

Im Objektmodell (Abb. 7) ist für alle Objekte, für die typgleiche Objekte an anderer Stelle des Modells existieren, ihr Typ angegeben. Dort, wo kein Typ angegeben ist, tragen das Objekt und sein Typ den gleichen Namen. Weitere Typzusammenhänge (wie z. B. Spezialisierungen) müssen in einem zusätzlichen Typmodell beschrieben werden.

Die im Kapitel 2 aufgeführten drei Schwachstellen sind im Objektmodell in Abb. 7 nicht mehr vorhanden:

Keine Überlappung von Kompositionen

- Im Objektmodell taucht an drei verschiedenen Orten ein prototypisches Objekt mit dem Namen Current Floor Display auf[3]. Alle drei Objekte gehören derselben Klasse an. Aufgrund der oben beschriebenen Semantik der Dekomposition ist aber sichergestellt, daß alle drei Objekte eine unterschiedliche Identität besitzen. Zur Unterscheidung solcher Objekte in Textspezifikationen, wird der Kompositionsname vorangestellt, z. B.: Floor.Current_Floor_Display). Eine Überlappung der Teilsysteme ist daher nicht notwendig.
- Typzusammenhänge werden im Typmodell beschrieben und müssen nicht im Objektmodell mit aufgeführt werden.

Teilsystemzusammenhänge durch Oberbeziehungen

Dieser Punkt wird im folgenden Kapitel «Beziehungshierarchie in Teil/Ganzes-Hierarchien» erläutert.

Ausdrucksstarke und präzise Objektmodelle

- Das Objektmodell beschreibt, daß mit Left Cage und Right Cage genau *zwei* Kabinen im Modell vorhanden sind und sich diese beiden Kabinen gegenseitig synchronisieren (diese Aussage war mit Klassenmodellen in dieser Präzision nicht möglich). Siehe dazu auch Abb. 5 und die zugehörigen Erläuterungen.
- Im obigen Objektmodell findet sich im Gegensatz zum ursprünglichen Klassenmodell (Abb. 2) keine einzige partielle Beziehung wieder. Die ursprünglich getroffenen partiellen Aussagen beziehen sich somit nicht auf das eigentliche Problem, sondern wurden durch den klassenorientierten Ansatz erzwungen.

Zum Beispiel von Abb. 7 sind folgende Bemerkungen zu machen:

- Potentielle Schwachstellen wie etwa die unterspezifizierte Zusammenarbeit der Floor-Komponenten mit anderen Objekten sind hier gegenüber Abb. 1 und 2 offensichtlicher. Ähnliches gilt für Komponenten von Control Panel.
- Die Verschachtelung von prototypischen Objektmengen führt zu einer Multiplikation der maximal möglichen Objekte. Die maximale Anzahl der Objekte Current Floor Display in der Komposition Floor ist somit 3·n.

Beziehungshierarchie in Teil/Ganzes-Hierarchien

Im letzten Teil wurde eine Systemzerlegung in Form von Kompositionen und Komponenten vorgestellt. Bisher wurde jedoch nicht geklärt, wie aus der detaillierten Beschreibung von Kompositionen und ihren Komponenten eine vergröberte Beschreibung auf der Stufe der Kompositionen hervorgeht. Hierzu müssen Detailzusammenhänge zwischen verschiedenen Kompositionen bzw. zwischen deren Komponenten auf abstrakte Zusammenhänge zwischen Kompositionen abgebildet werden. Zu diesem Zweck wird das Konzept der *Beziehungshierarchie* für Teil/Ganzes-Hierarchien eingeführt. Die Vorstellung dabei ist, daß unter den Objektausprägungen zu jedem Zeitpunkt zu einer existierenden Beziehung auf einer detaillierten Stufe eine zugeordnete Beziehung auf der Stufe der Kompositionen existiert. Diese Vorstellung wird mit Hilfe einer Beziehungshierarchie für Objektmodelle umgesetzt. Eine Beziehungshierarchie liegt vor, wenn die Beziehungen zwischen den Objekten in einem hierarchisch gegliederten Model so in Ober- und Unterbeziehungen geordnet werden können, daß die folgenden Bedingungen erfüllt sind:

(i) Zu jeder Beziehung *B* zwischen einer Komponente einer Komposition *K1* und einer Komponente einer anderen Komposition *K2* gibt es genau eine Beziehung *A* zwischen K1 und *K2*. *A* ist Oberbeziehung von *B*.

(ii) Zu jeder Beziehung *B* zwischen einer Komponente einer Komposition *K* und einem Objekt *O*, das keine Komponente irgendeiner Komposition ist, gibt es genau eine Beziehung *A* zwischen *K* und *O*. *A* ist Oberbeziehung von *B*.

(iii) Die Aussagen „*A* ist Oberbeziehung von *B*“ und „*B* ist Unterbeziehung von *A*“ sind äquivalent.

Dabei ist zu bemerken, daß verschiedene Beziehungen zwischen den Komponenten zweier Kompositionen nicht notwendig dieselbe Oberbeziehung haben müssen.

Wir fordern nun, daß die Beziehungen eines hierarchisch gegliederten Modells eine Beziehungshierarchie bilden. Damit wird der in Kapitel 2 geforderte Zusammenhang von Teilsystemen zumindest syntaktisch erfüllt: Zu jeder Beziehung auf Detailstufe gibt es eine zugeordnete Beziehung auf Kompositionsstufe. Semantisch wird der Zusammenhang allerdings erst dann sinnvoll, wenn Beziehungen in semantisch sinnvoller Weise zu Oberbe-

[3] Das dritte Current Floor Display-Objekt ist in der Abbildung in der Komposition Left Cage verborgen.

ziehungen zusammengefaßt werden und die Oberbeziehungen einen Namen bekommen, der zu dem paßt, was sie abstrahieren sollen. Durch die syntaktischen Forderungen (i) - (iii) und die geeignete Benennung der Oberbeziehungen ergibt sich auf Ebene von Kompositionen eine abstrakte Beschreibung von Zusammenhängen der Detail- bzw. Komponentenebene.

Für Beziehungshierarchien sind zusätzliche Restriktionen zwischen den Kardinalitäten von Ober- und Unterbeziehungen vorstellbar. Die Beschreibung möglicher Restriktionen würde allerdings den Rahmen dieses Beitrags sprengen.

4 Schlußbemerkung

Die derzeitigen objektorientierten Spezifikationsmethoden weisen essentielle Schwachstellen in zentralen Modellierungskonzepten auf. Diese Schwachstellen lassen sich nicht durch einfache Ergänzungen und zusätzliche Aspektsichten beheben. Daher erscheint auch der momentane Trend, defizitäre Methodenansätze zu fusionieren (siehe *UML* [Booch97], *Fusion Method* [Colem94]), kein geeigneter Lösungsweg, da dadurch die eigentlichen Ursachen nicht behoben werden. Ebenso ungeeignet erscheint der häufig erkennbare Ansatz, Entity-Relationship-artige Modelle durch einfache Ergänzungen für die objektorientierte Anforderungsspezifikation tauglich zu machen. Der Einsatz von Werkzeugen kann kaum zur Lösung dieser Probleme beitragen, da Visualisierungskonzepte ebenfalls geeignete Modellierungskonzepte voraussetzen.

Der hier vorgestellte Ansatz beschreibt eine Alternative zu den bekannten objektorientierten Methoden, bei der mit Ausnahme der Generalisierung alle Modellstrukturen auf der Basis von prototypischen Objekten und prototypischen Objektmengen anstelle von Klassen modelliert werden. Die Arbeit mit Klassen wird auf Typdefinitionen und Generalisierungshierarchien beschränkt. Auf diese Weise ist eine saubere und anschauliche Definition einer Modelldekomposition möglich, welche eine Abstraktion der Modellstruktur gestattet. Wir versprechen uns davon – analog zu den Leistungen entsprechender Konstrukte im Systementwurf und in der Programmierung – erhebliche Verbesserungen bei der Verständlichkeit und Wartung großer Modelle.

Auf Grundlage der hier skizzierten Konzepte entwikkeln wir eine Spezifikationssprache und eine dazu passende Methodik.

Literatur

[Booch94] Booch, G. (1994): *Object-Oriented Analysis and Design with Applications*. The Benjamin/ Cummings Publishing Company, Inc.

[Booch96] Booch, G., Rumbaugh, J., Jacobson I. (1996): *Unified Modelling Language for Object-Oriented Development*. Documentation Set, Version 0.8 + 0.91, RATIONAL Software Corporation.

[Booch97] Booch, G., Rumbaugh, J., Jacobson, I. (1997): *Unified Modelling Language for Object-Oriented Development*. Documentation Set, Version 1.0, RATIONAL Software Corporation.

[Chen76] Chen, P. (1976): The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems* vol. 1(1), 9-36.

[Coad91] Coad, P., Yourdon, E. (1991): *Object-Oriented Analysis*. Yourdon Press Computing Series.

[Colem94] Coleman, D., Meyer, B. (1994): *Object-Oriented Development: The Fusion Method.* Prentice-Hall, Englewood Cliffs, NJ.

[DeMar78] DeMarco, T. (1978): *Structured Analysis and System Specification*. Yourdon Press, New York.

[Ecker94] Eckert, G. (1994): Types, Classes and Collections in Object-Oriented Analysis. *Proceedings ICRE´94, International Conference on Requirements Engineering*, Colorado Springs, Colorado, USA.

[Emble92] Embley, D.W., Kurtz, B.D., Woodfield, S.N. (1992): *Object-Oriented Systems Analysis*. Prentice-Hall International, Inc.

[Glinz93] Glinz, M. (1993): Hierarchische Verhaltensbeschreibung in objektorientierten Systemmodellen – eine Grundlage für modellbasiertes Prototyping. In Züllighoven, Heinz / Altmann, Werner / Doberkat, Ernst-Erich (Herausgeber) *Requirements Engineering 1993: Prototyping*. Teubner, Stuttgart, 175-192.

[Jacob94] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1994): *Object-Oriented Software Engineering*. Addison-Wesley Publishing Company.

[Parna72] Parnas, D. L. (1972): On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM* vol. 15(12), Dec., 1053-1058.

[Rumba91] Rumbaugh, J., et al. (1991): *Object-Oriented Modeling and Design*. Prentice-Hall International, Inc.

[Shlae96] Shlaer, S., Mellor, S.J. (1996): *Shlaer-Mellor Method: The OOA96 Report, Version 1.0*. Project Technology, Inc., Berkeley.

[Wirfs90] Wirfs-Brock, R., Wilkerson, B., Wiener, L. (1990): *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, NJ.