Erschienen in: Informationssystem-Architektur. Rundbrief des GI-Fachausschusses 5.2, 3. Jg., Heft 1, Sept. 1996, S. 16-19

Delegation: Eine sinnvolle Ergänzung gängiger objektorientierter Modellierungskonzepte

Ulrich Frank & Sören Halter

1. Einleitung

Im Rahmen verschiedener Projekte haben wir des öfteren die Erfahrung gemacht, daß weder Vererbung noch gängige Beziehungsarten (wie Interaktion oder Aggregation) eine zufriedenstellende Abbildung realer Sachverhalte erlaubten. In solchen Fällen erwies sich Delegation häufig als ein angemessenes Konzept. Delegation wird seit langem in der Informatik verwendet - nicht zuletzt im Bereich der KI-Forschung. Auch in der Literatur über objektorientierter Systementwicklung findet man eine Reihe von Arbeiten, in denen Delegation thematisiert wird - wenn auch nicht in einheitlicher Terminologie (LIEBERMANN, KIM/LOCHOVSKY, SCIORE). Demgegenüber ist festzustellen, daß neuere populäre Modellierungsmethoden ein solches Konzept nicht beinhalten. Ein Umstand, der angesichts der erheblichen Bedeutung von Delegation einigermaßen überraschend anmutet. Er ist u.E. wohl vor allem auf zwei Gründe zurückzuführen. Zum einen ist daran zu denken, daß die expressiven Möglichkeiten, die mit Vererbung verbunden sind, häufig überschätzt werden - was sich mitunter darin artikuliert, daß selbst in populären Beispielen Vererbung in unangemessener Weise verwendet wird. Daneben ist zu berücksichtigen, daß die meisten objektorientierten Programmiersprachen keine Konzepte für die unmittelbare Implementierung von Delegation beinhalten.

Im folgenden Beitrag werden die Bedeutung von Delegation als Modellierungskonzept sowie die Voraussetzungen für seine angemessene Verwendung dargestellt. Darüber hinaus wird ein in Smalltalk realisiertes Entwurfsmuster ("Design Pattern") zur Verwendung von Delegation auf der Implementierungsebene beschrieben.

2. Grenzen der Vererbung

Vererbung stellt unzweifelhaft ein zentrales Konzept objektorientierten Entwurfs dar. Nicht nur, daß Generalisierung und Spezialisierung Wartbarkeit und Wiederverwendbarkeit begünstigen, darüber hinaus ermöglicht die "is a"-Beziehung eine intuitive und damit natürliche Beschreibung der Realität. Manchmal jedoch führt Vererbung zu unangemessenen Konzepten - auch dann, wenn sie in einer intuitiven und plausiblen Weise angewandt wurde. Betrachten wir das folgende Beispiel: Für den Entwurf des Informationssystems einer Universität benötigt man u.a. Objekte, die Studenten, wissenschaftliche Mitarbeiter und Professoren repräsentieren. Da diese Objekte gemeinsame Eigenschaften wie Name, Vorname und Geschlecht etc. aufweisen, liegt es nahe, über diese Eigenschaften zu generalisieren und eine entsprechende Oberklasse - etwa `Person` - einzuführen. Auf diese Weise erhält man eine leicht nachvollziehbare Konzeptualisierung: ein Student ist eine Person, ein Professor ist eine Person etc. Im Anschluß daran stellt sich heraus, daß wir auch Objekte benötigen, die Programmierer, Systemadministratoren und studentische Mitarbeiter repräsentieren. Auch hier scheint Vererbung der richtige Ansatz zu sein: Auch Programmierer, Systemadministratoren und studentische Mitarbeiter sind nach gängigem Verständnis Personen.

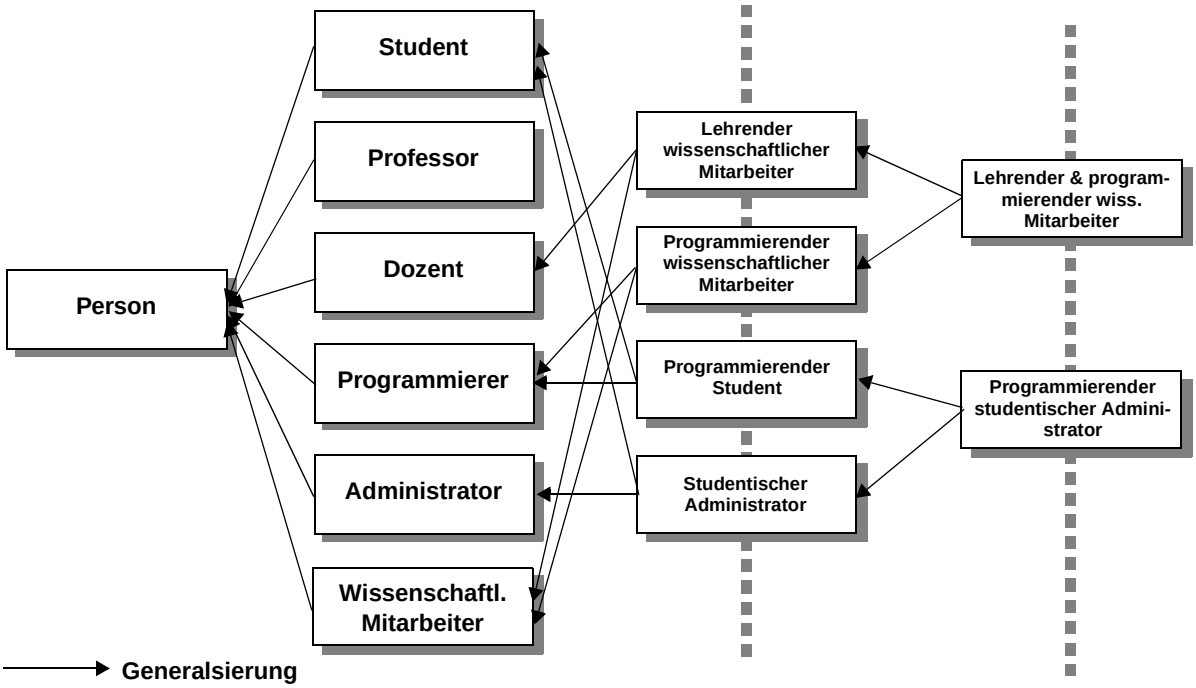

Abb. 1: Konzepte, die durch Mehrfachvererbung entstehen

Allerdings ist zu berücksichtigen, daß Studenten oder wissenschaftliche Mitarbeiter auch Programmierer sein können - oder gleichzeitig Programmierer und Systemadministrator. Mit Einfachvererbung lassen sich diese Zusammenhänge kaum darstellen. Mehrfachvererbung scheint zunächst erfolgversprechender. Wenn wir unser kleines Beispiel unter Verwendung von Mehrfachvererbung rekonstruieren, erhalten wir das in Abbildung 1 dargestellte Modell. Die in diesem Modell definierten Klassen würden prinzipiell erlauben, die oben genannten Kombinationen von Kompetenzen auszudrücken. Es kann allerdings kaum behauptet werden, daß dieses Modell dem Bemühen um eine möglichst natürliche Rekonstruktion realer Sachverhalte gerecht wird: Ein Konzept wie "lehrender und programmierender wissenschaftlicher Mitarbeiter" wird man in der Diskurswelt einer Universität kaum finden. Noch bedeutsamer ist der Umstand, daß Vererbung - einerlei, ob es sich um Einfach- oder Mehrfachvererbung handelt - auch aus softwaretechnischer Sicht zu dysfunktionalen Konzepten führt, die eine ernstzunehmende Gefährdung der Wartbarkeit und Integrität eines Informationssystems darstellen. So würde die Umsetzung des in Abbildung 1 dargestellten Modells dazu führen, daß aus unterschiedlichen Klassen Objekte instantiert würden, die aber jeweils die gleiche Entität in der Realität repräsentieren. So würde man beispielsweise eine Instanz der Klasse Person mit den Angaben einer bestimmten realen Person initialisieren, um später in einem anderen Kontext das gleiche mit einer Instanz der Klasse Student zu tun. Auf diese Weise entsteht eine nicht akzeptable Redundanz.

Unser kleines Beispiel zeigt, daß die Anwendung von Vererbung zu unangemessenen Konzepten führen kann, obwohl jede einzelne "is a"-Beziehung plausibel erscheint. In der Diskurswelt einer Domäne wie der oben skizzierten verwenden wir offenbar Abstraktionen, die vom aktuellen *Kontext* abhängen. Manchmal ist unser Augenmerk auf einen Dozenten gerichtet und es ist irrelevant, ob die entsprechende Person auch programmieren kann. In einem anderen Kontext betrachten wir diese Person als Systemadministrator. Vererbung läßt es allerdings nicht zu, Veränderungen des Kontextes, die während der Lebenszeit eines Objekts stattfinden mögen, auszudrücken. Mit anderen Worten: Generalisierung macht es nötig, Abstraktionen gleichsam einzufrieren, bevor das erste Objekt instanziert wurde.

Auch wenn das oben dargestellte Beispiel nach einer Konzeptualisierung verlangt, die es erlaubt, bestimmte Abstraktionen während der Lebenszeit eines Objekts zu ändern, heißt das nicht, daß auf ein zentrales Merkmal der Vererbung verzichtet werden sollte: Es ist unzweifelhaft, daß ein Student eine Person ist, daß ein entsprechendes Objekt sich also verhalten sollte wie ein Objekt der Klasse Person. Aus diesem Grund ergibt sich zusammen mit der Mehrdeutigkeit natürlichsprachlicher Verwendung von "is a" eine zunächst verwirrende Ausgangslage. Es ist deshalb hilfreich, für die betrachteten Fälle statt des Bezeichners "is a" einen anderen, weniger mehrdeutigen Bezeichner einzuführen. Je nach Betrachtungsrichtung verwenden wir dazu die Bezeichner "repräsentiert" oder "fungiert als": Ein Progammierer repräsentiert eine Person, eine Person fungiert als Programmierer. Mit anderen Worten: Wir betrachten Programmierer als eine *Rolle.* Im Unterschied zu Vererbung würde nicht die Klasse Programmierer von der Klasse Person erben, stattdessen würde eine Instanz der Klasse Person ihre Eigenschaften (Zustand und Protokoll) an eine Instanz der Klasse Programmierer *delegieren* (wir könnten auch sagen: propagieren).

Die Bedeutung von Delegation läßt sich auch an einem anderen Beispiel veranschaulichen, daß in dieser oder ähnlicher Weise des öfteren diskutiert wird: Eine Versicherungsgesellschaft verwaltet die minderjährigen Kinder ihrer Versicherungsnehmer mit Hilfe der Klasse NachwuchsKunde. Sobald ein Kind - das nötige Alter vorausgesetzt - selbst Versicherungsnehmer wird, muß die entsprechende Instanz gelöscht und anschließend eine neue Instanz der Klasse Versicherungsnehmer erzeugt werden. Ambitioniertere Ansätze zur Pflege von Objektverwaltungssystemen zielen auf eine Wandlung der Klasse eines Objekts während seiner Lebenszeit. Man spricht dabei von "Class Migration" (vgl. etwa WIERINGA ET AL.). Der konzeptionelle und systemtechnische Aufwand einer solchen Migration ist allerdings erheblich. Im Unterschied dazu würde Delegation vorsehen, daß sowohl NachwuchsKunde als auch Versicherungsnehmer *Rollen* etwa von Objekten der Klasse Person wären - ggfs. ergänzt durch eine Integritätsbedingung, wonach eine Instanz der Klasse Person nicht gleichzeitig mit einer Instanz der Klasse NachwuchsKunde sowie einer Instanz der Klasse Versicherungsnehmer assoziiert sein darf.

3. Die Semantik von Delegation

Angesichts des Umstands, daß Delegation ein bedeutsames Konzept für eine natürliche Modellierung gewisser realweltlicher Sachverhalte darstellt, erscheint es überraschend, daß keine der heute populären Methoden der objektorientierten Modellierung (vgl. RUMBAUGH ET AL., BOOCH, JACOBSON) Delegation explizit enthält. Wie eingangs bereits erwähnt, mag dies auf eine gewisse Überschätzung der Möglichkeiten von Vererbung zurückzuführen sein - nicht zuletzt wegen der gezeigten Mehrdeutigkeit des Bezeichners "is a". Ein anderer Grund mag darin liegen, daß man Aggregation als ein geeignetes Konzept für die Abbildung der skizzierten Sachverhalte hält: Ein Programmierer beinhaltet eine Person (oder sollten wir sagen: eine Person beinhaltet einen Programmierer - wobei hier sinnvollerweise andere Kardinalitäten zu wählen wären). Bei näherer Betrachtung zeigt sich allerdings, daß Delegation zwar gewisse Ähnlichkeiten zu Vererbung oder Aggregation aufweist, aber letztlich ein eigenständiges Konzept ist. Wir definieren Delegation als eine spezielle Assoziation mit folgenden allgemeinen Eigenschaften:

1. Es handelt sich um eine gerichtete binäre Assoziation zwischen einem Objekt, das wir Rolleninhaber ("Delegator") nennen, und einem anderen Objekt, der Rolle ("Delegate"). Das Rolleninhaber-Objekt delegiert sein Verhalten (und damit seinen Zustand) an das Rollen-Objekt.
2. Das Rollen-Objekt bietet zusätzlich zur eigenen Schnittstelle in transparenter Weise die-

Schnittstelle des Rolleninhaber-Objekt an. Für den Fall, daß Dienste von Rollen-Objekt und Rollenhinhaber-Objekt den gleichen Namen haben, wird der Dienst des Rollen-Objekts ausgeführt. Im Unterschied zu Vererbung werden die nicht zu befriedigenden Nachrichten an eine Instanz (nämlich ein Rolleninhaber-Objekt) weitergeleitet. Auf diese Weise wird erreicht, daß das Rollen-Objekt auch einen transparenten Zugriff auf den aktuellen Zustand des Rolleninhaber-Objekts erlaubt: Nach außen repräsentiert es gleichsam das Rolleninhaber-Objekt. Diese Forderung entspricht der tatsächlichen Betrachtungsweise in der Realität: Man würde wohl kaum einen Programmierer nach seinem Rolleninhaber fragen, um anschließend diesen nach seinem Namen zu fragen. Stattdessen würde man den Programmierer direkt nach seinem Namen fragen.

Darüber hinaus schlagen wir eine Reihe von Integritätsbedingungen vor:

1. Nur Klassen, die als spezielle Rollen-Klasse ausgezeichnet sind, können verwendet werden, um Rollen-Objekte innerhalb einer Delegationsbeziehung zu repräsentieren. Dafür gibt es zwei Gründe. Zunächst ist zu berücksichtigen, daß nicht die Objekte einer jeden Klasse konzeptionell geeignet sind als Rollen-Objekte zu fungieren, wobei Beurteilung dieser grundsätzlichen Möglichkeit von den Gegebenheiten der jeweiligen Domäne abhängt. Darüber hinaus ist daran zu denken, daß die spezielle Semantik von Rollen-Klassen geeignete Erweiterungen auf der Implementierungsebene nötig macht (s. 4).
2. Einem Rolleninhaber-Objekt können grundsätzlich keine oder mehrere Rollen-Objekte zugeordnet werden. Für eine bestimmte Delegationsbeziehung kann die Kardinalität innerhalb dieser Bandbreite eingeschränkt werden. Dabei kann ein Rolleninhaber-Objekt durchaus mit mehreren Objekten einer bestimmten Rollen-Klasse assoziiert sein. Beispiel: Eine Instanz der Klasse Person kann gleichzeitig an zwei Instanzen der Klasse Programmier delegieren - in einem Fall etwa an eine Instanz, deren Zustand auf Smalltalk-Kompetenz verweist, im anderen Fall an eine Instanz, die Kenntnisse in C++ verheißt (womit nicht gesagt ist, daß eine solche Art der Modellierung grundsätzlich angeraten ist).
3. Ein Rollen-Objekt ist einem und genau einem Rolleninhaber-Objekt zugeordnet.

In der Literatur wird das Konzept Delegation nicht in einheitlicher Weise dargestellt. Das gilt einerseits für die Terminologie. So wird häufig die Bezeichnung "delegieren an" genau in die andere Richtung verwendet (z.B. GAMMA ET AL.; JOHNSON/ZWEIG). Dabei hat man allerdings weniger eine Rekonstruktion realweltlicher Betrachtungsweisen im Sinn, man denkt vielmehr stärker an die Implementierung: Wenn ein Objekt, das eine Rolle darstellt, eine Nachricht gesendet bekommt, die es aufgrund des eigenen Protokolls nicht versteht, delegiert dieses Objekt den Aufruf an den Rolleninhaber. SCIORE spricht anstelle von Delegation von "object specialization". In der Programmiersprache SELF (SMITH/UNGAR) wird das Rolleninhaber-Objekt "parent-object" genannt. Wir meinen uns auch an die Wendung "Vererbung auf Instanzenebene" erinnern zu können. Auch wenn wir damit die ohnehin beachtliche Konfusion nicht eben verringern, bevorzugen wir die oben dargestellte Sichtweise, da sie uns im Hinblick auf die Anschaulichkeit und Lesbarkeit von Modellen angemessener erscheint. Andererseits wird die Semantik von Delegation sowie der Stellenwert des Konzepts für die objektorientierte Modellierung nicht immer in gleicher Weise eingeschätzt. Bezeichnend dafür die Feststellung von STEIN: "Delegation is inheritance".

Um die Verwendung von Delegation in der objektorientierten Modellierung zu ermöglichen, erweiterten wir die Modellierungsmethode MEMO ("Multi Perspective Enterprise Modeling", vgl. FRANK) um eine spezielle Assoziationsart. Dazu gehört neben einer entsprechenden Notation (s. Abb. 2) die Berücksichtigung der Semantik von Delegation. In der auf der Methode aufbauenden Entwicklungsumgebung MEMO Center werden die sich daraus ergebenden Inte-

gritätsbedingungen verwaltet. Außerdem erlaubt MEMO Center das Generieren von Smalltalk-Code, der der Erzeugung und Verwaltung von Delegationsbeziehungen dient.

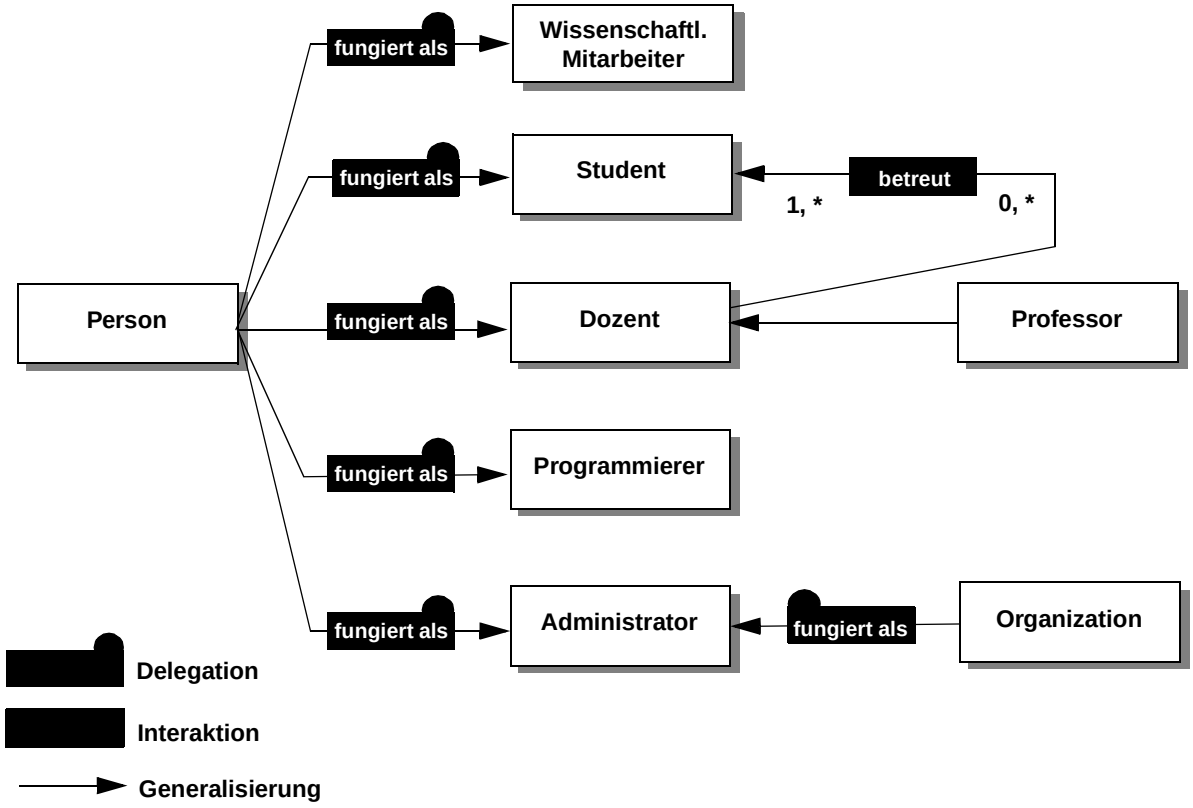

Abb. 2: Darstellung von Delegation in der MEMO-Notation

4. Delegation auf der Implementierungsebene: Eine Erweiterung am Beispiel Smalltalk

Nach unseren Erfahrungen ist Delegation ein häufig benötigtes und dabei sehr nützliches Modellierungskonzept. In diesem Sinne KATHURIA/SUBRAMANIAM (S. 39), die ein ähnliches Konzept, das sie "Assimilation" nennen, vorschlagen: "As practitioners we have a strong need for a concept like assimilation.". Unter bestimmten Voraussetzungen (s. 5) stellt Delegation ein ausgesprochen natürlich wirkendes Konzept zur Abbildung realer Sachverhalte dar. Es ist deshalb u.E. empfehlenswert, Delegation bei der Modellierung auch dann zu verwenden, wenn sie durch die eingesetzte Implementierungssprache nicht unterstützt wird. Die Verwendung von Delegation wird allerdings deutlich attraktiver, wenn keine semantische Lücke zwischen Objektmodell und Implementierungssprache zu berücksichtigen ist. Gibt es eine Chance, existierende objektorientierte Sprachen mit vertretbarem Aufwand um Delegation zu erweitern?

Die wesentliche Herausforderung besteht darin, für Rollen-Objekte einen transparenten Zugriff auf die Dienste des jeweils zugeordneten Rolleninhaber-Objekts zu ermöglichen. Dazu sollten alle Nachrichten, für die die Klasse des Rollen-Objekts (oder eine Oberklasse) keine entsprechende Methode bereitstellt, an das Rolleninhaber-Objekt weitergeleitet werden. Um dies zu erreichen, könnte man die Schnittstelle des Rolleninhaber-Objekts in das Rollen-Objekt kopieren und dort jeweils Methoden implementieren, die eine Weiterleitung der Nachricht an das Rollen-Objekt durchführen. Ein solcher Ansatz ist allerdings mit einem gravierenden Nachteil verbunden: Immer dann, wenn sich die Schnittstelle eines Rollen-Objekts ändert, müssen alle betroffenen Rollen-Klassen entsprechend angepaßt und neu kompiliert werden.

Vor dem Hintergrund des u.U. erheblichen Wartungsaufwands und der damit verbundenen Gefährdung der Systemintegrität ist deshalb von einem solchen Vorgehen abzuraten. Wesentlich leistungsfähiger wäre eine Lösung, die eine Weiterleitung nicht zu befriedigender Methodenaufrufe über einen generellen Mechanismus erlaubten. Da über eine solche Weiterleitung erst zur Laufzeit entschieden werden könnte, scheiden Sprachen mit statischer Typprüfung wie etwa Eiffel per definitionem aus. Es bleiben also allenfalls Sprachen, die eine dynamische Typprüfung vorsehen. Im folgenden zeigen wir eine entsprechende Erweiterung für Smalltalk. Zudem skizzieren wir zwei alternative Ansätze zur Unterstützung der konsistenten Verwaltung von Assoziationen im allgemeinen, von Delegationsbeziehungen im besonderen. Die Implementierung erfolgte für VisualWorks® 2.5, sollte allerdings auch in anderen Smalltalk-Umgebungen ohne nennenswerten Aufwand eingesetzt werden können.

Die Implementierung des erwähnten generellen Mechanismus kann in Smalltalk in sehr einfacher Weise durchgeführt werden. Dazu muß man wissen, daß - der dynamischen Typprüfung entsprechend - in Smalltalk erst zur Laufzeit überprüft wird, ob eine nachgefragte Methode von einem Objekt tatsächlich ausgeführt werden kann. Für den Fall, daß ein Objekt nicht über eine nachgefragte Methode verfügt, schickt es die Nachricht doesNotUnderstand: aMessage (aMessage ist dabei die unverstandene Nachricht) an sich selbst. In der Regel wird dann eine Ausnahmebehandlung aufgerufen, die mit der Unterbrechung des Programmablaufs verbunden ist. Durch das Überschreiben dieser Methode kann die Weiterleitung an das Rolleninhaber-Objekt gewährleistet werden:

```
doesNotUnderstand: aMessage
    ^self delegator perform: aMessage selector
            withArguments: aMessage arguments
```

Mit dieser kleinen, wenn auch weitreichenden Erweiterung ist jedoch nur ein Teil dessen bewältigt, was für die Implementierung von Delegationsbeziehungen nötig ist. Darüber hinaus ist es wünschenswert, die Etablierung und Pflege solcher Beziehungen zu unterstützen. Dazu gehören das Ausschließen von Anomalien, die beim Anlegen oder Löschen von Beziehungen entstehen können, sowie - durchaus damit zusammenhängend - die Berücksichtigung von Kardinalitätsbedingungen. Wir haben dazu zwei alternative Lösungen realisiert. Dabei handelt es sich einerseits um ein Entwurfsmuster, der in erster Linie eine Erweiterung der eigentlichen Smalltalk-Entwicklungsumgebung darstellt. Die Verwendung des Entwurfsmusters erfolgt durch die komfortable Spezialisierung weniger Klassen. Die zweite Lösung besteht darin, aus einem mit MEMO Center verwalteten Objektmodell Code für die Realisation und Verwaltung von Delegationsbeziehungen zu generieren. In Abbildung 3 ist das Objektmodell des Entwurfsmusters dargestellt. Die Methoden zum Hinzufügen und Löschen von Assoziationen sowie diejenigen zum Initialisieren und Freigeben (für den Garbage Collector) der beteiligten Objekte sind so implementiert, daß die Ausführung der notwendigen Folgeoperationen garantiert ist. Dabei wird davon ausgegangen, daß jeweils nur ein Benutzer auf einem Image arbeitet. Das Entwurfsmuster hat sich bei der Realisation des Informationssystems des Instituts im Rahmen des Projekts MORE ("Migrating Objects in a Research Environment") nachhaltig bewährt. Es kann zusammen mit einem ausführlichen Beispiel, das seine Verwendung verdeutlicht, über das World Wide Web (http://www.uni-koblenz.de/~iwi/) frei bezogen werden.

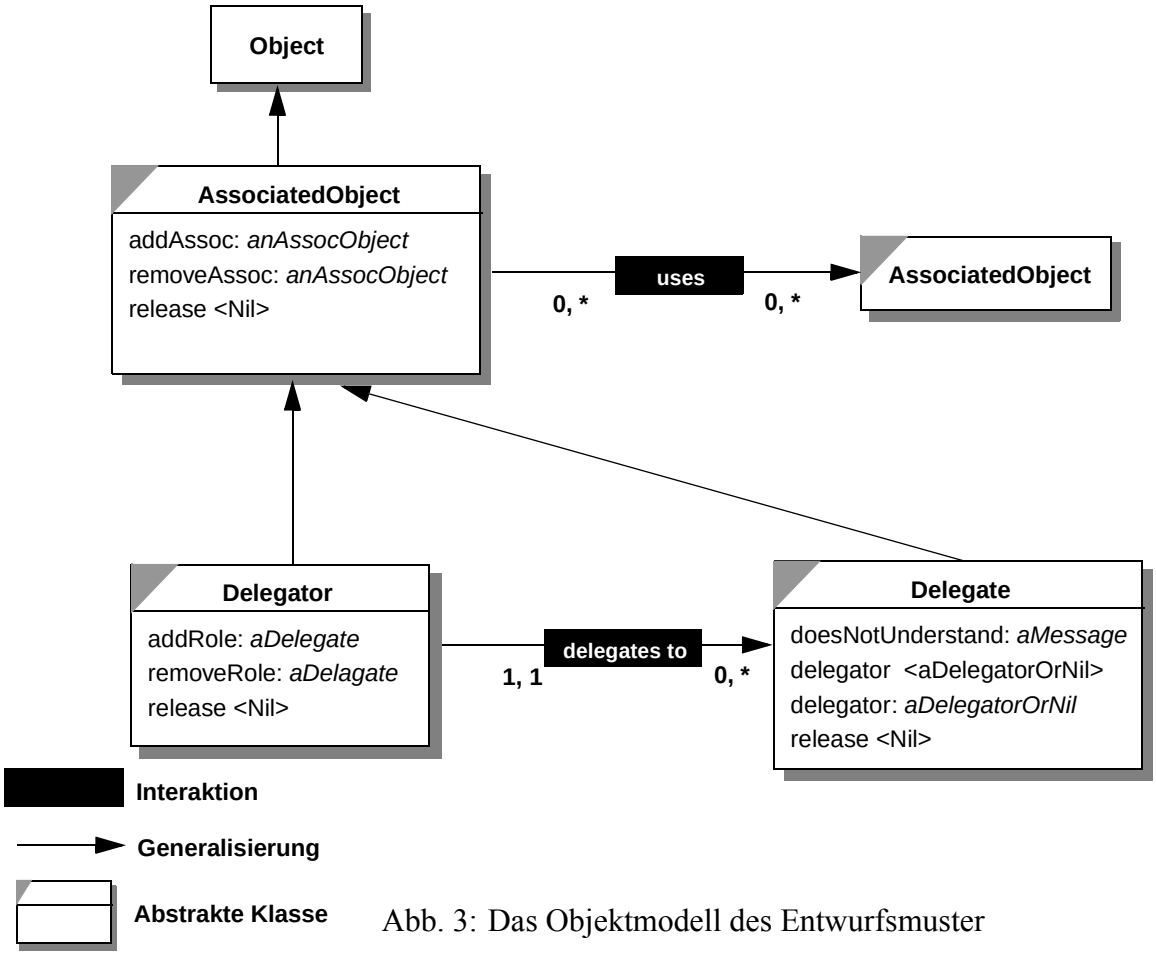

Abb. 3: Das Objektmodell des Entwurfsmuster

Wenn in MEMO Center eine Delegationsbeziehung angelegt werden soll, wird zunächst geprüft, ob das beabsichtigte Rollen-Objekt einer Unterklasse der abstrakten Klasse MEMO-Delegate angehört und dies gleichzeitig für das Rolleninhaber-Objekt nicht gilt. Falls dies der Fall ist, kann die Kardinalität des Rollen-Objekts festgelegt werden. Anschließend werden - zunächst nur im Modell - die entsprechenden Zugriffsdienste generiert. Daraus kann dann Smalltalk-Code generiert werden. Die entsprechenden Methoden sind im Unterschied zu denen, die das Entwurfsmuster bereitstellt, an den Namenskontext der jeweiligen Beziehung angepaßt. So heißt etwa eine Methode, die das Hinzufügen eines Rollen-Objekts der Klasse Programmierer erlaubt, nicht addRole: aRole, sondern addProgrammer: aProgrammer (wobei natürlich das Entwurfsmuster eine entsprechende - allerdings manuell durchzuführende - Spezialisierung ebenfalls erlaubt). Wie auch im Entwurfsmuster garantieren die generierten Methoden die konsistente Pflege von Beziehungen. Optional erzeugt MEMO Center zudem Code für einen transaktionsorientierten Zugriff auf persistente Objekte, die von einer objektorientierten Datenbank (Gemstone®) verwaltet werden.

Da ein Rollen-Objekt die entsprechenden Nachrichten an das Rolleninhaber-Objekt weiterleitet, verhält es sich - von außen betrachtet - wie das Rolleninhaber-Objekt. In den Fällen, in denen im Code die Zugehörigkeit eines Objekts zu einer Klasse bzw. einer derer Unterklassen (in Smalltalk: "isKindOf:") überprüft werden soll, um eine bestimmte Objektsemantik zu garantieren, wird häufig (aber nicht immer) auch ein Rollen-Objekt, das transparent an ein geeignetes Rolleninhaber-Objekt weiterleitet, akzeptabel sein. Um dies zu ermöglichen, ist an solchen Stellen ergänzend zu prüfen, ob ein Rollen-Objekt vorliegt. Falls dies der Fall ist, kann mittels der Nachricht "behavesLike:" die "kindOf:"-Nachricht an das zugehörige Rolleninhaber-Objekt weitergeleitet.

5. Voraussetzungen für die Verwendung von Delegation

Delegation kann eine sinnvolle Alternative zu Vererbung sein, sie ist jedoch keinesfalls geeignet, Vererbung generell zu ersetzen. Grundsätzlich muß die Frage, ob Delegation ein angemessenes Konzept zur Abbildung eines realen Sachverhalts darstellt, vor dem Hintergrund einer sorgfältigen Analyse der jeweiligen Diskurswelt geklärt werden. Zur Unterstützung dieser Analyse seien folgende Anhaltspunkte genannt:

- Mögliche Mehrdeutigkeiten von Generalisierungen ("is a") sollten sorgfältig geprüft werden. Dazu ist es hilfreich, sich zu fragen, ob man eine "is a"-Beziehung nicht auch "repräsentiert" oder "fungiert als" benennen könnte. Wenn das der Fall ist, hat man zumindest eine mögliche Delegation gefunden.
- Delegation entspricht weitgehend dem alltagsweltlichen Rollenkonzept. Wenn in der Diskurswelt Ausdrücke wie "Aufgabe", "Aufgabenträger", "Stelle", "Stelleninhaber", "arbeitet als", "dient als" etc. vorkommen, ist dies ein Indiz für die Anwendbarkeit von Delegation.
- Einige Entitäten der realen Welt sind per se Kandidaten für die Abbildung auf Rolleninhaber-Objekte: Personen, Organisationen oder vielseitig verwendbare Maschinen. Es ist sinnvoll, solche Objekte bereits während der Analyse entsprechend zu kennzeichen - indem man sie etwa einer dazu bestimmten Kategorie zuordnet.

Daneben ist zu berücksichtigen, daß die Anwendung von Delegation mit einschränkenden Bedingungen verbunden sein kann. So mag es in bestimmten Domänen Rollen geben, die sich gegenseitig ausschließen. Neben dem bereits erwähnten Beispiel aus dem Versicherungsbereich könnte in einer Hochschule etwa die Bedingung gelten, daß ein Professor nicht gleichzeitig Student sein darf. Mögliche Verletzungen impliziter Integritätsbedingungen durch die Anwendung von Delegation können während der Analyse dadurch ermittelt werden, daß alle Kombinationen generiert werden und durch einen sachkundigen Betrachter auf Plausibilität geprüft werden. Falls auf diese Weise ein Konflikt zwischen verschiedenen Rollen entdeckt wird, ist das Objektmodell um eine entsprechende Integritätsbedingung zu erweitern.

Selbst in den Fällen, in denen Delegation eine besonders natürliche Rekonstruktion der Realität verspricht, sollte man sich vor einer übertriebenen Anwendung dieses Konzepts hüten. So ist es durch die in 3 genannten Integritätsbedingungen nicht ausgeschlossen, daß die Instanzen einer Rollen-Klasse den Instanzen verschiedener Rolleninhaber-Klassen zugeordnet werden können - z.B. eine Instanz der Klasse Fahrzeugführer einer Instanz der Klasse Person oder einer Instanz der Klasse Systemsoftware. Trotz der damit verbundenen Einschränkung der Flexibilität empfehlen wir nachhaltig die Einschränkung aller Rollen-Klassen auf eine mögliche Rolleninhaber-Klasse. Eine entsprechende Integritätsbedingung ist deshalb sowohl in MEMO Center wie auch in dem skizzierten Entwurfsmuster enthalten. Der Grund für eine solche Einschränkung liegt auf der Hand. Je mehr Rolleninhaber-Klassen mit einer Rollen-Klasse assoziiert werden dürfen, desto verwirrender wird das Objektmodell für den Betrachter: Ein transparenter Zugriff auf die Schnittstelle des Rolleninhaber-Objekts ist nur solange erfreulich wie man weiß, wie die insgesamt verfügbare Schnittstelle aussieht. Darüber hinaus würde die mögliche Zuordnung zu mehreren Rolleninhaber-Klassen erhebliche Konsequenzen für die maschinelle Prüfbarkeit der Konsistenz eines Objektmodells nach sich ziehen: Ein im Modell referenzierter Dienst mag zur Laufzeit verfügbar sein - oder auch nicht. Aus ähnlichen Gründen ist davon abzuraten, daß eine Rollen-Klasse auch als Rolleninhaber-Klasse fungieren darf. In diesem Sinn warnen GAMMA ET AL. (S. 21) - die im übrigen Delegation nachhaltig empfehlen - vor einer leichtfertigen Verwendung: "Delegation is a good design choice only when it simplifies more than it complicates".

Literatur

Blair, G.S.; Gallagher, J.J.; Malik, J.: Genericity vs. Inheritance vs. Delegation vs. Conformance vs .... In: Journal of Object-Oriented Programming. Sep./Oct., 1989, S. xx-xx

Booch, G.: Object-Oriented Analysis and Design with Applications. 2. Aufl., Redwood City 1994

Frank, U.: MEMO: A Tool Supported Methodology for Analyzing and (Re-) Designing Business Information Systems. In: Ege, R.; Singh, M.; Meyer, B. (Ed.): Technology of Object-Oriented Languages and Systems. Englewood Cliffs 1994, S. 367-380

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Reading/Mass. etc.: Addison-Wesley 1995

Jacobson, I.; Christerson, M.; Jonsson, P.; Overgaard, G.: Object-Oriented Engineering. A Use Case Driven Approach. Reading/Mass. 1992

Johnson, R.E.; Zweig, J.: Delegation in C++. In: Journal of Object-Oriented Programming. Vol. 4, No. 11, S. 22-35

Kathuria, R. and Subramaniam, V.: Assimilation: A New and Necessary Concept for an Object Model. REPORT ON OBJECT ANALYSIS & DESIGN, Vol. 2, No. 5, 1996, S. 36-39

Lieberman, H.: Using prototypical objects to implement shared behavior in object-oriented systems. In: OOPSLA, 1986, S. 214-223

Kim, W.; Lochovsky, F. (Hg.) Object-Oriented Concepts, Applications, and Databases. New York 1989

Rumbaugh, J. et al.: Object Oriented Modeling and Design. Englewood Cliffs, NJ 1993

Sciore E.: Object specialization. In: ACM Transactions on Office Information Systems, Vol. xx, No. xx, 1989, S.

Smith, R.B.; Ungar, D.: Programming as an Experience. The Inspiration for Self. In: Proceedings of the ECOOP 95

Stein, L.A.: Delegation is Inheritance; oopsla, 1987

fehlende Literaturangaben ergänzen (auch im Text fehlen Seitenangaben)
Referenz zu SELF, Referenz zu Class Migration, Unified Methode
Verfasser im Text kennzeichnen, Abb. 3 fertigstellen, Text zu 4 ergänzen