Ebenen der Abstraktion und ihre Abbildung auf konzeptionelle Modelle - oder: Anmerkungen zur Semantik von Spezialisierungs- und Instanzierungsbeziehungen

Ulrich Frank
Institut für Wirtschaftsinformatik
Universität Koblenz

1 Einleitung

Die Analyse realweltlicher Domänen lässt mitunter eine Reihe verschiedener Abstraktionsmöglichkeiten erkennen. Deren Abbildung mit Hilfe gängiger Sprachen der konzeptionellen Modellierung führt teilweise zu erheblichen Problemen, die kaum thematisiert werden. So kann die scheinbar offenkundige Bedeutung von Spezialisierungsbeziehungen zu kontra-intuitiven Konsequenzen führen, die die Qualität von Informationssystemen nachhaltig gefährden. Die Ursache solcher oft ungewollten Modellierungsanomalien liegt darin, dass die nach verbreiteter Meinung so natürlichen Begriffe 'Klasse' und 'Objekt' in Programmier- wie auch in Modellierungssprachen in einer Bedeutung verwendet werden, die in subtiler Form von deren Verwendung in der Umgangssprache wie auch in der Mathematik abweicht. Daneben empfiehlt die Analyse realweltlicher Phänomene mitunter eine Differenzierung von mehr Abstraktionsebenen als durch gängige Modellierungssprachen darstellbar. Dies führt zu der unangenehmen Konsequenz, dass eine als angemessen erachtete konzeptionelle Beschreibung nicht semantisch entsprechend in einem konzeptionellen Modell umgesetzt werden kann – ebenfalls mit unerfreulichen Konsequenzen für die Qualität des zu erstellenden Informationssystems. In einigen Anwendungsbereichen der (Meta-) Modellierung gibt es gleichzeitig Bedarf an Instanzierungs- und an Spezialisierungsbeziehungen. Sie können jedoch nicht gemeinsam verwendet werden, was zu erheblichen Herausforderungen führt.

Im folgenden Beitrag werden die skizzierten Probleme zunächst dargestellt und analysiert. Die Betrachtung zeigt, dass der Stand der Kunst bei Modellierungs- und Implementierungssprachen noch nicht befriedigend ist. Während dieser Umstand zu weiterer Forschung in diesen Bereichen rät, bleibt zu klären, wie man den dargestellten Probleme mit heute verfügbaren Sprachen begegnen kann. Zu diesem Zweck werden pragmatische Lösungsansätze vorgestellt und diskutiert.

2 Kontra-intuitive Semantik von Spezialisierungsbeziehungen

Generalisierung ist ein zentrales Abstraktionskonzept des menschlichen Denkens: Wir abstrahieren von der Varianz der Spezialfälle und konzentrieren uns auf einen gemeinsamen, für bestimmte Betrachtungen als wesentlich erachteten Kern. Die Umkehrung der Generalisierung, die Spezialisierung, ist ebenfalls ein gängiges Konzept zur Strukturierung unserer Weltsicht: Immer dann, wenn uns ein genereller Begriff für eine spezielle Betrachtung – etwa um verschiedenartige realweltliche Objektarten zu unterscheiden - nicht hinreicht, erlaubt uns die Einführung einer Spezialisierung eine differenziertere Beschreibung. Dabei entsteht ein spezialisierter Begriff durch das Hinzufügen weiterer Eigenschaften. Die durch den Oberbegriff festgelegten Eigenschaften gelten also weiterhin im Unterbegriff – sie werden „geerbt". Im menschlichen Denken wird diese Regel allerdings nicht immer konsequent angewendet: Mitunter werden Generalisierungen gebildet, obwohl man weiß, dass nicht für alle Spezialisierungen alle Eigenschaften der Generalisierung gelten. Ein bekanntes Beispiel dafür ist die Aussage 'Vögel können fliegen'. Wenn man einer Vogelart begegnet, für die diese Aussage nicht zutrifft, gerät das Weltbild nicht ins Wanken. Anders ist dies bei entsprechenden

Formalisierungen. Sie führen zu Widersprüchen. Im Zusammenhang mit objektorientierten Programmiersprachen ist hier an die Redefinition geerbter Eigenschaften zu denken, die zu dem berüchtigten Ko- bzw. Kontravarianzproblem ([Meye97], S. 621 ff.) führen. In der Künstliche Intelligenz Forschung gab es einige Ansätze, durch mehrwertige Logiken ein „non-monotonic reasoning“ formal zu rekonstruieren. Wir werden diese Problematik im Folgenden allerdings ausklammern und allein solche Generalisierungen betrachten, für die keine Ausnahmen zu berücksichtigen sind.

2.1 Ein Beispiel

Generalisierung und Spezialisierung sind Konzepte, die für unseren Umgang mit einer komplexen Welt unerlässlich sind. Dementsprechend intuitiv scheint die Bedeutung von Generalisierungs- bzw. Spezialisierungsbeziehungen in der objektorientierten Modellierung. Betrachten wir dazu folgendes Beispiel: Um die Implementierung eines Informationssystems für eine Universität vorzubereiten, soll ein Objektmodell erstellt werden, in dem u. a. Studenten, wissenschaftliche Mitarbeiter und Professoren zu repräsentierten sind. Es liegt auf der Hand, dass es sich dabei jeweils um Personen handelt. Es bietet sich also an, die Klasse ’Person’ als Generalisierung der Klassen ’Student’, ’Assistent’ und ’Professor’ einzuführen. Die weitere Analyse der Domäne ergibt, dass auch Programmierer, Dozenten und Administratoren abzubilden sind. Auch dabei handelt es sich offensichtlich um Spezialisierungen von ’Person’. Es ist allerdings möglich, dass ein Programmierer gleichzeitig Student oder Assistent ist. Einfache Spezialisierungsbeziehungen erlauben es nicht, solche Zusammenhänge abzubilden. Demgegenüber scheint mehrfache Spezialisierung bzw. ’Mehrfachvererbung’ ein angemessener Ansatz zu sein, um den dargestellten Sachverhalt abzubilden. Abb. 1 zeigt ein Klassendiagramm, in dem entsprechende Spezialisierungsbeziehungen verwendet werden.

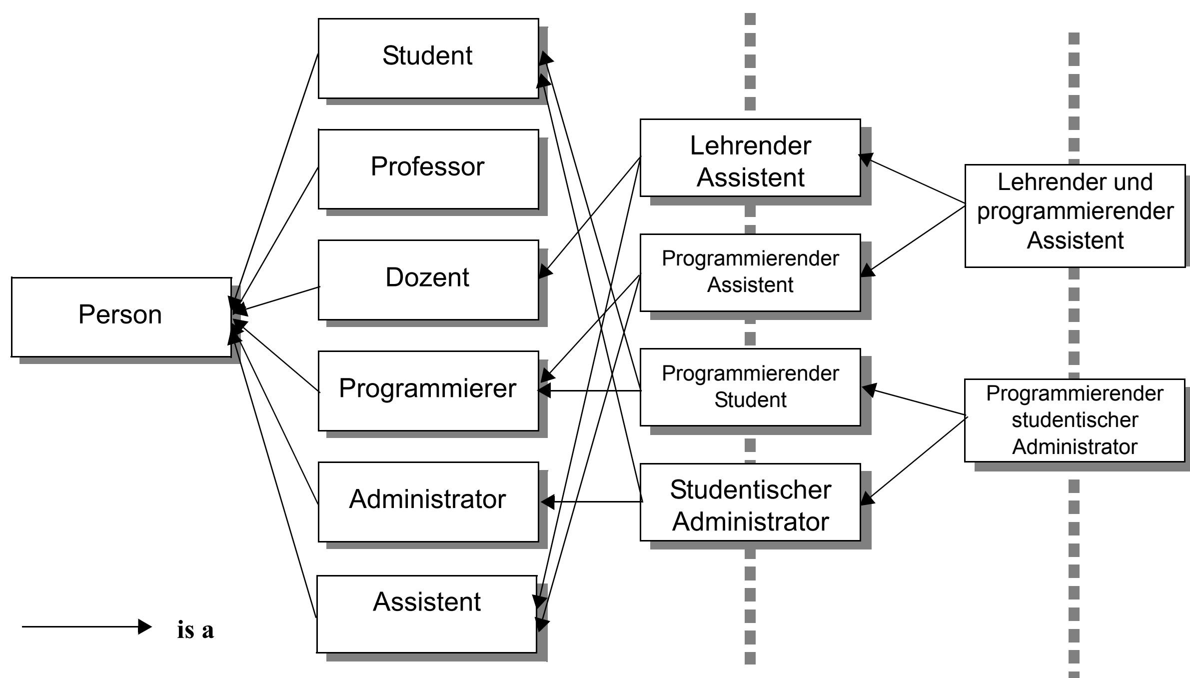

Abb. 1: Klassendiagramm mit mehrfacher Spezialisierung

Bei näherer Betrachtung der spezialisierten Klassen fällt zunächst auf, dass die Klassenbezeichner nicht ’natürlich’ sind, in dem Sinn, dass sie gängigen Begriffen in der Domäne entsprechen. Wir reden nicht von einem ’Programmierenden studentischen

Administrator'. Nun könnte man diesen Umstand allein für einen Ausdruck eingeschränkter Verständlichkeit des Klassendiagramms halten. Tatsächlich handelt es sich hier aber um eine schwerwiegende Anomalie: Die Bedeutung des Modells weicht erheblich von der indentierten, naheliegenden Bedeutung ab. Dies gilt jedenfalls dann, wenn man - wie dies häufig in der Software-Entwicklung geschieht - die Semantik von Spezialisierungsbeziehungen in objektorientierten Programmiersprachen zugrunde legt. Auch wenn objektorientierte Modellierungssprachen zumeist keine präzise Semantik für Spezialisierungsbeziehungen festlegen, wird bei der Interpretation bzw. Transformation von Klassendiagrammen zumeist die Semantik verwendet, die in Programmiersprachen üblich ist. Das drückt sich etwa darin aus, dass die Klassen des Entwurfsmodells üblicherweise in korrespondierende Klassen der eingesetzten Programmiersprache transformiert werden. Gängige UML-Werkzeuge verfügen über Code-Generatoren, die eine entsprechende Umsetzung des Klassendiagramms in Code vorsehen [KiFr02].

Die erwähnte Anomalie drückt sich darin aus, dass Änderungen der aus einem solchen Modell instanzierten Objekte mit umständlichen, wenig natürlich erscheinenden Operationen verbunden sind, die zudem die Integrität eines Informationssystems gefährden. Betrachten wir dazu ein Objekt der Klasse 'Student'. Sobald dieser Student hinreichende Programmierkenntnisse erworben hat, kann er der Klasse 'Programmierender Student' zugeordnet werden. In unserer alltagsweltlichen Sicht der Dinge ist eine solche Änderung wenig spektakulär: Der betrachtete Mensch bleibt nach wie vor Student und ist ergänzend dazu auch ein Programmierer. In der objektorientierten Implementierung des Modells gestaltet sich diese Änderung sehr viel aufwendiger: Zunächst ist eine neue Instanz der Klasse 'Programmierender Student' anzulegen. Dann ist der Zustand der korrespondierenden Instanz der Klasse 'Student' auf diese Instanz zu übertragen - einschließlich aller Referenzen auf diese Instanz! Schließlich ist die Instanz der Klasse 'Student' aus dem System zu entfernen. Es liegt auf der Hand, dass eine solche Operation eine ernsthafte Gefährdung der Integrität eines Informationssystems darstellt. Die Ursache für diese unterschiedlichen Interpretationen von Spezialisierungsbeziehungen liegt in den jeweils unterschiedlichen Klassenbegriffen. Während wir in der Alltagswelt implizit von einem extensionalen Klassenbegriff ausgehen, wird in objektorientierten Programmiersprachen zumeist ein intensionaler Klassenbegriff verwendet. Eine Klasse wird, wie bei der Beschreibung von Begriffsinhalten, intensional durch Eigenschaften (Attribute, Operationen, vertragliche Zusicherungen etc.) definiert. Objekte werden gleichsam aus dieser Schablone instanziert. In extensionaler Wendung wird eine Klasse als eine Menge gleichartiger Objekte definiert. Dies entspricht auch dem in der Logik üblichen Klassenbegriff, wo 'Klasse' und 'Menge' zumeist synonym verwendet werden.[1]

Aus dem Blickwinkel der Logik - wie auch im alltagsweltlichen Verständnis - markieren intensionaler und extensionaler Klassenbegriff lediglich zwei Perspektiven auf den gleichen Gegenstand. In der objektorientierten Software-Entwicklung ist allerdings ein bedeutsamer Unterschied zwischen beiden Begriffswendungen zu berücksichtigen. Der intensionale Klassenbegriff objektorientierter Programmiersprachen impliziert, dass ein Objekt jeweils Instanz einer und nur einer Klasse ist. Der extensionale Klassenbegriff erlaubt es, dass ein Objekt gleichzeitig mehreren Klassen angehören kann. In diesem Fall entspricht Spezialisierung der *Subordination* [Wolt96], d. h. eine spezialisierte Klasse ist eine echte Teilmenge einer oder mehrerer Oberklassen. Eine Klasse, die aus mehreren Oberklassen spezialisiert wird, ist die Schnittmenge der Oberklassen. Wendet man den extensionalen Klassenbegriff auf unser Beispiel an, reicht es hin, sich auf die Klassen zu beschränken, die unmittelbar von der Klasse 'Person' spezialisiert werden. Es ist nicht zwingend nötig, weitere

1. Es gibt Axiomensysteme der Mengenlehre, in denen eine Menge nur dann eine Klasse ist, wenn sie selbst Element einer anderen Menge ist (vgl. [Lore84]).

Subklassen zu bilden, da ein Objekt gleichzeitig mehreren Klassen angehören kann. Die mengenorientierte Darstellung in Abb. 2 verdeutlicht diesen Umstand. In Datenbankschemata wird i. d. R. ein extensionaler Klassenbegriff verwendet - was allerdings in dem hier betrachteten Zusammenhang nur dann von Bedeutung ist, wenn es möglich ist, Spezialisierungsbeziehungen auszudrücken. Wenn ein Student Programmierkenntnisse erwirbt, wird das entsprechende Objekt nicht gelöscht, sondern lediglich auch der Klasse 'Programmierer' zugeordnet (*Subsumption* [Wolt96]).

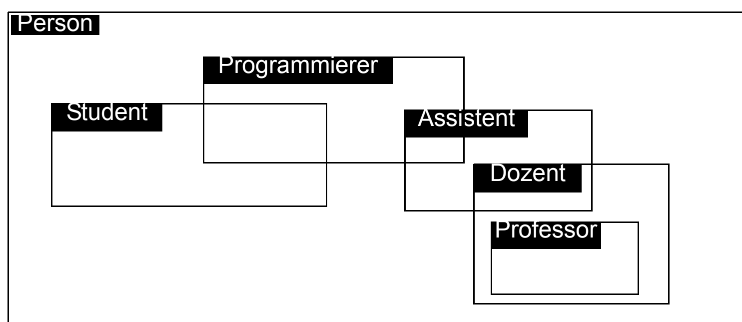

Abb. 2: Klassen in extensionaler Darstellung

Vor dem Hintergrund dieser Überlegungen ist der Schluss naheliegend, bei der Modellierung die im Hinblick auf gut nachvollziehbare Konzeptualisierungen angemessener erscheinende extensionale Semantik von 'Klasse' zu verwenden. Tatsächlich wird in der aktuellen Version 1.4 der UML wird offen gelassen, ob eine Klasse intensional oder extensional zu interpretieren ist. Ein Objekt kann mehreren Klassen zugeordnet werden[1]. Leider ist ein solches Vorgehen mit einem gravierenden Nachteil verbunden. Da die Implementierung von Klassendiagrammen i. d. R. mit objektorientierten Programmiersprachen erfolgt, würde die Verwendung eines extensionalen Klassenbegriffs in der Modellierung zu einem semantischen Bruch führen, der der gewünschten Integration von konzeptionellen Modellen und Code entgegenliefe.

2.2 Ein Lösungsansatz

Um zu verhindern, dass Spezialisierungsbeziehungen in Klassendiagrammen so verwendet werden, dass sie der eigentlichen Intention des Modellierers nicht entsprechen, ist zunächst Aufklärung nötig: Modellierer müssen sich der möglicherweise kontra-intuitiven Semantik von Spezialisierungsbeziehungen bewußt sein. Ein Blick in gängige Lehrbücher zur objektorientierten Modellierung zeigt, dass hier Nachholbedarf besteht, da dieses Problem i. d. R. ignoriert wird. Nun ist in die Semantik von Spezialisierungsbeziehungen in objektorientierten Systemen ein ausführlicher erforschter Gegenstand. Einige Autoren warnen ausdrücklich vor der vorschnellen Verwendung von Spezialisierung auf der Basis eines intensionalen Klassenbegriffs (Szyperski, Meyer). Gleichzeitig ist in sog. klassenlosen Programmiersprachen ein von der Spezialisierung bzw. Vererbung abweichendes Konzept zur Förderung der Wiederverwendung vorgesehen. Ein Objekt, das die Eigenschaften eines anderen Objekts erben soll, wird so mit diesem assoziiert, dass eingehende Methodenaufrufe, die nicht im Protokoll des erbenden Objekts enthalten sind, transparent an das vererbende Objekt weitergeleitet werden. Dabei ist allerdings zu berücksichtigen, dass es sich hierbei nicht um Spezialisierung von Klassen handelt. Eine differenzierte Betrachtung von klassenlosen Programmiersprachen findet sich in [Male95].

1. Gleichzeitig wird allerdings die Semantik von Spezialisierungsbeziehungen nicht spezifiziert.

Ein pragmatischer Ansatz, um Spezialisierungsbeziehungen mit gängigen Modellierungssprachen sinnvoll auszudrücken und gleichzeitig den semantischen Bruch zwischen konzeptionellen Modellen und Code zu vermeiden, ist die Einführung einer ergänzenden Delegationsbeziehung. Dazu wird auf das Konzept einer 'Rolle' zurückgegriffen. Eine Klasse wird über eine Delegationsbeziehung als Rolle einer anderen Klasse, des Rolleninhabers, definiert. Dadurch erbt sie nicht die Eigenschaften der Rolleninhaber-Klasse. Vielmehr sind ihre Objekte Rollen korrespondierender Rolleninhaber-Objekte. Wenn die Rollenobjekte eine Nachricht erhalten, die in ihrem Protokoll nicht enthalten ist, wird diese transparent an den Rolleninhaber weitergeleitet - ähnlich wie bei klassenlosen Programmiersprachen. Auf diese Weise wird also nicht nur das Protokoll des Rolleninhaber-Objekts 'vererbt', sondern auch sein Zustand: Die Nachricht 'nachName', die an ein Objekt der Klasse 'Student' gesendet wird, wird an ein assoziiertes Objekt der Klasse 'Person' weitergeleitet und der dort abgelegte Name wird zurückgereicht. Abb. 3 zeigt ein Modell unserer Beispieldomäne, in dem Delegation verwendet wurde.

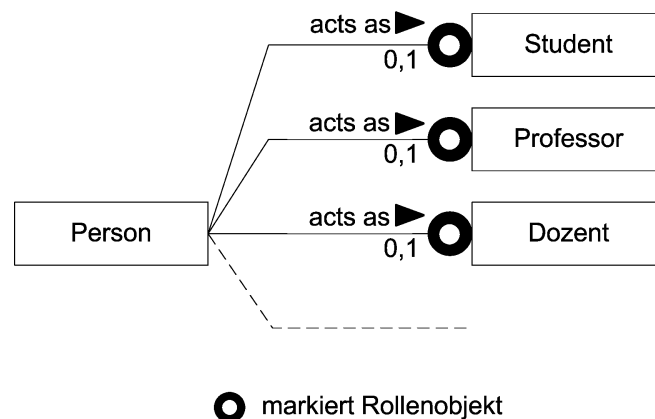

Abb. 3: Beispielmodell unter Verwendung von Delegationsbeziehungen

Eine Spezifikation der Semantik von Delegation als Modellierungskonzept findet sich in [Fran00]. Eine angemessene Verwendung dieses Konzepts empfiehlt die Berücksichtigung einiger Richtlinien:

Grundsätzlich gilt, dass man sich nicht durch den überladenen Bezeichner 'is a' verwirren lassen sollte.

- Falls eine Beziehung zwischen zwei Klassen sinnvoll 'repräsentiert' oder 'fungiert als' genannt werden kann, handelt es sich um einen Kandidaten für Delegation.
- Falls eine Generalisierung/Spezialisierung nicht denknotwendig für die gesamte Lebenszeit eines Systems gelten muss, kann Delegation eine bessere Option sein. Beispiel: Ein Professor muss nicht notwendigerweise ein Angestellter sein.
- Es gibt typische Kandidaten für Rolleninhaber-Klassen: Personen, Organisationen, Maschinen.

Die Einführung eines zusätzlichen Konzepts wie Delegation hilft dem versierten Modellierer, den Unzulänglichkeiten einer Spezialisierungssemantik entgegenzuwirken, die aus einem intensionalen Klassenbegriff resultieren. Gleichzeitig kann auf diese Weise ein semantischer Bruch zur Implementierungssprache weitgehend vermieden werden. Dennoch bleibt ein solcher Ansatz unbefriedigend, denn er beseitigt nicht die Problemursache, die in einem

Klassenkonzept liegt, das vom bewährten alltagsweltlichen - und auch in den Formalwissenschaften verbreiteten - Klassenbegriff abweicht.

3 Unzureichende Abstraktionsebenen

In der Software-Entwicklung werden i. d. R. drei grundlegende Abstraktionsebenen unterschieden. Die Meta-Ebene legt die Sprache(n) zur Spezifikation der darunter liegenden Schema- oder Typ-Ebene fest. Ein solches Schema, z. B. ein Klassendiagramm, kann dann als Instanz eines korrespondierenden Metamodells angesehen werden. Die Instanzenebene schließlich dient der Beschreibung von Objekten, die aus einem Schema instanziert wurden. In der konzeptionellen Modellierung kann i. d. R. nur eine Ebene, nämlich die Schema-Ebene, manipuliert werden. Die Instanzebene ergibt sich daraus implizit. Ergänzend zu diesen grundlegenden Abstraktionsebenen kann das Abstraktionsniveau von Teilen eines Modells durch Konzepte wie Verkapselung, Polymorphie oder Generalisierung/Spezialisierung variiert werden. Die auf diese Weise verfügbaren Abstraktionsmöglichkeiten reichen allerdings häufig nicht aus, um realweltliche Sachverhalte angemessen zu beschreiben. Betrachten wir dazu die Modellierung von Ressourcen, also etwa von peripheren Geräten. In Abb. 4 sind verschiedene Abstraktionsebenen dargestellt, die wir im täglichen Umgang mit Ressourcen ohne nennenswerte Probleme differenziert werden können.

Abstraktionsniveau	Beispiel
Meta Modellierungssprache	Resource_Type: name String — 1,1 … 0,* — Feature_Type: name String, type String
Typ Ressourcentyp	Drucker: bezeichnung String, seitProMin Integer, aufloesung Resolution
Initialisierter Typ konkreter Ressourcentyp	Drucker: bezeichnung HP-8L, seitProMin 14, aufloesung 1200x1200
Spezialisierter Typ Spezialisierter Ressourcentyp	Laserdrucker: bezeichnung String, seitProMin Integer, aufloesung Resolution, seitProKass Integer
"Clone" konkreter Ressourcentyp als Erweiterung	Drucker: bezeichnung HP-8LS, seitProMin 14, aufloesung 1200x1200, beidSeitig true
Konkrete Instanz konkrete Ressourceninstanz	Drucker: bezeichnung HP-8LS, seitProMin 14, aufloesung 1200x1200, beidSeitig true, geraeteNr AD-6721

Abb. 4: Mögliche Abstraktionsebenen bei der Modellierung von Ressourcen

Die Analyse der in Abb. 4 dargestellten Abstraktionsebenen macht zweierlei deutlich. So ist es einerseits leicht vorstellbar noch weitere Ebenen einzuführen. Beispielsweise könnten weitere Druckerklassen wie z.B. „Seitendrucker" eingeführt werden. Im Hinblick auf die Verwendung entsprechender Konzepte zeigt sich andererseits, dass die Unterscheidung zwischen Spezialisierung und Instanzierung nicht immer intuitiv ist: Ist ein Laserdrucker eine Instanz oder eine Spezialisierung der Klasse 'Drucker'? Bei präziser Unterscheidung zwischen Typ und Instanz wäre immer dann eine Spezialisierung zu wählen, wenn unter 'Laserdrucker' ein Typ und nicht eine konkrete Instanz verstanden werden soll. Für den Systementwurf ist dies jedoch nicht zwingend. Vielmehr ist es möglich, Typen über Attribute zu differenzieren, im Beispiel etwa über ein Attribut 'Druckertyp'. Insofern hängt die Wahl zwischen Instanzierung und Spezialisierung auch von den Anforderungen an das jeweils zu entwerfende System ab. Eine Spezialisierung bietet sich an, wenn alle folgenden Anforderungen *gemeinsam* zutreffen:

- Das Resultat soll ein Typ (bzw. eine Klasse) sein - und zwar auf der gleichen Abstraktionsebene wie der Obertyp (bzw. die Oberklasse). Dies ist im Hinblick auf den Systementwurf immer dann sinnvoll, wenn sich ein solcher Typ im Hinblick auf das Systemverhalten signifikant von anderen Typen unterscheidet. Wenn Typen auf Attribute abgebildet werden, ist eine statische Überprüfung dieser spezifischen Eigenschaften nicht möglich - mit den bekannten negativen Folgen für die Integrität des Systems.
- Die Varianz des spezialisierten Typs gegenüber dem Obertyp beschränkt sich auf Erweiterungen der Eigenschaftsmenge. Eine Redefinition geerbter Eigenschaften ist dabei i. d. R. auszuschließen - wegen der Implikationen des bereits erwähnten Ko- bzw. Kontravarianzproblems.
- Es wird eine Abstraktion über Instanzen angestrebt. Die Typen repräsentieren jeweils Instanzen, d. h. die Eigenschaften der Typen bilden Eigenschaften ab, die für jede Instanz gelten.

Im Unterschied dazu ist eine Instanzierung immer dann angezeigt, wenn das Resultat eine Instanz sein soll. Dieser Fall spielt allerdings in der konzeptionellen Modellierung eine eher untergeordnete Rolle, weil Instanzen selten beschrieben werden. Andernfalls sollten die beiden folgenden Anforderungen erfüllt sein:

- Die Varianz der 'Instanzen' sollte nicht allein auf eine Erweiterung von Eigenschaften beschränkt sein. Eine Instanzierung von Typen aus Metamodellen erlaubt eine tendenziell eine deutlich höhere Gestaltungsbandbreite. Die 'Instanzen' liegen auf einer geringeren Abstraktionsebene als die Typen der Metamodelle.
- Es wird auf der oberen Ebene eine Abstraktion über Typen angestrebt. Die Typen der Metamodelle beschreiben Eigenschaften von Typen, nicht von Instanzen (s. 4).

Allerdings legen die Beispiele die Frage nahe, ob überhaupt alle Abstraktionsebenen für die konzeptionelle Modellierung benötigt werden. Zunächst kann man tendenziell feststellen, dass die Beschreibung konkreter Instanzen i.d.R. nicht Gegenstand der konzeptionellen Modellierung ist. Vielmehr soll ein konzeptionelles Modell ja bewusst von den Teilen des abgebildeten Realitätsbilds abstrahieren, die absehbaren Änderungen unterliegen. Ähnliches gilt für konkrete Ressourcentypen. Im Einzelfall mag es allerdings wichtig sein, konkrete Ressourcentypen zu erfassen (etwa einen bestimmten Rechnertyp), weil er spezifische Implikationen für die Gestaltung von Informationssystemen mit sich bringt, die nicht vernachlässigt werden dürfen. Während die Abbildung konkreter Instanzen nicht zu den Anforderungen an eine konzeptionelle Modellierungssprache zählt, sind allerdings Instanzen dennoch in zweifacher Hinsicht zu berücksichtigen. So sollten Sprachkonzepte vorhanden sein, die die Identität von Instanzen auszudrücken erlauben. Wenn beispielsweise zwei Teilprozessen eines Workflow-Typs ein Sachbearbeiter zugeordnet ist, sollte es möglich sein,

auszudrücken, ob es sich dabei jeweils um die gleiche Instanz handeln muss – bzw. nicht handeln darf. Um eine differenzierte Modellierung zu unterstützen, sollte eine Sprache zur konzeptionellen Modellierung in jedem Fall erlauben, das Abstraktionsniveau der angebotenen Konstrukte explizit zu machen.

Gängige Modellierungssprachen - wie auch Implementierungssprachen - bieten hier offensichtlich keine angemessene Unterstützung. Die einzige Möglichkeit, die dem Modellierer bzw. Software-Entwickler bleibt, ist die Überladung von Abstraktionsebenen. So können etwa einzelne Klassen in einem Datenbankschema als Metaklassen interpretiert werden. Ein Beispiel dafür findet sich in [Fran02]. Ein solcher Ansatz bleibt allerdings unbefriedigend, weil er einerseits zu kaum verständlichen Modellen führt und andererseits erfordert, dass die jeweils vorgesehenen Interpretationen implementiert werden müssen.

4 Abstraktionskonflikte bei der Metamodellierung

Die Unterscheidung zwischen Spezialisierung und Instanzierung erscheint häufig, wenn auch nicht immer (s. 3), wenig problematisch. Sieht man von der Mehrdeutigkeit des in beiden Fällen gern benutzten Bezeichners ’is a’ ab, sind die Verwendungskontexte beider Konzepte i. d. R. deutlich unterschiedlich. Es gibt allerdings Fälle, in denen beide Modellierungskonzepte angemessen erscheinen, eine gemeinsame Verwendung sich allerdings ausschließt. So zielt die Metamodellierung darauf, für einen bestimmten Verwendungskontext, etwa die Beschreibung von Geschäftsprozessen, eine Modellierungssprache zu definieren. Die Instanzen des Metamodells sind dann Typen, die zur Modellierung auf der Objektebene verwendet werden. Im Hinblick auf die Unterstützung des Modellierers wäre es wünschenswert, die Gemeinsamkeiten, die für alle Prozesstypen, aber auch für alle Prozessinstanzen gelten, zu erfassen. Auf diese Weise wird der Modellierer davon entlastet, im Einzelfall Konzepte zu (re-) konstruieren, die allen Modellierungsgegenständen gemeinsam sind. In einem Metamodell werden allerdings nur die Gemeinsamkeiten von Typen berücksichtigt. So sind aggregierte Prozesstypen aus anderen Prozesstypen zusammengesetzt. Es können allen Prozesstypen Attribute wie ’Anzahl von Instanzen’, ’durchschnittliche Laufzeit’ etc. zugeordnet werden. Diese gemeinsamen Eigenschaften von Prozesstypen werden sinnvollerweise in der Modellierungssprache spezifiziert und damit für den Modellierer wiederverwendbar gemacht. Der Ausschnitt eines entsprechenden Metamodells in Abb. 5 verdeutlicht diesen Umstand.

Die Metamodellierung hat jedoch Grenzen, da sie es nicht erlaubt, Gemeinsamkeiten von Instanzen auszudrücken. So gilt für jede Prozessinstanz, dass sie eine Anfangszeit und eine Endzeit hat. Dieses Wissen kann aber in einem Metamodell nicht untergebracht werden, da es sich bei beiden Attributen ja nicht um Eigenschaften von Prozesstypen handelt. Im Unterschied dazu würde die Verwendung generischer Typen, die durch Spezialisierung (und nicht durch Instanzierung) an individuelle Bedürfnisse angepasst werden können, die Zuweisung von Attributen, die für alle Instanzen gelten, erlauben. Hinsichtlich des Ziels, die Wiederverwendung gemeinsamer Eigenschaften zu unterstützen, konkurrieren Instanzierung und Spezialisierung also. Sie können allerdings nicht gemeinsam in einem Metamodell verwendet werden, weil die jeweils intendierten Abstraktionsebenen nicht kompatibel sind.

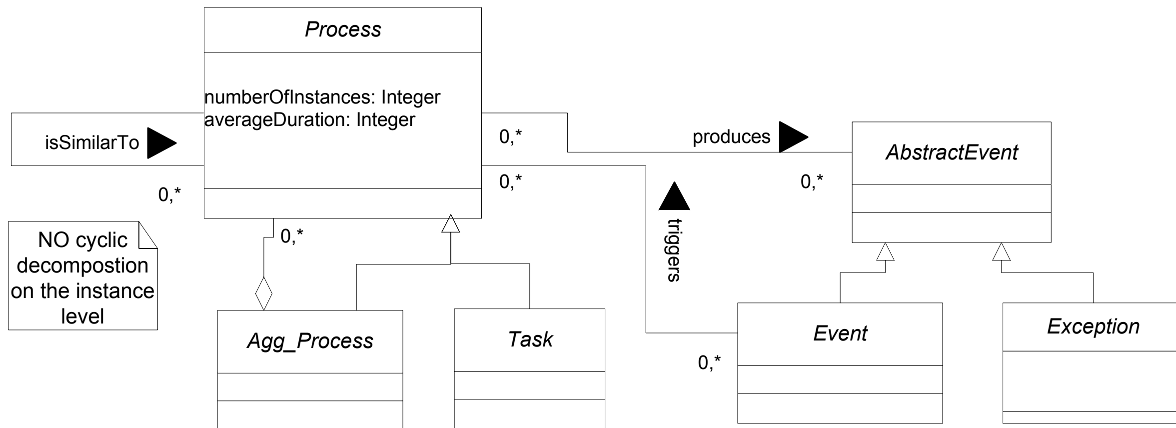

Abb. 5: Ausschnitt des Metamodells einer Prozessmodellierungssprache

Wendet man die in 3 diskutierten Kriterien zur Entscheidung zwischen Spezialisierung und Instanzierung auf diesen Fall an, stellt man fest, dass hier sowohl eine Abstraktion über Instanzen, die für eine Spezialisierung spricht, vorliegt, als auch eine Abstraktion über Typen, die eine Instanzierung aus einem Metamodell empfiehlt. Die Unterscheidung von Spezialisierung und Instanzierung auf dieser Ebene ist von wesentlicher Bedeutung für die das jeweils zu erreichende Niveau der Wiederverwendbarkeit. Sie entspricht der Unterscheidung zwischen einer dedizierten Modellierungssprache und Referenzmodellen. Eine in Form eines Metamodells spezifizierte Modellierungssprache erlaubt die Instanzierung von Typen - etwa von Geschäftsprozesstypen, während eine Referenzmodell den Entwurf eines konkreten Modells erheblich erleichtert, da nur noch Spezialisierungen vorgenommen werden müssen. Einschränkend ist dabei allerdings zu berücksichtigen, dass eine Spezialisierung die Bandbreiten möglicher konkreter Modelle erheblich einschränkt und dass es für bestimmte Modelltypen, etwa für Prozessmodelle, noch keine überzeugenden Spezialisierungssemantiken gibt. Wenn man in einem Metamodell auch Eigenschaften von Instanzen ausdrücken möchte, bleibt lediglich die Einführung zusätzlicher Integritätsbedingungen für die Zustände der Instanzen der Instanzen des Metamodells. Das ist nicht eben komfortabel und führt zudem zu schwer verständlichen Modellen. Komfortabler für den Modellierer sind Referenzmodelle, die mit Hilfe dedizierter Modellierungssprachen erstellt wurden, da sich so auf beiden Abstraktionsebenen Wiederverwendungseffekte darstellen lassen.

Im Unterschied zu den zuvor diskutierten Problemen handelt es sich hier allerdings nicht vordergründig um einen Mangel von Modellierungssprachen allgemein, sondern um ein Defizit solcher Sprachen, die zur Metamodellierung verwendet werden. Da dabei oft wie auch auf der Objektebene ERM-ähnliche Sprachen verwendet werden, bleibt zu vermuten, dass die Metamodellierung dedizierte Sprachen erfordert.

5 Abschließende Bemerkungen

Die Standardisierung konzeptioneller Modellierungssprachen, die seit einiger Zeit vor allem im Rahmen der UML erfolgt, verspricht eine Reihe unstrittiger Vorteile. Gleichzeitig zeigt die Betrachtung zentraler Konzepte wie Klasse, Objekt, Spezialisierung und Instanzierung, dass gängige Modellierungssprachen wie auch die UML nicht überzeugend sind. Denn sie unterstützen einen zentralen Anspruch der konzeptionellen Modellierung, nämlich eine natürliche, also mit vertrauten Konzeptualisierungen korrespondierende Beschreibung von Systemen, nur eingeschränkt. Dabei ist allerdings zu berücksichtigen, dass diese Defizite nicht allein den Modellierungssprachen angelastet werden können. Vielmehr resultieren sie zum

einen aus den Eigentümlichkeiten gängiger Programmiersprachen, zum anderen aus der doppelten Zielsetzung der konzeptionellen Modellierung: möglichst natürliche Repräsentation *und* die gleichzeitige Vorbereitung anschließender, eng integrierter Implementierungsschritte.

Da die betrachteten Probleme keinesfalls als semantische Spitzfindigkeiten abgetan werden können, sondern u. U. zu einer erheblichen Beeinträchtigung der Qualität von Informationssystemen beitragen, bleibt zu fragen, wie man ihnen wirksam begegnen kann. Zunächst ist es wichtig, in der Lehre mit Nachdruck auf diese Probleme hinzuweisen und so bei den Modellierern das Bewußtsein schaffen, das nötig ist, um diesen Fallstricken des objektorientierten Entwurfs zu entgehen. Für die Forschung ergeben sich m. E. zwei wesentliche Konsequenzen. So sollte trotz der scheinbaren Konsolidierung, die mit der Standardisierung der UML verbunden ist, die Forschung im Bereich der Modellierungssprachen weiter geführt werden - unabhängig davon, ob die Ergebnisse von der UML abweichen oder nicht. Die zweite Konsequenz ist grundsätzlicher Art. Angesichts des Umstands, dass es sich hier nicht um ein isoliertes Problem von Modellierungssprachen handelt, scheint es mir angemessen, gegenwärtige Formen der Arbeitsteilung und Koordination in der Informatik zu überdenken. So würde eine bessere Zusammenarbeit zwischen betroffenen Arbeitsgebieten, hier ist u. a. an die Software-Technik, Datenbanken, aber auch an die Künstliche Intelligenz-Forschung zu denken, nicht nur Redundanzen vermeiden und Synergien befördern, sondern könnte auch dazu beitragen, die aufgezeigten Friktionen zwischen Modellierungssprachen und Implementierungssprachen zu vermeiden.

Literatur

[Fran00] Frank, U.: Delegation: An Important Concept for the Appropriate Design of Object Models. In: Journal of Object-Oriented Programming. Vol. 13, No. 3, June 2000, pp. 13-18

[Fran02] Frank, U.: Modeling Products for Versatile E-Commerce Platforms - Essential Requirements and Generic Design Alternatives. In: Mayr, H. (Ed.): Proceedings of the 2nd International Workshop on Conceptual Modeling Approaches for E-Business (ECOMO). Berlin, Heidelberg etc.: Springer 2002

[Lore84] Lorenz, K.: Klasse (logisch). In: Mittelstraß, J. (Hg.): Enzyklopädie Philosophie und Wissenschaftstheorie. Bd. 2, Mannheim: BI Wissenschaftsverlag 1984, S. 403-405

[Meye97] Meyer, B.: Object Oriented Software Construction. 2. Aufl., Prentice Hall 1997

[Szyp98] Szyperski, C. A.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley 1998

[Wolt96] Wolters, G.: Subordination. In: Mittelstraß, J. (Hg.): Enzyklopädie Philosophie und Wissenschaftstheorie. Bd. 3, Mannheim: BI Wissenschaftsverlag 1996, S. 132-132

[KiFr03] Kirchner, L.; Frank, U.: Evaluierung von UML-Modellierungswerkzeugen. In: Objektspektrum, Nr. 1, 2003, S. 45-50

[Male95] Malenfant, J.: On the Semantic Diversity of Delegation-Based Programming Languages. In: Proceedings of the OOPSLA95. New York: ACM 1995, pp. 215-230