Erschienen in: B. Rumpe, W. Hesse (Hrsg) Modellierung 2004, LNI 45, (GI, 2004) 121-133

Tiefe Charakterisierung

Thomas Kühne[1] und Friedrich Steimann[2]

[1] Fachbereich Informatik
FG Metamodellierung
Technische Universität Darmstadt, 64283 Darmstadt
kuehne@informatik.tu-darmstadt.de
[2] Institut für Informationssysteme
Wissensbasierte Systeme
Universität Hannover, 30167 Hannover
steimann@kbs.uni-hannover.de

Zusammenfassung Mehrstufige Beschreibungshierarchien kommen in zunehmendem Maße in den verschiedensten Gebieten zum Einsatz. Ob in der Definition von Modellierungssprachenstandards, der Prozeßmodellierung oder in der Beschreibung von (Domänen-)Referenzmodellen mit dynamischer Typebene, immer werden Beschreibungshierarchien mit mindestens drei Ebenen genutzt. Die Beziehung der Beschreibungsebenen untereinander beruht typischerweise auf der Instanziierung, welche aber traditionell nur für zwei Ebenen ausgelegt ist. Dieser Beitrag thematisiert das Bedürfnis, Eigenschaften dennoch über mehr als eine Ebenengrenze hinweg festzulegen, und schlägt, nach der Erläuterung einiger konventioneller Lösungsmöglichkeiten, die Anwendung von "tiefer Instanziierung" vor, die als konservative Erweiterung der zweistufigen Instanziierung eine natürliche und prägnante Charakterisierung von Modellelementeigenschaften über mehrere Ebenengrenzen hinweg erlaubt.

1 Einleitung

Die durch das objektorientierte Paradigma populär gemachte Unterscheidung zwischen statischer Typebene (Klassen) und dynamischer Instanzebene (Objekte) brachte viele Vorteile für die Entwicklung komplexer Systeme. Die statische Typebene beschreibt modular und übersichtlich die universellen Eigenschaften eines Systems, die für alle Ausführungen unveränderlich bleiben, während die dynamische Instanzebene alle auftretenden Ausführungszustände erfassen und im Falle einer Programmiersprache auch realisieren kann. In verschiedenen Anwendungsgebieten wurde allerdings deutlich, daß es vorteilhaft ist, die Typebene nicht als fixe Größe anzunehmen, sondern als—unter Einhaltung gewisser Spielregeln—veränderlich. Abschnitt 2 beschreibt drei wichtige Anwendungsfälle. Typischerweise werden die Spielregeln (also die Spezifikationen, wie gültige Typebenen aussehen dürfen) mit dem gleichen Formalismus beschrieben, der auch zur Beschreibung der Instanzebenen zum Einsatz kam. Es wird eine Beschreibungshierarchie errichtet, ohne daß der Beschreibungsformalismus zwischen den (Meta-) Ebenen gewechselt wird. Interessanterweise kommt es so für die Elemente der Typebene zu einem Rollentausch. Aus der

Sicht der zweiten Schicht (Zählung beginnt mit 0 bei der untersten Instanzebene) sind sie nämlich keine Typen mehr, die Spezifikationen für Instanzen darstellen, sondern sie sind selbst Instanzen ihrer übergeordneten (Meta-) Typen. Somit besitzen die Elemente der Schicht 1 eine dualen Charakter, da sie gleichzeitig Typen und Instanzen sind.

Mit der herkömmlichen Instanziierung ergibt sich nun das Problem, daß ein Element der zweiten Ebene nicht Elemente der nullten Ebene beeinflussen kann, obwohl dies häufig wünschenswert wäre (siehe Abschnitt 2). Dieser Umstand ist darin begründet, daß Elemente der zweiten Ebene Elemente der ersten Ebene nur bezüglich ihrer Instanzfacette spezifizieren können. Solange nur zwei Ebenen—wie bei traditionellen objektorientierten Systemen—betrachtet werden, ist dies auch völlig ausreichend, da die Instanzen auf der untersten Ebene nur eine Instanzfacette besitzen. Sobald Instanzen (wie diejenigen auf der Ebene 1) aber auch eine Typfacette, mit der sie die Instanzfacette ihrer Instanzen spezifizieren, besitzen, muß es auch möglich sein, diese Typfacette zu spezifizieren. Ist dies möglich, kann man von der zweiten Ebene aus über die Typfacette der Elemente auf der ersten Ebene Einfluß auf die Instanzfacette der Elemente auf der nullten Ebene nehmen. Da es auf diese Art und Weise möglich ist, Elemente über zwei (oder auch mehr) Ebenengrenzen hinweg zu spezifizieren/charakterisieren, ist dafür der Ausdruck “tiefe Charakterisierung” angemessen.

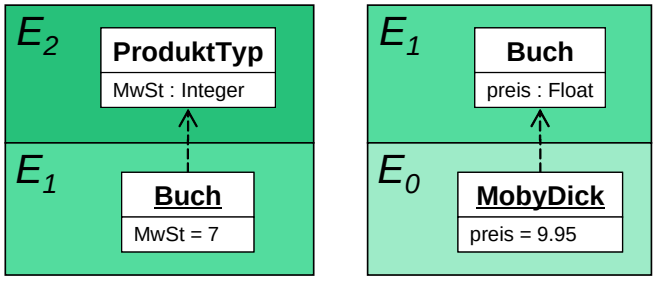
Nach der in Abschnitt 2 folgenden Beschreibung von drei Anwendungsbeispielen für tiefe Charakterisierung erläutert Abschnitt 3 einen konventionellen Weg, tiefe Charakterisierung zu erreichen. Abschnitt 4 beschreibt eine konservative Erweiterung des Instanziierungsbegriffs, mit welcher der gleiche Effekt mit geringerem Aufwand erreicht werden kann und die darüberhinaus noch weitere Vorteile bietet.

2 Beispiele für Beschreibungshierarchien

Die folgenden drei Anwendungsfälle sind Beispiele dafür, daß zunächst mit zwei Ebenen begonnen wurde (Typen und Instanzen), sich dann jedoch herausstellte, daß es nützlich ist auch die Typebene flexibel zu halten und für deren Spezifikation (was darf verändert werden und was nicht) wiederum eine übergeordnete (Meta-)Typebene einzuführen. In allen Fällen zeigt sich, daß schnell der Wunsch entsteht, von der zweiten Typebene direkt Einfluß auf die Elemente der untersten Ebene Einfluß zu nehmen.

2.1 Dynamische Typmodelle

Es ist heutzutage schon fast eine Selbstverständlichkeit, von einem Softwaresystem flexible Erweiterungsmöglichkeiten zu erwarten. Für das Beispiel eines Internetshops bedeutet dies, daß man nicht nur Artikel für bestehende Produktsparten aufnehmen können muß—Abb. 1(b) zeigt, wie das angebotene Buch Moby Dick als Instanz des Produkttyps Buch modelliert wird—sondern auch darauf vorbereitet sein muß, jederzeit die angebotene Produktpalette erweitern zu können, also z.B. den Produktyp DVD mit aufzunehmen. Hierzu werden die Produkttypen selbst als Instanzen von Produkttyp modelliert (siehe Abb. 1(a)).



(a) Buch als Instanz (b) Buch als Typ

Abbildung 1. Dynamisches Typemodell

Man beachte, wie auf diese Weise auch Eigenschaften von Produkttypen—wie z.B. unterschiedliche Mehrwertsteuersätze—spezifiziert werden können. Im Beispiel ist der Mehrwertsteuersatz nicht einer einzelnen Produktinstanz (wie Moby Dick), sondern dem Produkttyp (hier Buch) zugeordnet.

In einer objektorientieren Programmiersprache könnte man MwSt (in Abb. 1(a)) als sogenannte Klassenvariable realisieren. Für unsere Zwecke soll MwSt aber einfach eine Objektvariable auf Klassenebene darstellen, womit—im Gegensatz zur Klassenvariable— keine globale Sichtbarkeit für alle Instanzen impliziert ist. Sollte man über eine Instanz (z.B., Moby Dick) auf die für Bücher geltenden Mehrwertsteuer zugreifen wollen, so müßte man zunächst zur Klasse (hier Buch) navigieren und dort dann den Wert von MwSt abfragen.

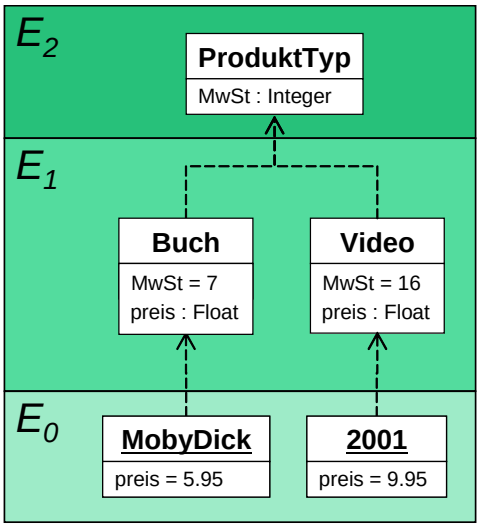


Abbildung 2. Produktdefinition

Abb. 2 zeigt eine kombinierte Sichtweise, bei der in Ebene 1 das Element Buch seine beiden Rollen—als Instanz von Produkttyp und als Typ von Moby Dick—offenbart. Durch den Einsatz von Entwurfsmustern, z.B. “Type Object” [JW98], läßt sich die dreistufige Hierarchie aus Abb. 2 mit nur zwei Ebenen (Klassen und Objekte) realisieren. Allerdings kann damit nicht erreicht werden, daß folgende Spielregeln für das Hinzufügen neuer Produkttypen (wie DVD) automatisch überprüft werden:

- Neuen Produktypen muß ein Mehrwertsteuersatz zugewiesen werden.
- Ihre Instanzen müssen einen Wert für ihren Verkaufspreis erhalten.

Tatsächlich will man also eigentlich schon für das Element Produkttyp festlegen, daß Instanzen dessen Instanzen (sozusagen Instanzen zweiter Ordnung) einen Wert für ihren Preis bekommen müssen. Mit den in Abb. 2 angewendeten Mitteln ist diese Anforderung nur durch die Einhaltung einer Konvention erfüllbar: Wann immer eine neue Produkttyp-Instanz erzeugt wird, so muß diese auch ein Attribut preis definieren. Auch das Herausfaktorisieren von preis in eine Oberklasse Produkt würde hieran nichts ändern. Nach wie vor müßte man eine Konvention einhalten: Neue Produkttypen müssen von Produkt erben (siehe auch Abschnitt 3). Ein alternativer Lösungsversuch besteht darin, auf Produkttyp zu verzichten und MwSt mit dem Wertebereich 7-16 in Produkt zu definieren. Eine kovariante Redefinition von MwSt in Produkttypen auf die Singletontypen 7 bzw. 16 brächte dann zwar ein nutzbares Modell, aber noch immer müßte eine Konvention "Neue Produktinstanzen müssen MwSt kovariant redefinieren" eingehalten werden.

Naturgemäß sind Konventionen jedoch nicht von Standardwerkzeugen überprüfbar, so daß man an dieser Stelle alleine auf die Disziplin der Entwickler angewiesen wäre. Um auch hier eine automatisierte Überprüfung zu ermöglichen, müssen entsprechende Spezifikationsmöglichkeiten geschaffen werden. Dies ist das Thema der Abschnitte 3 & 4.

2.2 Prozessmodellierung

Gegenstand der Prozessmodellierung ist die systematische Erfassung von Arbeitsabläufen. Die Abläufe selbst werden von realen Personen ausgeführt, deren Objekt-Stellvertreter auf Ebene 0 angeordnet sind (Szenarienmodellierung). Auf der ersten Typebene darüber befindet sich die Spezifikation der erlaubten/gewünschten Abläufe. Diese Spezifikation hängt, z.B. für den Fall von Softwareentwicklungsprozessen, sehr von der Art des Projekts (z.B., Embedded Systems oder Großrechner) und der gewählten Vorgehensweise ab (z.B., Extreme Programming oder V-Modell). Dennoch sind allen Spezifikationen gewisse Elemente gemeinsam. So kommen typischerweise Akteure, Aktivitäten und Produkte in allen Spezifikationen vor. Diese Gemeinsamkeiten und die Grundspielregeln, wie Spezifikationen für Prozessausführungen aussehen dürfen, werden auf der zweiten Ebene festgelegt (siehe Abb. 3).

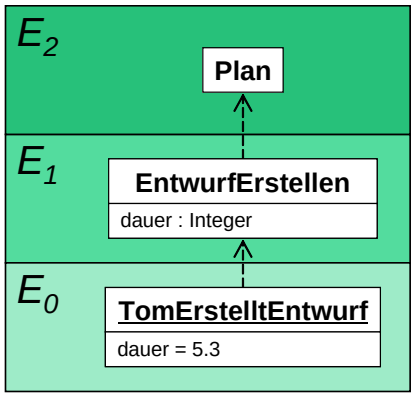


Abbildung 3. Prozessdefinition

Auch hier gilt wieder, daß man bereits auf der zweiten Ebene festlegen will, daß ausgeführte Aktivitäten (z.B., TomErstelltEntwurf) einen Wert für die Dauer der Ausführung tragen sollen. Das entsprechende Attribut muß auf Ebene 1 angesiedelt werden, dies soll aber bereits auf Ebene 2 festgeschrieben werden, damit es nicht möglich ist, Prozesspläne zu erstellen, deren Aktivitäten keine Ausführungsdauer zugeordnet ist.

Hierzu das Element Aktivität mit dem Attribut dauer auf Ebene 2 zu definieren wäre (in zweierlei Hinsicht) falsch. Zunächst würden Ausführungsdauerwerte dann auf Ebene 1 zugewiesen, also z.B. an EntwurfErstellen. Tatsächlich möchte man aber nicht eine Höchstausführungsdauer oder einen Mittelwert für diese *Art* der Aktivität angeben, sondern für einzelne Ausführungen angeben wie lange diese gedauert haben oder ggf. dauern dürfen, je nach Interpretation des Attributs. Auf jeden Fall bezieht sich ein Wert für eine Ausführungsdauer sinnvollerweise immer auf tatsächliche Ausführungen (Ebene 0) und nicht auf Aktivitätsarten (Ebene 1). Damit wird auch das zweite Problem des Ansatzes, das Element Aktivität auf der zweiten Ebene zu definieren, deutlich: Da die Elemente auf der ersten Ebene (noch) gar keine Aktivitäten sind, sondern nur Spezifikationen derselben, oder anders gesagt Aktivitätsarten, ist die Benennung des Ebene 2 Elements Aktivität fehlerhaft. Vielmehr wäre z.B. Plan oder auch Planungsartefakt angebracht. Ein Element Aktivität muß auf jeden Fall auf Ebene 1 plaziert werden, was aus der Korrektheit der Aussage "TomErstelltEntwurf ist eine Aktivität" folgt. Ein Ebene 0 Element (TomErstelltEntwurf) kann nur unter ein Ebene 1 Konzept fallen, nicht jedoch unter ein Ebene 2 Konzept. In unserem Beispiel könnte Aktivität als Oberklasse von EntwurfErstellen definiert werden.

Daß Plan dagegen eine korrekte Wahl für das entsprechende Ebene 2 Element ist, folgt aus der Richtigkeit der Aussage "EntwurfErstellen ist ein Plan"[1] und der Falschheit der Aussage "TomErstelltEntwurf ist ein Plan". Wäre die letzte Aussage wahr, dann wäre dies ein Hinweis darauf, daß Plan eine Oberklasse von EntwurfErstellen ist. Die Anwendbarkeit dieser Klassifizierungsaussagen folgt aus mengentheoretischen Überlegungen. Elemente einer Untermenge sind auch transitiv in deren Obermenge enthalten (erster Fall "TomErstelltEntwurf ist eine Aktivität"). Elemente einer Menge sind jedoch nicht auch Elemente einer zweiten Menge, die erstere enthält (zweiter Fall "TomErstelltEntwurf ist kein Plan"). Analog sollten auf der zweiten Ebene nicht Produkt und Akteur zum Einsatz kommen, sondern Spezifikation bzw. Rolle.

2.3 Modellierungsstandards

Zunächst war die UML [OMG00] fix auf der Ebene 2 definiert (in der OMG Hierarchie enthält Ebene 1 die Benutzertypen und Ebene 0 die Benutzerdaten). Mit der Einsicht, daß "Unified" nicht gleichbedeutend mit "Universal" ist, also der Einsicht, daß zur breiten Anwendbarkeit der UML diese auch anpaßbar sein mußte, kamen Möglichkeiten, die Sprachdefinitionsebene zu beeinflussen.

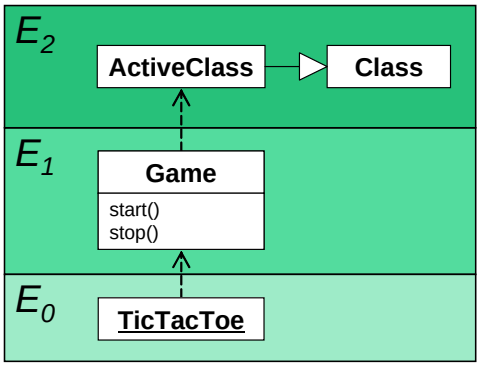


Abbildung 4. Sprachdefinition

Die Idee der UML als "Sprachfamilie" [CK+99] sollte für bessere Anwendbarkeit sorgen, brachte aber auch das Problem fehlender "tiefer Charakterisierung". Will man bei-

[1] Man beachte, daß EntwurfErstellen noch keine Aktivität ist, sondern nur der Typ für eine. Analog steht die Klasse Person nicht für eine Person, sondern für einen Personentyp.

spielsweise die UML um das Konzept aktiver Klassen—deren Instanzen z.B. ein- und auschaltbare interne Uhren besitzen—erweitern, dann muß dies auf Ebene 2 (der UML Sprachdefinitionsebene) geschehen, unberührt davon, ob dazu das sogenannte Metamodell direkt verändert wird oder der Stereotypen-Mechanismus verwendet wird. Man möchte, ja muß, aber zudem erreichen, daß aktive Objekte z.B. die Nachricht start und stop verstehen (siehe Abb. 4). Dazu müssen die Methoden auf Ebene 1 deklariert werden, aber in der Typfacette der enstsprechenden Klasse. Gerade die Typfacette ist aber von Ebene 2 aus mit traditioneller Instanziierung nicht beeinflußbar [AKHS03].

3 Power Types

In Abschnitt 2.1 wurde bereits darauf hingewiesen, daß "tiefe Charakterisierung" durch die Einhaltung von Konventionen erreicht werden kann. Abb. 5 zeigt nicht nur, wie das Attribut preis in eine Oberklasse herausfaktorisiert wurde und damit die einzuhaltende Konvention zu "Jede Instanz von ProduktTyp muß von Produkt erben" wird, sondern sie zeigt auch einen Mechanismus, die Einhaltung dieser Konvention zu erzwingen. Powertypes [Ode94] können immer dann zur Anwendung kommen, wenn ein Element sowohl von einem Typ (durch Instanziierung) als auch von einer Oberklasse (durch Spezialierung) definiert wird. Aus der Perspektive dieses Beitrags gesehen erlauben Powertypes sowohl die Instanzfacette (durch den Typ) als auch die Typfacette (durch die Oberklasse) eines Elements zu definieren. In der UML kann man den "Powertyp" einer Klasse, wie in Abb. 5 geschehen, durch eine mit dem Stereotyp ≪powertype≫ dekorierte Abhängigkeitsverbindung (dependency link) bestimmen. Mit dieser Verbindung wird ausgedrückt, daß jede Instanz (z.B., Video) des Powertypes von einer bestimmten Oberklasse (z.B., Produkt) erben muß. Obwohl entsprechende Werkzeugunterstützung in der Praxis nicht existiert, könnte dies auch automatisch überprüft werden. Damit wäre erreicht, daß z.B. Instanzen von ProduktTyp Instanzen erzeugen, die einen Wert für preis besitzen.

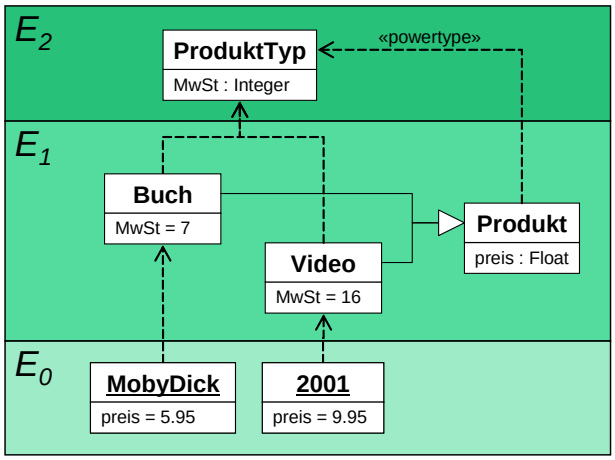


Abbildung 5. Powertype Produktdefinition

Mit "Powertypes" scheint also ein Mechanismus für "tiefe Charakterisierung" zu existieren. Allerdings ist dieser nicht vollends durchspezifiziert. Sowohl UML [OMG00] als auch Originalquelle [Ode94] lassen z.B., offen, ob es möglich ist, für mehrere Oberklassen den

gleichen “Powertyp” zu benutzen und was dies ggf. bedeuten würde. Darüberhinaus lassen sich konkrete Nachteile unterschiedlicher Schwere identifizieren:

1. Mit der ≪powertype≫ Abhängigkeitsverbindung wird eine Ebenengrenze überschritten, während in einem strikten Modellierungsansatz dies nur Instanziierungsbeziehungen gestattet ist [AK02]. Nicht grundsätzlich ein Problem, stellt dies doch eine unerwünschte Komplikation bezüglich einer sauberen Konstruktion von Beschreibungshierarchien dar.
2. Die Definition der Typ- und Instanzfacetten von Powertypinstanzen ist auf zwei Stellen verteilt. Um vollständig zu ermitteln welche (Typ- und Instanz-) Eigenschaften eine Powertypinstanz besitzt, müssen zwei verschieden Definitionsstellen konsultiert werden (Powertyp und Oberklasse). Hier wäre eine kompaktere Formulierung an einer einzigen Definitionsstelle von Vorteil.
3. Man kann sich ohne Mühe vorstellen, daß das Powertype-Konzept schlecht skaliert. Sobald die Beschreibungshierarchie drei Ebenen übersteigt, ist es unter Umständen nötig tiefe Charakterisierung über mehr als zwei Ebenen zu erreichen. Die dann entstehenden Beziehungen, mehrere Ebenen übergreifend und mehrere Rollen (Instanz, Typ, Oberklasse) auf einzelne Elemente vereinigend, werden schnell unübersichtlich.

Der im folgenden Abschnitt vorgestellte Ansatz vermeidet all diese Nachteile.

4 Tiefe Instanziierung

Ursprünglich zur Behebung einiger Probleme (duale Klassifizierung und Replikation von Konzepten) in der UML Sprachdefinitionshierarchie entwickelt [AK01], ist der Ansatz der “tiefen Instanziierung” auch außerhalb des Bereichs Sprachdefinition anwendbar. Im Nachhinein erscheint es nur logisch, daß bei einer Erweiterung der Ebenenanzahl von zwei auf drei oder mehr auch der entsprechende Instanziierungsbegriff mit erweitert werden muß. Bei der “tiefen Instanziierung” handelt es sich um eine konservative Erweiterung, welche die traditionelle (flache) Instanziierung für den Fall von nur zwei Ebenen subsumiert. Dies trifft z.B. auf die Ebenen E_1 und E_0 in Abb. 6 zu. Man beachte wie die jeweiligen Potenzwerte es

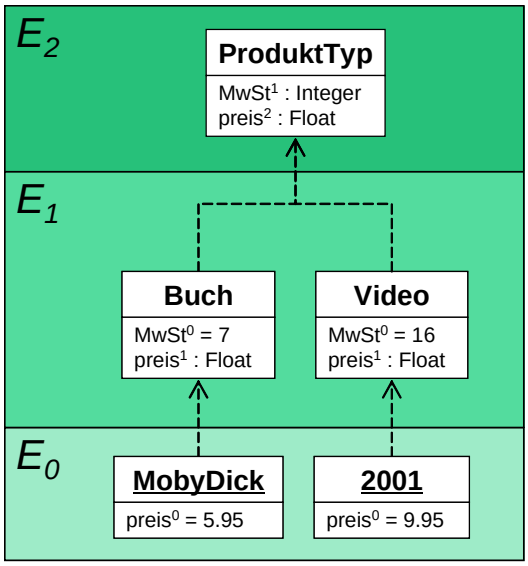


Abbildung 6. Tiefe Instanziierung

ermöglichen, Klassenattribute (preis, Potenz 1) leicht von Objektvariablen auf Klassenebene (MwSt, Potenz 0) zu unterscheiden[2].

Sobald drei oder mehr Ebenen involviert sind, kommt tiefe Instanziierung richtig zum tragen. Wie Element ProduktTyp in Abb. 6 zeigt, besitzen normale Attribute (z.B. MwSt) die Potenz 1. Solche normalen Attribute werden bei Instanziierung zu normalen Objektvariablen (mit Potenz 0), die einen Wert halten können. Für eine weitere Instanziierung stehen sie dann wegen der Potenz 0 nicht mehr zur Verfügung, weshalb z.B. MwSt weder in Moby Dick noch 2001 auftauchen kann.

Attribut preis des Elements ProduktTyp in Abb. 6 dagegen besitzt die Potenz 2, die sich bei der Instanziierung (z.B. zu Buch) zu 1 vermindert. Somit entsteht wie gewünscht in Buch ein normales Attribut, das in Instanzen von Buch zu Objektvariablen instanziiert wird. Wie man leicht sieht, besteht eine einfache Korrespondenz zwischen der Powertyp-Lösung aus Abb. 5 und der Potenzen-Lösung aus Abb. 6: Attribute, die im ersten Fall in Produkttyp definiert wurden, bleiben dort (mit Potenz 1). Attribute, die in Produkt definiert wurden, wandern von dort nach Produkttyp (mit Potenz 2).

Auf diese Art und Weise erreichen Potenzen, daß die Typfacetten von Instanzen von Produkttyp genauso einfach und von der gleichen Definitionsstelle aus definiert werden können, wie das für die Instanzfacetten der Fall ist. Mit tiefer Instanziierung ist es also möglich, tiefe Charakterisierung zu erreichen, ohne dabei auf den Vererbungsmechanismus zur Definition von Typfacetten zurückzugreifen. Soll eine Charakterisierung über noch mehr Ebenen hinweg erfolgen, so erhöht sich einfach die entsprechend zugeordnete Potenz eines Attributs. Mit tiefer Instanziierung steht also ein Mechanismus bereit, der tiefe Charakterisierung analog zum Powertyp-Mechanisms leisten kann, des letzteren Nachteile (nicht strikte Hierarchien, zwei Definitionsstellen, schlechte Skalierbarkeit) jedoch alle vermeidet.

Es sei in diesem Zusammenhang nur am Rande erwähnt, daß tiefe Instanziierung auch die Unifikation der Begriffe Objektvariable, Attribut und Meta-Attribute, usw. zu einem einzigen “Feld”-Begriff ermöglicht indem den Feldern einfach entsprechende Potenzen zugeordnet werden (0, 1, 2, usw.). Eine namentliche Unterscheidung über die Potenzwerte hinaus ist dann eigentlich überflüssig und konzeptuell ohnehin unnötig, da es sich in allen Fällen nur um jeweils entweder der Instanz- oder der Typfacette zugeordnete Eigenschaften von Elementen handelt.

4.1 Typentheoretische Interpretation tiefer Instanziierung

Tiefe Instanziierung besitzt als Erweiterung des traditionellen, flachen Instanziierungsbegriffs, den sie auf natürliche Art und Weise subsumiert, eine intuitive Semantik. Diese ist sogar leichter formulierbar als die üblichen Abbildungsvorschriften zwischen Klassen und

[2] In [AK01] wird beschrieben wann die Angabe von Potenzwerten optional ist, so daß kein unnötiger Aufwand entsteht.

Objekten bzw. Klassenattributen und Objektvariablen, da nur eine Dekrementierung von Potenzwerten nötig ist. Dennoch soll in diesem Abschnitt kurz auf eine typentheoretische Interpretation eingegangen werden, um dies auch formal nachzuvollziehen.

Eine gängige Interpretation von Klassen und deren Eigenschaften ist die als Mengen und Abbildungen zwischen diesen. Eine Klasse C wird demnach als Menge C interpretiert, und ein Attribut a von C vom Typ D als eine Funktion a von C nach D. Die Instanziierung der Klasse C entspricht dem Übergang von der Menge C zu einem ihrer Elemente und der Zuordnung dieses Elements zu einem "Wert" aus D per Attribut a. Mathematisch würde man schreiben

$$a : C \mapsto D, \text{bzw.} \tag{1}$$

$$a(c) = d, \text{mit } c \in C \text{ und } d \in D \tag{2}$$

Die Umkehrung der Instanziierung entspricht der Klassifikation oder, in Russells Typentheorie, dem Übergang von einem Element zu einer Menge, die es enthält, und damit der Anhebung des "Typs"[3] um eine Stufe bzw. Ebene. Wenn man diese Typerhöhung noch einmal vornimmt landet man auf der nächsten Stufe bei einer Mengen von Mengen. Geht man zudem davon aus, daß die Stufenhierarchie nach unten (durch die Stufe 0) begrenzt ist, also auf der untersten Stufe nur Atome (Elemente, die selbst keine Mengen mehr sind) vorkommen, so läßt sich eine klare Analogie zu den zuvor genannten Modellierungsebenen herstellen: von Instanzen (E_0) zu Klassen (E_1) zu Metaklassen (E_2) und so fort.

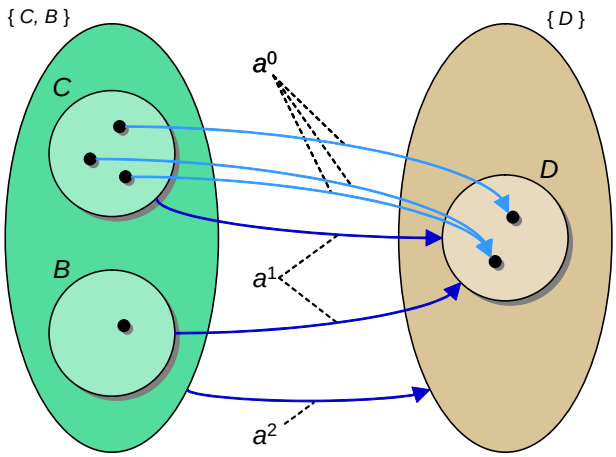


Abbildung 7. Mengentheoretische Interpretation

Was aber bedeutet diese Analogie für die tiefe Instanziierung? Wenn man statt Klassenattributen tiefe Metaklassenattribute (z.B. preis^2) betrachtet, dann muß man nicht nur den Definitionsbereich des Attributs ändern, also von Menge (siehe Definition 1) zu einer Menge von Mengen übergehen, sondern auch den Wertebereich:

$$a^2 : \{C, \ldots\} \mapsto \{D, \ldots\} \tag{3}$$

Allerdings möchte man für den bisher diskutierten Zweck der tiefen Charakterisierung auf der rechten Seite nicht beliebige Mengen außer D zulassen, damit auf der untersten

[3] Nicht zu verwechseln mit den Typen heutiger Programmiersprachen, die man in diesem Zusammenhang eher als Sorten bezeichnen würde.

Stufe nur ein Element aus D gewählt werden kann. Aus diesem Grund ist z.B. folgende Definition sinnvoll:

$$a^2 : \{B, C\} \mapsto \{D\} \tag{4}$$

Dann bleibt bei der Instanziierung nichts anderes übrig, als sich beim Wertebereich für eine einzige Menge (hier D) zu entscheiden.

In Abb. 7 wird dieser Zusammenhang grafisch dargestellt. Wie man sieht verschwindet beim Übergang von a^2 (tiefes Metaattribut), über a^1 (normales Attribut) zu a^0 (Objektvariable) jeweils eine "Mengenschale" im Definitions- und Wertebereich. Dies korrespondiert in dieser Reihenfolge zu den Definitionen 4, 1 und 2, wobei bei letzterer wohl nur aus Tradition eine andere Syntax ($a(c) = d$, statt $a^0 : c \mapsto d$) zum Einsatz kommt.

4.2 Beispielszenarien mit ihrer Bedeutung

Im folgenden werden drei Modellierungszenarien mit ihrer mengentheoretischen Interpretation angegeben. Das tiefe Attribut preis^2 aus Abb. 6 entspricht folgenden Definitionen:

$$\begin{array}{lll} \textit{preis}^2 : \textit{Produkttyp} & \mapsto \{Float\} \\ \textit{preis}^1 : \textit{Buch} & \mapsto Float \\ \textit{preis}^1 : \textit{Video} & \mapsto Float \\ \textit{preis}^0 \quad (\textit{MobyDick}) & = 5.95 \\ \textit{preis}^0 \quad (2001) & = 9.995 \end{array} \qquad \begin{array}{rl} \textit{Produkttyp} = & \{\textit{Buch}, \textit{Video}, \ldots\}^4 \\ = & \{\{\textit{MobyDick}, \ldots\}, \\ & \{2001, \ldots\}, \ldots\} \\ \textit{Buch} = & \{\textit{MobyDick}, \ldots\} \\ \textit{Video} = & \{2001, \ldots\} \end{array}$$

Das Attribut MwSt^1 aus Abb. 6 ist zwar auf E^2 angesiedelt hat aber nur Potenz 1. Die entsprechenden Definitionen folgen unten links. Auf der rechten Seite dagegen wird angenommen, daß Mehrwertsteuersätze zwar bereits auf der Typebene festgelegt jedoch auch auf der Objektebene sichtbar sein sollen. Beide Varianten der MwSt Modellierung sind sinnvoll und es hängt nur von der Modellierungsabsicht ab welche die "richtige" ist [AKHS03].

$$\begin{array}{lll} \textit{MwSt}^1 : \textit{Produkttyp} & \mapsto \textit{Integer} \\ \textit{MwSt}^0 \quad (\textit{Buch}) & = 7 \\ \textit{MwSt}^0 \quad (\textit{Video}) & = 16 \end{array} \qquad \begin{array}{lll} \textit{MwSt}^2 : \textit{Produkttyp} & \mapsto \{\{7\}, \{16\}\} \\ \textit{MwSt}^1 : \textit{Buch} & \mapsto \{7\} \\ \textit{MwSt}^1 : \textit{Video} & \mapsto \{16\} \\ \textit{MwSt}^0 \quad (\textit{MobyDick}) & = 7 \\ \textit{MwSt}^0 \quad (\textit{2001}) & = 16 \end{array}$$

5 Schluß

Der Einsatz von mehr als zwei Beschreibungsebenen zur Modellierung besitzt ein großes Potential, das nur zum Teil schon genutzt wird. Es ist davon auszugehen, daß die Vorteile der Einbeziehung mehrerer Ebenen zukünftig immer häufiger in Anspruch genommen

[4] Diese extensionale (aufzählende) Definition dient nur der Illustration. Die eigentliche Definition hat den üblichen intensionalen (beschreibenden) Charakter.

werden. Neben den in Abschnitt 2 vorgestellten Beispielen wird es auch immer wieder vorkommen, daß man Generalisierungsdiskriminatoren, wie z.B. Antriebsart, in Typen auf der zweiten Ebene umwandelt, um auch entsprechende Attribute und Methoden für diese angeben zu können, oder daß Typen der zweiten Ebene als Markierungsmöglichkeit für einen generativen Softwareentwicklungsansatz [Fra03] benutzt werden.

In allen Fällen zeigt sich, daß Eigenschaften von Elementen über mehr als eine Ebenengrenze hinweg festgelegt werden sollen, etwas, das der traditionelle Instanziierungsbegriff mit seiner alleinigen Bestimmung der Instanzfacette von Instanzen nicht leisten kann.

Während es mit "Powertypes" einen sogar in der UML verankerten Mechanismus gibt, um tiefe Charakterisierung zu erreichen, hat dieser einige in diesem Beitrag angesprochene Nachteile. Insbesondere das Problem der Verteilung der Spezifikation eines Elements auf mehrere definierende Elemente und die schlechte Skalierbarkeit machen "Powertypes" für eine Anwendung im größerem Maßstab ungeeignet.

Der in diesem Beitrag als Ersatz vorgeschlagene Ansatz der tiefen Instanziierung ist unter mehreren Gesichtspunkten vorteilhaft: Er leistet das, was "Powertypes" bieten, ohne deren Nachteile einzubringen. Er subsumiert die traditionelle Instanziierung, so daß vorhandene Modelle und Werkzeuge nicht völlig umgekrempelt, sondern ggf. nur erweitert werden müssen. Letztlich ermöglicht er auch noch die Unifikation der Konzepte Objektvariable, Attribut, Metaattribut, usw. zum Konzept "Feld" mit entsprechender Potenz.

Das Ziel dieses Beitrags war es zum einen auf die Dringlichkeit aufmerksam zu machen, mit welcher der ohne Zweifel vorhandene Bedarf an tiefer Charakterisierung mit einem unterstützenden Mechanismus bedient werden muß, und zum anderen auf "Tiefe Instanziierung" als einen diesbezüglich geeigneten Mechanismus hinzuweisen.

Literatur

[AK01] Colin Atkinson and Thomas Kühne, *The essence of multilevel metamodeling*, Proceedings of UML 2000, LNCS 2185, Springer Verlag, October 2001, pp. 19–33.

[AK02] Colin Atkinson and Thomas Kühne, *Profiles in a strict metamodeling framework*, Journal of the Science of Computer Programming **44**, Issue 1, (2002), 5–22.

[AKHS03] Colin Atkinson, Thomas Kühne, and Brian Henderson-Sellers, *Systematic stereotype usage*, Journal on Software and Systems Modeling **2** (2003), no. 3, 153–163.

[CK+99] S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. Wills, *Defining UML family members using prefaces*, In Proceedings TOOLS'99 Pacific (Christine Mingins and Bertrand. Meyer, eds.), IEEE Computer Society, 1999.

[Fra03] David S. Frankel, *Model driven architecture: Appyling MDA to enterprise computing*, OMG Press, 2003.

[JW98] Ralph Johnson and Bobby Woolf, *Type object*, Pattern Languages of Program Design (Robert C. Martin, Dirk Riehle, and Frank Buschmann, eds.), Addison-Wesley, 1998.

[Ode94] Jim Odell, *Power types*, Journal of Object-Oriented Programming **7** (1994), no. 2, 8–12.

[OMG00] OMG, *Unified modeling language specification, version 1.4*, 2000, Version 1.4, OMG document ad00-11-01.