Frameworkbasierte Anwendungsentwicklung (Teil 4):

Fachwerte

Wert und Objekt sind die fundamentalen Konzepte bei der Modellierung und Implementierung von Anwendungssoftware. Bisher hat allerdings die Modellierung und Implementierung von Werten in objektorientierten Systemen wenig Aufmerksamkeit gefunden. Unsere Erfahrung zeigt, daß ein grund-legendes Verständnis von Werten die Programmierung wesentlich vereinfacht und die Entwicklung wesentlich beschleunigt. In diesem Artikel werden wir die Auswirkungen von Werttypen auf objektorientierte Systeme darlegen und auf deren Implementierung sowie auf das Design im Kontext des JWAM-Frameworks eingehen.

Ein Beispiel

Im Bankbereich ist der (Währungs-)Betrag eine fundamentale Abstraktion. Neben dem reinen Wert des Betrags besteht er zusätzlich aus der Währung. Diese wird typischerweise durch einen dreistelligen Buchstabencode dargestellt: z. B. ‚DEM", ‚USD" oder ‚CHF". Der Wert des Betrags selbst wird üblicherweise als eine doppelt genaue Fließkommazahl repräsentiert. Als Beispiel dient eine Konto-Klasse, die den Saldo über ein Exemplar des Typs Betrag modelliert (siehe Abb. 1).
Die gängige Vorgehensweise wäre, den Betrag als eine Klasse zu modellieren, so daß ein konkreter Betrag im System als ein Objekt existiert. Nehmen wir nun an, daß Betrag als eine Klasse zur Verfügung steht und Operationen anbietet, die einen Klienten den Zustand eines Objekts verändern lassen. Ein Kontoobjekt wird auf die Frage nach dem Saldo ein Betragsobjekt zurück liefern, das den aktuellen Saldo darstellt. Wenn nun die Objektreferenz des Kontos und die des Klienten auf dasselbe Betragsobjekt zeigen, ist der Klient in der Lage, den Saldo des Kontos von außen durch Modifizieren des Saldoobjekts zu ändern. Dadurch werden alle Sicherungen umgangen, die gegebenenfalls in der Kontoklasse eingebaut wurden, um sicherzustellen, daß der Saldo korrekt geändert wird und die Konsistenz der Kontoobjekte erhalten bleibt. Wir haben somit einen klassischen Seiteneffekt, der durch das Alias-Problem herbeigeführt wurde (siehe Abb. 2).
Um dieses auf den ersten Blick technische Problem zu umgehen, gibt es eine Reihe von Lösungen:

- Eine Variante wäre sicherlich, daß das Kontoobjekt jeweils nur Kopien des Saldoobjekts herausgibt.
- Eine andere Lösung wäre die Anwendung des Body-Handle-Musters (siehe unten), bei dem die Exemplare des Typs Betrag jeweils beim Schreiben automatisch eine Kopie anfertigen.
- Eine dritte Lösung wäre die Veränderung der Schnittstelle von Betrag, so daß nur noch lesende Operationen angeboten werden.

Alle diese Lösungen deuten auf eine grundsätzliche Eigenschaft hin: Der (Währungs-)Betrag wird am besten durch einen Wert (im Sinne der Wertsemantik) dargestellt und nicht durch ein Objekt (im Sinne der Referenzsemantik). Da es sich bei dem (Währungs-)Betrag um ein fachliches Konzept handelt, das aus dem von uns analysierten Anwendungsgebiet Bank stammt, bezeichnen wir den Betrag als Fachwert. Fachwerte sind sozusagen die Ergänzung zu den in jeder Programmiersprache vorhandenen Standarddatentypen, wie int, boolean, char und float, die sich ebenfalls nach Wertsemantik verhalten.

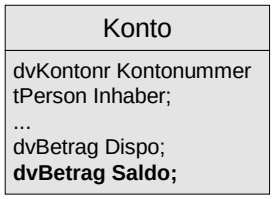


Abb. 1: Die Klasse "Konto"

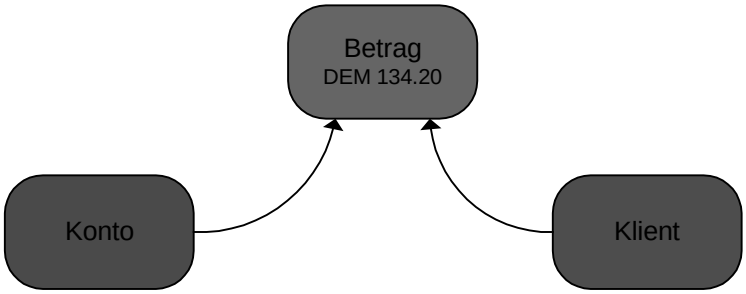


Abb. 2: Die Objekte "Konto" und "Klient" verweisen auf denselben Betrag.

Modellierung mit Objekten und Werten

Werte und Objekte stellen zwei der grundlegenden Konzepte der Informatik dar, die für moderne Programmiersprachen sowie für die damit entwickelten Anwendungssysteme unerläßlich sind. Im folgenden soll auf der Basis von [Mac82], [Bäu98b], [Mül99] und [Zül98] ein kurzer Überblick über die Eigenschaften von Werten und Objekten gegeben werden.

Werte

Werte sind zeit- und ortslos, d. h. Begriffe wie Zeit und Ort sind auf sie nicht anwendbar, womit sie auch keinen Lebenszyklus besitzen. Ferner können sie weder hergestellt noch verbraucht werden. Sie existieren also in ihrem eigenen ‚Werte-Universum", auf das von außen kein Zugriff besteht.
Werte abstrahieren von ihrem Kontext, d. h. sie sind nicht an die Existenz eines konkreten Gegenstandes in einer Anwendungsdomäne gebunden. Gleichzeitig verfügen sie über keine Identität, sondern es gibt nur einen einzigen Wert, auf den man sich beziehen kann. Deshalb kann es auch nicht mehrere Exemplare eines Wertes geben, sondern es handelt sich dann immer nur um verschiedene Repräsentationen des gleichen Werts.
Werte sind unveränderlich. Das bedeutet, daß sie zwar berechnet bzw. aufeinander bezogen werden können, der Wert an sich aber immer der gleiche bleibt. Folglich haben sie keinerlei Seiteneffekte.
Werte sind referentiell transparent. Das besagt, daß Ausdrücke auf der Basis von Werten jederzeit aufgrund ihrer Teilausdrücke verstanden werden können. So repräsentieren sowohl ‚40 + 2", ‚6*7" als auch ‚42" den eigentlichen Wert 42. Der Wert eines solchen Ausdrucks hängt also nur von seinen Argumenten ab und ist - unabhängig von der Reihenfolge - immer gleich. Die verschiedenen Teilausdrücke könnten also einfach durch andere, wertgleiche Ausdrücke ersetzt werden, ohne daß sich das Ergebnis ändern würde. Allerdings kann ein Wert, auf den sich ein Bezeichner in einem Kontext bezieht, undefiniert sein. Daneben sollte immer bedacht werden, daß Werte verschiedene Repräsentationen besitzen können.

Werte können nicht gemeinsam genutzt werden, da sie weder eine Identität noch einen Zustand haben. Es kann folglich keine Kommunikation anhand von Werten vorgenommen werden, da Werte nicht ausgetauscht oder bearbeitet werden können. Um aber dennoch mit Werten kommunizieren zu können, muß ein Wert an den Einsatzkontext gebunden werden. Dabei wird ein Wert über eine eindeutige Identifikationsmöglichkeit in einem Kontext eingeordnet. So ist es nur sinnvoll, über ‚diese 50 DM" zu reden, wenn ‚50 DM" zum Beispiel einer Banknote zugeordnet sind.

Werte	**Objekte**
• haben keinen Lebenszyklus	• haben einen Lebenszyklus
• abstrakt, ohne Identität, nur Gleichheit	• Repräsentationen von Objekten in Einsatzkontext
• definiert oder undefiniert, aber unveränderlich	• veränderbar bei Wahrung der Identität
• keine gemeinsame Nutzung	• gemeinsame Nutzung
• referentiell transparent	

Tabelle 1: Übersicht der Eigentschaften von Wert und Objekt

Objekte

Objekte haben im Gegensatz zu Werten einen Lebenszyklus. Sie können erzeugt und gelöscht werden und sich zu einem bestimmten Zeitpunkt an einem bestimmten Ort befinden.
Objekte verfügen über eine Identität. Es kann also beliebig viele Exemplare eines allgemeinen Konzepts geben, die sich allerdings in ihrer Identität voneinander unterscheiden.
Objekte können in ihrem Lebenszyklus von ihren Benutzern verändert werden, d. h. sie können verschiedene interne Zustände annehmen.
Objekte können über Referenzen von mehreren Benutzern gemeinsam benutzt werden, was allerdings die Konsequenz hat, daß Seiteneffekte auftreten können. Ein Objekt kann daher in einem Kontext verändert werden, ohne daß dies in einem anderen bemerkt wird.
In Tabelle 1 sind die wichtigsten Eigenschaften von Werten und Objekten nochmals aufgelistet.
Bei der Softwareentwicklung können wir mindestens drei Modellebenen unterscheiden: Domänenmodell, Systemdesign und Implementierungsmodell. Wie wir bereits gezeigt haben, sind Objekte und Werte nicht nur Implementierungskonzepte, die durch Programmiersprachen vorgegeben werden, sondern auch Modellierungskonzepte, die bei Analyse und Entwurf verwendet werden können.
Es gibt kein Kochrezept, nach dem wir festlegen können, ob etwas als ein Wert oder ein Objekt modelliert werden muß. Die Entscheidung, etwas als Objekt oder als Wert zu modellieren, ist immer auch von Pragmatismus geprägt, der in den folgenden Fragen zum Ausdruck kommt:

- Wie gehen Benutzer mit diesem Konzept um? (Domänenmodell),
- Wie viele Exemplare werden wahrscheinlich existieren? (Designmodell),
- Wie schwergewichtig ist eine Standardimplementierung (Implementierungsmodell).

Tatsächlich hängt die Wahl der Typen von den modellierten Konzepten ab. Dafür ist die Adresse ein gutes Beispiel. Konzeptionell ist eine Adresse ein Wert. Wenn jemand umzieht, ändert sich nicht die Adresse, sondern die Person zieht zu einer neuen Adresse. Für das Domänenmodell wird hier Adresse als Wert verstanden. Leider ist eine Adresse kein leichtgewichtiges Konzept. Wenn wir Adresse als Wert mit vollständiger Kopiersemantik implementieren, müssen wir Geschwindigkeits-einbußen in Kauf nehmen. Obwohl wir keine feste Regel kennen und jede Entwurfsentscheidung pragmatisch begründet werden muß, formulieren wir als Faustregel, daß Werttypen eher bei leichtgewichtigen Abstraktionen verwendet werden.

Implementierung von Fachwerten

Die meisten objektorientierten Programmiersprachen unterstützen die Implementierung und Verwendung von Fachwerten nicht direkt. Der wesentliche Mechanismus, um benutzerdefinierte Datentypen einzuführen, sind Klassen. Eigene Konstruktoren für wertsemantische Datentypen existieren nicht.
Eine Begründung dafür mag sein, daß Werte aus der Sicht der Programmiersprachenentwickler gewichen sind, weil sie sich auf Objekte konzentriert haben. Ein weiterer Grund mag sein, daß es keine ideale Lösung für die Implementierung von Fachwerten gibt. Es ist nicht immer angebracht, Kopiersemantik zu verwenden, weil z. B. schwergewichtige Werte dadurch das System bremsen und der Speicherverbrauch unangemessen ansteigt.
Die Programmiersprache Java geht mit ihrer im Standard-Package enthaltenen Klasse String eigentlich genau in die richtige Richtung. Objekte der Klasse String arbeiten nach Wertsemantik und setzten technisch eine Reihe der im folgenden diskutierten Implementierungstechniken (Body-Handle und Flyweight) um. Leider haben die Sprachdesigner von Java diesen Ansatz für andere Werttypen, wie ‚Datum", ‚Punkt" und ‚Rechteck", nicht weiterverfolgt. Alle anderen Kandidaten für Werttypen sind mit Referenzsemantik implementiert und zeigen alle damit verknüpften Problemen. Der Programmierer muß sicherstellen, daß die Objekte richtig und sinnvoll verwendet werden.
Da keine verfügbare objektorientierte Programmiersprache eine direkte Unterstützung für Fachwerte anbietet, müssen diese mit Hilfe von Klassen implementiert werden, die den eigentlichen Wert des Fachwerts kapseln. Verschiedene Implementierungsansätze berücksichtigen unterschiedliche Ansprüche und verfolgen spezielle Ziele. Wir unterscheiden Implementierungsvarianten anhand von zwei Kriterien:

1. Wie wird die Wertsemantik eines Objekts sichergestellt? Wir sehen zwei Implementierungsmöglichkeiten:
   1.1: Unveränderliche Objekte, die nur lesenden aber keinen schreibenden Zugriff auf ihren internen Zustand ermöglichen.
   1.2: ‚Copy-on-Write"-Objekte, die einen Mechanismus anbieten, der gewährleistet, daß beim Schreiben zuerst eine Kopie angefertigt wird, bevor verändernde Operationen aufgerufen werden. Durch diesen Ansatz wird sichergestellt, daß zwei Klienten A und B desselben Wertobjekts V nicht die Möglichkeit besitzen, den Zustand des anderen durch direktes Verändern des Objekts V zu beeinflussen. Dadurch bleibt der Effekt der Veränderung auf den Klienten begrenzt, der diese Veränderung ausgelöst hat.
2. Inwieweit werden die Fähigkeiten der Programmiersprache ausgenutzt? Der entscheidende Unterschied zwischen den Programmiersprachen liegt darin, wie ein Objekt erzeugt wird. Java und Smalltalk unterstützen ausschließlich dynamische Objekterzeugung. In C++ können Objekte auch statisch erzeugt werden. Da wir uns in dieser Artikelreihe mit Java befassen, werden wir hier auf die Spezifika von C++ nicht eingehen (näheres hierzu findet sich in [Bäu98b] und [Zül98]).

Unveränderliche Objekte

Ansatz 1.1 der unveränderlichen Fachwertobjekte ist in jeder objektorientierten Sprache, also auch in Java, einfach zu realisieren. Es muß lediglich sichergestellt werden, daß der Wert des Fachwerts nach der Erzeugung nicht verändert werden kann. Bei jeder Änderung muß also ein neuer Wert angelegt werden. Dabei ist der Speicherverbrauch allerdings enorm.
Dieses Problem läßt sich durch eine Kombination des Flyweight- mit dem Factory-Muster (siehe [Gam98]) lösen. Diese Muster beruhen darauf, daß im System eine Factory vorhanden ist, die als einzige Instanz in der Lage ist, einen Fachwert zu erzeugen. Außerdem stellt die Factory durch ihre Implementierung nach dem Flyweight-Muster sicher, daß zu jedem Wert aus dem Wertebereich des Fachwerts nur jeweils ein Objekt existiert. An der Schnittstelle der Factory müssen die notwendigen Methoden für die Erzeugung von Fachwerten zur Verfügung gestellt werden. Alle bereits erzeugten Fachwertobjekte werden in der Factory gespeichert. So kann ver-glichen werden, ob ein zu erzeugender Fachwert bereits existiert. Ist dies der Fall, wird kein neues Objekt erzeugt, sondern nur eine Referenz auf das bestehende Objekt als Resultat zurückgeliefert. Das Fachwertobjekt kann also von mehreren Objekten

gemeinsam genutzt werden. Da es nach der Erzeugung nicht mehr verändert werden kann, treten dabei keine Seiteneffekte auf. Diese Konstruktion eignet sich besonders für Fachwerte mit einem relativ kleinen Wertebereich, dessen Werte bei Programmstart erzeugt und erst bei Programm-ende wieder gelöscht werden.

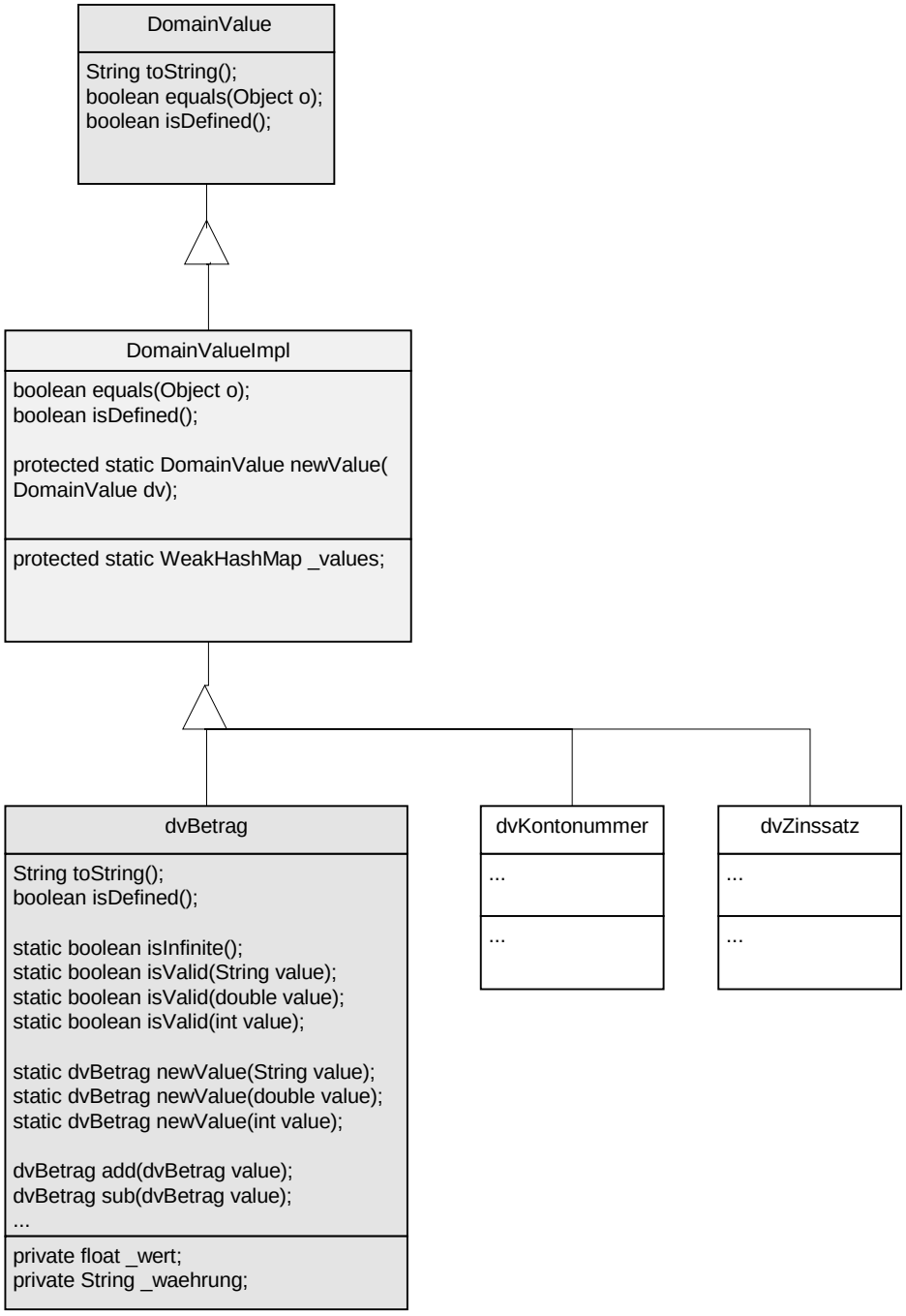
‚Copy-on-Write"-Objekte
Bei Implementierungsvariante 1.2 (siehe oben) wird ein anderer Weg beschritten. Bei dieser Variante werden veränderliche Fachwertobjekte benutzt. Diese werden aber zur Erhaltung der Wertsemantik bei Bedarf kopiert - daher ‚Copy-on-Write". In der einfachsten Form muß der Anwendungsentwickler dafür sorgen, daß alle Klienten einer Fachwertklasse diese Kopien selbst erstellen, damit keine Seiteneffekte auftreten. Der Entwickler der Fachwertklasse muß lediglich einen Kopiermechanismus entwerfen - alles andere liegt in der Verantwortung des Anwendungsprogrammierers.
Eine verfeinerte Form des ‚Copy-on-Write" ist die Verwendung des Body/Handle-Entwurfsmusters (siehe [Cop92] und [Koe97]). Der Fachwert wird hierbei durch zwei Klassen implementiert: Body und Handle. Der eigentliche Wert des Fachwertob-jekts wird im Body-Objekt gekapselt. Der mit Fachwertobjekten arbeitende Programmierer kann nur über das Handle-Objekt, welches eine Referenz auf das Body-Objekt enthält, auf den Wert zugreifen. Alle Methodenaufrufe am Handle-Objekt werden direkt an das zugehörige Body-Objekt weitergeleitet. Da auf jedes Body-Objekt mehrere Handle-Objekte verweisen können, vermerkt das Body-Objekt mittels eines Referenzzählers die Anzahl der Verweise von Handle-Objekten. Wird eine verändernde Operation am Handle-Objekt aufgerufen, so muß - falls der Referenzzähler größer als eins ist - eine Kopie des Body-Objekts angelegt und der Zähler dementsprechend um eins verringert werden. Fällt die Anzahl der Referenzen auf null, so kann das Objekt gelöscht werden. Um allerdings die Wertsemantik zu sichern, muß auch das Handle-Objekt bei einer Änderung immer kopiert werden, da ansonsten mehrere Exemplare auf dieses Objekt verweisen könnten und es wiederum zu Seiteneffekten käme.
Im folgenden stellen wir anhand des JWAM-Frameworks vor, welche Unterstützung ein Framework dem Entwickler bei der Implementierung von Fachwerten bieten kann. Dabei gehen wir auch auf die Möglichkeiten der Sprache Java zur Realisierung von Fachwerten ein.

Fachwerte im JWAM-Framework

Das JWAM-Framework (siehe [JWAM99]) stellt zur Implementierung der Fachwerte zwei Klassen zur Verfügung:

- Das Interface DomainValue legt die Schnittstelle der Fachwerte auf dem allgemeinsten Niveau fest.
- Mit Hilfe der abstrakten Klasse DomainValueImpl wird bereits eine Teilimplementierung angeboten, die verwendet werden kann, wenn neue Fachwerte für ein Anwendungssystem implementiert werden sollen.

Unterhalb dieser Klassenhierarchie befinden sich domänenspezifische Fachwerte. Zum JWAM-Framework wird eine Reihe von Standardfachwerten und von speziellen Beispielfachwerten angeboten. Sie können als Anleitung für die eigene Entwicklung von Fachwertklassen verstanden werden.
Als Entwurfsmuster wurde eine abstrakte Factory gewählt. Die Schnittstelle dieser Factory befindet sich in der statischen Schnittstelle der Fachwertklasse. Zur Speicherung der Exemplare wird eine WeakHashMap (ein Datentyp aus JDK 1.2) verwendet, um unnötig von der Factory referenzierte Objekte speicherplatzsparend freizugeben.
An der Schnittstelle des konkreten Fachwertes (hier dvBetrag) finden sich für jeden Basisdatentyp, aus dem der Fachwert konstruiert werden kann, eine Klassenmethode isValid, die eine syntaktische Prüfung und eine Wertebereichsprüfung erlaubt, sowie einen Konstruktor (hier newValue genannt). Die Implementierung von newValue stützt sich dabei auf die gleichnamige Methode protected in der Oberklasse ab. Diese sorgt für die Verwendung bereits erzeugter Werte und das gegebenenfalls notwendige Erzeugen eines Exemplars. Der eigentliche Konstruktor der Fachwertklassen ist protected, damit neue Fachwerte nur über die Methode newValue erzeugt werden und so der gesamte Mechanismus der Fachwert-Factory nicht umgangen werden kann.



Zur Speicherung der Fachwerte wird ebenfalls ein geeignetes Verfahren benötigt. Da der Serialisierungsmechanismus von Java BLOBs (‚Binary Large Object Blocks") verwendet, ist er nicht geeignet, um effizient mit relationalen Datenbanken zusammenzuarbeiten. Wir benutzen ihn daher nur für den Transport von Fachwerten über Prozeßgrenzen hinweg als einfaches vorhandenes Mittel. Beim Lesen der Objekte von einem Datenstrom, der aus einem anderen Prozeß entgegengenommen wird, ist es notwendig, die Fachwert-Factory dazwischenzuschalten, um zu garantieren, daß es für jeden Wert nur genau ein Exemplar eines Fachwertobjekts gibt.
Um den Aufwand zu verkleinern, haben wir uns bei der Speicherung der Fachwerte in einer relationalen Datenbank für eine eigene Serialisierung entschieden, die wir im folgenden zur besseren Unterscheidung Atomisierung nennen. Hierbei nutzen wir den Umstand, daß bei der Implementierung eines Fachwerts bekannt ist, welche skalaren Werte zulässig sind. Deshalb kann in der Fachwertklasse selbst angegeben werden, wie ein Fachwert beim Atomisieren zu schreiben ist. Ein einfacher Ansatz ist, jeden Fachwert als String intern zu speichern und mit geeignetem Parsing beim Einlesen zu prüfen. Der Mehrverbrauch an

Speicher ist heutzutage nur in Ausnahmefällen ein bedeutendes Argument. Ein Vorteil ist sicherlich, daß diese Repräsentation überall (beim Debugging, in einer Tabelle der Datenbank) lesbar und verständlich ist. Zur Effizienzsteigerung kann dann neben der String-Repräsentation zusätzlich eine effiziente Verwaltung (beim Betrag z. B. durch float) eingefügt werden.

Auswirkungen von Fachwerten auf die Performanz

Die Verwendung von Fachwerten in objektorientierten Systemen hat nicht nur im Bereich der Analyse und der Modellierung seine Vorteile, sondern verbessert - wenn richtig eingesetzt - auch die Performanz der Systeme.

Verteilte Systeme

In verteilen Systemen werden Objektreferenzen zwischen Prozeßgrenzen ausgetauscht. Obwohl aus unserer Sicht naiv, suggerieren eine Reihe von Produkten und Publikationen eine vollständig transparente Verteilung. Unter bestimmten Umständen können dann Objektkonstellationen auftreten, in denen z. B. das Kontoobjekt ein Betragsobjekt in einem anderen Prozeß referenziert. Dadurch werden vermehrt Aufrufe zwischen Prozessen nötig, die massive Performanzeinbußen bewirken. Wäre der (Währungs-)Betrag anstelle eines Objekts als Wert modelliert und implementiert, könnte er immer mit dem ummantelnden Objekt kopiert werden, was eine Performanzreduktion verhindert.
Es gibt eine Reihe von Lösungen, die bei verteilten Systemen verwendet werden, um diese Probleme zu lösen. Zum Beispiel lassen sich Objekte als unveränderlich kennzeichnen, oder ein Laufzeitsystem bietet ein ausgefeiltes Objektmigrationsverfahren an, bei dem nicht nur einzelne, sondern ganze Objektbäume transportiert werden, um die Anzahl der Aufrufe zwischen Prozessen zu minimieren. Stellt man Werttypen explizit für verteilte Systeme zur Verfügung, bietet dies für alle diese Konstrukte eine gute Basis.

Sperrmechanismen (Locking)

Beim Einsatz von Multithreading kann die Verwendung von Werttypen Sperrmechanismen reduzieren oder ganz unnötig machen. Da Werte konzeptionell keinen sich ändernden Zustand haben, können sie als unveränderbare Objekte implementiert werden. Das wiederum erfordert keine Sperrmechanismen. Andere Implementierungsvarianten können zumindest das Problem teilweise reduzieren: Wird ein Wert als ein Objekt repräsentiert, kann von Kopiersemantik ausgegangen werden; wird ein Wert mittels des Body-Handle-Musters implementiert, kann der Locking-Overhead zumindest für den Klienten vermieden werden.

Speichern in Datenbanken

Bei der Serialisierung von Objekten, zum Beispiel beim Speichern in einer Datenbank, ist der Serialisierungsalgorithmus dafür verantwortlich, zyklische Objektreferenzen zwischen Objekten aufzulösen. Jede Überprüfung, ob eine Objektreferenz einen Zyklus darstellt, reduziert natürlich die Ausführungsgeschwindigkeit. Offensichtlich muß ein Objekt, das einen Wert repräsentiert, nicht in diese Überprüfung einbezogen werden. Der Wert kann direkt in den Puffer geschrieben werden. Da die Anzahl der Wertobjekte in einem System sehr hoch werden kann - typischerweise wesentlich höher als die Anzahl der regulären Objekte - wird die Ausführungsgeschwindigkeit so signifikant steigen.
Hinzu kommt, daß keine Objektidentitäten (sogenannte ‚OIDs") für Werte vergeben werden müssen. Dadurch verringert sich die Größe des ‚Datenabdrucks" von Werten gegenüber vergleichbaren Objekten. Ein Wert kann direkt aus dem Hauptspeicher geschrieben werden, ohne seine interne Struktur näher zu betrachten, und vor allen Dingen, ohne auf eingeschlossene Objektreferenzen zu achten.
Die genannten Vorteile beziehen sich vorwiegend auf leichtgewichtige Werte, d. h. Werte mit einem kleinen implementierten Zustandsraum. Schwergewichtige Werte reduzieren die Vorteile, weil sie mit Objektreferenzen implementiert sein können, denen nachgegangen werden muß. Zusätzlich kann die Serialisierung eines komplexen Objektgraphen mit einer hohen Anzahl gleicher Werte zu einem

unerwartet großen Datenabdruck führen, wenn die Werte schwergewichtig sind und eher kopiert als referenziert werden.
Werden Objekte in einer relationalen oder objektorientierten Datenbank gespeichert, benötigen sie zusätzlichen Verwaltungsaufwand. In relationalen Datenbanken muß jedes Objekt ungeachtet seiner Größe in einer speziellen Tabelle abgelegt werden. Das anschließende Wiedereinlesen erfordert aufwendige Joins, was die Ausführungsgeschwindigkeit drückt. Wenn aber klar ist, daß ein Objekt konzeptionell eigentlich ein Wert ist, dann kann es effizient in sein umgebendes Objekt eingebunden werden. Danach kann der Wert mit seinem umgebenden Objekt sowohl schnell gelesen, als auch schnell geschrieben werden. Das Objekt und alle seine Attribute können als ein Tupel aufgefaßt werden und demnach auch in einer anstelle von vielen Tabellen verwaltet werden.
Eine Reihe von verfügbaren Datenbanken geht mit den von ihnen angebotenen Funktionen in diese Richtung. Zum Beispiel bieten Sybase und Oracle die Möglichkeit, neben den allgemein üblichen Werttypen (wie integer und string) zusätzliche Werttypen (wie Datum und Uhrzeit) einzuführen, die typischerweise nicht als Werte sondern als Objekte aufgefaßt werden. Trotzdem werden diese Werttypen direkt in Tabellen geschrieben. Wird den Anwendern von Datenbanken die Möglichkeit gegeben, ihre eigenen Werttypen einzuführen, kann die Geschwindigkeit also signifikant erhöht werden.
Diese Aussagen treffen aber - wie bereits gesagt - hauptsächlich auf leichtgewichtige Werte zu. Schwergewichtige Werte können eine eigene Tabelle erfordern, wie es auch für Objekte üblich wäre. Eine weitere Schwierigkeit tritt auf, wenn Werte polymorph genutzt werden sollen. Dann müssen sie entweder referenziert werden, was zu den oben erwähnten aufwendigen Joins führt, oder sie werden als BLOB gespeichert, was verhindert, daß sie über Abfragen erreichbar sind.
Bei der Abfrage von relationalen Datenbanken können Werttypen ebenso eine Hilfe sein. Sie machen die Abfragen wesentlich einfacher und ermöglichen es, Indizes zu generieren. Nehmen wir z. B. einen Währungsbetrag, der in einer Klasse Konto als Objekt modelliert ist. Eine SQL Abfrage wie

select from Account where MonetaryAmount = '100 SFR'
müßte in ein Join übersetzt werden, bei dem die Tabellen Account und MonetaryAmount zusammengebunden werden. Es ist zudem nicht möglich, mit Hilfe der Standard-Indexierung relationaler Datenbanken einen Index für MonetaryAmount in der Tabelle Account zu erzeugen. Das bewirkt weitere Geschwindigkeitseinbußen.
Diese Beispiele illustrieren, daß die Modellierung mit Werttypen an angemessenen Stellen eine signifikante Verbesserung der Performanz von objektorientierten Softwaresystemen erreicht.

Ausblick und Dank

In den letzten Jahren haben wir in wissenschaftlichen und industriellen Projekten ein tragfähiges Modell für den Einsatz und die Konstruktion von Fachwerten erarbeitet. Dieses Modell wurde bereits in mehreren Zyklen überarbeitet und in verschiedenen Programmiersprachen umgesetzt. Die daraus resultierenden Erfahrungen sind in das JWAM-Framework eingeflossen und werden zur Zeit weiter ausgebaut. Ein aktuelles Thema ist die Kombination von Fachwerten zu komplexen Werten.
Wir danken in diesem Zusammenhang ganz besonders folgenden Kollegen, die uns mit ihren Erfahrungen die Grundlage für diesen Artikel geliefert haben: Dirk Bäumer, Dirk Riehle, Wolf Siberski, Daniel Megert, Karl-Heinz Sylla. Wir danken Klaus Müller und allen seinen Diskussionspartnern für die Umsetzung des Fachwert-Konzepts für das JWAM-Framework.

Literatur

[Bäu98a] D. Bäumer, Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme, Dissertationsschrift am Fachbereich Informatik der Universität Hamburg, 1/98
[Bäu98b] D. Bäumer, D. Riehle, W. Sieberski, C. Lilienthal, D. Meggert, K.-H. Sylla, H. Züllighoven, Values in Object Systems, Ubilab Technical Report 98.10.1
[Ble99a] W.-G. Bleek, G. Gryczan, C. Lillienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven, Framework-basierte Anwendungsentwicklung (Teil 2): Die Konstruktion interaktiver Anwendungen, in

OBJEKTspektrum 2/99, S. 78
[Ble99b] W.-G. Bleek, G. Gryczan, C. Lillienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven, Framework-basierte Anwendungsentwicklung (Teil 3): Die Anbindung von Benutzungsoberflächen und Entwicklungsumgebungen an Frameworks, in OBJEKTspektrum 3/99, S. 90
[Cop92] J. Coplien, Advanced C++: Programming Styles and Idioms, Addison-Wesley, 1992
[Gam98] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software, 2. Auflage, Addison-Wesley, 1998
[Gry99] G. Gryczan, C. Lillienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven, Framework-basierte Anwendungsentwicklung (Teil 1), in: OBJEKTspektrum 1/99, S. 90
[JWAM99] JWAM: Java Framework for the Tools and Materials Approach, Universität Hamburg, FB Informatik, Arbeitsbereich Softwaretechnik, 1999
(siehe http://swt-www.informatik.uni-hamburg.de/Software/JWAM)
[Koe97] A. Koenig, B. Moo, Ruminations on C++, Addison-Wesley, 1997
[Mac82] B. J. MacLennan, Values and Objects in Programming Languages, ACM SIGPLAN Notices, Vol.17, Nr. 12/82
[Mül99] K. Müller: Konzeption und Umsetzung eines Fachwertkonzepts, Universität Hamburg, FB Informatik, Arbeitsbereich Softwaretechnik, Studienarbeit 1999
[Zül98] H. Züllighoven, Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz, dpunkt-Verlag, 1998