Objektorientiertes Programmieren mit Smalltalk

Nachrichten-Zustellung

Smalltalk ist rein objektorientiert und Vorbild für neuere Entwicklungen wie Objective-C oder Ruby. Als Grundelemente dienen Objekte und Nachrichten, mitgelieferte Klassen bestimmen den Funktionsumfang. Der Artikel stellt zwei Implementierungen vor, die Einsteigern und Profis gerecht werden. Friedrich Dominicus

Scheinbar neuartige Konzepte der objektorientierten Programmierung stammen häufig aus der Smalltalk-Welt. Obwohl Firmen damit durchaus professionelle Software schreiben, führt die Sprache ein Nischendasein. Dabei eignet sich Smalltalk gut für Einsteiger, denn sie vermittelt die objektorientierte Lehre in Reinkultur. Alle Smalltalk-Implementierungen bringen eine Vielzahl von Klassen mit, die der Programmierer für eigene Anwendungen erweitert.

Sprache mit System

Die meisten Smalltalk-Systeme sind Entwicklungsumgebungen mit vielen Komponenten wie einem Workspace zur Eingabe und zum interaktiven Test, einem Klassenbrowser und noch einigem mehr. In diesem Artikel kommen das freie Squeak [1] und Visual Age Smalltalk von IBM zum Einsatz. Letzteres gibt es als Trial-Version [2] zum Download. Der Einstieg ist aber nicht ganz einfach: Das System ist für große Entwicklerteams ausgelegt und unterstützt verteiltes Entwickeln mit vielen Usern. Entsprechend viele Optionen bringt die Entwicklungsumgebung mit, die beinahe einen eigenen Systemverwalter erfordert.

IBMs Smalltalk ist stark auf Business-Applikationen ausgelegt und besitzt für den professionellen Bereich ein schier unerschöpfliches Potenzial. Für erste Schritte eignet sich Squeak besser, da es weniger komplex ist und sich schon die Installation einfacher gestaltet. Squeaks GUI ist zwar textorientiert, Applikationen können aber auch eine Grafikbibliothek (für so genannte Morphs) benutzen (siehe Abbildung 1).

Andere Implementierungen, teils Open Source, teils proprietär, sind Cincom Smalltalk [3], Smalltalk/X [4] oder Little Smalltalk [5]. Auch das GNU-Projekt führt einen eigenen Smalltalk-Compiler, allerdings ohne grafische Oberfläche. Weitere Implementierungen finden sich in den Smalltalk-FAQ [6].

Objekte überall

Smalltalk ist strikt objektorientiert, im Gegensatz zu Sprachen wie C++ und Java, die auch nicht objektorientierte Elemente besitzen. Es gibt zwei Standards, die den Sprachumfang festlegen: Smalltalk-80 und den neueren Ansi-Standard. Squeak erfüllt nur den ersten, aber es gibt Patches, die es Ansi-kompatibel machen. IBMs Produkt dagegen ist ein Ansi-Smalltalk.

Jede Entität in Smalltalk ist ein Objekt und alle Objekte sind Elemente einer Klasse. Die Vererbungshierarchie beginnt bei der Klasse »Object«. Prozeduren und Funktionen heißen in der Smalltalk-Welt Meldungen (Messages). Jedes Objekt kann Meldungen erhalten oder versenden. Im Code steht zuerst der Empfänger, also das Objekt, und dann die Nachricht:

```
Objekt Meldung
```

In Smalltalk geben alle Nachrichten ein Objekt zurück, das wiederum Empfänger einer Nachricht sein kann. Es ergeben sich Messagekaskaden, die auch von funktionalen Programmiersprachen bekannt sind.

Der Smalltalk-Programmierer gibt seinen Code im so genannten Workspace ein, der ähnlich funktioniert wie eine Shell, allerdings nicht zeilenorientiert: egal wo der eingegebene Smalltalk-Code steht, er ist beispielsweise mit einem Mausklick ausführbar. Damit lassen sich an jeder Stelle Smalltalk-Ausdrücke auswerten, ausführen, anhalten oder debuggen. Ausgaben erfolgen auf dem Transcript, einem speziellen Ausgabefenster.

Hello World

Am Anfang der Beschäftigung mit jeder Programmiersprache steht das traditionsreiche Programm, das nichts anderes tut, als einen String auszugeben. In Smalltalk sieht das so aus:

```
Transcript show: 'Hello World' printString
```

Darauf erscheint im Transcript die Ausgabe »'Hello World'«, siehe Abbildung 2. Das Beispiel wendet »printString« auf ein Objekt der Klasse »String« an. Das Ergebnis besteht aus einer Zeichenkette, die als Argument für die Nachricht »show« dient. Die Nachricht wird an eine globale Klassenvariable

»Transcript« gesendet, die dem Transcript-Fenster entspricht, das folgerichtig das Ergebnis anzeigt.

Ein Punkt trennt einzelne Anweisungen. Erfolgt die Trennung durch ein »;«-Zeichen wird die nachfolgende Nachricht an das zuvor aktive Objekt verschickt (in diesem Fall »Transcript«). Smalltalk unterscheidet zwischen Groß- und Kleinschreibung, dabei gelten folgende Konventionen:

- Klassen und Konstanten beginnen mit einem Großbuchstaben.
- Nachrichten werden in gemischter Schreibweise bezeichnet und beginnen mit Kleinbuchstaben.

Jedes Objekt unterstützt Basisoperationen: die Klasse abfragen, Repräsentation als Zeichenkette ausgeben, Kopieren, auf Gleichheit beziehungsweise Ungleichheit testen, Elemente kopieren und einige mehr. Erhält beispielsweise ein Objekt die Nachricht »class«, gibt es die Klasse zurück, zu der es gehört. Eine Suche im Class Browser findet die Klasse, ein Mausklick darauf zeigt die zugehörigen Methoden.

Klassen erweitern

Programme mit Smalltalk entwickeln bedeutet meist bestehende Klassenstrukturen erweitern. Der Entwickler benutzt dazu den Klassenbrowser, sucht sich die zu erweiternde Klasse aus und implementiert dann die Erweiterung. Auf diese Art lässt sich die Klasse »Integer« um eine Fakultätsfunktion erweitern (siehe Listing 1) .

Da jede Funktion immer zu einem Objekt gehört, besitzt sie mit diesem einen impliziten Parameter. Sie benötigt also keinen expliziten Parameter wie in C beispielsweise bei »fact (int n)«. Für Funktionen gibt es keine besondere Abgrenzung durch geschweifte Klammern oder andere Kennzeichen. Der Name der Smalltalk-Funktion lautet »fact«, der in Anführungszeichen eingeschlossene Satz ist ein Kommentar.

Die Zeile »| result |« legt eine lokale Variable fest. Der Operator »:=« weist einer Variablen einen Wert zu. Da in Smalltalk keine statische Typüberprüfung erfolgt, kann eine Variable im Lauf ihres Lebens auf unterschiedliche Objekte verweisen. Im Beispiel handelt es sich um ein Objekt der Klasse »SmallInteger« – die Eingabe von »1 class.« demonstriert diese Klassenzugehörigkeit.

Die nächste Zeile stellt nahezu alle syntaktischen Elemente vor, die es in Smalltalk gibt. Der Name der Nachricht lautet »to:do:« und ist an die Klasse »Number« gerichtet. Eine solche Nachricht heißt in Smalltalk Keyword Message. Im Beispiel besitzt sie zwei Argumente, die durch die Schlüsselwörter »to:« und »do:« gekennzeichnet sind. Der Empfänger der Nachricht ist im Beispiel die Zahl 1. Die Iteration endet, wenn die Schleife den Wert von »self« erreicht. Mit »self« ist der Empfänger der Nachricht gemeint. Bei einem Aufruf von »5 fact« wird »self« an die Zahl 5 gebunden.

Die »to:do:«-Nachricht erwartet nach dem »do:« einen Block, also ein Funktionsobjekt. In anderen Sprachen wie Lisp oder Scheme heißen Blocks auch Closures, in C gibt es dazu keine direkte Entsprechung. Blocks sind Code-Abschnitte, die den Zustand von sichtbaren Variablen einfrieren. Wird ein Block zu einem späteren Zeitpunkt ausgewertet, gelten die dort gesetzten Werte.

Am Beginn des Blocks steht das Argument »n«. Es hat zunächst den Wert 1, dann 2 – bis zum Wert von »self«. Der Block setzt die Variable »result« auf einen neuen Wert, der wiederum von »n« abhängt. Der Block multipliziert die »result« mit 1, dann mit 2 und so fort. Die Funktion gibt am Ende »result« zurück. Das Karet »^« entspricht einem »return« in C. Gibt der Programmierer den Return-Wert nicht explizit an, liefert Smalltalk implizit »self« zurück.

Ja oder nein?

Auch Fallunterscheidungen funktionieren anders. Da Smalltalk immer mit Objekten arbeitet, gilt auch hier die Reihenfolge Empfänger – Nachricht. Fallunterscheidungen operieren auf boolesche Objekte, deshalb nimmt die Fallunterscheidung die Stelle des Objekts ein; die Nachricht an das Objekt folgt danach:

```
1 < 2 ifTrue: ['Stimmt']
      ifFalse: ['Stimmt nicht.'].
```

Das Ergebnis des Vergleichs ist ein Objekt der Klasse »True«, das ein Subtyp der Klasse »Boolean« ist. Bei »ifTrue:ifFalse:« handelt es sich um eine Nachricht an ein Objekt vom Typ »Boolean« oder eine davon abgeleitete Klasse. Bei

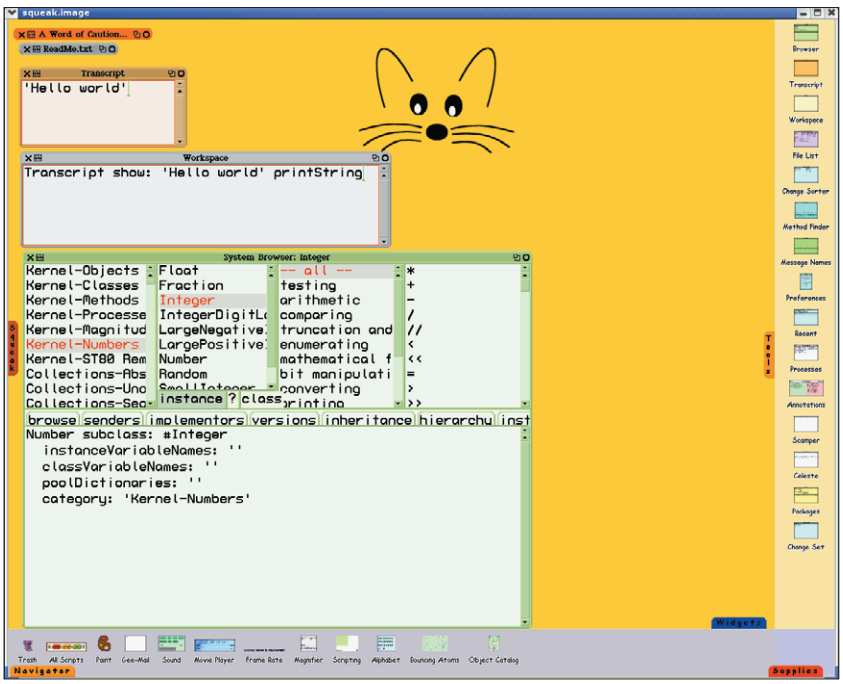

Abbildung 1: Transcript-Fenster, Workspace und Klassenbrowser auf Squeaks bunter Oberfläche.

Listing 1: Fakultät in Smalltalk

```
01 fact
02         "Berechnet die Fakultät."
03         | result |
04         result := 1.
05         1 to: self do: [:n | result := result * n].
06         ^result
```

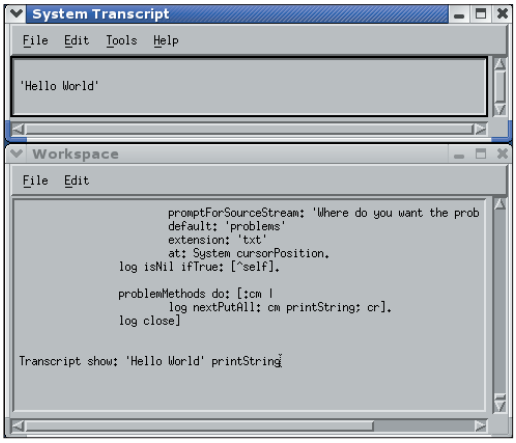

Abbildung 2: „Hello World" in IBMs Visual Age Smalltalk mit einem Workspace- und Transcript-Fenster.

der Entwicklung eigener Programme empfiehlt es sich, nach einer passenden Klasse zu suchen und diese zu erweitern. Zur Auswahl stehen dabei, eine neue Methode der gefundenen Klasse hinzuzufügen (wie im Beispiel mit der Fakultät) oder eine Ableitung der Klasse vorzunehmen.

Ableiten und verändern

Gibt es keine passende Klasse, bleibt zuletzt das generische »Object« als Superklasse. Listing 2 definiert eine Klasse für eine einfach verkettete Liste. Diese Deklaration zeigt, dass die Definition einer Klasse nicht vom üblichen Aufrufschema in Smalltalk abweicht. Das »Object« erhält die Nachricht »subclass:instanceVariableNames:... «. Der Name der definierten Unterklasse lautet »MyLinkedList«. Das »#« vor diesem Namen zeigt an, dass es sich nicht um eine Variable handelt, die ausgewertet werden soll, sondern um einen Namen.

Listing 2: Klassendefinition in Smalltalk

```
01 Object subclass: #MyLinkedList
02     instanceVariableNames: 'first last item next '
03     classVariableNames: ''
04     poolDictionaries: ''
```

Listing 3: Zugriffsfunktionen in Smalltalk

```
01 last
02        "Return last element of MyLinkedList (Reader)"
03         ^last
04
05 last: anItem
06        "Set last to equal anItem (Writer)"
07        last := anItem.
```

Der zweite Parameter »instanceVariableNames« legt fest, welche Variablen eine Instanz der Klasse besitzen wird. Die darauf folgenden Klassenvariablen »classVariableNames« gelten für alle Instanzen einer Klasse. Der letzte Parameter »poolDictionaries« kann eine oder mehrere Hashtabellen enthalten, die innerhalb der Klasse direkt zugänglich sind. Diese Hashtabellen bieten sich an, um die Daten von Klassen abzuspeichern.

Die hier eingeführte Klasse besitzt die Instanzvariablen »first«, »last«, »item«, »next« und ist direkt von »Object« abgeleitet. Günstiger wäre es gewesen, diese Klasse von einer »Collection« abzuleiten; dazu wären alle Methoden dieser Klasse zu implementieren, doch das würde den Rahmen des Artikels sprengen.

Instanzvariablen sind private Daten, auf die nur Objekte der Klasse zugreifen dürfen. Sollen andere Programmteile auf solche Variablen zugreifen können, muss der Programmierer zusätzlich geeignete Zugriffsfunktionen definieren (siehe Listing 3).

Smalltalk grafisch

Die meisten Implementierungen bringen umfangreiche Bibliotheken mit. Sie sind zum großen Teil grafischen Benutzeroberflächen gewidmet und unterstützen das MVC-Pattern (Model View Controller). Das resultiert aus Smalltalks langer Geschichte: Viele der ersten Implementationen grafischer Bedienelemente entstammen der Smalltalk-Welt.

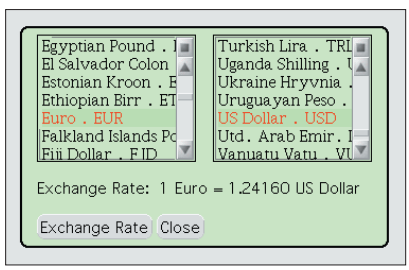

Abbildung 3: Ein Währungsumrechner, implementiert mit einem Morph, einem grafischen Objekt der Squeak-Umgebung.

Auch Visual Age und Squeak sind mit Bibliotheken reich bestückt. Squeak dehnt sie auf multimediale Elemente aus wie Sound, Grafiken und Animation. Visual Age legt mehr Wert auf Interoperabilität mit anderen Werkzeugen aus dem Hause IBM, speziell aus dem Business-Bereich (Datenbanken, Domino, Web).

Morphing

Eine Mini-Anwendung demonstriert einige der grundlegenden Fähigkeiten von Squeak. Sie sucht im Netz nach einer Umrechnungstabelle für Währungen und und stellt diese in einem grafischen Objekt dar (siehe Abbildung 3). Die Quellen dazu sind unter [7] zu finden Das Beispiel benutzt Squeaks grafische Elemente, die Morphs. Sie können sich an Begrenzungen wie ein Ausgabefenster halten oder solche einfach ignorieren. So ist es kein Problem, eine Linie quer über alle Fenster zu zeichnen. Mit diesen Mitteln lässt sich in Squeak vergleichsweise einfach eine eigene Fenster-Oberfläche erstellen.

Weitere Informationen über Smalltalk finden sich im Internet. So gibt es ein Squeak-Wiki und freie Smalltalk-Bücher online, die den Einstieg erleichtern, aber auch fortgeschrittene Entwickler befriedigen dürften [8]. *(fjl/ofr)* ■

Infos

[1] Squeak für Unix: [http://www-sor.inria.fr/~piumarta/squeak/]
[2] IBM Visual Age: [http://www-3.ibm.com/software/awdtools/smalltalk/]
[3] Cincom Smalltalk: [http://www.cincom.com/scripts/smalltalk.dll/index.ssp]
[4] Little Smalltalk: [http://www.smalltalk.org/versions/LittleSmalltalk.html]
[5] Smalltalk/X: [http://www.exept.de/sites/exept/deutsch/Smalltalk/frame_ueber-sicht.html]
[6] Smalltalk-FAQ: [http://www.infi.net/~vmalik]
[7] Listings online: [http://www.linux-magazin.de/Service/Listings/2004/04/Smalltalk]
[8] Freie Smalltalk-Bücher online: [http://www.iam.unibe.ch/~ducasse/FreeBooks.html]

Der Autor

Friedrich Dominicus ist Geschäftsführer der Q Software Solutions GmbH.