Werttypen in objektorientierten Programmiersprachen

Beate Ritterbach, Axel Schmolitzky
Arbeitsbereich Softwaretechnik, Universität Hamburg

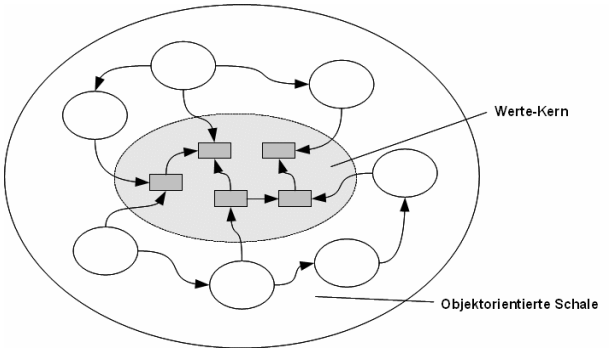
Ausgangspunkt unserer Arbeit sind objektorientierte Programmiersprachen. Eine wesentliche Leitlinie solcher Sprachen lautet „Alles ist ein Objekt“, zumindest gilt das für die Exemplare der benutzerdefinierbaren Typen. Dem steht entgegen, dass in der Praxis immer wieder der Bedarf nach Abstraktionen auftaucht, die sich in vielerlei Hinsicht ähnlich wie primitive Typen verhalten und die mit Objekttypen nur schwer abgebildet werden können: fachlich motivierte Werttypen.
Werte und Objekten sind grundlegend verschiedene Abstraktionen. Werte sind notwendig. Wenn es keine explizite Unterstützung für sie gibt, dann müssen sie mit Objekten durch Einsatz von Konventionen und Mustern simuliert werden. Derartige Umwege machen ein System schwer verständlich und fehleranfällig.
1. Die hier präsentierte Arbeit stellt einen konzeptionellen Wertbegriff vor, unabhängig von potentiellen Implementationen. 2. Darauf aufbauend skizziert sie eine Sprachunterstützung, welche die Eigenschaften von Werten gewährleistet; dabei stehen software-technische Gesichtspunkte wie Sicherheit, Verständlichkeit und Wartbarkeit im Vordergrund. 3. Schliesslich untersucht sie, welche Wechselwirkungen Werttypen mit anderen Sprachkonzepten eingehen.

1. Was sind Werte/ Werttypen?

Aufbauend insbesondere auf der Arbeit von MacLennan [2] verstehen wir Werte als unveränderliche, zeitlose Abstraktionen, deren Operationen keine Seiteneffekte hervorrufen und referentiell transparent sind. Die genannten Eigenschaften sind konzeptionell zu verstehen, als durch Klienten beobachtbare Eigenschaften, unabhängig von einer konkreten Implementation. Die Zeitlosigkeit von Werten manifestiert sich dadurch, dass Werte nicht erzeugt werden. In etlichen Publikationen, die sich mit Werten befassen, steht die Unveränderlichkeit als prägende Eigenschaft von Werten im Vordergrund ([1], [5], [8]), Unerzeugbarkeit dagegen wird oft vernachlässigt. Unerzeugbarkeit führt u. a. dazu, dass das für Objekttypen zentrale Konzept eines Konstruktors auf Werttypen keine Anwendung findet.

2. Sprachunterstützung für Werte

Während Zahlentypen abhängig von der gewählten Programmiersprache teilweise als primitive Typen vordefiniert sind (z. B. int, float), stehen fachlich motivierte Werttypen wie Datum, Punkt, Intervall, rationale oder komplexe Zahl in der Regel nicht per se zur Verfügung. Es ist wünschenswert, für Werttypen die gleiche Flexibilität zu haben wie für Objekttypen, sie sollen frei definierbar sein.
Der hier vorgestellte Ansatz nimmt seinen Ausgangspunkt im objektorientierten Paradigma und sucht dieses um Werte und wert-typische (und damit funktionale) Verarbeitung zu erweitern. Unsere Arbeit schlägt eine dedizierte Sprachunterstützung für Werttypen vor: einen eigenständigen Wert-Typkonstruktor, getrennt vom Typkonstruktor für Objekttypen. Sie diskutiert einen Satz sprachseitig, zur Compilezeit überprüfbarer Regeln, welche die konzeptionellen Eigenschaften von Werten sicherstellen. Durch dieses Regelwerk entsteht zwischen Werten und Objekten ein asymmetrisches Abhängigkeitsverhältnis, Objekte können Werte benutzen, nicht aber umgekehrt. Damit teilt sich jedes System in einen „Werte-Kern“, in dem die für Werte charakteristischen Eigenschaften wie z. B. referentielle Transparenz garantiert sind, und eine „Objektorientierte Schale“, die von zustandsbehafteter Verarbeitung geprägt ist.



Im Ergebnis ist der Wertekern „pure functional“, in der „objektorientierten Schale“ werden Objekte und ihre Zustandsänderungen geradlinig unterstützt. Diese Trennung unterscheidet die hier vorgestellte Arbeit von

etlichen aktuellen Ansätzen, z. B. von Sprachen, die das ausdrückliche Ziel haben, objektorientierte und funktionale Ansätze miteinander zu vereinen (z. B. Scala[7], OCaml [3], Python [4], Ruby [6]). Diese sehen keine expliziten Werttypen und keine Trennung des objektorientierten und des funktionalen Paradigmas innerhalb der Sprache vor und werden damit „impure“. Werttypen in objektorientierten Sprachen ermöglichen, die Vorteile beider Paradigmen zu vereinen.

3. Wechselwirkungen von Werttypen mit anderen Sprachkonzepten

Werte sind in vielerlei Hinsicht anders als Objekte. Das erweist sich insbesondere bei ihrer Einbettung in eine Programmiersprache - sei es eine bestehende objektorientierte Sprache, die um Werttypen erweitert werden soll, oder eine neue Sprache, die von vornherein auf einer Trennung des Typsystems in Werte und Objekte aufbauen soll. Sprachkonzepte wie z. B. Vererbung/ Subtyping mit Inklusionspolymorphie (die als grundlegend für Objektorientierung angesehen werden) finden auf Werttypen keine oder nur eingeschränkte Anwendung. Auf Konzepte wie Identität und Gleichheit erlauben Werttypen einen neuen Blickwinkel, ebenfalls auf die Funktionsweise von Variablen.

Literatur:

1. Bacon, D.F.: Kava: A Java dialect with a uniform object model for lightweight classes. Concurrency—Practice and Experience 15(3–5), 185–206, 2003
2. MacLennan, B. J.: Values and Objects in Programming Languages, ACM SIGPLAN Notices, 17:12, S. 70-79, 1982.
3. OCaml, URL: http://caml.inria.fr/ocaml/.
4. Python. URL: http://www.python.org.
5. Riehle, D.: Value object. Proceedings of the 2006 conference on Pattern languages of programs,), 1-6, 2006
6. Ruby. URL: http://www.ruby-lang.org/
7. Odersky, M. ; Spoon, L., Venners, B.: Programming in Scala. Mountain View, Calif., 2008
8. Vaziri, M., Tip, F., Fink, S., Dolby, J.: Declarative Object Identity Using Relation Types. ECOOP 2007: 54-78