Implementierung eines objektorientierten Entwurfs in der funktional-logischen Sprache Curry

Herbert Kuchen
Universität Münster*

Im Software-Engineering setzt sich die objektorientierte Software-Entwicklung aufgrund der größeren Wartungsfreundlichkeit und Wiederverwendbarkeit der erstellten Komponenten immer mehr durch. Ein objektorientierter Entwurf wird heute vorzugsweise in der Unified Modeling Language (UML) notiert [BRJ99], d.h. unter Verwendung von u.a. Klassendiagrammen, Interaktionsdiagrammen und Zustandsdiagrammen.

Die u.a. wegen ihres hohen Programmierniveaus, ihrerer leichteren Lesbarkeit und ihrer Vorteile beim Beweisen von Programmeigenschaften geschätzten deklarativen Sprachen haben in diesem Umfeld nur eine Chance, wenn sich in ihnen objektorientierte Entwürfe leicht implementieren lassen. Dies ist zunächst nicht der Fall. Objekte haben einen Zustand, den es in rein deklarativen Sprachen nicht gibt. Der z.B. in funktionalen Sprachen übliche Ansatz, bei Anwendung einer Funktion auf eine Datenstruktur die "veränderte Datenstruktur" als (Teil-)Ergebnis zu liefern, wie z.B. bei der bekannten Stack-Operation `push: Stack` $\times$ `Element` $\rightarrow$ `Stack`, ist in einem objektorientierten Kontext (insbesondere bei verteilten, nebenläufigen Objekten) nicht verwendbar, da i. allg. mehrere Objekte mit dem Stackobjekt kommunizieren, der neue "Zustand des Stacks" aber nur an einer Stelle verfügbar ist. Eine Alternative hierzu ist, ein zustandsbehaftetes Objekt durch einen (in Auswertung befindlichen) Berechnungsausdruck zu repräsentieren. Dieser Ausdruck wird typischerweise eine Anwendung einer rekursiven Funktion auf Argumente sein, die den momentanen Zustand des Objekts repräsentieren. Dieser Ansatz wird z.B. in [Ha99] verwendet. Die "Objekte" kommunizieren hier über Streams mit ihren Nachbarn. Es wird allerdings ein statisches "Objektnetz" verwendet, was für die Implementierung eines objektorientierten Entwurfs i.allg. unzureichend ist, da Objekte üblicherweise dynamisch erzeugt und mit anderen verknüpft werden.

In unserem Ansatz wird dieses dynamische Erzeugen von Objekten und die Kommunikation zwischen ihnen ermöglicht. Jedes Objekt ist konzeptionell mit einem ausgezeichneten Broker-Objekt über Streams verbunden. Die Kommunikation zwischen Objekten läuft über den Broker. Für eine effiziente Implementierung kann dies ggfs. durch direkte Kommunikation ersetzt werden.

Um die für die objektorientierte Software-Entwicklung unverzichtbare Vererbung zu simulieren, wird auf einen Ansatz von [WB89] zurückgegriffen. Hierbei erhält jede Funktion für jedes Argument-"Objekt" als zusätzliches Argument ein Verzeichnis, in dem vermerkt

*Institut für Wirtschaftsinformatik, Steinfurter Str. 109, 48149 Münster, `kuchen@uni-muenster.de`

ist, welche Funktionen (Methoden) bezogen auf die Vererbungshierachie für das zu verarbeitende Objekt relevant sind.

Da eine explizite Durchführung dieser Transformation für die Entwickler(in) sehr umständlich und daher unzumutbar ist, wird eine an objektorientierte Sprachen angelehnte Syntax bereitgestellt. Dieser syntactic sugar wird von einem Precompiler in (reines) Curry übersetzt. Weitere Ansätze mit (teilweise) ähnlicher Zielrichtung findet man in [HS95, NC97, BLOP97].

Literatur

[Ha99] M. Hanus: Distributed Programming in a Multi-Paradigm Declarative Language, PPDP'99, LNCS 1702, 188-205, 1999.

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language User Guide, Addison Wesley, 1999.

[HS95] J. Hughes, J. Sparud: Haskell++: An Object-Oriented Extension of Haskell, Haskell Workshop, 1995, `http://www.cs.chalmers.se/~rjmh/Software/h++.html`

[NC97] J. Nordlander, M. Carlsson: A Rendezvous of Functions and Reactive Objekts — Escaping the Evil I, In Proceedings of the Haskell Workshop, Amsterdam, 1997, `http://www.cs.chalmers.se/~nordland/ohaskell/papers.html`

[BLOP97] S. Breitinger, R. Loogen, Y. Ortéga-Mallen, R. Peña: The Eden Coordination Model for Distributed Memory Systems, HIPS 1997, `http://www.mathematik.uni-marburg.de/~loogen`

[WB89] P. Wadler, S. Blott: How to make ad-hoc polymorphism less ad hoc, ACM Symposium on Principles of Programming Languages, 60-76, 1989.