Weiterführende Konzepte für prozedurale Programmiersprachen (*A*dvanced *P*rocedural *P*rogramming *L*anguage *E*lements)

Christian Heinlein
Abt. Rechnerstrukturen
Universität Ulm

Prozedurale Programmiersprachen (wie z. B. Pascal, C, Modula-2, Ada) wurden in den 1980er und 1990er Jahren zu objektorientierten Sprachen (wie z. B. Eiffel, C++, Java) weiterentwickelt (obwohl die Idee der Objektorientierung bekanntlich wesentlich älter ist). Im wesentlichen wurden Module, Verbundtypen (record types) und Prozeduren zu Klassen kombiniert und um die Konzepte Vererbung, Polymorphie und dynamisches Binden erweitert. Obwohl sich damit zahlreiche Programmierprobleme einfacher als mit prozeduralen Sprachen lösen lassen, kauft man sich auf der anderen Seite zahlreiche neue Probleme ein. Insbesondere sind objektorientierte Programme nur bedingt *modular erweiterbar*, obwohl Erweiterbarkeit oft als einer der Vorzüge von Objektorientierung gepriesen wird.

Ende der 1990er Jahre wurde durch die Entwicklung aspektorientierter Sprachen versucht, dieses „rissige" Kleid objektorientierter Sprachen zu „flicken", was im großen und ganzen auch gelungen ist. Beispielsweise kann man in AspectJ − im Gegensatz zur Basissprache Java − zu einer bestehenden Klasse nachträglich Datenelemente und Methoden hinzufügen („horizontale Erweiterung"), zusätzliche `implements`-Beziehungen vereinbaren („inkrementelle Erweiterung der Typhierarchie" in begrenztem Umfang) sowie bestehende Methodenimplementierungen nachträglich erweitern oder verändern („Verhaltenserweiterung"), ohne den Quellcode der Klasse verändern zu müssen. Allerdings wirkt AspectJ in der Tat wie ein großes Flickwerk: Für alle gravierenden Mängel von Java, die im Laufe der Jahre offensichtlich wurden (außer das Fehlen parametrischer Polymorphie) wird ein „Patch" angeboten.

Ausgehend von prozeduralen Programmiersprachen, hat man also auf dem Umweg über objekt- und aspektorientierte Sprachen einen bestimmten Punkt in der „Landschaft" der Programmiersprachen erreicht, den man als „moderne imperative Sprachen" bezeichnen könnte.[1] Es stellt sich jedoch die berechtigte Frage, ob man denselben (oder zumindest einen benachbarten) Punkt nicht auch einfacher und direkter erreichen kann, indem man noch einmal zum Ausgangspunkt prozeduraler Sprachen zurückkehrt und diese in einer anderen Richtung erweitert.

Konkret wird vorgeschlagen, zwei Grundelemente prozeduraler Sprachen − Datenstrukturen und Prozeduren − *unabhängig voneinander* zu verallgemeinern, indem man starre Arrays und Records durch flexiblere *offene Typen* und einfache Prozeduren durch *dynamische Prozeduren* ersetzt. Die daraus resultierenden *verbesserten prozeduralen Programmiersprachen* decken einen Großteil der Möglichkeiten objekt- und aspektorientierter Sprachen ab (zum Teil gehen sie sogar darüber hinaus), obwohl sie strukturell wesentlich einfacher (und somit leichter erlernbar, verwendbar und implementierbar) sind. Beispielsweise decken dynamische Prozeduren alle bekannten Formen von *dynamic dispatch* (single, multiple, predicate dispatch) sowie das aspektorientierte Konzept von *advice* durch ein einziges einheitliches Sprachkonzept ab.

Im Vortrag werden die wesentlichen Ideen verbesserter prozeduraler Programmiersprachen zur Diskussion gestellt, die bisher in diesem Bereich durchgeführten Arbeiten skizziert sowie ein Ausblick auf zukünftige Arbeiten gegeben.

[1] Da sowohl prozeduralen als auch objekt- und aspektorientierten Sprachen − im Gegensatz beispielsweise zu funktionalen oder logikbasierten Sprachen − die Abarbeitung einer Befehlsfolge als fundamentales Berechnungsprinzip zugrundeliegt, werden diese hier unter dem Oberbegriff imperativer Sprachen zusammengefasst. Im Gegensatz dazu werden die Bezeichnungen imperativ und prozedural häufig synonym verwendet, was bis zum Auftreten objektorientierter Sprachen auch zutreffend war.