Montages
engineering of computer languages

Author(s):
Kutter, Philipp W.

Publication Date:
2004

Permanent Link:

https://doi.org/10.3929/ethz-a-004751228 →

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Diss. ETH Nr. 15421

Montages

—

Engineering of Computer Languages

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Sciences

presented by

PHILIPP W. KUTTER
Dipl. Inf.-Ing. ETH Zürich
born 04.07.1971
citizen of Küsnacht, Zürich

accepted on the Recommendation of

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Martin Odersky, co-examiner

2004

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

Introduction of higher-level programming languages has constantly been the main driving force behind progress in the software industry. While many people knew for instance how to structure their complex software systems in an object-oriented way before they had object-oriented programming languages at hand, only the introduction of these languages allowed to spread and reuse this knowledge. Similarly, complex databases have been implemented and queried before the introduction of data-base query languages such as SQL, but only these languages made the technique widely available. The natural evolution towards higher level languages continues, for instance with the uniform modeling language UML and its executable variants.

The subject of this work is the design of a language that allows to specify new languages in a simple way, and to make the introduction of new languages a common process for software engineers. We call this process *language engineering*. Language engineering is most interesting if it is applied to small domains, where often a few people know the essential details and where the possiblity to spread and reuse their knowledge is vital. The abstractions common to such a domain can typically be captured with new languages constructs, which are then combined with common constructs from well known general-purpose languages. Therefore the main requirements to such a language description formalism are

- ease of use, in order to let the domain-experts participate in the design, as well as
- describability and reuasability of common language features, such as inheritance or exception handling.

In the first part of this work, we discuss the requirements of language-engineering and analyze what may be the major obstacle for such an approach in practice. Based on this analysis, in the second part, we formally define the language description formalism *Montages*. The semantics of Montages is given as a new kind of state based system, called *tree state-machine*. The states of a tree state-machine are pairs, composed from a position in the abstract syntax tree of a program, and a state in a classical state-machine, which is associated with the kind of language construct currently processed. States are associated with actions, and intermediate results of the computation are stored in the attributes of the nodes in the syntax tree. It is shown, how such language descriptions can be executed directly, and how alternatively a description can be partially evaluated into a specialized interpreter of the described language. Finally, the specialized interpreter together with a program can be further specialized into an executable version of the program. A number of techniques are introduced to control the form of the generated code. The last part shows a specification of the Java programming language in order to demonstrate modularity and reusability of the resulting descriptions.

Zusammenfassung

Die Einführung von Programmiersprachen auf einem höheren Abstractionslevel kann als die treibende Kraft für Fortschritte in der Softwareindustrie angesehen werden. Viele Leute wussten zum Beispiel schon lange bevor OO Sprachen zur Verfügung standen, wie man ein komplexes Softwaresystem objektorientiert strukturiert; aber erst als diese Sprachen eingeführt waren konnte dieses Wissen verbreitet und wiederverwendet werden. Auf eine ähnliche Weise wurden komplexe Datenbanken schon entwickelt und abgefragt bevor man Datenbankabfragesprachen wie SQL kannte, diese Techniken wurden jedoch erst breit einsetzbar nachdem solche Sprachen entwickelt worden sind. Die natürliche Evolution zu Sprachen mit höherem Abstractionslevel geht auch heute noch weiter, wie man am Beispiel der uniform modeling language UML und deren ausführbaren Varianten sieht.

Das Thema dieser Arbeit ist der Entwurf einer Sprache die es erlaubt neue Sprachen auf eine einfache Art und Weise zu spezifizieren, und es somit möglich macht das Einführen einer neuen Sprache in den Software Ingenieurprozess einzubetten. Diesen Prozess nennen wir *Language Engineering*. Die Anwendung von Language Engineering ist am interessantesten in kleinen, spezialisierten Anwendungsbereichen, wo das essentielle Detailwissen oft von einer kleinen Zahl von Spezialisten getragen wird. Dadurch wird die Möglichkeit dieses Wissen zu verbreiten und wiederzuverwenden emminent wichtig. Die Abstraktionen in einem solchen Anwendungsgebiet werden typisch mit neuen Sprachkonstrukten erfasst, welche dann mit allgemeinen, bekannten Programmierkonstrukten aus existierenden Sprachen kombiniert werden. Daher sind die wichtigsten Anforderungen an einen solchen Sprachbeschreibungsformalismuss, dass

- der Formalismus so einfach zu benutzen ist, dass die Spezialisten in den Entwurfsprozess einbezogen werden können, als auch dass
- allgemeine Programmierkonstrukte aus existierenden Sprachen einfach beschrieben werden können, und diese Beschreibungen einfach wiederverwendet werden können.

Im ersten Teil dieser Arbeit besprechen wir die Anforderungen an eine Language Engineering Disziplin and analysieren welches die Hauptprobleme im praktischen Einsatz sind. Basierend auf dieser Analyse, definieren wir im zweiten Teil den Sprachbeschreibungsformalismus *Montages*. Die Semantik von Montages wird mit einer neuen Art von zustandbasierten Systemen gegeben, welche wir *Tree State-Machine* nennen. Die Zustände einer solchen Tree State-Machine sind Paare, bestehend aus einer Position im abstrakten Syntaxbaum eines Programs, und einem Zustand eines klassischen Zustandautomatens bestehen, welcher wiederum mit der Klasse des prozessierten Sprachkonstrukt assoziiert ist. Die Zustände sind mit Aktionen verbunden, und Zwischenresultate werden als Attribute der Knoten des Syntaxbaumes gespeichert. Wir zeigen, wie solche Sprachbeschrei-

bungen direkt ausgeführt werden können, und wie man alternativ eine Beschreibung partiell evaluieren kann, um aus ihr einen speziallisierten Interpreter der beschriebenen Sprache zu gewinnen. Schlussendlich kann der Interpreter zusammen mit einem Program in eine ausführbare Version des Programs übergeführt werden. Eine Reihe von Techniken werden eingeführt um die Form des Resultierenden Codes zu kontrolieren. Der letzte Teil der Arbeit beinhalted eine Spezifikation der Java Programmiersprache um die Modularität und Wiederverwendbarkeit der resultierenden Beschreibungen zu zeigen.