Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

490

J.A. Bergstra L.M.G. Feijs (Eds.)

Algebraic Methods II: Theory, Tools and Applications

Springer-Verlag

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

490

J.A. Bergstra L.M.G. Feijs (Eds.)

Algebraic Methods II: Theory, Tools and Applications

Springer-Verlag
Berlin Heidelberg New York London Paris
Tokyo Hong Kong Barcelona Budapest

Volume Editor

Jan A. Bergstra
Department of Computer Science, University of Amsterdam
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

Loe M. G. Feijs
Philips Research Laboratories
P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands

CR Subject Classification (1991): C.2.2, D.1–3, F.3

ISBN 3-540-53912-3 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-53912-3 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

Preface

This volume originates from a workshop organized by ESPRIT project no. 432 METEOR in Mierlo, The Netherlands, September 1989. The workshop was a successor to an earlier one held in Passau, Germany, June 9-11 1987, the proceedings of which have been published as Lecture Notes in Computer Science Volume 394.

At the workshop, six invited talks were given by A. Finkelstein, C.B. Jones, P. Klint, C.A. Middelburg, E.-R. Olderog and H.A. Partsch.

The program committee consisted of

M. Wirsing,
H. Perdrix,
J.A. Bergstra,
J.C.M. Baeten,
L.M.G. Feijs,
J. Hagelstein,
F. Ponsaert,
M.-C. Gaudel,
R. Zicari.

This volume contains five invited contributions and ten papers by the METEOR team based on talks that were presented at the workshop. The invited talk of Jones led to a paper by Feijs on modularizing the formal description of a database which has been included as well.

The program committee would like to thank P. Wodon (project leader of METEOR), A. Bradier (ESPRIT project officer) and PRLE (organizer of the workshop). The financial support of the following partners of the METEOR project

Philips Research Laboratories Brussels,
Philips Research Laboratories Eindhoven,
Compagnie Générale d'Electricité,
LRI – Université Paris-Sud,
ATT & Philips Telecommunications,
Centrum voor Wiskunde en Informatica,
TXT,
Politechnico di Milano,
Universität Passau

is gratefully acknowledged.

Finally, as the editors of the volume we would like to thank R.D. van den Bos for his initiative and help in preparing this volume and Springer-Verlag for their excellent cooperation concerning the publication of this volume.

Eindhoven, January 1991 Jan A. Bergstra, Loe M. G. Feijs

Introduction

This volume is divided in four parts. Part I contains the invited lectures. These lectures cover a variety of topics ranging from requirements engineering and transformational design to the construction of programming environments and the design of wide-spectrum languages. Part II, III and IV contain papers from the METEOR team.

The rationale for the grouping of papers is the following: as COLD is a major result of the METEOR project all information about COLD has been collected in one part (III). COLD is an algebraic technique because it starts out from sorts, functions and algebras. It extends the conventional algebraic paradigm by incorporating features from sequential imperative programming, dynamic logic and first and second order predicate logic.

Because conventional algebraic specification techniques based on equational logic have played a key role in METEOR, contributions in that area have been collected in a single part as well.

Part II collects papers on topics that were of secondary, but still vital importance to METEOR: requirements engineering, design and transformation.

Introduction

We briefly survey the invited papers.

Van Diepen & Partsch discuss the formalisation of informal requirements acquisition. The discussion is based on a case study and leads to requirements on formalisms for requirements definition and a description of formalization as a process.

Finkelstein et al. introduce viewpoint oriented software development. Their method is technically based on FOREST and its underlying mathematical foundation MAL (Modal Action Logic). To these ideas it adds so-called structured common sense (SCS). The method is illustrated by examples.

Klint describes a meta environment for generating programming environments. His work was done in ESPRIT project no. 348 GIPE. The environment is a part of the computer system that incorporates formalisms and subsystems such as TYPOL, ASF, SDF and METAL. In particular Klint describes the environment generator for ASF and SDF, where ASF constitutes a Spartan syntax for structured algebraic specifications to which ASD adds a significant amount of user oriented syntactic freedom.

The paper "Using transformations to Verify Parallel Programs" by K. Apt and E.-R. Olderog addresses the construction of parallel programs that formally satisfy a pre- and postcondition style specification. The approach is to use program transformations which leads to significant simplifications and which can be used in combination with the proof method of Owicki and Gries.

The paper of C. Middelburg reports on the integration of language concepts into a wide-spectrum language. He combines VDM with a language of temporal logic. There are several links with METEOR here. First the role of temporal logic has been investigated for requirements engineering in METEOR; in particular ERAE is based on temporal logic also. Secondly VVSL reflects strong influences from COLD; from COLD it gets its modularization and parameterization mechanisms. Also the way of translating VVSL to MPL_ω is derived from the formal semantics of COLD. Finally the integration of languages and concepts into a wide-spectrum language is very difficult and will be a research topic for the near future.

Table of Contents

PART I
Invited Contributions

Introduction

This volume is divided in four parts. Part I contains the invited lectures. These lectures cover a variety of topics ranging from requirements engineering and transformational design to the construction of programming environments and the design of wide-spectrum languages. Part II, III and IV contain papers from the METEOR team.

The rationale for the grouping of papers is the following: as COLD is a major result of the METEOR project all information about COLD has been collected in one part (III). COLD is an algebraic technique because it starts out from sorts, functions and algebras. It extends the conventional algebraic paradigm by incorporating features from sequential imperative programming, dynamic logic and first and second order predicate logic.

Because conventional algebraic specification techniques based on equational logic have played a key role in METEOR, contributions in that area have been collected in a single part as well.

Part II collects papers on topics that were of secondary, but still vital importance to METEOR: requirements engineering, design and transformation.

PART I
Invited Contributions

Introduction

We briefly survey the invited papers.

Van Diepen & Partsch discuss the formalisation of informal requirements acquisition. The discussion is based on a case study and leads to requirements on formalisms for requirements definition and a description of formalization as a process.

Finkelstein et al. introduce viewpoint oriented software development. Their method is technically based on FOREST and its underlying mathematical foundation MAL (Modal Action Logic). To these ideas it adds so-called structured common sense (SCS). The method is illustrated by examples.

Klint describes a meta environment for generating programming environments. His work was done in ESPRIT project no. 348 GIPE. The environment is a part of the computer system that incorporates formalisms and subsystems such as TYPOL, ASF, SDF and METAL. In particular Klint describes the environment generator for ASF and SDF, where ASF constitutes a Spartan syntax for structured algebraic specifications to which ASD adds a significant amount of user oriented syntactic freedom.

The paper "Using transformations to Verify Parallel Programs" by K. Apt and E.-R. Olderog addresses the construction of parallel programs that formally satisfy a pre- and postcondition style specification. The approach is to use program transformations which leads to significant simplifications and which can be used in combination with the proof method of Owicki and Gries.

The paper of C. Middelburg reports on the integration of language concepts into a wide-spectrum language. He combines VDM with a language of temporal logic. There are several links with METEOR here. First the role of temporal logic has been investigated for requirements engineering in METEOR; in particular ERAE is based on temporal logic also. Secondly VVSL reflects strong influences from COLD; from COLD it gets its modularization and parameterization mechanisms. Also the way of translating VVSL to MPL_ω is derived from the formal semantics of COLD. Finally the integration of languages and concepts into a wide-spectrum language is very difficult and will be a research topic for the near future.

Formalizing Informal Requirements
Some Aspects

N.W.P. van Diepen
H.A. Partsch

University of Nijmegen*
Department of Computer Science
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Abstract

Formal specifications are nowadays considered as an important intermediate stage in the software development process. There are various approaches for constructing an efficient program satisfying a given formal specification. The formalization process, however, has not yet been investigated as thoroughly. Thus, it is still one of the main sources for inconsistencies between the wishes of the customer and the program finally delivered. Some problems to be solved during formalization are identified and illustrated with a real-world example.

1 Introduction

In its widest sense, software development means

"given a problem, find a program (or a set of programs) that (efficiently) solves the problem"

where program may be taken as synonymous with software.

The major difficulty in software development is caused by the fact that the original problem description usually consists of a bunch of half-baked wishes which are neither precise or detailed, nor even complete. The program, by nature, has to be precisely defined and fully detailed up to each single instruction. It is obvious that software development done in one large step to bridge the huge gap between these extreme positions is doomed to fail, i.e., the resulting software probably does not work as expected.

There are various reasons why software might not work properly. Very often, the problem given originally was simply misunderstood or misinterpreted. Therefore, it is widely accepted today that the process of software development should be broken into smaller, manageable, steps in the framework of so-called "life cycle models". A minimum requirement is a decomposition into two steps (frequently called "requirements engineering" and "program construction") with a precise, possibly formal, statement of the problem as an intermediate stage (cf., e.g., also [Balzer et al. 83], [Agresti 86], [Bauer et al. 89]).

Such a formal problem specification states precisely and unambiguously the "task" to be fulfilled by the software, i.e., it describes what the problem is without giving a direct solution or even the details about its implementation. Additionally, it entails a "separation of concerns" which allows early checks on whether the informal wishes are properly reflected and thus prevents superfluous implementation work.

There are various approaches focusing on the program construction part of this development paradigm, viz. how to construct an efficient program that satisfies a given formal specification, e.g., by transformations (for overviews, see [Partsch Steinbrüggen 83], [Feather 86]), or assertional techniques ([Dijkstra 76], [Gries 81], [Backhouse 86]). The requirements engineering part, although at least as important, has not yet as thoroughly been investigated in the context of these new approaches and paradigms. However, a lot of work in this area has been done within traditional software engineering. Therefore, in the following

*Support has been received from the Netherlands Organization for Scientific Research N.W.O. under grant NF 63/62-518 (the STOP — Specification and Transformation Of Programs — project).