The Knowledge Engineering Review, Vol. 11:4, 1996, 301

Guest Editorial

NORBERT E. FUCHS

This issue of *The Knowledge Engineering Review* is dedicated to logic engineering. The phrase 'logic engineering' has two roots: computational logic, the programming paradigm based on an executable subset of logic, and software engineering, the craft of efficiently and economically developing software of a high quality. Thus, logic engineering concerns the application of computational logic to the formulation and solution of problems that arise in software engineering in the broadest sense.

Though its full potential remains to be utilised, the marriage between computational logic and software engineering is a very fortunate one, since computational logic

- can serve as a high level specification language and as a programming language,
- can be used to develop tools, even complete development environments,
- allows meta-programming and thus self-applicability.

Arguably, computational logic can be applied to most fields of software engineering. The collection of papers in this issue bears witness to this claim.

In his paper "Declarative Process Modelling using Logic Programming", Paolo Ciancarini addresses the complete software development process. He shows how this process can be effectively formalised by rule-based languages, specifically logic languages, describing the specification, modelling, enactment and coordination of the process. Process modelling ranges from simple animation to complex programming environments using project databases.

In their contribution "Declarative Specifications", Norbert E Fuchs and David Robertson concentrate on requirements engineering. They introduce application-specific specification languages as semantically equivalent representations of logic specifications. By doing this they effectively bridge the conceptual gap between an application domain and the domain of formal methods, and make formal methods available to application specialists who may not be familiar with them.

Leon Sterling and Ümit Yalçinalp show in "Logic Programming and Software Engineering—Implications for Software Design" how programming experience impacts the design of logic programs. Programming experience can be abstracted in reusable patterns that allow systematic program development, design for provability and meta-programming. The use of program skeletons expressing the basic control flow and of techniques adding functionality to a skeleton allows program development by step-wise enhancement.

The software development process can be divided into two phases, one problem-oriented and one efficiency-oriented. Once we have a correct logic specification, we need to derive from it an efficient program. Computational logic provides a wealth of techniques for doing this, and Alberto Pettorossi and Maurizio Proietti in "Developing Correct and Efficient Logic Programs by Transformation" provide a comprehensive overview, describing rule-based program transformation, schema-based program transformation and partial evaluation.

I would like to thank John Fox and Paul Krause for inviting me to edit this special issue on logic engineering for *The Knowledge Engineering Review*, and the authors for their contributions.