Preface

One of the goals of software production in future decades must be to reduce production costs and time to market considerably, while improving the quality and stability of the product. This can be achieved most effectively if the broad mass of application programmers is given the opportunity to apply their expert knowledge in their application domain but is not expected to have in-depth knowledge of the issues of software engineering and implementation.

The considerable gap between application-level problem solutions and efficient implementations at the level of today's source programs as written in C or Java must be bridged by sophisticated optimizing techniques of program generation. At best, these techniques should be fully automatable – at least, they should have strong automatic support. If this can be achieved, program generation has the potential to revolutionize software development just like automation and components revolutionized manufacturing.

This book is about domain-specific program generation. It is a condensation of contributions all but one of which were presented at a five-day seminar with the same title at Schloss Dagstuhl, Germany, in March of 2003. The seminar brought together researchers from four different communities:

- *Domain-specific languages:* Language developers in a specific application domain have often been unaware of the domain-independent aspects of their domain-specific work. Vice versa, researchers who do not work in a specific domain are often unaware of some of the factors that make an application work.
- *High-performance parallelism:* This is one application domain which has led to the development of a particular form of domain-specific language (so-called skeletons). Researchers in this community are becoming interested in the wider aspects of domain-specific program generation.
- *Program generators:* This domain is concerned with the fast and reliable generation of members of a program family, also called a product line. There are many applications of product lines in industry, commerce, and the military.
- *Metaprogramming:* Researchers in this community develop a technology for combining and specializing program fragments. This requires at least two levels of code: one in which the fragments are coded and one which combines and specializes the fragments. This technology can be used for customizing compilation and translation systems for domain-specific purposes. Multi-staging is a special form of metaprogramming, in which each level is coded in the same programming language.

This volume has four parts.

Surveys. Six surveys attempt to give some structure to the diverse world of domain-specific programming technology.

1. Batory retraces the evolution of the very successful domain of data base query optimization and discusses what lessons can potentially be learned for other domains.

2. Consel makes the point that a domain is best defined as a set of existing programs and sketches how one might derive a domain-specific language from such a set, with which one can then specify other programs in this set.
3. Taha illustrates the technique of multi-stage programming on the example of a staged interpreter.
4. Czarnecki et al. describe staged interpreters and templates as a suitable way of extending a host language with domain-specific constructs. They evaluate and compare three languages for template programming: MetaOCaml, Template Haskell and C++.
5. One step beyond domain-specific program generation lies the goal of domain-specific program optimization. Lengauer reviews different optimization techniques in the domain of high-performance parallelism.
6. Smaragdakis offers a personal assessment of the approaches and attitudes in the research community of generative programming.

Domain-Specific Languages. Five contributions describe domain-specific programming languages or language enhancements.

1. Bischof, Gorlatch and Leshchinskiy present the skeleton library DatTeL for the domain of high-performance parallelism. The library's two key features are (1) its user interface, which is similar to the C++ Standard Template Library (STL) and which facilitates a smooth transition from sequential to parallel programming, and (2) an efficient implementation of STL-like constructs on parallel computers.
2. Hammond and Michaelson describe the language Hume for the domain of real-time embedded systems. Hume has high-level features typical for functional languages. Since it consists of three layers, Hume also allows for domain-specific metaprogramming.
3. O'Donnell describes an embedding of the language Hydra for the domain of digital circuit design into the host language Template Haskell.
4. Consel and Réveillère present a programming paradigm for the domain of services for mobile communication terminals. This domain is subject to frequent changes in technology and user requirements. The paradigm enables the quick development of robust communication services under these challenging circumstances.
5. Cremet and Odersky choose the π-calculus for mobile processes as their domain. They describe the domain-specific language PiLib, which is implemented as a library in Odersky's new language Scala. With the features of Scala, calls to PiLib can be made to look almost like π-formulas.

Tools for Program Generation. Three further contributions stress issues of tool support for program generation in a domain-specific language.

1. Gregg and Ertl work with their language vmIDL for describing virtual machine instructions. Their tool vmgen takes the specification of such instructions in their domain-specific language and returns efficient implementations of the instructions in C.

2. Visser presents the Stratego language for the domain of rewriting program transformations, and the corresponding toolset Stratego/XT.
3. Fischer and Visser work with AutoBayes, a fully automatic, schema-based program synthesis system for applications in the analysis of statistical data. It consists of a domain-specific schema library, implemented in Prolog. In their contribution to this volume, they discuss the software engineering challenges in retro-fitting the system with a concrete, domain-specific syntax.

Domain-Specific Optimization. Finally, four contributions describe domain-specific techniques of program optimization.

1. Kuchen works with a skeleton library for high-performance parallelism, similar to the library DatTel of Bischof et al. However, Kuchen's library is not based on STL. Instead, it contains alternative C++ templates which are higher-order, enable type polymorphism and allow for partial application. In the second half of his treatise, Kuchen discusses ways of optimizing (i.e., "retuning") sequences of calls of his skeletons.
2. Gorlatch addresses exactly the same problem: the optimization of sequences of skeleton calls. His skeletons are basic patterns of communication and computation – so-called collective operations, some of which are found in standard communication libraries like MPI. Gorlatch also discusses how to tune compositions of skeleton calls for a distributed execution on the Grid.
3. Beckmann et al. describe the TaskGraph library: a further library for C++ which can be used to optimize, restructure and specialize parallel target code at run time. They demonstrate that the effort spent on the context-sensitive optimization can be heavily outweighed by the gains in performance.
4. Veldhuizen pursues the idea of a compiler for an extensible language, which can give formal guarantees of the performance of its target code.

Each submission was reviewed by one person who was present at the corresponding presentation at Schloss Dagstuhl and one person who did not attend the Dagstuhl seminar.[1] There were two rounds of reviews. Aside from the editors themselves, the reviewers were:

Ira Baxter	Christoph M. Kirsch	Ulrik Schultz
Claus Braband	Shriram Krishnamurthy	Tim Sheard
Krzysztof Czarnecki	Calvin Lin	Satnam Singh
Albert Cohen	Andrew Lumsdaine	Yannis Smaragdakis
Marco Danelutto	Anne-Françoise Le Meur	Jrg Striegnitz
Olivier Danvy	Jim Neighbors	Andrew Tolmach
Prem Devanbu	John O'Donnell	Todd Veldhuizen
Sergei Gorlatch	Catuscia Palamidessi	Harrick Vin
Kevin Hammond	Susanna Pelagatti	David Wile
Christoph A. Herrmann	Simon Peyton Jones	Matthias Zenger
Zhenjiang Hu	Frank Pfenning	
Paul H. J. Kelly	Laurent Réveillère	

[1] An exception is the contribution by Cremet and Odersky, which was not presented at Schloss Dagstuhl.

IFIP Working Group. At the Dagstuhl seminar, plans were made to form an IFIP TC-2 Working Group: WG 2.11 on Program Generation. In the meantime, IFIP has given permission to go ahead with the formation. The mission statement of the group follows this preface.

Acknowledgements. The editors, who were also the organizers of the Dagstuhl seminar, would like to thank the participants of the seminar for their contributions and the reviewers for their thoughtful reviews and rereviews. The first editor is grateful to Johanna Bucur for her help in the final preparation of the book.

We hope that this volume will be a good ambassador for the new IFIP WG.

March 2004

Christian Lengauer
Don Batory
Charles Consel
Martin Odersky