Peter Van Roy (Ed.)

Multiparadigm Programming in Mozart/Oz

Second International Conference, MOZ 2004
Charleroi, Belgium, October 7-8, 2004
Revised Selected and Invited Papers

Springer

Volume Editor

Peter Van Roy
Université catholique de Louvain
Department of Computing Science and Engineering
Place Sainte Barbe, 2, B-1348 Louvain-la-Neuve, Belgium
E-mail: pvr@info.ucl.ac.be

Library of Congress Control Number: 2005921638

CR Subject Classification (1998): D.3, F.3, D.2, D.1, D.4

ISSN 0302-9743
ISBN 3-540-25079-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper   SPIN: 11398158   06/3142   5 4 3 2 1 0

Foreword

To many readers, Mozart/Oz represents a new addition to the pantheon of programming systems. One way of evaluating a newcomer is through the eyes of the classics, for example Kernighan and Pike's "The Practice of Programming," a book that concludes with six "lasting concepts": simplicity and clarity, generality, evolution, interfaces, automation, and notation. Kernighan and Pike concentrate on using standard languages such as C and Java to implement these concepts, but it is instructive to see how a multiparadigm language such as Oz changes the outlook.

Oz's concurrency model yields simplicity and clarity (because Oz makes it easier to express complex programs with many interacting components), generality, and better interfaces (because the dataflow model automatically makes interfaces more lightweight).

Constraint programming in Oz again yields simplicity and clarity (because the programmer can express what needs to be true rather than the more complex issue of how to make it true), and offers a powerful mathematical notation that is difficult to implement on top of languages that do not support it natively.

Mozart's distributed computing model makes for improved interfaces and eases the evolution of systems. In my own work, one of the most important concerns is to be able to quickly scale up a prototype implementation into a large-scale service that can run reliably on thousands of computers, serving millions of users. The field of computer science needs more research to discover the best ways of facilitating this, but Mozart provides one powerful approach.

Altogether, Mozart/Oz helps with all the lasting concepts except automation, and it plays a particularly strong role in notation, which Kernighan and Pike point out is an underappreciated area. I believe that providing the right notation is the most important of the six concepts, one that supports all the others. Multiparadigm systems such as Oz provide more choices for notation than single-paradigm languages.

Going beyond Kernighan and Pike's six concerns, I recognize three more concerns that I think are important, and cannot be added on to a language by writing functions and classes; they must be inherent to the language itself.

The first is the ability to separate concerns, to describe separate aspects of a program separately. Mozart supports separation of fault tolerance and distributed computation allocation in an admirable way.

My second concern is security. Sure, you can eliminate a large class of security holes by replacing the `char*` datatype with `string`, but strong security cannot be guaranteed in a language that is not itself secure.

My third concern is performance. David Moon once said, in words more pithy than I can recall, that you can abstract anything except performance. That is, you can add abstraction layers, but you can't get back sufficient speed if the underlying language implementation doesn't provide it. Mozart/Oz has a 10-year history of making choices that provide for better performance, thereby making the system a platform that will rarely run up against fundamental performance problems.

We all look for tools and ideas to help us become better programmers. Sometimes the most fundamental idea is to pick the right programming environment.

Peter Norvig
Director of Search Quality, Google, Inc.
Coauthor, *Artificial Intelligence: A Modern Approach*

Preface

Multiparadigm programming, when done well, brings together the best parts of different programming paradigms in a simple and powerful whole. This allows the programmer to choose the right concepts for each problem to be solved. This book gives a snapshot of the work being done with Mozart/Oz, one of today's most comprehensive multiparadigm programming systems. Mozart/Oz has been under development since the early 1990s as a vehicle to support research in programming languages, constraint programming, and distributed programming.[1] Since then, Mozart/Oz has matured into a production-quality system with an active user community. Mozart/Oz consists of the Oz programming language and its implementation, Mozart. Oz combines the concepts of all major programming paradigms in a simple and harmonious whole. Mozart is a high-quality open source implementation of Oz that exists for different versions of Windows, Unix/Linux/Solaris, and Mac OS X.[2]

This book is an extended version of the proceedings of the 2nd International Mozart/Oz Conference (MOZ 2004), which was held in Charleroi, Belgium on October 7 and 8, 2004. MOZ 2004 consisted of 23 technical talks, four tutorials, and invited talks by Gert Smolka and Mark S. Miller. The slides of all talks and tutorials are available for downloading at the conference website.[3] This book contains all 23 papers presented at the conference, supplemented with two invited papers written especially for the book. The conference papers were selected from 28 submissions after a rigorous reviewing process in which most papers were reviewed by three members of the Program Committee. We were pleasantly surprised by the high average quality of the submissions.

Mozart/Oz research and development started in the early 1990s as part of the ACCLAIM project, funded by the European Union. This project led to the Mozart Consortium, an informal but intense collaboration that initially consisted of the Programming Systems Lab at Saarland University in Saarbrücken, Germany, the Swedish Institute of Computer Science in Kista, Sweden, and the Université catholique de Louvain in Louvain-la-Neuve, Belgium. Several other institutions have since joined this collaboration. Since the publication in March 2004 of the textbook *Concepts, Techniques, and Models of Computer Programming* by MIT Press, the Mozart/Oz community has grown significantly. As a result, we are reorganizing the Mozart Consortium to make it more open.

Security and Concurrency

Two important themes in this book are security and concurrency. The book includes two invited papers on language-based computer security. Computer secu-

[1] In the early days before the Mozart Consortium the system was called DFKI Oz.
[2] See `www.mozart-oz.org`.
[3] See `www.cetic.be/moz2004`.

rity is a major preoccupation today both in the computer science community and in general society. While there are many short-term solutions to security problems, a good long-term solution requires rethinking our programming languages and operating systems. One crucial idea is that languages and operating systems should thoroughly support the principle of least authority. This support starts from the user interface and goes all the way down to basic object invocations. With such thorough support, many security problems that are considered difficult today become much simpler. For example, the so-called trade-off between security and usability largely goes away. We can have security without compromising usability. The two invited papers are the beginning of what we hope will become a significant effort from the Mozart/Oz community to address these issues and propose solutions.

The second important theme of this book is concurrent programming. We have built Mozart/Oz so that concurrency is both easy to program with and efficient in execution. Many papers in the book exploit this concurrency support. Several papers use a multiagent architecture based on message passing. Other papers use constraint programming, which is implemented with lightweight threads and declarative concurrency. We find that both message-passing concurrency and declarative concurrency are much easier to program with than shared-state concurrency. The same conclusion has been reached independently by others. Joe Armstrong, the main designer of the Erlang language, has found that using message-passing concurrency greatly simplifies building software that does not crash. Doug Barnes and Mark S. Miller, the main designers of the E language, have found that message-passing concurrency greatly simplifies building secure distributed systems. E is discussed in both of the invited papers in this book.

Joe Armstrong has coined the phrase *concurrency-oriented programming* for languages like Oz and Erlang that make concurrency both easy and efficient. We conclude that concurrency-oriented programming will become increasingly important in the future. This is not just because concurrency is useful for multiagent systems and constraint programming. It is really because concurrency makes it easier to build software that is reliable and secure.

Diversity and Synergy

Classifying the papers in this book according to subject area gives an idea of the diversity of work going on under the Mozart banner: security and language design, computer science education, software engineering, human-computer interfaces and the Web, distributed programming, grammars and natural language, constraint research, and constraint applications. Constraints in Mozart are used to implement games (Oz Minesweeper), to solve practical problems (reconfiguration of electrical power networks, aircraft sequencing at an airport, timetabling, etc.), and to do complex symbolic calculation (such as natural language processing and music composition). If you start reading the book knowing only some of these areas, then I hope that it will encourage you to get involved with the others. Please do not hesitate to contact the authors of the papers to ask for software and advice.

The most important strength of Mozart, in my view, is the synergy that comes from connecting areas that are usually considered as disjoint. The synergy is strong because the connections are done in a deep way, based on the fundamental concepts of each area and their formal semantics. It is my hope that this book will inspire you to build on this synergy to go beyond what has been done before. Research and development, like many human activities, are limited by a psychological barrier similar to that which causes sports records to advance only gradually. It is rare that people step far beyond the boundaries of what has been done before. One way to break this barrier is to take advantage of the connections that Mozart offers between different areas. I hope that the wide variety of examples shown in this book will help you to do that.

In conclusion, I would like to thank all the people who made MOZ 2004 and this book a reality: the paper authors, the Program Committee members, the Mozart developers, and, last but not least, the CETIC asbl, who organized the conference in a professional manner. I thank Peter Norvig of Google, Inc., who graciously accepted to write the Foreword for this book. And, finally, I give a special thanks to Donatien Grolaux, the local arrangements chair, for his hard work in handling all the practical details.

November 2004 Peter Van Roy
Louvain-la-Neuve, Belgium

Organization

MOZ 2004 was organized by CETIC in cooperation with the Université catholique de Louvain. CETIC asbl is the Centre of Excellence in Information and Communication Technologies, an applied research laboratory based in Charleroi, Belgium.[1] CETIC is focused on the fields of software engineering, distributed computing, and electronic systems. The Université catholique de Louvain was founded in 1425 and is located in Louvain-la-Neuve, Belgium.

Organizing Committee

Program Committee

[1] See `www.cetic.be`.

Table of Contents

Author Index