Peter Van Roy (Ed.)

LNCS 3389

Multiparadigm Programming in Mozart/Oz

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

Springer

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

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

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

Lecture Notes in Computer Science 3389

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
Louvain-la-Neuve, Belgium

Peter Van Roy

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

Donatien Grolaux, CETIC, Belgium (local arrangements chair)
Bruno Carton, CETIC, Belgium
Pierre Guisset, director, CETIC, Belgium
Peter Van Roy, Université catholique de Louvain, Belgium

Program Committee

Per Brand, Swedish Institute of Computer Science, Sweden
Thorsten Brunklaus, Saarland University, Germany
Raphaël Collet, Université catholique de Louvain, Belgium
Juan F. Díaz, Universidad del Valle, Cali, Colombia
Denys Duchier, INRIA Futurs, Lille, France
Sameh El-Ansary, Swedish Institute of Computer Science, Sweden
Kevin Glynn, Université catholique de Louvain, Belgium
Donatien Grolaux, CETIC, Belgium
Seif Haridi, KTH – Royal Institute of Technology, Sweden
Martin Henz, FriarTuck and the National University of Singapore
Erik Klintskog, Swedish Institute of Computer Science, Sweden
Joachim Niehren, INRIA Futurs, Lille, France
Luc Onana, KTH – Royal Institute of Technology, Sweden
Konstantin Popov, Swedish Institute of Computer Science, Sweden
Mahmoud Rafea, Central Laboratory for Agricultural Expert Systems, Egypt
Juris Reinfelds, New Mexico State University, USA
Andreas Rossberg, Saarland University, Germany
Camilo Rueda, Pontificia Universidad Javeriana, Cali, Colombia
Christian Schulte, KTH – Royal Institute of Technology, Sweden
Gert Smolka, Saarland University, Germany
Fred Spiessens, Université catholique de Louvain, Belgium
Peter Van Roy, Université catholique de Louvain, Belgium (Program Chair)

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

Lecture Notes in Computer Science

For information about Vols. 1–3317

please contact your bookseller or Springer

Vol. 3452: F. Baader, A. Voronkov (Eds.), Logic Programming, Artificial Intelligence, and Reasoning. XI, 562 pages. 2005. (Subseries LNAI).

Vol. 3436: B. Bouyssounouse, J. Sifakis (Eds.), Embedded Systems Design. XV, 492 pages. 2005.

Vol. 3418: U. Brandes, T. Erlebach (Eds.), Network Analysis. XII, 471 pages. 2005.

Vol. 3416: M. Böhlen, J. Gamper, W. Polasek, M.A. Wimmer (Eds.), E-Government: Towards Electronic Democracy. XIII, 311 pages. 2005. (Subseries LNAI).

Vol. 3414: M. Morari, L. Thiele (Eds.), Hybrid Systems: Computation and Control. XII, 684 pages. 2005.

Vol. 3412: X. Franch, D. Port (Eds.), COTS-Based Software Systems. XVI, 312 pages. 2005.

Vol. 3411: S.H. Myaeng, M. Zhou, K.-F. Wong, H.-J. Zhang (Eds.), Information Retrieval Technology. XIII, 337 pages. 2005.

Vol. 3410: C.A. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization. XVI, 912 pages. 2005.

Vol. 3409: N. Guelfi, G. Reggio, A. Romanovsky (Eds.), Scientific Engineering of Distributed Java Applications. X, 127 pages. 2005.

Vol. 3406: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. XVII, 829 pages. 2005.

Vol. 3404: V. Diekert, B. Durand (Eds.), STACS 2005. XVI, 706 pages. 2005.

Vol. 3403: B. Ganter, R. Godin (Eds.), Formal Concept Analysis. XI, 419 pages. 2005. (Subseries LNAI).

Vol. 3401: Z. Li, L. Vulkov, J. Waśniewski (Eds.), Numerical Analysis and Its Applications. XIII, 630 pages. 2005.

Vol. 3398: D.-K. Baik (Ed.), Systems Modeling and Simulation: Theory and Applications. XIV, 733 pages. 2005. (Subseries LNAI).

Vol. 3397: T.G. Kim (Ed.), Artificial Intelligence and Simulation. XV, 711 pages. 2005. (Subseries LNAI).

Vol. 3396: R.M. van Eijk, M.-P. Huget, F. Dignum (Eds.), Agent Communication. X, 261 pages. 2005. (Subseries LNAI).

Vol. 3395: J. Grabowski, B. Nielsen (Eds.), Formal Approaches to Software Testing. X, 225 pages. 2005.

Vol. 3393: H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, G. Taentzer (Eds.), Formal Methods in Software and Systems Modeling. XXVII, 413 pages. 2005.

Vol. 3391: C. Kim (Ed.), Information Networking. XVII, 936 pages. 2005.

Vol. 3390: R. Choren, A. Garcia, C. Lucena, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems III. XII, 291 pages. 2005.

Vol. 3389: P. Van Roy (Ed.), Multiparadigm Programming in Mozart/Oz. XV, 329 pages. 2005.

Vol. 3388: J. Lagergren (Ed.), Comparative Genomics. VIII, 133 pages. 2005. (Subseries LNBI).

Vol. 3387: J. Cardoso, A. Sheth (Eds.), Semantic Web Services and Web Process Composition. VIII, 147 pages. 2005.

Vol. 3386: S. Vaudenay (Ed.), Public Key Cryptography - PKC 2005. IX, 436 pages. 2005.

Vol. 3385: R. Cousot (Ed.), Verification, Model Checking, and Abstract Interpretation. XII, 483 pages. 2005.

Vol. 3383: J. Pach (Ed.), Graph Drawing. XII, 536 pages. 2005.

Vol. 3382: J. Odell, P. Giorgini, J.P. Müller (Eds.), Agent-Oriented Software Engineering V. X, 239 pages. 2005.

Vol. 3381: P. Vojtáš, M. Bieliková, B. Charron-Bost, O. Sýkora (Eds.), SOFSEM 2005: Theory and Practice of Computer Science. XV, 448 pages. 2005.

Vol. 3379: M. Hemmje, C. Niederee, T. Risse (Eds.), From Integrated Publication and Information Systems to Information and Knowledge Environments. XXIV, 321 pages. 2005.

Vol. 3378: J. Kilian (Ed.), Theory of Cryptography. XII, 621 pages. 2005.

Vol. 3377: B. Goethals, A. Siebes (Eds.), Knowledge Discovery in Inductive Databases. VII, 190 pages. 2005.

Vol. 3376: A. Menezes (Ed.), Topics in Cryptology – CT-RSA 2005. X, 385 pages. 2004.

Vol. 3375: M.A. Marsan, G. Bianchi, M. Listanti, M. Meo (Eds.), Quality of Service in Multiservice IP Networks. XIII, 656 pages. 2005.

Vol. 3374: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems. X, 279 pages. 2005. (Subseries LNAI).

Vol. 3372: C. Bussler, V. Tannen, I. Fundulaki (Eds.), Semantic Web and Databases. X, 227 pages. 2005.

Vol. 3369: V.R. Benjamins, P. Casanovas, J. Breuker, A. Gangemi (Eds.), Law and the Semantic Web. XII, 249 pages. 2005. (Subseries LNAI).

Vol. 3368: L. Paletta, J.K. Tsotsos, E. Rome, G.W. Humphreys (Eds.), Attention and Performance in Computational Vision. VIII, 231 pages. 2005.

Vol. 3366: I. Rahwan, P. Moraitis, C. Reed (Eds.), Argumentation in Multi-Agent Systems. XII, 263 pages. 2005. (Subseries LNAI).

Vol. 3365: G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 415 pages. 2005.

Vol. 3363: T. Eiter, L. Libkin (Eds.), Database Theory - ICDT 2005. XI, 413 pages. 2004.

Vol. 3362: G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, T. Muntean (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 257 pages. 2005.

Vol. 3361: S. Bengio, H. Bourlard (Eds.), Machine Learning for Multimodal Interaction. XII, 362 pages. 2005.

Vol. 3360: S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D. McLeod, M.E. Orlowska, L. Strous (Eds.), Journal on Data Semantics II. XI, 223 pages. 2005.

Vol. 3359: G. Grieser, Y. Tanaka (Eds.), Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets. XIV, 257 pages. 2005. (Subseries LNAI).

Vol. 3358: J. Cao, L.T. Yang, M. Guo, F. Lau (Eds.), Parallel and Distributed Processing and Applications. XXIV, 1058 pages. 2004.

Vol. 3357: H. Handschuh, M.A. Hasan (Eds.), Selected Areas in Cryptography. XI, 354 pages. 2004.

Vol. 3356: G. Das, V.P. Gulati (Eds.), Intelligent Information Technology. XII, 428 pages. 2004.

Vol. 3355: R. Murray-Smith, R. Shorten (Eds.), Switching and Learning in Feedback Systems. X, 343 pages. 2005.

Vol. 3353: J. Hromkovič, M. Nagl, B. Westfechtel (Eds.), Graph-Theoretic Concepts in Computer Science. XI, 404 pages. 2004.

Vol. 3352: C. Blundo, S. Cimato (Eds.), Security in Communication Networks. XI, 381 pages. 2005.

Vol. 3351: G. Persiano, R. Solis-Oba (Eds.), Approximation and Online Algorithms. VIII, 295 pages. 2005.

Vol. 3350: M. Hermenegildo, D. Cabeza (Eds.), Practical Aspects of Declarative Languages. VIII, 269 pages. 2005.

Vol. 3349: B.M. Chapman (Ed.), Shared Memory Parallel Programming with Open MP. X, 149 pages. 2005.

Vol. 3348: A. Canteaut, K. Viswanathan (Eds.), Progress in Cryptology - INDOCRYPT 2004. XIV, 431 pages. 2004.

Vol. 3347: R.K. Ghosh, H. Mohanty (Eds.), Distributed Computing and Internet Technology. XX, 472 pages. 2004.

Vol. 3346: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 249 pages. 2005. (Subseries LNAI).

Vol. 3345: Y. Cai (Ed.), Ambient Intelligence for Scientific Discovery. XII, 311 pages. 2005. (Subseries LNAI).

Vol. 3344: J. Malenfant, B.M. Østvold (Eds.), Object-Oriented Technology. ECOOP 2004 Workshop Reader. VIII, 215 pages. 2005.

Vol. 3343: C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, T. Barkowsky (Eds.), Spatial Cognition IV. Reasoning, Action, and Interaction. XIII, 519 pages. 2005. (Subseries LNAI).

Vol. 3342: E. Şahin, W.M. Spears (Eds.), Swarm Robotics. IX, 175 pages. 2005.

Vol. 3341: R. Fleischer, G. Trippen (Eds.), Algorithms and Computation. XVII, 935 pages. 2004.

Vol. 3340: C.S. Calude, E. Calude, M.J. Dinneen (Eds.), Developments in Language Theory. XI, 431 pages. 2004.

Vol. 3339: G.I. Webb, X. Yu (Eds.), AI 2004: Advances in Artificial Intelligence. XXII, 1272 pages. 2004. (Subseries LNAI).

Vol. 3338: S.Z. Li, J. Lai, T. Tan, G. Feng, Y. Wang (Eds.), Advances in Biometric Person Authentication. XVIII, 699 pages. 2004.

Vol. 3337: J.M. Barreiro, F. Martin-Sanchez, V. Maojo, F. Sanz (Eds.), Biological and Medical Data Analysis. XI, 508 pages. 2004.

Vol. 3336: D. Karagiannis, U. Reimer (Eds.), Practical Aspects of Knowledge Management. X, 523 pages. 2004. (Subseries LNAI).

Vol. 3335: M. Malek, M. Reitenspieß, J. Kaiser (Eds.), Service Availability. X, 213 pages. 2005.

Vol. 3334: Z. Chen, H. Chen, Q. Miao, Y. Fu, E. Fox, E.-p. Lim (Eds.), Digital Libraries: International Collaboration and Cross-Fertilization. XX, 690 pages. 2004.

Vol. 3333: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part III. XXXV, 785 pages. 2004.

Vol. 3332: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part II. XXXVI, 1051 pages. 2004.

Vol. 3331: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part I. XXXVI, 667 pages. 2004.

Vol. 3330: J. Akiyama, E.T. Baskoro, M. Kano (Eds.), Combinatorial Geometry and Graph Theory. VIII, 227 pages. 2005.

Vol. 3329: P.J. Lee (Ed.), Advances in Cryptology - ASIACRYPT 2004. XVI, 546 pages. 2004.

Vol. 3328: K. Lodaya, M. Mahajan (Eds.), FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science. XVI, 532 pages. 2004.

Vol. 3327: Y. Shi, W. Xu, Z. Chen (Eds.), Data Mining and Knowledge Management. XIII, 263 pages. 2005. (Subseries LNAI).

Vol. 3326: A. Sen, N. Das, S.K. Das, B.P. Sinha (Eds.), Distributed Computing - IWDC 2004. XIX, 546 pages. 2004.

Vol. 3325: C.H. Lim, M. Yung (Eds.), Information Security Applications. XI, 472 pages. 2005.

Vol. 3323: G. Antoniou, H. Boley (Eds.), Rules and Rule Markup Languages for the Semantic Web. X, 215 pages. 2004.

Vol. 3322: R. Klette, J. Žunić (Eds.), Combinatorial Image Analysis. XII, 760 pages. 2004.

Vol. 3321: M.J. Maher (Ed.), Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making. XII, 510 pages. 2004.

Vol. 3320: K.-M. Liew, H. Shen, S. See, W. Cai (Eds.), Parallel and Distributed Computing: Applications and Technologies. XXIV, 891 pages. 2004.

Vol. 3319: D. Amyot, A.W. Williams (Eds.), System Analysis and Modeling. XII, 301 pages. 2005.

Vol. 3318: E. Eskin, C. Workman (Eds.), Regulatory Genomics. VII, 115 pages. 2005. (Subseries LNBI).

Table of Contents

Distributed Programming

Grammars and Natural Language

Constraint Research

Constraint Applications

The Development of Oz and Mozart

Gert Smolka

Saarland University
Saarbrücken, Germany
smolka@ps.uni-sb.de

In this talk I will review the development of the programming language Oz and the programming system Mozart. I will discuss where in hindsight I see the strong and the weak points of the language. Moreover, I will compare Oz with Alice, a typed functional language we developed after Oz.

The development of Oz started in 1991 at DFKI under my lead. The initial goal was to advance ideas from constraint and concurrent logic programming and also from knowledge representation and to develop a practically useful programming system. After a number of radical and unforeseen redesigns we arrived in 1995 at the final base language and a stable implementation (DFKI Oz). In 1996 we founded the Mozart Consortium with SICS and Louvain-la-Neuve. Oz was extended with support for persistence, distribution and modules and Mozart 1.0 was released in January 1999.

P. Van Roy (Ed.): MOZ 2004, LNCS 3389, p. 1, 2005.

The Structure of Authority: Why Security Is Not a Separable Concern

Mark S. Miller[1,2], Bill Tulloh[3,**], and Jonathan S. Shapiro[2]

[1] Hewlett Packard Labs
[2] Johns Hopkins University
[3] George Mason University

Abstract. Common programming practice grants excess authority for the sake of functionality; programming principles require least authority for the sake of security. If we practice our principles, we could have both security and functionality. Treating security as a separate concern has not succeeded in bridging the gap between principle and practice, because it operates without knowledge of what constitutes least authority. Only when requests are made – whether by humans acting through a user interface, or by one object invoking another – can we determine how much authority is adequate. Without this knowledge, we must provide programs with enough authority to do anything they *might* be requested to do.

We examine the practice of least authority at four major layers of abstraction – from humans in an organization down to individual objects within a programming language. We explain the special role of object-capability languages – such as *E* or the proposed Oz-E – in supporting practical least authority.

1 Excess Authority: The Gateway to Abuse

Software systems today are highly vulnerable to attack. This widespread vulnerability can be traced in large part to the excess authority we routinely grant programs. Virtually every program a user launches is granted the user's full authority, even a simple game program like Solitaire. All widely-deployed operating systems today – including Windows, UNIX variants, Macintosh, and PalmOS – work on this principle. While users need broad authority to accomplish their various goals, this authority greatly exceeds what any particular program needs to accomplish its task.

When you run Solitaire, it only needs the authority to draw in its window, to receive the UI events you direct at it, and to write into a file you specify in order to save your score. If you had granted it only this limited authority, a corrupted Solitaire might be annoying, but not a threat. It may prevent you from

** Bill Tulloh would like to thank the Critical Infrastructure Protection Project at George Mason University for its financial support of this research.

P. Van Roy (Ed.): MOZ 2004, LNCS 3389, pp. 2–20, 2005.