A Survey of Oz

Gert Smolka
Programming Systems Lab
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
email: smolka@dfki.uni-sb.de

January 1995

Oz is a high-level programming language designed for concurrent symbolic computation. It is based on a new computation model providing a uniform and simple foundation for several programming paradigms, including higher-order functional, constraint logic, and concurrent object-oriented programming. Oz is designed as a successor to languages such as Lisp, Prolog and Smalltalk, which fail to support applications that require concurrency, reactivity, and real-time control.

DFKI Oz is an interactive implementation of Oz featuring a programming interface based on GNU Emacs, a concurrent browser, an object-oriented interface to Tcl/Tk, powerful interoperability features (sockets, C, C++), an incremental compiler, a garbage collector, and support for stand-alone applications. Performance is competitive with commercial Prolog and Lisp systems.

Oz Highlights

Logic variables and concurrent control. Oz is based on logic variables and fair concurrent control. Variables can be used before they are assigned values, and multiple computations are advanced fairly. If a computation requires the value of a not yet assigned variable, the computation suspends and automatically resumes when the value of the variable becomes available.

Oz is a higher-order language. Functions, procedures, objects, classes, methods, messages, and modules are created dynamically and are designated by first-class values, which may be passed as arguments, returned as results, tested for equality, and stored in the attributes of objects.

Full compositionality. Every computation in Oz is described by an expression. Expressions are closed under concurrent composition, local declaration, and procedural abstraction. Oz is lexically scoped. Since full compositionality in Oz is not hampered by a static type discipline, parameterized modules come for free.

Concurrent objects and multiple inheritance. Objects are the primary concurrent structuring concept of Oz. They combine data encapsulation through procedural abstraction with state and mutual exclusion. Objects can be seen as service-providing agents. The services of an object are provided through methods and can be requested by sending messages to the object. Objects are created as instances of classes. Classes define methods, attributes and features. The definition of a class may involve inheritance from other classes. Objects and classes are first-class citizens. They are created dynamically. Objects can be spawned as concurrent agents. Creating communicating agents in Oz is as simple as writing functions in Lisp.

Values and Data structures. Values in Oz include arbitrary-precision integers, floats, strings, tuples, lists, and records. Arrays and dictionaries are provided as objects. Composite data structures combine with logic variables and higher-order values. Cyclic data structures and infinite streams (e.g., the lazy list of all primes) are easily expressible.

First-class names. In Oz, the names of record fields, methods and attributes are first-class values that can be specified through variables. Values in Oz include a class of primitive names that are created dynamically and cannot be faked. In conjunction with lexical scoping, first class names provide for safety features so far available only in typed languages: one can express abstract data types as well as private methods and attributes of objects.

Soft real-time control. Oz is well-suited for reactive programming with soft real-time requirements. Oz employs an asynchronous, preemptive and priority-controlled task scheduling policy. Timers and access to real time are available. Control with time-outs is easily expressible.

Computing with partial information. Oz can compute with variables whose values are only partially specified. Information about the values of variables is specified by means of constraints, which are automatically combined by a simplification method generalizing unification. A computation can be synchronized on the event that a guard constraint is entailed by the constraint store.

Search. Oz comes with powerful predefined search abstractions, including depth-first one solution, demand-driven multiple solution, all solutions, and best solution (branch and bound) search. Search is performed as speculative computation with constraints taking place in local computation spaces. Branching is postponed until a stability condition is met. There is no backtracking. All search abstractions are obtained from a single higher-order search combinator.

Finite domain constraints. Oz comes with powerful constraints for variables constrained to finite sets of nonnegative integers (so-called finite domain variables), including addition, multiplication and comparisons. Finite domain constraints are essential for combinatorial search problems such as scheduling, configuration, and placement.

Deep guards. Oz is a deep guard language. It has two concurrent guarded choice combinators, committed choice and disjunction.

New computation model. Oz has been designed hand-in-hand with a new concurrent computation model incorporating ideas from functional programming, logic programming, and concurrent computation (the π-calculus, in particular). A guiding principle was the requirement that Oz be definable by reduction to a lean kernel language. The quest for semantic minimality led to the discovery of new primitives for expressing familiar computational abstractions such as functions, objects, and search. Objects, for instance, can be reduced to procedural abstraction and cells; cells are a new primitive providing for concurrent state.

Logic sublanguage. Oz has a logic sublanguage. Horn clause programs will execute in a logically sound manner. Since Oz has no backtracking, their operational behavior will be quite different from Prolog, however. Don't know choices (i.e., disjunctions) are suspended until they become determinate (similar to the so-called Andorra Principle). A Horn clause program can be executed with search by submitting it to a suitable search abstraction (i.e., one can choose the search strategy). Due to constraint propagation, Oz often explores much smaller search spaces than Prolog. Oz can express logically sound negation.

DFKI Oz

DFKI Oz is an interactive implementation of Oz featuring a programming interface based on GNU Emacs, a concurrent browser, an object-oriented interface to Tcl/Tk, powerful interoperability features (sockets, C, C++), an incremental compiler, a garbage collector, and support for stand-alone applications. Performance is competitive with commercial Prolog and Lisp systems.

Interactive programming environment. At any time, the user can enter expressions to be executed concurrently with already existing activities. Results can be reported in the browser or in user-defined windows. The browser is a concurrent display tool that shows the values of variables as soon as they become available. Expressions are entered through an interface realized with the Gnu Emacs Editor. One can start computations with partially described inputs and advance them stepwise by adding further information.

Object-oriented interface to Tcl/Tk. DFKI Oz comes with an object-oriented interface to Tcl/Tk, a powerful tool for programming graphical user interfaces. The functionality of all graphical objects comes in form of classes, which are subject to possibly multiple inheritance. Windows are described compositionally by means of object hierarchies and are subject to dynamic and interactive modification. The interface substitutes Oz for the low-level Tcl (an event-driven script language) exploiting concurrency, objects and first-class procedures. Nevertheless, one can freely integrate Tcl/Tk scripts if desired.

Interoperability. DFKI Oz comes with powerful interoperability features. High-level socket functionality is provided through predefined classes, making it straightforward to connect applications written in other languages, possibly running on different computers. Starting and controlling external processes is supported. Files are available through predefined classes. Unix functionality is available through a special interface. If necessary, the functionality of DFKI Oz can be extended by dynamically linking functions written in C and C++.

Demo. If DFKI Oz is installed on your system, you can start an interactive Oz demo by typing `ozdemo` to a shell. By graphical interaction you can play with many interesting demos illustrating different aspects of Oz. The source code of the demos is available through the DFKI Oz programming interface.

Availability. DFKI Oz is available for many platforms running Unix/X, including Sparcs and 486 PCs. DFKI Oz can be obtained free by anonymous ftp from `ps-ftp.dfki.uni-sb.de`, or through the WWW from

`http://ps-www.dfki.uni-sb.de/`.

History and Future

The research leading to the design and implementation of Oz began in 1990, when Gert Smolka founded the Programming Systems Lab at DFKI. The first implementation of Oz was operational two years later in September 1992. From there Oz evolved rapidly, until a fixed point was reached in January 1995. The language design is now frozen, with the exception of finite domain constraints. The DFKI Oz system is being improved upon steadily.

Current research issues at DFKI concerning Oz include more efficient implementation, type diagnosis (avoiding a static type discipline), concurrent exception handling, parallel implementation, concepts for distributed programming, and theoretical foundations.

Further Reading

The DFKI Oz Documentation Series currently consists of 10 documents offering more than 600 pages of tutorial and defining material. The document

- An Oz Primer [15]

is a tutorial introduction to Oz. The document

- Constraint Programming in Oz [8]

is a mainly tutorial presentation of constraint programming in Oz. The document

- Programming in Oz [2]

covers advanced programming techniques, excluding constraint programming. The documents

- DFKI Oz User's Manual [6]
- Window Programming in DFKI Oz [5]
- Open Programming in DFKI Oz [9]

concern issues related to DFKI Oz. The document

- DFKI Oz Demos [7]

describes a collection of selected demos coming with DFKI Oz. The official language definition consists of three documents

- The Definition of Kernel Oz [13]
- The Oz Notation [1]
- The Oz Standard Modules [3]

defining the semantics, the syntax, and the predefined functionality of Oz, respectively.

There are many research papers on issues related to Oz. The articles [14, 12] concern the computation model underlying Oz. The main ideas behind objects in Oz are discussed in [4]. Oz's higher-order search combinator is introduced in [11, 10]. The article [16] concerns logic records as used in Oz.

The DFKI Oz Documentation Series and many related research papers are available by anonymous ftp from `ps-ftp.dfki.uni-sb.de`, or through the WWW from `http://ps-www.dfki.uni-sb.de/`.

Acknowledgements

The following persons have contributed directly to the design and/or implementation of Oz: Martin Henz, Michael Mehl, Martin Müller, Tobias Müller, Joachim Niehren, Konstantin Popov, Ralf Scheidhauer, Christian Schulte, Gert Smolka, Ralf Treinen, and Jörg Würtz. Peter van Roy contributes since October 1994.

The development of Oz is supported by the Bundesminister für Forschung und Technology (Hydra, ITW 9105), the Esprit Basic Research Project ACCLAIM (contract EP 7195) and the Esprit Working Group CCL (contract EP 6028).

References

[1] Martin Henz. The Oz notation. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[2] Martin Henz and Martin Müller. Programming in Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[3] Martin Henz, Martin Müller, Christian Schulte, and Jörg Würtz. The Oz standard modules. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[4] Martin Henz, Gert Smolka, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, chapter 2, pages 27–48. The MIT Press, Cambridge, MA, 1995. To appear.

[5] Michael Mehl. Window programming in DFKI Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[6] Michael Mehl, Tobias Müller, Konstantin Popow, and Ralf Scheidhauer. DFKI Oz user's manual. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[7] Martin Müller, Tobias Müller, Christian Schulte, Ralf Treinen, and Jörg Würtz. DFKI Oz demos. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[8] Tobias Müller, Konstantin Popow, Christian Schulte, and Jörg Würtz. Constraint programming in Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[9] Christian Schulte. Open programming in DFKI Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[10] Christian Schulte and Gert Smolka. Encapsulated search in higher-order concurrent constraint programming. In Maurice Bruynooghe, editor, *Logic Programming: Proceedings of the 1994 International Symposium*, pages 505–520, Ithaca, New York, USA, 13–17 November 1994. MIT-Press.

[11] Christian Schulte, Gert Smolka, and Jörg Würtz. Encapsulated search and constraint programming in Oz. In A.H. Borning, editor, *Second Workshop on Principles and Practice of Constraint Programming*, Lecture Notes in

Computer Science, vol. 874, pages 134–150, Orcas Island, Washington, USA, 2-4 May 1994. Springer-Verlag.

[12] Gert Smolka. A calculus for higher-order concurrent constraint programming with deep guards. Research Report RR-94-03, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, February 1994.

[13] Gert Smolka. The definition of Kernel Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[14] Gert Smolka. A foundation for higher-order concurrent constraint programming. In Jean-Pierre Jouannaud, editor, *1st International Conference on Constraints in Computational Logics*, Lecture Notes in Computer Science, vol. 845, pages 50–72, München, Germany, 7–9 September 1994. Springer-Verlag.

[15] Gert Smolka. An Oz primer. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[16] Gert Smolka and Ralf Treinen. Records for logic programming. *Journal of Logic Programming*, 18(3):229–258, April 1994.

Remark. Papers of authors from the Programming Systems Lab of DFKI are available through anonymous ftp from `ps-ftp.dfki.uni-sb.de` or through WWW from `http://ps-www.dfki.uni-sb.de/`.