AI Magazine Volume 9 Number 1 (1988) (© AAAI)

WORKSHOP REPORT

# Concurrent Logic Programming, Metaprogramming, and Open Systems

Kenneth M. Kahn

*An informal workshop on concurrent logic programming, metaprogramming, and open systems was held at Xerox Palo Alto Research Center (PARC) on 8-9 September 1987 with support from the American Association for Artificial Intelligence.*
*The 50 workshop participants came from the Japanese Fifth Generation Project (ICOT), the Weizmann Institute of Science in Israel, Imperial College in London, the Swedish Institute of Computer Science, Stanford University, the Massachusetts Institute of Technology (MIT), Carnegie-Mellon University (CMU), Cal Tech, Science University of Tokyo, Melbourne University, Calgary University, University of Wisconsin, Case Western Reserve, University of Oregon, Korea Advanced Institute of Science and Technology (KAIST), Quintus, Symbolics, IBM, and Xerox PARC. No proceedings were generated; instead, participants distributed copies of drafts, slides, and recent papers.*

The workshop began with the visions of concurrent logic programming and open systems that are driving research at Weizmann, ICOT, CMU, MIT, and Xerox PARC. A shared vision emerged from the morning session with concurrent logic programming fulfilling the same role that C and Assembler do now. Languages such as Flat Concurrent Prolog and Guarded Horn Clauses are seen as general-purpose, parallel machine languages and interface languages between hardware and software and not, as a newcomer to this field might expect, as high-level, AI, problem-solving languages. This view is one of the major departures from mainstream logic programming and Prolog that most of the workshop participants have taken, which is reflected in the way these languages have traded Prolog's exhaustive search capability for controllable concurrency.

Another major departure from conventional (logic) programming is the incorporation of an open-system vision. In an open system, a growing predominance of computational services is provided and used by independent entities. Computation is envisioned as becoming more and more distributed between entities from different organizations, with different goals, that cooperate without requiring trust. Open, distributed implementations of concurrent logic-programming languages are seen by researchers from Weizmann and Xerox as a way of providing a uniform notation for computation in open systems. Dan Bobrow of Xerox presented the open-system challenge from the viewpoint of a participating agent; for example, how can an agent know what others might do for it and what assumptions will limit interactions, and how can the agent deal with "breakdown" in the sense discussed by Terry Winograd and Fernando Flores. Carl Hewitt of MIT presented actors as an alternative computational model for open-system programming. Many were surprised at the close correspondence between the actor model and concurrent logic programming. This similarity has become increasingly apparent to researchers in both fields. Both present a scalable, computational model with fine-grained, implicit concurrency and explicit synchronization control. Both are well suited for defining robust, securely encapsulated servers.

I presented the vision of the Vulcan group at Xerox PARC, which takes as its challenge the building of high-level languages, abstractions, and tools upon a distributed implementation of a concurrent logic-programming language. The Vulcan group sees metaprogramming as an important methodology in building concurrent logic-programming abstractions. The group shares with Weizmann, ICOT, and others the belief that enhanced metainterpreters are a conceptually clear and simple way to capture many programming abstractions. It is hoped that the use of suites of enhanced interpreters can be an effective and practical technique when combined with the automatic specialization that partial evaluation provides.

Vijay Saraswat from CMU provided the personal vision that tightly integrates notions of controlled constraint programming, transactions, production systems, and synchronous computation with the concurrent logic-programming framework. The connection between transactions and production systems followed from his proposal to permit the simultaneous reduction of multiple processes with a multiheaded definite clause. In a later talk by John Connery from the University of Oregon, a similar idea came up as a proposed extension to Prolog.

Mark S. Miller and K. Eric Drexler presented the Vulcan group's market-based approach to adaptive resource management in open systems. Takashi Chikayama presented ICOT's latest developments in concurrent logic programming.

The rest of the workshop consisted of technically oriented presentations and discussions. Several discussions were on the differences between Guarded Horn Clauses, Parlog, Saraswat's family of CP languages, and Flat Concurrent Prolog. The focus of these discussions was on whether unification should be an atomic transaction and whether data flow synchronization annotations should be interpreted statically (as a code annotation) or dynamically (as a data annotation). Differences between these two points lead to languages with different properties, for example, the ability or inability to describe mutual exclusion. Only Flat Concurrent Prolog supports dynamic annotations (read-only variables), and it was widely acknowledged that this capability confuses beginning programmers. Eric Tribble from Xerox sparked much discussion by presenting his analysis of the uses of read-only variables and proposing alternative simpler mechanisms.

Presentations about higher-level languages included one by Jim Rauen of MIT and Xerox about Lexical Flat Concurrent Prolog. The talk described a higher-order concurrent logic-programming language that enables programming with predicate arguments and closures and provides a principled means of defining and using modules. This work was directly inspired by the T dialect of Scheme.

One session was devoted to discussions of object-oriented programming languages built on concurrent logic programming. Andrew Davison of Imperial College presented Polka, Takashi Chikayama of ICOT presented KL1-U, and I talked about Vulcan. All these languages consider objects as recurrent processes consuming streams of messages. They differ, however, in their handling of inheritance, delegation, messages to self, and synchronization of state change.

We had a session on metainterpreters and partial evaluation. Leon Sterling of Case Western presented his ideas on the Flavors-like mixing of enhanced metainterpreters. Akira Okumura of ICOT presented rules for unfolding Guarded Horn Clauses programs. Udi Shapiro from Weizmann talked about metainterpreter-based algorithmic debugging and the ways in which specialized views of the computation can be useful for different kinds of debugging. He also talked about a new semantics for Flat Concurrent Prolog computations. Will Winsborough from the University of Wisconsin talked about abstract interpretation in a logic-programming framework.

Some of the talks were on implementation issues. Yasunori Kimura of ICOT presented a multiple reference count scheme that supported Guarded Horn Clauses optimizations, which are essential in keeping bus traffic down on a shared-memory, snoopy cache machine. Annika Waern and Fredrik Holmgren of the Swedish Institute of Computer Science talked about how to compile Guarded Horn Clauses programs to Prolog implementations extended with "freeze"—the capability to suspend goals. Vijay Saraswat from CMU talked about a similar technique. He also discussed constraint programming in a concurrent logic-programming framework. John Cleary from the University of Calgary presented the distributed simulation technique called time warps and its applicability to parallel implementations of Prolog and distributed implementations of unification. Seif Haridi of the Swedish Institute of Computer Science talked about work on parallel implementations of Prolog.

One session consisted of live demos that were projected onto a large screen. Dongwook Shin from KAIST demonstrated Aflog, a functional extension to concurrent logic programming. Bob Cassels and John Hotchkiss of Symbolics demonstrated their graphic debugger for logic programs. Bill Silverman from the Weizmann Institute showed the prototype of a distributed implementation of the Logix operating system. Logix is a programming system for Flat Concurrent Prolog written in Flat Concurrent Prolog. The distributed version supports remote computations and distributed code management.

Overall, the workshop exceeded our expectations. Many presentations were given of new results and work in progress. The level, extent, and liveliness of the discussions during and following the presentations obviated the need for the several panels that had been planned. We think the workshop successfully tied together the threads of concurrent logic programming, metaprogramming, and open systems. Many participants came to the workshop with some version of one or two of these threads and went home with a heightened awareness of all three.

## References

Ehud Shapiro, ed., *Concurrent Prolog: Collected Papers,* MIT Press, Cambridge, Mass. 1987. A collection of papers on Concurrent Prolog and related languages, this book includes classic papers that started the field of concurrent logic programming as well as recent work on embedded languages, implementations, tools, metaprogramming, partial evaluation, applications, and semantics.

Kazunori Ueda, *Guarded Horn Clauses,* MIT Press, Cambridge, Mass. 1987. This book describes the language that has become the kernel language for the latest research at ICOT.

Steve Gregory, *Parallel Logic Programming in PARLOG,* Addison-Wesley, Wokingham, England 1987. This text serves as a general introduction to Parlog, the concurrent logic-programming language from Imperial College.

Bernardo Huberman, ed., *The Ecology of Computation,* North Holland, Amsterdam 1988. This book is a good collection of papers on open systems.

Kenneth Kahn, "Partial Evaluation, Programming Methodology, and Artificial Intelligence", *AI Magazine,* Vol. 5, No. 1, pages 53-57. This article is a general introduction to partial evaluation and its application to the specialization of interpreters.