Interleaving of Ideas from ICOT, Xerox PARC, and Weizmann Institute: Progress in Concurrent Logic Programming

Ken Kahn (kahn@parc.xerox.com)

June 12, 1992

This is a draft of a paper to be submitted to the special issue of the Communications of the ACM on the Fifth Generation project.

Abstract

This article presents an informal intellectual history of some ideas about logic programming, concurrency, programming language design, open distributed systems, systems programming, and programming language abstractions. Over the last six years there has been considerable interaction between researchers at ICOT, Weizmann Institute and Xerox PARC. The viewpoint presented here is from Xerox PARC where the author lead the Vulcan project whose goal was to invent higher-level abstractions upon the foundation of concurrent logic programming and where work on distributed and concurrent constraint programming continues today.

1 The Early Years

When ICOT was formed in 1982, I was on the faculty of the University of Uppsala, Sweden doing research at UPMAIL (Uppsala Programming Methodology and Artificial Intelligence Laboratory). There was great excitement in the laboratory because of the shared belief that logic programming could offer much to AI and, in general, symbolic computing. Koichi Furukawa (at that time an ICOT lab manager, now deputy director of ICOT) and some of his colleagues visited UPMAIL that year to present the plan for the Fifth Generation Project and to explore possible collaborations.

About a year later I was invited to be a guest researcher at ICOT for a month. My research at that time was on LM-Prolog, an extended Prolog well-integrated with Lisp, implemented on MIT-style Lisp Machines (LMI and Symbolics).[CK83] One of the driving motivations behind this work was that there were lots of good things in Prolog, but Prolog could be much better if many of the ideas from the Lisp and object-oriented programming communities could be imported into the framework. I was also working on a partial evaluator for Lisp written in LM-Prolog.[Kah84] This program was capable of automatically specializing Lisp programs. One goal of this effort was to generate specializations of the LM-Prolog interpreter each of which could only interpret a single LM-Prolog program. The performance of these specialized interpreters of programs was comparible with the compiled versions of those programs.

Researchers at ICOT were working on similar things. There was good work going on in partial evaluation of Prolog programs.[?] There was work on ESP a Prolog extended with objects and macros.[?] Efforts on a system called Mandala had begun which combined ideas of meta-interpretation and object-oriented programming in logic programming framework.[FAK+84]

When I arrived at ICOT, I was eager to demo my research and was delayed by more than a week by the fact that ICOT had a later version of the Symbolics operating system and it was incompatible with the release LM-Prolog had been ported to. While my demonstrations and seminars about LM-Prolog and partial evaluation went well and my discussions with ICOT researchers were productive, the most important event during my visit was my introduction to Concurrent Prolog.

Ehud Shapiro from the Weizmann Institute of Science in Israel was visiting then, working closely with Akikazu Takeuchi of ICOT. Concurrent

Prolog was conceived as an extension of Prolog[1] to introduce programmer-controlled concurrency.[?] It was based upon the concept of a "read-only variable" which I had found very confusing when I had read about it before my ICOT visit. Part of the problem was simple nomenclature: a variable doesn't become read-only; what happens is that there are occurrences of a variable which only have a read capability instead of the usual situation where all occurrences have read-write privledges.

Shapiro and Takeuchi had written a paper about how Concurrent Prolog could be used as an actor or concurrent object language.[ST83] I was very interested in this since I had worked on various actor languages as a doctoral student at MIT. Again my difficulty in grasping read-only variables interfered with a good grasp of the central ideas in this paper. I understood it only after Shapiro carefully explained the ideas to me. After understanding the paper, I felt that some very powerful ideas about concurrent objects or actors were hidden under a very verbose and clumsy way to express them in Concurrent Prolog. The idea of "incomplete messages" in which the receiver (or receivers) fills in missing portions of messages was particularly attractive. Typically there are processes suspended waiting for those parts to be filled in. It seemed clear to me that this technique was a good alternative to the continuation passing of actors and Scheme.

At this time the Fifth Generation project was designing parallel hardware and its accompanying kernel language. A distributed memory machine seemed to make the most sense since it could scale well, while shared memory machines seemed uninteresting because they were limited to a small number of processing elements. Shapiro was working on a parallel machine architecture called the "Bagel".[?] I collaborated with him on a notation for mapping processes to processors based upon the ideas of the Logo turtle. A process had a heading and could spawn new processes forward (or backwards) along its heading and could change its heading.

At this time it seemed that single-language machines were a good idea. There was lots of excitement about Lisp machines which benefited from a tight integration of components and powerful features. During my visit to ICOT it seemed clear to most people that doing Prolog or Concurrent Prolog machine was the way to go. And unlike the Lisp Machines these new machines would be designed with parallelism in mind.[2]

[1]There never was an implementation of Concurrent Prolog that retained Prolog as a sublanguage. Eventually Concurrent Prolog was redefined as a different language which provided concurrency and sacrificed the ability of Prolog programs to do implicit search.

[2]With the advantage of hindsight, this was a mistake because it cut of FGCS research

As I recall, there was some debate at that time about whether the kernel language of parallel inference machines should be based upon a parallel ESP (Prolog extended with objects) or something like Concurrent Prolog. I argued for Concurrent Prolog because it did actors so well and it seemed clear to me that actors should be the base concept for programming parallel machines.

Soon afterwards, ICOT did decide to go with concurrent logic programming as the basis for their kernel language (KL1) but chose to base it upon Guarded Horn Clauses[Ued85] instead of Concurrent Prolog. It wasn't until about six years later that I came to agree that this was the wisest choice. At the time I believed that GHC was too weak for the task since it lacked what I believed were some very important features of Concurrent Prolog such as atomic unification and dynamic read/write capabilities. This mistaken view was held by all the members of the project I lead at Xerox PARC during the late 1980s.

A few months after my visit to ICOT I was visiting the Free University of Brussels. I gave a seminar about my research and then was asked to give another one about the Fifth Generation project. There was tremendous interest in this project. I recall explaining the "middle-out" strategy of FGCS. ICOT concentrated initially on the language and operating system side of things and then moved upwards towards applications and downward towards hardware. This seemed like a good strategy at the time and looking at it today, I think the most significant successes of the Fifth Generation project are in the middle.

At the Free University of Brussels I visited Luc Steels and his students. They were interested in actors and failed to get too excited about Concurrent Prolog. My response to this was to try to develop a higher-level language which compiled directly into Concurrent Prolog. I didn't get too far the short time I was there and dropped the idea when I returned to Uppsala since there was much to do on other projects. It wasn't until two years later that I again took up the task of designing a higher-level language that preserved the power of "actors" in Concurrent Prolog while avoiding the clumsy verbose means of expression.

---

from tools and platforms of other researchers. This approach was too closed and only now is ICOT doing serious work on porting their software to standard platforms.

2 The Pre-Vulcan Period at PARC

A few months later I joined Xerox PARC. Multi-paradigm programming was very "in" and they had developed Loops to support Lisp, objects, and rules. My first task at PARC was to replace the rule component. The design was based upon my experience with LM-Prolog and work on integrating logic programming and objects. I was also asked to consult on the joint project between Xerox and Quintus Computer Systems to build Xerox Quintus Prolog, a micro-code implementation of Prolog on the Xerox Lisp Machines based upon the Warren Abstract Machine (WAM). At this point, Symbolics was selling a very similar product.[3] Comparing these different systems and running various InterLisp-D benchmarks I became discouraged that the rule component of Loops implemented in Lisp could ever have anywhere near the performance of these other systems.

Long before I came to PARC, Larry Masinter at Xerox PARC had been trying to convince to the LOOPS group that the object component should be based upon the idea of generic functions and multi-methods. He quickly convinced me, partly because it fit much better with the Prolog style of defining predicates. The idea which later became incorporated in InterLoops, CommonLoops, and CLOS was that a function could be defined as a collection of methods each one of which could do type discrimination on any of its arguments to determine if the method was applicable. Conventional object-oriented programming is based on discriminating only on the first argument. This proposal both generalized object-oriented programming and fit better with Lisp's functional style than earlier Lisp/object combinations. The caller of a function should not need to know whether that function was implemented as ordinary Lisp code or as a collection of multi-methods. In addition, I was excited about how Prolog-like computation could fit in by having multi-methods that do pattern-matching or unification instead of (or in addition to) the type discrimination of object-oriented programming. Methods could be combined in the object-oriented manner so that the most specific method was chosen or could be combined in the Prolog manner with a backtrackable choice where upon failure another method could be tried.

While doing this work on what became CommonLoops (and CLOS) and on my extension which I called CommonLog, I became less and less satisfied with the complexity of the resulting language. And on top of all this

[3] I believe both of these implementations owe their existence to the interest in Prolog that the Fifth Generation project had generated.

complexity it did not seem (and still doesn't to me) that it could be extended to give good support for parallel and distributed computation. Also I began to question the very premise of multi-paradigm programming; was it so desirable to mix at a very fine-grain very different ways of thinking about computation? In one line of code it was possible to mix functional, object-oriented and logic programming styles.

3 The Vulcan Years

Because of my dissatisfaction with CommonLoops, I returned to the idea of building a higher-level language for Concurrent Prolog. Together with my PARC colleagues Mark Miller and Danny Bobrow, we quickly designed a language in early 1986 which we called Vulcan.[4] We wrote a paper about it[KTMB87] and suddenly there was lots of interest. The paper was reprinted in several books. Other PARC researchers (Eric Dean Tribble and Curtis Abbott) joined the group. Ehud Shapiro began to consult for us. Students (Jim Rauen and Andy Cameron) joined the project. A DARPA official encouraged us to write a grant proposal.[5] I think the strong interest can be explained by the fact that the project combined, in a coherent and simple manner, several fashionable items: object-oriented programming, parallel programming, distributed applications including groupware, and Fifth Generation computing.

During this period we actively followed research at ICOT and the Weizmann Institute.[6] We developed a vision of distributed open-systems computing which placed Flat Concurrent Prolog (FCP) as the foundation.[7] We

[4]The name came from the fact that this was an actor language based upon logic programming. Vulcan is a fictional planet whose inhabitants are very logical and are portrayed by actors.

[5]In 1987, we did submit a large project proposal just before there were a series of major management changes at DARPA. Each change delayed a decision by many months. After more than a year PARC lost patience and withdrew the proposal.

[6]Very related research was going on in the Parlog group at Imperial College, London. In fact the precursor to Concurrent Prolog, the Relational Language was developed there.[?]. We followed the work there, visited each other, and so on but we were not influenced as much by the work there. Their Parlog system was too large and complex for our tastes. We did interact with Andrew Davidson on his Pool and Polka languages which were object-oriented extensions of Parlog. Years later we were strongly influenced by the Strand language which was largely based upon the work on Flat Parlog [?] done at Imperial College.

[7]Flat Concurrent Prolog is a subset of Concurrent Prolog which permits a much more light-weight and simple implementation.

saw FCP as a small yet powerful kernel upon which to build many layers of abstraction. We foresaw tools and programming methodologies which relied upon the dual readings of logic programs: declarative and process-oriented. Clients and servers could be built in FCP in a portable fashion as well as connected together by FCP. The boundary between client and server computation could be very flexible and could be specified at run time. We saw unification as a single, conceptually simple, computational mechanism that provided the functionality of assignment, binding, argument passing, return of values, inter-process communication, atomic transactions and the construction, testing, and access of records. While our colleagues at ICOT and Weizmann were exploring how these languages support the exploitation of parallel hardware, we saw the same language supporting distributed computing. We saw how these languages support secure encapsulated state necessary for distributed applications. We were excited about ICOT and Weizmann Institute research which demonstrated how easy it was to implement various abstractions as short and simple meta-interpreters. We saw how these languages are well-suited for partial evaluation which could both make code reuse and meta-interpreters more practical.

The Vulcan project continued to grow. In 1988 we hired Jacob Levy whose thesis work [?] at the Weizmann Institute was on the implementation of a parallel functional programming language on top of FCP. We hired Vijay Saraswat whose CMU thesis [Sar89] was on concurrent constraint programming: an elegant synthesis of concurrent logic programming and constraint logic programming. Saraswat had proposed a language he called Herbrand that we favored since it was just slightly weaker than FCP but much cleaner, simpler, and probably more efficient to implement.

The Vulcan project had become rather large and PARC management began to question why Xerox should be funding this since it was a project whose results could clearly benefit the world at large but it was not very clear how Xerox would benefit in particular. Management lost patience waiting for a reply from DARPA about our proposal and the project was stopped. Three of us (Saraswat, Levy, and myself) continued doing related research.

4 The Post-Vulcan Years

Saraswat and I began work on a visual syntax for concurrent logic programs. The syntax was based upon the topology of drawings. It was designed so that it was well-suited, not just for program sources, but also as the basis

for generating animations of program executions.[KS90] One discovery was that object-oriented programs did not come out so clumsy and verbose when drawn instead of typed.

I collaborated with Shapiro on a pre-processor for logic programs designed to support object-oriented programming.[KS88] My thinking had changed from believing that concurrent logic programs were too low-level, to believing they just needed some simple syntactic support. I came to realize that the Vulcan language, by trying to be an actor language, had sacrificed some very important expressive power of the underlying language. In 1989, I presented an invited paper at the European Conference on Object-Oriented Programming on this topic.[Kah89] The essence of the paper is that concurrent logic programming, by virtue of its first-class communication channels, is an important generalization of actors or concurrent objects. Multiple input channels are very important, as is the ability to communicate input channels. During this period I interacted with Kauro Yoshida of ICOT during her development of A'UM, an object-oriented system on top of FGHC which retains the power of multiple, explicit, input channels.[YC88]

We hosted an extended visit by Kazunori Ueda and M??? from ICOT. Ueda, the designer of Flat Guarded Horn Clauses (FGHC), slowly won us over to the view that his language, while weaker than Herbrand, was simpler and that there were programming techniques that compenstate for its inabilities. Essentially, we moved from the view of unification as an atomic operation to viewing it as an eventual publication of information. I began to program in the FGHC subset of the Weizmann Institute implementation of FCP. I would have seriously considered using an ICOT implementations had one been available for Unix workstations.[8]

AI Limited in the UK then announced a commercial concurrent logic programming language called Strand88. We became a beta test site and received visits by one of the language designers (Steve Taylor) and later by officials of the company. We were very eager to collaborate because the existence of a commercial product gave these languages a certain realness and respectability within Xerox.[9]

Our first reaction was that they had simplified the language too much: they had replaced unification by assignment and simple pattern matching. What we had once believed was the essence of concurrent logic programming

[8]ICOT today is working on porting their work to Unix workstations and has made its software freely available.

[9]Also Xerox had had a long history of business relations with AI Limited on other products.

was gone. As was the case with FGHC, we were won over to the language by being shown how its deficiencies were easily compensated for by certain programming techniques. I stopped using the FGHC subset of FCP and became a Strand programmer. I even offered a well-attended in-house tutorial on Strand at PARC.

Saraswat quickly became disenchanted with Strand because the way it provided assignment interfered with giving it a good declarative semantics. Strand assignment is single assignment, so it avoids the problems associated with assignment in a concurrent setting. But Strand signals error if an attempt is made to assign the same variable twice. Saraswat then discovered a two-year old paper by Masahiro Hirata from the University of Tskuba?? on a language called DOC.[Hir86] The critical idea in the paper was that if every variable had only a single writer then no inconsistencies or failures could occur.

Saraswat, Levy and I picked up on this idea and designed a concurrent constraint language called Janus.[SKL90] We introduced a syntax to distinguish between an "asker" and a "teller" of a variable. We designed syntactic restrictions (checkable at compile time) which guarantee that a variable cannot receive more than one value. We discovered that these restrictions also enable some very significant compiler optimizations, including compile-time garbage collection.

Because we lacked the resources to build a real implementation of Janus we started collaborative efforts with various external research groups (the University of Arizona, McGill University, Saskatchwan University). Jacob Levy left and started a group at the Hebrew University in Israel.[10]

5 Today

Today work continues on Janus implementations. ICOT research on moded FGHC attempts to achieve the goals of Janus by sophisticated program analysis rather than syntactic restrictions.[?] An interesting aspect of this work is how it manages when appropriate to borrow implementation techniques from object-oriented programming. Also, work on directed logic variables at Weizmann Institute was inspired by Janus.[?]

Saraswat has had a major impact on the research community with his framework of concurrent constraint programming. He is active in a large joint Esprit/NSF project called ACCLAIM based upon his work. At ICOT

[10] He now works at Sun Microsystems and is implementing a Janus in his spare time.

there is a system called GDCC which directly builds upon his work. His work has also had significant impact on the theoretical computer science community interested in concurrency. Saraswat and a student (Clifford Tse) are working on the design and parallel implementation a programming language called Linear Janus which is based upon concurrent constraint programming and addresses the same goals as Janus but is based upon linear logic.

The work of the Vulcan project on exploring the feasibility and advantages of using concurrent logic programming as the foundation for building distributed applications has influenced Shapiro's group at the Weizmann Institute. They have been focusing on distributed applications for the last two years.

I have concentrated my efforts on a building an environment for "Pictorial Janus" which is a visual syntax for Janus. The system accepts program drawings in PostScript, parses them and produces animations of concurrent executions. A postdoc (Markus Fromherz) is using Pictorial Janus to model the behavior of paper paths in copiers. I see my work as making the ideas behind concurrent logic programming more accessible. Programs and their behaviors are much easier to understand when presented in a manner that exploits our very capable perceptual system.

6 A Personal View of the Fifth Generation Project

So what is my view of the Fifth Generation project after ten years of interactions? Personally I am very glad that it happened. There were many fruitful direct interactions and I am sure several times as many indirect positive influences. Without the Fifth Generation project there might not have been a Vulcan project, or good collaborations with the Weizmann Institute, or the Strand and Janus languages. More globally, I think the whole computer science research community has benefited a good deal from the project. As Hideaki Kumano, Director General, Machinery and Information Industries Bureau, Ministry of International Trade and Industry (MITI) said during his keynote speech at the 1992 FGCS conference:

> Around the world, a number of projects received their initial impetus from our project: these include the Strategic Computing Initiative in the USA, the EC's Esprit project, and the Alvey Project in the United Kingdom. These projects were initially launched to compete with the Fifth Generation Computer Systems Project. Now, however, I strongly believe that since our

> ideal of international contributions has come to be understood around the globe, together with the realization that technology cannot and should not be divided by borders, each project is providing the stimulus for the others, and all are making major contributions to the advancement of information proccessing technologies.

I think the benefits to the Japanese computer science community were very large. Comparing visits I made to Japanese computer science laboratories in 1979, 1983, 1988, and 1992 there has been tremendous progress. When the project started there were few world-class researchers in Japan on programming language design and implementation, on AI, on parallel processing, etc.. Today the gap has shrunk completely; the quality and breadth of research I have seen in 1992 is equal to that of the US or Europe. I think the Fifth Generation project deserves much credit for this. By taking on very ambitious and exciting goals, they got much further than if they had taken on more realistic goals. I do not think that the Fifth Generation project is a failure because they failed to meet many of their ambitious goals; I think it is a great success because it helped move computer science research in Japan to a world-class status.

References

[CK83] Mats Carlsson and Ken Kahn. LM-Prolog user manual. Technical Report 24, UPMAIL, Uppsala University, November 1983.

[FAK+84] K. Furukawa, Takeuchi. A., S. Kunifuji, H. Yasukawa, M. Ohki, and K. Ueda. Mandala: A logic based knowledge programming system. In ICOT, editor, *Proc of the International Conference on Fifth Generation Computer Systems*, 1984.

[Hir86] Masahiro Hirata. Programming language doc and its self-description, or, x=x considered harmful. In *3d Conference Proceedings of Japan Society for Software Science and Technology*, pages 69–72, 1986.

[Kah84] Ken Kahn. The compilation of Prolog programs without the use of a Prolog compiler. In *Proceedings of the Fifth Generation Computer Systems Conference*, 1984.

[Kah89] Kenneth Kahn. Objects – a fresh look. In Stephen Cook, editor, *Proceedings of the Third European Conference on Object-Oriented Programming*, pages 207–224. Cambridge University Press, July 1989.

[KS88] K. Kahn and E. Shapiro. Logic programs with implicit state. Technical report, Weizmann Institute of Science, Rehovot, Israel, 1988.

[KS90] Kenneth M. Kahn and Vijay A. Saraswat. Complete visualizations of concurrent programs and their executions. In *Proceedings of the IEEE Visual Language Workshop*, October 1990.

[KTMB87] K. Kahn, E. Tribble, M. Miller, and D. Bobrow. Vulcan: Logical concurrent objects. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 75–112. The MIT Press, 1987. Also in Concurrent Prolog, MIT Press, ed. Ehud Shapiro.

[Sar89] Vijay A. Saraswat. *Concurrent Constraint Programming Languages.* PhD thesis, Carnegie-Mellon University, January 1989.

[SKL90] Vijay A. Saraswat, Kenneth Kahn, and Jacob Levy. Janus—A step towards distributed constraint programming. In *Proceedings of the North American Logic Programming Conference.* MIT Press, October 1990.

[ST83] Ehud Shapiro and A. Takeuchi. Object oriented programming in concurrent Prolog. *New Generation Computing*, 1:25–48, 1983.

[Ued85] K. Ueda. Guarded Horn Clauses. Technical Report TR-103, ICOT, June 1985.

[YC88] K. Yoshida and T. Chikayama. A'um - a stream-based concurrent object-oriented language. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 638–649, 1988.

Kenneth M. Kahn
System Sciences Laboratory, Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304
Work: (415) 812-4390; Home: (415) 851-0890
kahn@parc.xerox.com

Educational Background:

Massachusetts Institute of Technology, Sept. 1973 to Feb. 1979
Ph.D in Electrical Engineering and Computer Science, Jan. 1979
Thesis: *Creation of Computer Animation from Story Descriptions*
M.S. in Electrical Engineering, Aug. 1975
Thesis: *Mechanization of Temporal Knowledge*

University of Pennsylvania, Sept. 1969 to June 1973
University of Stockholm, Junior Year Abroad, Sept. 1971 to June 1972
B.A. in Economics *magna cum laude* with distinction, June 1973
Thesis: *Bankruptcy, Information Costs, and Equilibrium*

Professional Experience:

Member of Research Staff, Xerox PARC, August 1984 to present

Guest Researcher, ICOT (Institute for New Generation Computing), Tokyo, Nov. 1983

Associate Professor, Computing Science Department, Uppsala University, July 1981 to August 1984

Research Associate, UPMAIL, Computing Science Department, Uppsala University, July 1980 to August 1984

Visiting Professor and Researcher, University of Stockholm, Jan. 1980 to June 1980,

MIT Research Scientist and Lecturer, Jan. 1979 to Dec. 1979

Research Assistant, Research Staff, and IBM Graduate Student Fellowship Recipient, Artificial Intelligence Laboratory, Project MAC (Laboratory for Computer Science), MIT, Sept. 1973 to Jan. 1979

Programmer of a statistical and graphical study of census data, Philadelphia Social History Project, Sept. 1972 to July 1973,

Applications programmer, consultant and liaison to the humanities departments, University of Pennsylvania Computer Center, Jan. 1970 to July 1971

Research Experience:

As part of my master's thesis at MIT, I implemented two versions of a time specialist, a computer program capable of accepting a wide range of temporal statements, checking their consistency, and making inferences to answer questions [1, 2]. Following that I joined the LOGO group and taught elementary school children to do natural language programming [3] and animation [4] using several small systems I built for this purpose [5]. (In 1983, while consulting for Atari, I resumed research on natural language tools for children, this time based upon Prolog [6].) From 1976 to 1980 I was the designer and implementor of an actor-based computer language called "Director" [4, 7, 8, 9]. Director was used for programming computer animation and knowledge-based systems. Simultaneously, I was working on my doctoral thesis, building a system capable of making simple computer-animated films in response to vague, incomplete story descriptions [10, 11, 12, 13, 14]. During this time, I took several courses in animation and filmmaking and made several films which were shown at film festivals and local theaters [15, 17, 18]. One of them was sold to cable [16]. Recently, as a refresher, I took three animation courses at De Anza College.

After a year as a post-doc at MIT, I went to Sweden, initially to the University of Stockholm, and then to Uppsala University, where I began exploring multi-paradigm programming. After learning about Prolog, I used it to implement an actor language, "Intermission" [19]. Next came "Uniform", a language based on extended unification [20, 21]. The language was an attempt to combine the important features of Lisp, actor languages, and Prolog into a simple coherent framework. My research on combining the best of Lisp and Prolog led to my design and, with a colleague, implementation of LM-Prolog [22, 23, 24, 25, 26], an extended Prolog system on Lisp Machines that was sold by Lisp Machines, Incorporated.

While in Sweden I become excited about the potential for partial evaluation to win back performance sacrificed in the quest for simple generic programs. As a realistic example I worked on automatically generating a compiler from LM-Prolog to Lisp by doing partial evaluation of the LM-Prolog interpreter written in Lisp [27, 28, 29]. The partial evaluator was written in LM-Prolog and generated efficient specializations of Lisp programs.

From Sweden I went to Xerox PARC to continue my research on multi-paradigm systems with the "LOOPS" group. During 1985, I was one of the two designers and implementers of CommonLoops [30], the basis of the Common Lisp Object System (CLOS) standard. At this time I was also chair of the Common Lisp object-oriented programming subcommittee. (Despite having been instrumental in bringing CLOS to the world, I am not particularly proud of it.) During this period, I also collaborated on other projects with several researchers at PARC [31, 32, 33].

In 1986 I started and led the Vulcan project, whose purpose was building high level programming abstractions within a concurrent logic programming framework [34, 35, 36, 37, 38, 39, 40]. For nearly two years I managed a project consisting of six researchers. Much of the research focused upon programming language support for open systems programming [41]. When funding was cut, similar efforts were continued, though on a smaller scale [42, 43, 44, 45]. I focused upon connections between concurrent logic (and more generally constraint) programming and concurrent object-oriented frameworks or actors [46, 47, 48]. The theme underlying the Vulcan and subsequent research was "clean and small but real".

For the last three years, I have enjoyed combining my interest in animation with my interests in language design and concurrency by leading a project to visualize concurrent programs and animate their executions in a coherent and general manner [49, 50]. My efforts have resulted in a picture parser and an animator capable of interpreting PostScript drawings of concurrent constraint

programs and automatically producing animations of program executions [51]. Most recently, I have been exploring applications of this system to visualizing object-oriented designs [52] and non-programming uses (e.g. [53]).

My professional interests have remained surprisingly constant over the last 19 years. Programming language design and animation have been central. My inclination has been to explore the esoteric – actors and object-oriented programming in the 70s, multi-paradigm, description, and logic programming languages in the early 80s, and most recently, concurrent languages for distributed computing based on logic and constraints – with the aim of providing very powerful, yet easy to use, generic tools for programming. I am enjoying and am very excited about my recent work on combining animation, concurrency, and programming. In the future, I hope to pursue research on these and related topics.

Teaching Experience:

Lecturer for undergraduate and graduate courses in Artificial Intelligence, Description Languages, Lisp Machines, and Logic Programming, Uppsala University, Sweden, Sept. 1980 to August 1984

Lecturer for two courses in Artificial Intelligence, University of Stockholm, Sweden, Jan. to June 1980

Lecturer for "Structure and Interpretation of Computer Languages", Department of Electrical Engineering and Computer Science, MIT, Feb. 1979 to June 1979

Teaching Assistant for "Problem-Solving Paradigms", Department of Electrical Engineering and Computer Science, MIT, January 1976 to May 1976

Taught elementary school children to program graphics, animation, and natural language with the Logo Group of the MIT AI Lab, Feb. 1975 to Dec. 1975

Editorial and Academic Activities:

Member of program committee: Second International Logic Programming Conference, Uppsala, Sweden, 1984; Third International Conference on Logic Programming, London, 1986; North American Conference on Logic Programming, Cleveland, Ohio, 1989; Joint Conference on Object-Oriented Programming and European Conference on Object-Oriented Programming, Ottawa, Canada, 1990; North American Conference on Logic Programming, Austin, Texas, 1990; Symposium on Partial Evaluation and Semantics-Based Program Manipulation, New Haven, Connecticut, 1991; International Logic Programming Symposium, San Diego, California, 1991.

General chair: International Logic Programming Symposium, San Diego, California, 1991 (over 200 attendees).

Area editor: Knowledge Representation, Reasoning, and Expert Systems, *The Journal of Logic Programming*, 1986–1991.

Editorial board: *The Journal of Logic Programming*, *Lisp and Symbolic Computing*, *New Generation Computing*.

I have been on four doctoral thesis committees, and I have been the host/advisor for five undergraduate summer students and two postdoctoral fellows at Xerox PARC.