TEACHING FIFTH GENERATION COMPUTING:

THE IMPORTANCE OF SMALLTALK

Wilf R. LaLonde
Dave A. Thomas
John R. Pugh

SCS-TR-64

October 1984

School of Computer Science, Carleton University, Ottawa K1S 5B6, Canada.

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

Teaching Fifth Generation Computing: The Importance of Smalltalk

Wilf R. LaLonde, Dave A. Thomas and John R. Pugh
School of Computer Science
Carleton University
Ottawa, Ontario, Canada K1S 5B6

Abstract

The obvious approach to preparing students for fifth generation computing is to focus on the two primary languages used by the Artificial Intelligence community: Lisp and Prolog. By dwelling on the features lacking in traditional didactic languages, e.g. symbolic manipulation and rule-based programming, an awareness and appreciation for the increased power of fifth generation systems can be imparted.

However, this emphasis is only a partial solution because it ignores one of the more important programming languages of the future. We claim that Smalltalk is at least as important as Lisp and Prolog from the point of view of fifth generation computing. The object-oriented paradigm that it supports, its emphasis on modularization for "programming in the large", and its potential as an Artificial Intelligence programming language combine to make it an important candidate for consideration.

1 Introduction

Fifth generation systems are distinguished from traditional systems primarily by their emphasis on symbolic computing and the highly parallel hardware expected to support it. Since Lisp and Prolog are the two main programming languages supporting symbolic manipulation, they are the obvious languages to focus on when the aim is preparing students for fifth generation computing. Their primary features, advantages and disadvantages can be studied and illustrated through applications in artificial intelligence.

But these are not the only languages worthy of consideration. In particular, Smalltalk is an equally valid candidate since it is an instance of a new class of programming languages based on the notion of actors or active objects. Although the design of fifth generation languages is in its infancy, we believe that features of all three languages will play an important role in a viable general-purpose fifth generation language of the future. Whether purposeful or not, ignoring the primary features of any one of these languages is tantamount to neglect. Our aim, in this paper, is to support the claim that Smalltalk is as important as Lisp and Prolog by briefly reviewing the three languages in juxtaposition with the object-oriented paradigm. The ultimate conclusion is that ignoring Smalltalk is as negligent as ignoring Prolog.

2 The Object-Oriented Paradigm

The object-oriented viewpoint treats objects as the primary focus in designing and implementing software systems. Objects are endowed with a representation (data) and methods (code) for responding to messages (queries). Since the objects are directly responsive, they are active (as opposed to passive) data structures. Such active objects are generally referred to as actors.

The actor paradigm is receiving renewed interest both as a consequence of the recent availability of Smalltalk processors[14] and as a side-effect of the actor-based research being pursued by the Artificial Intelligence community.[19,24]

Actors are the result of a convergence of several computer science/artificial intelligence developments and ideas:

1. abstract data types and modularity from programming languages such as Euclid and Ada,
2. message passing from operating systems and distributed processing,
3. inheritance from frame-based artificial intelligence systems,
4. objects and object-oriented programming from Lisp (Flavors) and the Smalltalk community.

It is the natural paradigm for describing real world objects since such objects can be made to correspond one-to-one with actors. The ease of matching actors with domain objects is expected to play a major role as systems become increasingly knowledge based.

Additionally, inheritance mechanisms that permit data/code sharing along hierarchies permit modularization to a degree which is not possible with the more primitive scoping mechanisms of traditional languages. In conjunction with generic operations provided automatically by the object-oriented approach, the inheritance mechanism provides effective support for programming-in-the-large[10].

The basic notions of general actor systems are embodied in Act/1 and described in several publications by Hewitt and Lieberman[1,16–19,24,25]. Other sources include Byrd et al[6], Laff[23], Reynolds[28] and Thalmann[32]. Smalltalk and its style of object-oriented programming is described in detail in Goldberg and Robson[14] and Goldberg[15]. Other sources include Borning[5], Curry et al[9], Green et al[13] and the Xerox group[36]. Applications of actors and object-oriented programming include Borning[4], Kahn[20] and Klahr[21].

The Act/l project is concerned with the design and implementation of a distributed system with a very large number of processors, the Apiary[19]. Problems associated with workload balancing and special hardware for performance are fundamental problems being considered. Smalltalk, on the other hand, is more concerned with providing an object-oriented programming system as a replacement for traditional systems. Implementations on conventional hardware (e.g. M68000 systems) with acceptable performance is the main goal. In spite of interesting performance improvements[11,31,33], system performance is still marginal but promising.

From the point of view of fifth generation systems, the actor approach competes with the logic approach in its promise of providing highly concurrent hardware/software systems for symbolic computing. However, it deviates by insisting that side-effects, as encapsulated in individual actors (actors can modify their data), are of paramount importance. The logic school, on the other hand, ignores the problems associated with operations that have side-effects; e.g., add-clause, delete-clause, ..., as it attempts to develop a highly parallel version of Prolog comparable to "pure" Lisp. The hope is that side-effects can be introduced after the fact without impacting or compromising the parallelism.

3 Lisp as a Fifth Generation Language

Traditionally, Lisp has been known for its list processing and language extension capabilities through macros and powerful readers (among other features). It is the most important A.I. language of the eighties and required background for any serious A.I. endeavour. However, such capabilities are not the exclusive property of Lisp or its derivatives. For instance, lists are also available in Smalltalk and so are control structure extension (although the methodology is different). On the other hand, in its original form Lisp lacks concurrency primitives, does not support abstract data types very well, nor is it message oriented. Extending the language to include these features in an integrated manner is a major effort.

Most recently, object-oriented programming (via Flavors) has been added to extend the power of the language. Although this is a step in the right direction, the extension is not well integrated with the existing data types. It is in the direction of helping Lisp become the PL/1 of the eighties. On the other hand, the MIT methodology[19] which consists of a redesign of the Lisp base using actors and message-passing as the underlying paradigm is a better approach in the long run.

4 Prolog as a Fifth Generation Language

Prolog[8] is the first practical instance of the logic family of programming languages. It provides unification and backtracking as the two most important programming concepts. With the impetus provided by the Japanese fifth generation project,[12,27] Prolog and its variants are receiving accelerating interest. New architectures[3,34] for variations of Prolog are being investigated along with new languages such as Concurrent Prolog[29,30].

From the point of view of a fifth generation language, Prolog suffers from several deficiencies. The most notable is that it requires and uses side-effects extensively (assert/retract) in spite of the fact that the language semantics ignores it. Current research is concerned with adding parallelism to the language but ignores the problems that will be faced by programmers as they attempt to manage assert/retract in the presence of concurrency. Actor systems, on the other hand, support both side-effects and concurrency as fundamental principles.

Another problem is that Prolog is very much a research tool. It lacks a comprehensive programming environment. Moreover, it is data structure oriented rather than data type oriented; the representation of the data must be known exactly in order to access relevant information (a regression from the point of view of computer science). It lacks modularization facilities although simple scoping facilities are appearing. Lastly, it is evolving towards actors since object-oriented features are being introduced[37].

5 Smalltalk as a Fifth Generation Language

Smalltalk[14] is the first member of a new family of actor languages. It consists of a small, well-defined language (similar to Pascal in size) along with a complete multi-window programming environment. The virtual machine specification is available along with the source code. Additionally, it includes extensive libraries with browsing capabilities that allows the system to be easily extended, modified, and experimented with.

Although Smalltalk is a small, well designed language and environment, it does not offer the full power of an actor system. For instance,

1. all instances of a specific class must have identical representations and methods. Thus,
    1. instances are not allowed to have personalized methods.
    2. multiple representations are not possible.
2. It doesn't allow specialization of classes with individual representations (subclasses must have a representation that **includes** the supertype representation).
3. It has a primitive multiprocessing facility based on semaphores - higher level facilities can be provided by more general actor systems.
4. Method inheritance and the is_a hierarchy are needlessly intertwined.

Nevertheless, Smalltalk does provide the major properties of an object-oriented system: modularity, generic operations, inheritance, and message-passing. Although not originally conceived as a symbolic manipulation language to rival Lisp, it does have the requisite facilities (e.g., list manipulation and built-in garbage collection), and extension capabilities to make it so.

6 Conclusion

Although fifth generation languages and systems have yet to appear, it is not too early to introduce students to the concepts and programming techniques that will play a dominant role. By focussing on the deficiencies of Lisp and Prolog, we have tried to point out the importance of Smalltalk as a vehicle for supplying the "missing" pieces.

Additional evidence for the importance of actors and the actor paradigm can be seen from the increasing popularity of object-oriented programming. Consider Green[13], Borning[4], Kahn[20], Klahr[21], Reynolds[28], and Thalmann[32]. More important is the fact that (a) new object-oriented languages such as Smallworld[23], Ross[26], Loops[2], and Act/1[24] are being designed and (b) object-oriented features have been added to Lisp via Flavors[7,35] and Prolog via extensions[37].

7 References

1. Atkinson, R. and Hewitt, C., *Synchronization in Actor Systems*, Conference Record of the 4th. ACM Symposium on Principles of Programming Languages, Los Angeles, California, Jan. 17-19, 1977.
2. Bobrow, D.G., and Stefik, M.J., *The LOOPS Manual (Preliminary Version)*, Knowledge-based VLSI Design Group Technical Report, KB-VLSI-81-13, Stanford University, August 1984.
3. Borgwardt, P., *Parallel Prolog Using Stack Segments on Shared-Memory Multiprocessors*, 1984 International Symposium on Logic Programming, Atlantic City, New Jersey, Feb. 1984, pp. 2-11.
4. Borning, A., *The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory*, ACM TOPLAS, Vol 3, No. 4, October 1981, pp 353-387.
5. Borning, A. and Ingalls, D.H., *Multiple Inheritance in Smalltalk-80*, Proceedings of the AAAI Conference, Pittsburgh, Aug. 1982.
6. Byrd, R.J., Smith, S.E. and de Jong, S.P., *An Actor-Based Programming System*, ACM SIGOA Conference on Office Information Systems, Univ. of Philadelphia, June 21-23, 1982.
7. Cannon, H.I., *Flavors*, Technical Report, MIT Artificial Intelligence Lab., 1980.
8. Clocksin, W.F. and Mellish, C.S., *Programming in Prolog.* Springer-Verlag, 1982.
9. Curry, B., Baer, L., Lipkie, D., and Lee, B., *Traits: An Approach to Multiple-Inheritance Inheritance Subclassing*, Proceedings ACM SIGOA Conference on Office Information Systems, published as ACM SIGOA Newsletter Vol. 3, Nos. 1 and 2, 1982.
10. DeRemer F., and Kron, H., *Programming-in-the-Large Versus Programming-in-the-Small*, IEEE Transactions on Software Engineering, SE-2, June 1976, pp 80-86.

11. Deutsch, L.P., *Efficient Implementation of the Smalltalk-80 System*, 11th ACM Symposium on Principles of Programming Languages, Jan. 1984, pp. 297-302.
12. Feigenbaum, E.A. and McCorduck, P., *The Fifth Generation*, Addison-Wesley, Reading, Mass., 1983.
13. Green, M. and Philp, P., *The Use of Object-oriented Languages in Graphics Programming*, Proceedings of NCGA Graphics Interface 1982 Conference, Toronto, 1982.
14. Goldberg, A. and Robson, D., *Smalltalk-80: The Language and Its Implementation*, Addison Wesley, Reading, Mass., 1983.
15. Goldberg, A., *Smalltalk-80: The Interactive Programming Environment*, Addison Wesley, Reading, Mass., 1984.
16. Hewitt, C., Bishop, P. and Steiger, R., *A Universal Modular Actor Formalism for Artificial Intelligence*. OJCAI-73 Stanford, California, Aug. 1973.
17. Hewitt, C., *Protection and Synchronization in Actor Systems*, ACM SIGCOMM-SIGOPS Interface Workshop on Interprocess Communication, Santa Monica, California, March 24-25, 1975.
18. Hewitt, C., *Viewing Control Structures as Patterns of Passing Messages*, A.I. Journal, Vol. 8, No. 3, June 1977, pp. 323-364.
19. Hewitt, C., *The Apiary Network Architecture for Knowledgeable Systems*, Conference Record of the 1980 Lisp Conference, Stanford University, Aug. 1980.
20. Kahn, K., *DIRECTOR Guide*. MIT AI Memo 482B, Dec. 79.
21. Klahr, P., McArthur, D., and Narian, S., *SWIRL: An Object-Oriented Air Battle Simulator*. Proc. AAAI-82, Pittsburgh, August, 1982.
22. Krasner, G., *Smalltalk-80: Bits of History, Words of Advice*, Addison Wesley, Reading Mass., 1983.
23. Laff, M.R., *Smallworld - An Object-Based Programming System*. IBM Research Report RC-9022, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1981.
24. Lieberman, H., *A Preview of ACT 1*. MIT AI Laboratory Memo No. 625, June 1981.
25. Lieberman, H., *Thinking About Lots of Things at Once Without Getting Confused - Parallelism in Act 1*, MIT AI Laboratory Memo No. 626, May 1981.
26. McArthur, D., and Klahr, P., *The ROSS Language Manual*, N-1854, The Rand Corporation, Santa Monica, 1982.
27. Moto-Oka, T. (ed.), *Fifth Generation Computer Systems*, Proc. of the International Conference on Fifth Generation Computer Systems, Tokyo, Japan, October 1981.
28. Reynolds, C.W., *Computer Animation with Scripts and Actors*. Proceedings of ACM SIGGRAPH Conference, July 1982.
29. Shapiro, E.Y., *A Subset of Concurrent Prolog and Its Interpreter*, ICOT Technical Report. TR-003, 1983.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-49 Out-of-print	**O(N) Election Algorithms in Complete Networks with Global Sense of Orientation** Jorg Sack, Nicola Santoro, Jorge Urrutia, May 1984
SCS-TR-50	**The Design of a Program Editor Based on Constraints** Christopher A. Carter and Wilf R. LaLonde, May 1984.
SCS-TR-51	**Discretized Linear Inaction-Penalty Learning Automata** B.J. Oommen and Eldon Hansen, May 1984.
SCS-TR-52	**Sense of Direction, Topological Awareness and Communication Complexity** Nicola Santoro, May 1984.
SCS-TR-53	**Optimal List Organizing Strategy Which Uses Stochastic Move-to-Front Operations** B.J. Oommen, June 1984.
SCS-TR-54	**Rectilinear Computational Geometry** J. Sack, June 1984.
SCS-TR-55	**An Efficient, Implicit Double-Ended Priority Queue** M.D. Atkinson, Jorg Sack, Nicola Santoro, T. Strothotte, July 1984.
SCS-TR-56	**Dynamic Multipaging: A Multidimensional Structure for Fast Associative Searching** E. Otoo, T.H. Merrett, August 1984.
SCS-TR-57	**Specialization, Generalization and Inheritance** Wilf R. LaLonde, John R. Pugh, August 1984.
SCS-TR-58	**Computer Access Methods for Extendible Arrays of Varying Dimensions** E. Otoo, August 1984.
SCS-TR-59	**Area-Efficient Embeddings of Trees** J.P. Corriveau, Nicola Santoro, August 1984.
SCS-TR-60	**Uniquely Colourable m-Dichromatic Oriented Graphs** V. Neumann-Lara, N. Santoro, J. Urrutia, August 1984.
SCS-TR-61	**Analysis of Distributed Algorithms for Extrema Finding in a Ring** D. Rotem, E, Korach and N. Santoro, August 1984.
SCS-TR-62	**On Zigzag Permutations and Comparisons of Adjacent Elements** M.D. Atkinson, October 1984.
SCS-TR-63	**Sets of Integers with Distinct Differences** M.D. Atkinson, A. Hassenklover, October 1984.
SCS-TR-64	**Teaching Fifth Generation Computing: The Importance of Small Talk** Wilf R. LaLonde, Dave A. Thomas, John R. Pugh, October 1984.
SCS-TR-65	**An Extremely Fast Minimum Spanning Circle Algorithm** B.J. Oommen, October 1984.
SCS-TR-66	**On the Futility of Arbitrarily Increasing Memory Capabilities of Stochastic Learning Automata** B.J. Oommen, October 1984. Revised May 1985.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-67	**Heaps in Heaps** T. Strothotte, J.-R. Sack, November 1984. Revised April 1985.
SCS-TR-68	**Partial Orders and Comparison Problems** M.D. Atkinson, November 1984.
SCS-TR-69	**On the Expected Communication Complexity of Distributed Selection** N. Santoro, J.B. Sidney, S.J. Sidney, February 1985.
SCS-TR-70	**Features of Fifth Generation Languages: A Panoramic View** Wilf R. LaLonde, John R. Pugh, March 1985.
SCS-TR-71	**Actra: The Design of an Industrial Fifth Generation Smalltalk System** David A. Thomas, Wilf R. LaLonde, April 1985.
SCS-TR-72	**Minmaxheaps, Orderstatisticstrees and their Application to the Coursemarks Problem** M.D. Atkinson, J.-R. Sack, N. Santoro, T. Strothotte, March 1985.
SCS-TR-73	**Designing Communities of Data Types** Wilf R. LaLonde, May 1985.
SCS-TR-74	**Absorbing and Ergodic Discretized Two Action Learning Automata** B. John Oommen, May 1985.
SCS-TR-75	**Optimal Parallel Merging Without Memory Conflicts** Selim Akl and Nicola Santoro, May 1985
SCS-TR-76	**List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations** B. John Oommen, May 1985.
SCS-TR-77	**Linearizing the Directory Growth in Order Preserving Extendible Hashing** E.J. Otoo, July 1985.
SCS-TR-78	**Optimal Semijoin Evaluation in Distributed Query Processing** E.J. Otoo, N. Santoro, D. Rotem, July 1985.
SCS-TR-79	**On the Problem of Translating an Elliptic Object Through a Workspace of Elliptic Obstacles** B.J. Oommen, I. Reichstein, July 1985.
SCS-TR-80	**Smalltalk - Discovering the System** W. LaLonde, J. Pugh, D. Thomas, October 1985.
SCS-TR-81	**A Learning Automation Solution to the Stochastic Minimum Spanning Circle Problem** B.J. Oommen, October 1985.
SCS-TR-82	**Separability of Sets of Polygons** Frank Dehne, Jorge-R. Sack, October 1985.
SCS-TR-83	**Extensions of Partial Orders of Bounded Width** M.D. Atkinson and H.W. Chang, November 1985.

SCS-TR-84	**Deterministic Learning Automata Solutions to the Object Partitioning Problem** B. John Oommen, D.C.Y. Ma, November 1985
SCS-TR-85	**Selecting Subsets of the Correct Density** M.D. Atkinson, December 1985