Issues paper for Ada/AI/RT WG Workshop, Summer '92 SigAda Meeting, June 24-25, 1992.

Ada9X Issues for AI Systems

Henry G. Baker

Nimble Computer Corporation, 16231 Meadow Ridge Way, Encino, CA 91436
(818) 986-1436 (818) 986-1360 (FAX)

We must offer suggestions to the Ada-9X Committee for allowing/enhancing AI programs in Ada.

Defense AI Applications in Ada-83? In Ada-9X? What are the issues?

Can AI programs be successfully written and deployed in Ada-83? Do the changes contemplated in Ada-9X make the writing and deployment of AI programs any easier? Do there still exist major "gotchas" in Ada-9X which will seriously decrease the performance and/or increase the cost of developing and deploying AI programs in Ada?

AI programs are some of the largest *programs* around, in terms of lines of code, complexity, cost, etc. I.e., they are large *programs*, but small amounts of *data*, relative to more traditional embedded systems programs. E.g., *compiled* program text of large AI system can take 20 megabytes, whereas "large" non-AI programs use 1-2 megabytes for both program and data (not counting large passive databases such as terrain mapping databases).

What are the design, implementation and maintenance implications of such large programs in an Ada environment? E.g., small changes may cause massive recompilations—it could take hours or days to "make" a new version of a system. With a program of this size, is it *ever* "delivered"? If only 10^{-3} of such a program is changed annually, this may still be 10,000 lines of code changed per year. Rumors exist of massive software changes/fixes for the Patriot missile system while the unit was already in battle. Significant changes may be required in a "Pilot's Associate" program *for every mission*. In other words, "program" may have become "data", to be loaded for each and every mission. Does this mean that "dynamic loading" a la Berkeley Unix or Unix V Rel. 4 should become part of an Ada run-time system? Does this require a "persistent Ada heap", a la current "object-oriented databases"?

A "signature" characteristic of AI programs is their "late binding" of control constructs, which are universally implemented by means of first-class function closures. These closures are dynamically constructed functions which can be passed as arguments, returned as values, and stored into data structures as values. Ada-83 was expressly forbidden by its Steelman Requirements to have no such capability. Ada-83 offers *generic* functions and procedures, which can emulate some, but not all, of these late binding constructs. Do the capabilities of Ada-9X provide enough relief to satisfy the AI developer, or should we send the Ada-9X team back to the drawing board?

AI people have been requesting garbage collection for Ada at least since 1980 [Schwartz80], yet no vendor provides it, and Ada compiler/runtime validation does not require garbage collection. Yet GC is an extremely valuable tool in allowing the decomposition of large systems without increasing the probability of failure due to *dangling references*. Such dangling references are becoming more and more likely with the dramatic increase in pointer-based programs due to the popularity of "object-oriented" programming. Can garbage collection be emulated on top of Ada with enough efficiency to support the heavy computational demands of AI programs?

Traditional AI systems require a large address space and the shared-memory paradigm. Yet many embedded systems are designed with hardware that supports a distributed-memory/message-passing model, and it may be quite difficult to map AI programs onto these platforms. The Ada parallel process model clearly prefers an explicit exchange of information via the rendezvous mechanism, and only grudgingly supports the notion of asynchronous access to shared data. Yet the most popular model for parallel, embedded real-time AI systems is the "blackboard" model, which has at its core a database shared and asynchronously updated by all processes!

Although Ada was standardized in 1983, production quality compilers were not available until the 1986-87 time frame, and significant bugs are still prevalent in Ada83 compilers today. For example, generics could not be used reliably in the first generation of Ada83 compilers, and "storage leaks" continue even in today's Ada83 runtime systems. Thus, it seems prudent to recognize that it may be 1995 before debugged, reliable compilers and runtime systems are available for Ada-9X. In this case, another generation of weapons systems will be developed in Ada83—i.e., they will not be able to take advantage of any of the newer Ada-9X capabilities. Since AI capabilities are being put into systems today, what is the near-term effect of doing this in Ada83 v. Ada-9X? What are the long-term effects—i.e., efficiency, maintenance, obsolete protocols, etc.—of this delay?

Some "100% Ada" projects are using Ada as "just another language", to be loaded into a separate address space on a classical operating system with multiple address spaces. Any synchronization between the separate address spaces is implemented by means of non-Ada capabilities—e.g., locking in a "file" system. The Ada strong typing system is side-stepped by reading and writing to external "files". Worst of all, Ada run-time checks within an "application" (i.e., address space) are disabled, with any protection provided by the hardware—e.g., "bus error". Do such system designs conform to the spirit, as well as the letter, of the Ada law, or are they a pragmatic solution to an inflexible language standard? Perhaps such systems recognize the inevitable need to interface Ada with COTS technologies—most likely C, C++ or even Lisp(!).

Issues of ancillary standards and tools. Are the MIL-STD's for software design, documentation, implementation and testing appropriate for AI programs, or are they too rigid? Can the complex notions of AI programs even be expressed in these "design methodologies" and "design tools", or are new methodologies and tools required. Do the proposed documentation and coding standards put the AI programmer into a straight-jacket (assuming that Ada itself hasn't already)? Are the APSE/KAPSE/... tools part of the solution, or part of the problem?

Are real-time AI programs a mirage? Can an AI program ever be expected to always respond within a fixed latency, or must we start planning for only stochastic response latencies? What sorts of scheduling capabilities do AI programs require beyond those useful for other real-time programs?

Overall Goal of Workshop and Summary

Potential contractor/developers of defense software systems have little incentive to make investments in standards or tools for the uncertain likelihood of future contracts. Since AI capabilities are new, there is no established pool of experience in the defense software contracting industry which can fight for the language changes and tools which will make AI programming easier, cheaper and more effective. There are, on the other hand, major established groups to argue for better signal processing support, better decimal/mainframe support, better network support, better real-time support for traditional software control loops, etc. It is therefore likely that significant "holes" exist in the Ada language and infrastructure, which will only become evident later, when projects become late and costs balloon.

The overall goal of this workshop is a document which clearly states the requirements for programming languages to support <u>real-time embedded AI programs for defense applications</u>. These requirements need to be prioritized, and the consequences and costs of not meeting the requirements need to be estimated. Since modern warfare puts the ultimate premium on up-to-date intelligence, efficient resource allocation, and pin-point accuracy, AI will play a pivotal role in making sure that the weapons are located at the right place and the right time, and used against the right target with the appropriate ammunition with sufficient accuracy and concentration to knock out the target once, but only once. We have to make sure that we are fighting the *next* war rather than the *previous* war.

REFERENCES

Ada83. *Reference Manual for the Ada® Programming Language*. ANSI/MIL-STD-1815A-1983, U.S. Gov't Printing Office, Wash., DC, 1983.

Baker, Henry. "List Processing in Real Time on a Serial Computer". *Comm. of the ACM 21*,4 (April 1978),280-294.

Baker, H.G. "The Automatic Translation of Lisp Applications into Ada". *Proc. 8'th Conf. on Ada Tech.*, Atlanta, GA (March 1990),633-639.

Baker, H.G. "Structured Programming with Limited Private Types in Ada: Nesting is for the Soaring Eagles". *Ada Letters XI*,5 (July/Aug 1991), 79-90.

Baker, H.G. "Object-Oriented Programming in Ada83—Genericity Rehabilitated". *Ada Letters XI*, 9 (Nov/Dec 1991), 116-127.

Baker, H.G. "CONS Should not CONS its Arguments, or A Lazy Alloc is a Smart Alloc". ACM *Sigplan Not. 27*,3 (March 1992), 24-34.

Baker, H.G. "The Treadmill: Real-Time Garbage Collection without Motion Sickness". ACM *Sigplan Not. 27*,3 (March 1992), 66-70.

Baker, H.G. "Equal Rights for Functional Objects or, The More Things Change, The More They Are the Same". ACM OOPS Messenger, 1992, to appear.

Baker, H.G. "Iterators: Signs of Weakness in Object-Oriented Languages". ACM *OOPS Messenger*, 1992, to appear.

Barnes, J.G.P. *Programming in Ada: Third Edition*. Addison-Wesley, Reading, MA, 1989,494p.

Hosch, Frederick A. "Generic Instantiations as Closures". *ACM Ada Letters 10*,1 (1990),122-130.

Kernighan, Brian W., and Ritchie, Dennis. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

Kownacki, Ron, and Taft, S. Tucker. "Portable and Efficient Dynamic Storage Management in Ada". *Proc. ACM SigAda Int'l Conf., Ada Letters*, Dec. 1987,190-198.

Mendal, Geoffrey O. "Storage Reclamation Models for Ada Programs". *Proc. ACM SigAda Int'l Conf., Ada Letters*, Dec. 1987,180-189.

Perez, E.P. "Simulating Inheritance with Ada". *ACM Ada Letters 8*,5 (1988),37-46.

Rosen, Steven M. "Controlling Dynamic Objects in Large Ada Systems". *ACM Ada Letters 7*,5 (1987),79-92.

Schwartz, Richard L., and Melliar-Smith, Peter M. "The Suitability of Ada for Artificial Intelligence Applications". Final Report, Contract #AAG29-79-C--0216, SRI Int'l., Menlo Park, CA, May 1980,48p.

Smith, D. Douglas. "ALEXI—A Case Study in Design Issues for Lisp Capabilities in Ada". *Wash. Ada Symp. 5* (June 1988),109-116.

Steele, Guy L. *Common Lisp, The Language; 2nd Ed*. Digital Press, Bedford, MA, 1990,1029p.

Taft, Tucker, *et al*. *[Ada-9X] DRAFT Mapping Document*. Ada-9X Proj. Rep., Feb. 1991.

Taft, Tucker, *et al*. *[Ada-9X] DRAFT Mapping Rationale Document*. Ada-9X Proj. Rep., Feb. 1991.

Yen, Mike. "Adapting an AI-Based Application from its Lisp Environment into a Real-Time Embedded System". *Proc. AIAA Comps. in Aerospace VII,* Monterey, CA, (Oct. 1989),1114-1122.

Yen, Mike. "Using a Dynamic Memory Management Package to Facilitate Building Lisp-like Data Structures in Ada". *Proc. AIDA-90*, Nov. 1990, 85-93.

