USING ADA* -- THE DEEPER CHALLANGES

David A. Feinberg, C.D.P.

Software Technology
Boeing Aerospace Company
P. O. Box 3999, M/S 82-53
Seattle, Washington 98124
Telephone: (206) 773-5485

ABSTRACT:

The Ada programming language and the associated Ada Programming Support Environment (APSE) and Ada Run Time Environment (ARTE) provide the potential for significant life-cycle cost reductions in computer software development and maintenance activities. The Ada programming language itself is standardized, trademarked and controlled via formal validation procedures. Though compilers are not yet as production-ready as most would desire, the technology for constructing them is sufficiently well known and understood that time and money should suffice to correct current deficiencies.

The APSE and ARTE are, on the other hand, significantly newer issues within most software development and maintenance efforts. Currently, APSE and ARTE are highly dependent on differing implementer concepts, strategies and market objectives. Complex and sophisticated mission-critical computing systems require the use of a complete Ada-based capability, not just the programming language itself; yet the range of APSE and ARTE features which must actually be utilized can vary significantly from one system to another. As a consequence, the need to understand, objectively evaluate, and select differing APSE and ARTE capabilities and features is critical to the effective use of Ada and the life-cycle efficiencies it is intended to promote. Methodologies for dealing with dissimilar APSE/ARTE systems are also in sore need of definition and understanding; particularly for industry contractors who will be developing similar capabilities (e.g., missile and air/space craft navigation, guidance, throttle control) for differing customers (e.g., Army, Navy, Air Force, NASA, Boeing, Airbus).

It is the selection, collection, and understanding of APSE and ARTE which provide the deeper challanges of using Ada for

---

* Ada is a registered trademark of the United States Government (Ada Joint Program Office)

real-life mission-critical computing systems. This paper discusses some of the current issues which must be clarified, often on a case-by-case basis, in order to sucessfully realize the full capabilities of Ada.

1. INTRODUCTION

In the early 1970's, the Department of Defense (DOD) recognized several problems related to the acquisition of software for major defense systems. Software systems were too frequently late, unreliable, and more expensive than planned. Additionally, there was a steadily rising trend in software costs while, at the same time, computer hardware costs were decreasing significantly.

At the time, the primary cause of these problems was identified as a deficiency in the computer programming process; particularly in the area of programming languages. There were over 450 general purpose languages and dialects being used for DOD systems with no single point of control for each. Many of these languages were poorly suited to their application, and/or did not take advantage of nor support good programming practices. The DOD was also beginning to recognize the long-term life-cycle advantages of using higher order languages (HOL's) rather than assembler code. By 1974, each of the military services was independently proposing development of a standard HOL for their service's mission-critical software development.

In January, 1975, a joint services HOL working group began identifying and defining requirements for all DOD HOL's and individual service efforts were halted. The "Strawman" document issued in April, 1975, started a multi-year effort which culminated in 1981 and 1983 with the establishment of ANSI/MIL-STD-1815A, "Reference Manual for the Ada Programming Language," as a single DOD standard for all future mission-critical computer software development efforts.

Unfortunately, during the six years required to produce the Ada standard, the understanding of the problems of developing large, complex software systems evolved. While the programming process was still important, newer full-life-cycle models of software project activities reduced programming's overall significance to only 20% of the whole; much less than was thought in the early 1970's.

In response to this changing perception, the HOL working group began to recognize that the new common DOD HOL alone would not be sufficient to ensure DOD's desired improvements in software development. The programming environment within which Ada would

operate needed significant improvement.

Following two years of work, an Ada Programming Support Environment (APSE) was defined in the 1980 "Stoneman" document. Even though this document provides criteria for assessment and evaluations of programming environments, it is not a standard and, as such, implementers of Ada tools are not bound by any hard and fast requirements. Rather, implementers are free to choose any of the four "Stoneman"-defined levels of Ada programming support. More importantly, they are also free to select, as they see fit, specific tools within each of the levels. Thus, while Ada, the language, is tightly controlled, APSE's are not controlled at all and vary significantly from one implementer's products to another's.

In a similar manner, an Ada Run Time Environment (ARTE) can also vary significantly. Once the necessities of the Ada language standard are satisfied, implementers are free to produce a wide variety of operating executives. In fact, ARTE development is even less constrained than development of an APSE; no assessment and evaluation document such as "Stoneman" even exists for run time requirements.

In response to the absence of APSE and ARTE system standardization, projects using Ada must, on a case-by-case basis, identify those features most necessary to their specific requirements. Once this is done, evaluation of the numerous implementer offerings is required in order to select the critical environmental capabilities which will be used. The following sections describe the key issues affecting selection of Ada Programming Support and Ada Run Time environments.

2. ADA PROGRAMMING SUPPORT ENVIRONMENT

An Ada Programming Support Environment (APSE) consists of a number of individual tools which provide software support to write, test and maintain Ada language programs. An APSE can also be used to provide orderly program development methodology. Tools within an APSE will vary from implementer to implementer; however, most implementers conform at some level to the "Stoneman" document. The cooperating ability of tools with each other, as opposed to merely "Stoneman" tools-database interfaces, can, however, vary significantly.

Typically, an APSE will consist of at least the minimum tool levels described in "Stoneman": an operating system, a Kernal APSE (KAPSE), and a Minimal APSE (MAPSE). With the exception of a debugger, it is virtually impossible to utilize Ada without the MAPSE tools: a compiler, linker/loader, editor, configuration manager, and job control language processor.

Additionally, a full APSE (i.e., any Ada Programming Support Environment with tools in excess of those called for by MAPSE) may consist of any number of augmenting tools such as a pretty printer, cross reference generator, test generator, program design language processor, source code control system, problem reporting system, etc.

Evaluation of an APSE is required in order to determine which available environment best fits the needs of a specific Ada-based project. This minimally requires analyzing the tools in a given APSE to determine their effectiveness, and where possible, to directly compare them to similar tools in other APSE's.

While quantitative methods can be used to examine many tools, this is not always possible. First, even though two (or more) tools perform the same function on the same computer using the same operating system, their performance characteristics may vary significantly based on computer load factors at the time of testing. Even if these factors can be controlled or mitigated, design parameters of the tools themselves can cause fluctuating performance data depending on individual account and session situations. In general, modern virtual memory multi-component computer systems can play havoc with what appear to be straight-forward quantitative evaluations.

The second reason is that quantitative evaluation methods are not always applicable. Discussions of such factors as "user friendliness" do not realistically lend themselves to quantitative accumulation. Even so, these factors can be significant issues when determining the overall effectiveness of a tool.

While individual tool evaluations are important, even more critical is extending any evaluation to the integration and cooperation of all of the tools which comprise an APSE. It is not uncommon for individual software tools to be efficacious as stand-alone entities, yet efforts to use the results of one as grist for another fail totally. Such an overall view of APSE effectivity and suitability cannot be obtained by simply summing the results of individual tool evaluations. An APSE must be reviewed as an integrated (or non-integrated) whole to determine if it fulfills a project's software development needs.

3. ADA RUN TIME ENVIRONMENT

An Ada Run Time Environment (ARTE) is the collaboration of program object code conventions with data structures used to interface to the underlying run time system. This system, in turn, consists of a series of library and/or executive routines that are necessary to support execution of Ada programs. Typical functions of an ARTE include general operating system services as well as Ada-specific features such as tasking, dynamic memory management, exception handling, interrupt processing and any other needed support deferred from a compiler's code generation phases.

Even though the Ada language is standardized, the ARTE for different computers and operating systems can vary widely. This can be due to differences in computer hardware, operating systems, compiler implementations of Ada semantics, or, the most frequent case, a combination of all of these. Additional variations can result from trade-offs for reasons of ARTE or program size, speed, overhead, capability, or portability.

In rare cases, a specific project using Ada will find one or more ARTE implementations which are universally best suited to its needs. Usually, however, compromises between various implementations in terms of project priorities will be required. Given the characteristics of most mission-critical software programs, the best ARTE may turn out to be the one that is easiest and safest to modify on a case-by-case basis.

Evaluation of ARTE elements depends on the depth to which a project is required to delve. Some elements (e.g., code size, coding language, implemented pragmas) are readily apparent by simple examination of external characteristics or implementer documentation. Others (e.g., subprogram call timing, arithmetic implementations) can be found through test program executions. Still others (e.g., delay overhead, task dispatch algorithm) can only be determined by detailed analysis (or even experimental modifications) of the run time code itself.

4. ENVIRONMENTAL PROLIFERATION

Even though the Ada programming language itself is standardized, trademarked, and controlled via formal validation procedures, Ada Programming Support Environments (APSE) and Ada Run Time Environments (ARTE) are not. The U. S. Army has already taken delivery of its APSE/ARTE system: the Ada Language System (ALS). The Air Force continues to make progress on key components of its support environment: the Ada Integrated Environment (AIE) and its supporting Ada Compilation System (ACS). Within the past few months, the Navy has let a contract

for its version of the ALS: ALS/N. NASA has also established its policy calling for an integrated Software Support Environment to support use of Ada for Space Station operational software.

Thus far, with the partial exception of ALS and ALS/N, none of the existing APSE/ARTE systems are compatible with each other; even though they execute on identical host and target computers. When systems on the drawing boards plus commercially available products (e.g., Systems Designers' "Perspective", Verdix's "VADS") are added to the list, the proliferation of dissimilar capabilities, facilities and functions will reach significant proportions. The late 1980's have all the potential to become highly reminiscent of the 1970's programming language proliferation which led to Ada in the first place (Figure 1).

ORGANIZATION	1970'S HOL	1980'S APSE
Air Force	JOVIAL	AIE/ACS
Army	TACPOL	ALS
Navy	CMS-2	ALS/N
NASA	HAL/S	SSE
Industry	FORTRAN Pascal C	Perspective VADS ADE etc.

Figure 1. Organizational Standards Proliferation

The potential proliferations in late 1980's APSE/ARTE are the well-intentioned result of attempts to "graft" enhancements onto the Ada programming language, which is in turn, the solution to the 1970's perception of the software development problem. Ada was initially designed to correct difficulties in programming. Current, 1980's, estimates allot only 20% of the software development cycle to programming, and consequently, Ada needed to be expanded to fit a newer, better, full-life-cycle model. Unfortunately, the "Stoneman" grafts have been done

"on-the-fly", and have, in turn, recreated a mutant of the initial problem. The Ada Language is standardized. APSE's and ARTE's are not. Moreover, differing organizations are beginning to require use of their incompatible APSE/ARTE even as full-life-cycle model methodologies for software development are beginning to coalesce (e.g., DOD-STD-2167).

5. CONCLUSION

The use of Ada and its associated Ada Programming Support Environment (APSE) and Ada Run Time Environment (ARTE) continues to provide a high potential for significant life-cycle methodology improvements and cost reductions in software development and maintenance activities. In order to move the significant advantages of Ada from potential to actual, several concurrent efforts must be completed. The first, development of high quality compilers and optimizing code generators, is already well underway. Over a dozen organizations currently offer Ada compilers and some form of minimal programming support tools. The technology necessary to improve these offerings has been in existence for over a dozen years. Time and incentive should produce the needed production quality compilers.

Development of full-function, integrated, APSE's is the second needed effort. While the goal of this effort is conceptually clear, the steps necessary to reach it remain unacceptably vague. Full scale software development environments have been proposed for years, but no universally usable one yet exists. Using Ada as a vehicle for producing such a capability has much merit and the "Stoneman" document provides some necessary guidance. Unfortunately, these items are not yet enough. Significant research into programming environment requirements and solution sets, particularly those dealing with human factors and expert systems, remains to be accomplished.

The third effort needed to move Ada from potential to actual usage is the development of a configurable ARTE. Ada is intended for "mission-critical" computing systems. These systems can range from ground-based surveillance and tracking systems (air, space, sea) to in-flight avionics (manned, unmanned) to simple sensor/actuator systems, and much much more. Even though all of these mission critical systems can be considered as "real time," many other widely varying characteristics can affect their execution environment constraints. A great deal of research and development remains to be done. The need for an ARTE criteria and evaluation document is barely even recognized. Yet, the ultimate key to mission-critical computing is its performance in the field; under "production" conditions.

Finally, and most difficult, is the need to recognize and begin to resolve the issue of incompatible APSE/ARTE systems. Using the full-life-cycle model demanded of today's software development process, the proliferation of differing services' tool sets can clearly become counter-productive; particularly for organizations performing similar work for different customers.

The work of many individuals and organizations will be required to complete the efforts described in this paper. The definition, rationalization, implementation and integration of APSE and ARTE into the Ada language to create complete software development environments are now the deeper challanges of using Ada. Only when they are accomplished will Ada be able to meet the ultimate goals for which it was created.

ACKNOWLEDGEMENTS:

The work of many individuals is necessary not only to answer the questions raised in this paper, but to raise and clarify them in the first place. Several members of Boeing's Ada Project have contributed ideas and concepts which led to this paper. Special recognition belongs to two: James B. Unkefer for his work on Ada Programming Support Environments, and Ruth A. Maule for her clear, effective approach to the enigmas of Ada Run Time Environments. Thanks are also due to Maretta Holden of Boeing Military Airplane Company who continues to see into the future farther than most of us.

REFERENCES:

1. AJPO, "Kernal Ada Programming Support Environment (KAPSE) Interface Team Public Report," Volumes I-V.

2. ARTEWG, "Draft Charter for the Ada Runtime Environment Working Group," July, 17, 1985.

3. ARTEWG, "Ada Implementation Dependencies," November 12, 1985 (draft).

4. United States Air Force, "Preliminary Program Manager's Guide to Ada," document numbers ESD-TR-83-255 and WP-25012, February, 1984.

5. United States Department of Defense, "Ada Methodologies: Concepts and Requirements (METHODMAN)," November, 1982.

6. United States Department of Defense, "Interim DoD Policy on Computer Programming Languages," Memorandum to Secretaries of the Military Departments, et. al., from Under Secretary of Defense Robert DeLauer, June 10, 1983.

7. United States Department of Defense, "Proposed Military Standard Common APSE Interface Set (CAIS), Version 1.4," October 31, 1984.

8. United States Department of Defense, "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A, February, 1983.

9. United States Department of Defense, "Requirements for Ada Programming Support Environments (STONEMAN)," February, 1980.

BIOGRAPHY:

David A. Feinberg, C.D.P., is a specialist in the development and use of software engineering tools and environments. He is employed by The Boeing Company and is currently in charge of the company's Ada Project. During the past twenty-three years, Mr. Feinberg's assignments have included creation of a software development facility used for the construction of commerical electric power distribution and control products; large scale network operations and communications management; and compiler and operating systems construction. He is the author of over twenty-five papers, essays and articles. Mr. Feinberg is a member of ACM, IEEE Computer Society and DPMA, and holds an M.S.A. degree from The George Washington University and a B.S. degree from Stanford.