ADA® IMPLEMENTATION IN MULTI-DEVICE CONFIGURATION

S. Ramachandran
D. McCabe

McDonnell Douglas Helicopter Company
Mesa, Arizona

ABSTRACT

This paper examines Ada implementation of a multi-device configuration in an engineering organization. The advantages and disadvantages of Ada are examined from this perspective. System architecture, software development environment, Ada compilers/cross-compilers and software development environment, Ada compilers/cross-compilers and software engineering methodologies are discussed. Simulation architecture selected by McDonnell Douglas Helicopter Company and lessons learned are presented.

INTRODUCTION

Ada has been mandated as the primary programming language by the Department of Defense (DoD) for mission critical programs. This emphasis on Ada has been reflected in recent training systems initiatives also. Since new training devices are built from the ground up, design and development of new simulators are somewhat straightforward. However, aircraft companies with established simulation facilities face the difficult choice of whether and how to make the transition from a mostly assembler/FORTRAN environment to Ada. McDonnell Douglas Helicopter Company is one of the companies that has been making this difficult and expensive transition. This paper examines the technical issues that are to be considered and resolved to make a successful transition. The discussion here is from the viewpoint of an organization that has to support aircraft development with multi-ship simulation. The major question is should the simulation organization switch over to Ada at all or maintain status quo to the software development? This requires an examination of not only the technical pros and cons of Ada software development but also of the charter for the simulation organization (engineering simulation versus training simulation) and the costs associated with the decision to be taken. The paper first discusses the pros and cons of choosing or not choosing Ada. Problems of transition in the light of various stages of development of different simulators are discussed. Once the decision to choose Ada has been made, the technical issues for a successful simulation implementation are reviewed. McDonnell Douglas Helicopter Company's approach to satisfy its simulation requirements is presented. The lessons learned in achieving Ada implementation are also presented.

ADVANTAGES AND DISADVANTAGES OF ADA

The advantages of using Ada in simulation applications are not much different from those for other applications. Engineering simulation is a large software intensive activity. However, in the context of aircraft development, simulation is a medium where the technical efforts of diverse engineering organizations within the company are brought to focus and where airplane concepts are validated. It is also a tool where software eventually intended for aircraft is developed and tested. For these reasons, the advantages of Ada are even more attractive for simulation application. One such consideration is the easier portability Ada offers. Portability of code has been an elusive goal of simulation software as it is with other software applications. It has been recognized for some time that standardizing on a single language will be a major part of making this goal a reality. At this time the Ada language is being far more tightly controlled than any other language in both the language revision aspect (there is only ONE version of the language) and the compiler implementation aspect (certified validation suites). This makes Ada a stable language to standardize on, making portable code far more feasible than with a non-regulated language. Since aircraft development has become software intensive, it is extremely important to reduce software costs. To achieve this, software must be portable between the simulator, hot bench and aircraft. With Ada mandated as the higher order programming language for aircraft development, adopting Ada as the programming language for the simulator also makes good economical sense as well.

Another advantage is that, in the long term, all programmers will use a common programming language and therefore will have a far easier time transitioning from one aircraft project to another. Ada will also permit efficient utilization of programmers since they could be moved around within the company between simulation and aircraft programs depending upon company needs.

The technical advantages of Ada are even more alluring in the context of commonality between simulators, hotbenches and aircraft. Ada encourages structured object oriented design, which closely resembles the way different systems/subsystems/components operate in an aircraft. Built into the Ada language is the construct of packages which allows a

®Ada is a registered trademark of DoD Ada Joint Program Office.

mechanism for putting all the code which describes an object and its processes into a logical unit. This package can be incorporated with other packages or subprograms thereby allowing the use of these code objects throughout a software system. It is a powerful way of letting the code reflect the objects and processes necessary to control a system in an understandable manner. Ada enforces a high degree of structure by imposing the principles of modularity, abstraction, information hiding, and localization. The interfaces are embodied in the package specification and can be totally defined prior to having to work out the algorithms associated with them. Ada decreases the possibility of having the wrong variables being passed from one software unit to another thereby increasing the understanding of the flow of the software and making maintainability and modification easier.

Ada permits depiction of parallel events in an understandable manner. Most languages address this problem by interfacing from their high level language to either operating system calls (which vary from operating system to operating system) or to assembler routines which schedule multiple programs simultaneously. In Ada this concept is addressed right in the language via the TASK construct. The scheduling of simultaneous events is no longer buried in code as a call to some routine which is written in a low-level language and one has to guess what it is doing. In Ada it is labeled as a TASK with clear rendezvous points.It is in the same language as the rest of the code. This is another invaluable advantage for understanding the code for either maintenance or modification purposes as well as emulating the aircraft hardware operation in the simulator.

However, these advantages have to be weighed against the disadvantages of using Ada. One problem area is the lack of compilers with the system dependent features described in the Ada Language Reference Manual's (LRM) Chapter 13. These features include many of the system programming capabilities which are necessary for simulation applications. These include pragmas (which provide the selection criteria for mapping an entity onto the underlying machine) such as PACK (elimination of gaps in storage areas allocated to consecutive components), INLINE (machine code insertion), and INTERFACE (calling subprograms written in another language). They also include REPRESENTATION CLAUSES (imposing certain characteristics of the mapping of an entity onto the underlying machine), ADDRESS CLAUSES (specification of a required address in storage which allow interrupts to be coded), UNCHECKED DEALLOCATION and UNCHECKED CONVERSION. Although there are plans to include these features in future validation efforts, they are not tested at this time. Therefore, one of the important criteria in considering an Ada compiler is what aspects of Chapter 13 are implemented and to what degree.

Another disadvantage with the implementation of the language is the lack of optimization in both host and target Ada compilers and cross-compilers. There are at least two reasons for this. One is that Ada embodies many aspects of computing that were previously rendered to the realm of operating systems. There is a learning curve involved in just being able to implement these aspects in a high order language. The other is that Ada is relatively new compared to other high order languages. Enough time has not passed for optimization to have been the prime focus of the implementors.

Another disadvantage is the conspicuous lack of software engineers capable of designing software which incorporates the unique features of Ada. A structured methodology of design is mandated by these features and changing the way software is normally developed in a simulation environment can be painful, especially if there are not enough experienced personnel to guide the inexperienced programmers. This disadvantage can be alleviated if concerted efforts are introduced to train programmers in the design aspects necessary to properly use this language.

The above disadvantages will diminish in impact if not totally disappear as the implementations of the language mature and the software engineering community gains experience with it.

Despite the implementation shortcomings of Ada, there are two important considerations for adopting Ada. One is whether the simulation facility is interested in training device contracts. Based on current trends in military training system procurement, it is obvious that if a switch is not made to Ada, the company will be left behind in the training market since more and more military training device contracts require Ada and the company will not have the experience or talent base to compete. If the simulation department does not choose Ada as its standard software development language, it will be required to change existing developed code for every new version of its current standard language, whatever it may be. This is due to a lack of tight control on other languages which has been imposed on the Ada language and implemented through the validation process.

However, the transition to Ada is an expensive proposition since most of the existing software is in a language other than Ada, and usually in FORTRAN. This existing code may have to be redesigned in Ada. This cost could be very high since a software redesign rather than a simple conversion is necessary to take advantage of Ada's strengths. New computers, operating systems, and compilers may have to be bought for the implementation since existing computing systems in many cases may not have Ada development and real-time execution capability. Some existing configurations permit Ada-only

implementations and not simultaneous implementations of new Ada code with existing FORTRAN software.

Putting off the switch to Ada will only make it even more expensive later since additional software will need to be eventually redone in Ada. It is also important to maintain a smooth transition to an Ada environment. Ada software conversion/development has to be achieved with an existing workforce and in an economical fashion while supporting the current simulator operation, developing software for new simulators and planning for future ones.

SIMULATION REQUIREMENTS

McDonnell Douglas Helicopter Company's Engineering and Training Simulation Department (ETSD) was faced with the issue of making a decision regarding Ada in early 1986. At that time ETSD had a full mission simulator in operation in support of a research rotorcraft program. At the same time the department was in the early stages of developing a second simulator for an advanced version of an existing rotorcraft. Plans were also being made for developing a third simulator in 1987 for this program. A major part of the simulation software on the first simulator was in FORTRAN with the rest in assembler and C.

All the simulated aircraft incorporate the latest and planned advances in combat rotorcraft technology including "glass cockpits" and a full suite of avionics, weapons, and sensors. The simulators provide full flight and mission simulation including visual and sensor simulation along with moving map systems. Since the first simulator was developed during 1984, Ada was not used. Consequently, the issue of Ada was taken up later when the second simulator program was started.

Along with the new simulators McDonnell Douglas Helicopter Company also faced the issue of interactive simulation. The aircraft could participate in joint missions in mixed roles of adversaries or friendlies. The development of a system control station for typical instructor/operator functions as well as for engineers to obtain and analyze aircraft data also required the networking of these three simulators plus other simulators as they were developed and brought on line. The requirement for the second and third simulators gave the opportunity to examine technological alternatives in terms of both hardware and software in the light of recent computer technology and Ada implementation.

TECHNICAL ISSUES

The technical issues to be resolved included computer hardware performance and requirements, software development environment, software tools, Ada compilers/cross-compilers, and software engineering.

System architecture:

One of the guidelines used in the hardware evaluation was the need to use existing minicomputers as well as a special purpose processor for high speed flight dynamics and control simulation. A modular design with minimum system upgrade costs was desired since the demand for computer power invariably increases with enhancements to aircraft capability. To facilitate effective man-machine interface between the pilot and the helicopter, minimum data transport delay between processors is required. The configuration must support implementation of a wide variety of algorithms ranging from artificial intelligence to aircraft electrical and hydraulic system. The system configuration must also permit simulation at rates up to 60 Hz. Most importantly, the hardware cost must be as low as possible. The popular criterion of cost per MIPS was used as a yardstick to determine hardware cost. A wide variety of systems were evaluated including multiprocessors, multicomputers, array processors and pipeline systems. Different vendor products within each group were also evaluated. Based on the above requirements and the implementation of Ada, it was found that distributed processing with multi-processors offered the best cost and technical solution.

Software Development Environment:

While the above resolves the target or the real-time implementation system, it is equally important to address the issue of the software development system for simulation. Developing software in Ada is different from developing software in other languages due to the many aspects of this language discussed earlier. Due to this difference, the development environment becomes an extremely important tool. The necessity to understand how different types of software modules interact with each other in a multi-tasking and generic software system drives the need for software tools which allow graphical/textual representations of design and automatic documentation, as well as appropriate compilers and cross-compilers.

During the preliminary design phase of development automatic software tools for graphical representations of data flows and module constructs such as packages, tasks, and subprograms allow the software designers a standardized way of creating their designs and an excellent method for documenting it in an understandable form. During the detailed design phase, textual representation of the algorithms and variables is more uniformly presented with the aid of Programming Design Languages (PDL) and Data Dictionaries. Many of the commercial PDLs on the market also include templates which produce them in military standard formats as well as supplying automatic metrics. As pointed out earlier, the implementors of the Ada language have been slow to implement all the

features in the Language Reference Manual. It therefore becomes critical to develop the criteria for evaluating compilers and cross-compilers for simulation applications in order to avoid problems at the coding phase. Appendix F of every compiler's manual and Chapter 13 of the Language Reference Manual provide the real-time features which may be necessary for simulation applications.

Once it is realized that software tools become more of a necessity when designing in this language, the question of whether the present development system is adequate becomes very important. It requires analyzing the existing development system, software tool requirements, software development tools that are currently available, and most importantly, if they will work together. If the tools do not work together, then the question is whether the present development system for Ada design and coding should be used.

Ada compilers/cross-compilers:

To be cost effective the real-time (target) system may be quite different from that of the software development system, as was the case at McDonnell Douglas Helicopter Company. In this case it is important to evaluate not only the compilers but also the cross-compilers. These are the most important software tools in the development environment. Designing a list of criteria for necessary and desirable features in an Ada compiler/cross-compiler for a given application can make the job of selecting the compiler/cross-compiler easier. Simulation code is run in real-time and therefore criteria such as ADDRESS CLAUSES (to allow interrupt capability) and pragma INTERFACE to the target systems assembly language may be necessary. The pragma PACK and REPRESENTATION CLAUSES may be added to the list of desirable compiler criteria if transporting of large volumes of data throughout the simulation system is necessary. Criteria may have to be set as to the speed of the compilation time, the run time, or both. Valid data on Ada compilers in this area is difficult to obtain as it is fairly easy for the vendors to skew the results of their tests with the mix of code they use in it. An unbiased source of information may be obtained from the Performance Issues Working Group (PIWG) of the Association for Computing Machinery (ACM). Their data are obtained from ACM volunteers running the test suites that the PIWG develops. However, the compiler being considered may not have been tested within the same machine configuration as that being considered. The PIWG also does not do an analysis on the results of the tests; they just publish the data and let the users draw their own conclusions.

- If a requirement to use a validated compiler does not exist for the application under consideration (as may be the case in an engineering simulation) one could consider a non-validated compiler for development needs. Often non-validated compilers claim faster compilation and execution times than validated compilers. But it is important to examine carefully any compiler that has not been validated or is not planned to be validated. As stated earlier, the validation suites do not test all of the system dependent features needed for simulation applications but they do assure that all aspects of the language which are not system dependent are tested. This may become vitally important for future modification or porting considerations of the developed Ada code. The compilation/execution speed advantages that may be realized initially need to be weighed against the cost of future redesigns or recoding.

Once the compiler criteria has been established, one more list needs to be established: the compromise list. What trade-offs are acceptable in both the compiler and cross-compiler? They may be related to the criteria mentioned above or the associated software tools which are included with the compiler. An automatic recompilation system may outweigh, in relative value, the compilation speed of a compiler, especially for very large applications. Or it may not be possible to get the bit packing desired in the cross-compiler but it does offer a source target code debugging capability. Different vendors are focusing on different aspects of their compiler systems. It is worth the effort to investigate their track records also if a vendor is promising some features which are not required now but are absolutely necessary for use in the future. (Asking for customers' names from vendors for this purpose is an accepted practice). Care should be exercised in the compromise as it is emphasised again that the compiler and cross-compiler are the most important software tools in the simulation development system.

Software Engineering:

Software engineering is the term applied to the activity of creating software in a disciplined and consistent way. Ada's rich set of constructs and capabilities give the software engineers the ability to create a software solution which maps more accurately the problem domain it is addressing than other higher order languages which incorporate operating system calls or low-level assembler routines to effect the same solution. Simulation tackles complex systems and this fact coupled with the enhanced capabilities Ada offers requires a well thought out methodology for designing simulation software.

There are several software development methodologies available. Few, if any, cover all the aspects of designing software from requirements through testing. Most of the software design methods embodied in these methodologies are either top-down structured, data-structure, or object-oriented design. The object-oriented method seems to be emerging as a

favorite although most successful projects seem to be employing a combination of methods.

Another aspect to designing simulation software in Ada is what to do with all the developed FORTRAN code. Three options are possible: incorporate it, convert it, or redesign it in Ada. The first option is dependent on the Ada compiler one chooses. If it supports the pragma INTERFACE to the FORTRAN language, it is possible to use most of that code with minimal rewrites. The second option is the least desirable of the three. It implies purchasing a FORTRAN to Ada software conversion tool. There are a number of them in the commercial market. However, the Ada code which emerges would neither resemble well designed Ada modules nor understandable FORTRAN. The last option is the most desirable since the final product is a simulation code in one language. It may, however,also be the most time consuming and expensive and may not be feasible initially. Whatever option is chosen, incorporation of FORTRAN code needs to be taken into account in the design methodology.

SIMULATION CONFIGURATION

Figure 1 shows the McDonnell Douglas Helicopter Company multi-ship configuration arrived at after reviewing all the issues discussed earlier. As mentioned earlier, the approach was to use a distributed processing system for real-time simulation. The configuration uses existing processors and FORTRAN and other code as they existed. All computing expansion is achieved through microprocessors thus permitting a low cost and very affordable solution. This configuration also permits the main goal of developing all new software in Ada via a development system which is cross-compiled to the target system. An existing super-mini computer with compiler and crosscompiler provide the main software

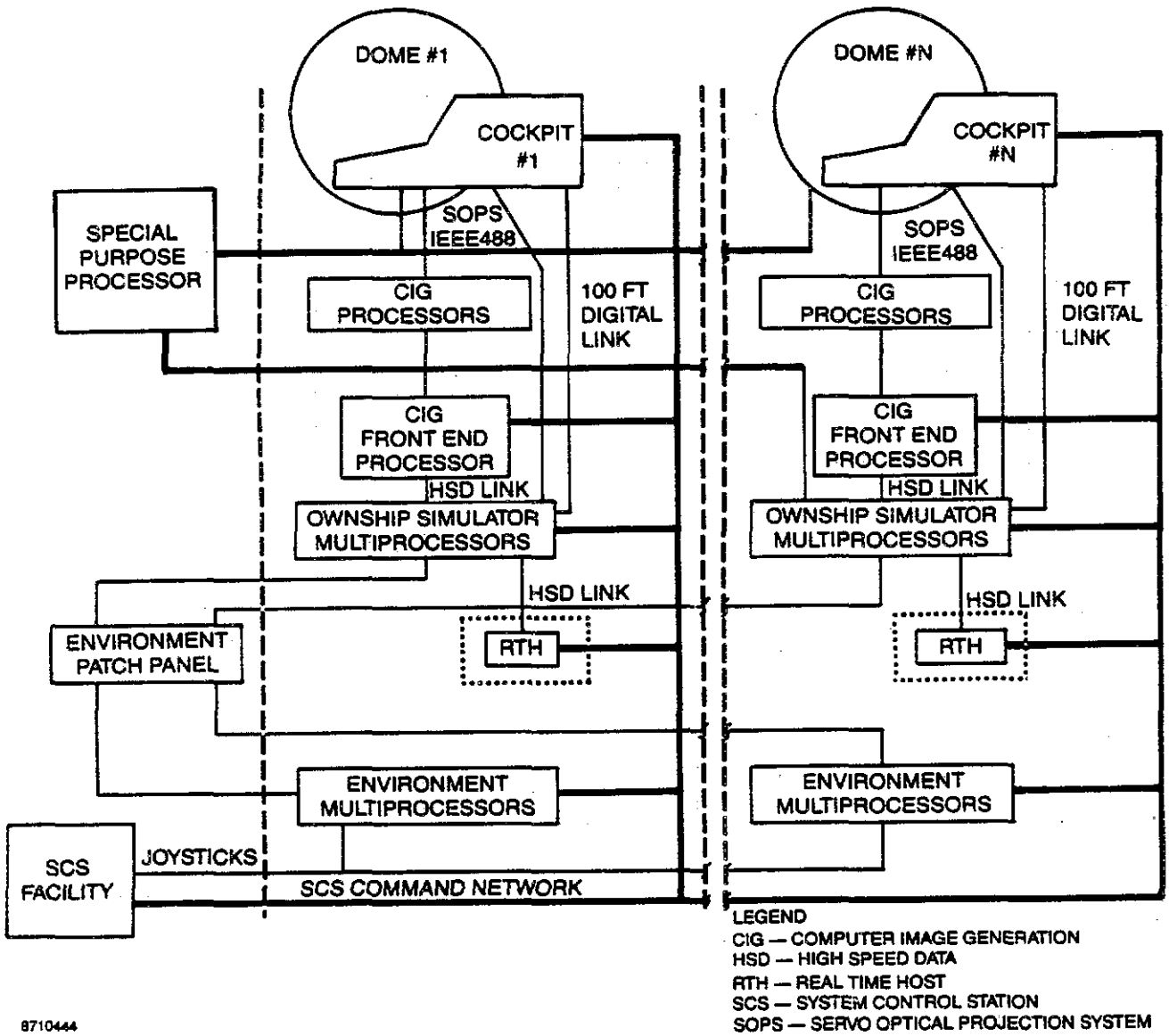

Fig. 1 Simulation Architecture

development capabilities. The configuration also permits gradual redesign of FORTRAN code to Ada and phasing in of processors to execute the redesigned code. PDLs, automatic document generator, and language sensitive editor are the primary software tools. These allow the development configuration to support high software productivity. Most importantly it forces a modular software/hardware approach in the long run.

LESSONS LEARNED

The McDonnell Douglas Helicopter Company simulation department has successfully moved into developing software in Ada, but this was not achieved without some difficulties. There were a number of lessons learned along the way.

Due to the fact that many Ada compiler and cross-compiler implementors are not quite "there" with all the features necessary for simulation software it is important to decide whether Ada will be the simulation language before investment is made into software and hardware for both the development and target systems. If money is invested in target systems before investigating the cross-compilers and debug tools, the advantages expected from a faster and more economical CPU may well be nullified by the fact that there are few, if any, good cross-compilers for the target system that has been purchased. As stated previously, development systems need enhanced capabilities when developing with Ada. Vendors who supply the software packages which allow this enhancement have not made these packages for every operating system. The development system and the software packages have to be considered as a whole and not in piece-meal. It is nearly impossible to retrofit software packages to any existing machine.

Designing software in Ada needs more training than that which is offered by the compiler vendors as part of the product they sell. Just knowing the syntax of the language is not enough. As stated earlier, a methodology for design does need to accompany the learning of Ada. Otherwise the price paid is creation of non-reuseable code if design issues are ignored in developing simulation Ada software. Unfortunately just selecting a methodology may not be enough. The methodology chosen may not fit the type of software that is being developed. One approach to this dilemma is to evaluate the PDL or code which is being produced by this methodology as soon as possible. If it is too complex, try another methodology or modify the existing one.

In the area of compilers and cross-compilers, a number of lessons were learned. On the issue of a validated versus a non-validated compiler, there is no question - use a validated compiler! This does not guarantee all the capabilities needed for simulation applications but it does mean the compiler has been extensively tested by validation suites and there will be fewer problems than with a non-validated compiler. The next lesson is that validation by itself is not enough. The buyer must be aware of what implementation dependent features of the LRM is needed for their applications. Knowing Chapter 13 and being able to understand Appendix F of a compiler vendor's manual is very helpful. The final lesson is to determine what compromises are acceptable, since it may not be possible to get all the features desired in a compiler. In the same vein, it is helpful to consider which vendor is making progress in the areas of capability we require, even if they do not have these capablities now.

CONCLUSION

This paper has reviewed the issues in implementing Ada in a multi-ship simulation organization. The issues were discussed in the light of MDHC's experience in achieving the transition to ADA. The advantages and disadvantages of Ada were examined. Technical issues such as hardware performance and requirements, software development environment, software tools, Ada compilers and cross-compilers, and software engineering were considered. MDHC's simulation configuration was discussed and lessons learned were presented. It was pointed out that the most important aspect is that all these issues must be examined in totality before any commitment is made to purchase hardware or software.

ABOUT THE AUTHORS

Dr. Ramachandran is a Senior Staff Engineer with the Engineering and Training Simulation Department. He has been involved in the setting up of the new Simulation Laboratory and the Apache and LHX simulator programs. His simulation experience covers a wide range of aircraft, helicopters and missiles.

His previous experience includes Goodyear Aerospace, Dynamics Research Corporation and NASA Langley Research Center. He has authored over twenty articles in the fields of simulation, aerodynamics and control systems. He received his B. Tech from the Indian Institute of Technology and M.S. and Ph.D. from the University of Cincinnati.

Mr. McCabe is presently a Systems Engineer with the Simulation Department and is involved in Ada Implementation for Simulation. He received his BSEE and MSEE from Arizona State University. He has many years of computer hardware and software experience.