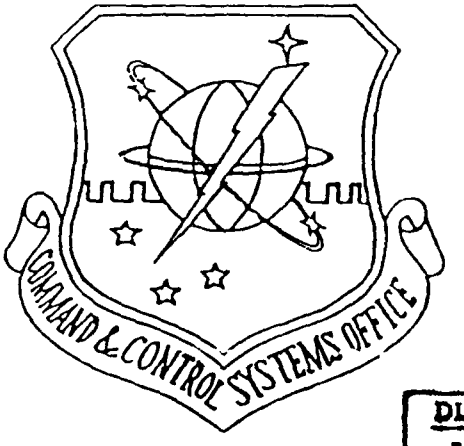
ADA* EVALUATION PROJECT

USAGE AND SELECTION OF ADA* MICROCOMPUTER COMPILERS

Prepared for

HEADQUARTERS UNITED STATES AIR FORCE
Assistant Chief of Staff of Systems for Command, Control, Communications, and Computers
Technology & Security Division



Prepared by
Standard Automated Remote to AUTODIN Host (SARAH) Branch
COMMAND AND CONTROL SYSTEMS OFFICE (CCSO)
Tinker Air Force Base
Oklahoma City, OK 73145-6340
COMMERCIAL (405) 734-2457/5152
AUTOVON 884-2457/5152

* Ada is a registered trademark of the U.S. Government (Ada Joint Program Office)

9 December 1986

THIS REPORT IS THE FOURTH OF A SERIES WHICH DOCUMENT THE LESSONS LEARNED IN THE USE OF ADA IN A COMMUNICATIONS ENVIRONMENT.

ABSTRACT

This paper discusses Ada microcomputer compilers and seeks to provide information to aid in successful selection of compilers for Ada software development. Selection of Ada compilers is significantly more complicated than selecting compilers for older programming languages. Effective requirements analysis and selection of Ada compilers will have a large bearing on how well project teams are able to develop Ada software products and how effectively the products will execute.

After an introduction in Section 1, the second section of this paper describes the benefits of using microcomputers to develop Ada software. Case studies are provided to give examples of how microcomputers can be integrated into a software production environment.

Choosing an Ada compiler is the topic of the third section. Specific criteria are highlighted and references are provided to publications which provide more detailed selection guidelines. Section 4 then discusses some of the currently available compilers.

Some of the current limitations and problems of Ada compilers are discussed in Section 5. Topics such as incremental compilation, distributed library systems, and low level features are covered so that prospective Ada developers are aware of the types of things that they should look for in future Ada compilers.

Section 6, on experiences and performance issues, provides information on many of the most commonly asked questions. The information presented refers mainly to the AlsyCOMP_003 compiler for the IBM PC-AT and compatible computers.

The final section deals with future directions, addressing issues such as compilation rates, optimization, and compiler environment integration.

Ada Evaluation Report Series by CCSO

Ada Training	March 13, 1986
Design Issues	May 21, 1986
Security	May 23, 1986
Micro Compilers	December 9, 1986
Ada Environments	December 9, 1986
Transportability	Winter 86-87
Modifiability	Winter 86-87
Runtime Execution	Winter 86-87
Module Reuse	Spring 87
Testing	Spring 87
Project Management	Spring 87
Summary	Fall 87

TABLE OF CONTENTS

1. INTRODUCTION

1.1. THE ADA EVALUATION TASK

This paper is one in a series which seeks to help potential Ada developers gain practical insight into what is required to successfully develop Ada software. With this goal in mind, Air Staff tasked the Command and Control Systems Office (CCSO) to evaluate the Ada language while developing real-time communications software. The task involves writing papers on various aspects of Ada development such as training, Ada design, environments and security issues. This paper discusses selection and usage of microcomputer compilers.

CCSO chose the Standard Automated Remote to AUTODIN (Automatic Digital Network) Host (SARAH)[1] project as the vehicle basis for the Ada evaluation. SARAH is a small to medium size project (approx. 40,000 lines of source code) which will function as a standard intelligent terminal for AUTODIN users and will be used to help eliminate punched cards and paper tape as a transmit/receive medium. The development environment for SARAH consists of a number of IBM PC AT and Zenith Z150 microcomputers, and a Digital Equipment Corporation (DEC) VAX 11/780. The source code produced is compiled on the PC ATs using Alsys Ada compilers and the object code is targeted to both the Z150 and PC AT. The VAX was intended to be used for configuartion management support and to function as a repository for reusable Ada code. The SARAH software will run on a range of PC XT, PC AT, and compatible microcomputers under the MSDOS operating system (version 2.0 or higher).

1.2. BACKGROUND

The dramatic increase in the power of microcomputers, particularly personal computers such as the IBM PC AT, has made it possible to use these machines as effective Ada development workstations. Ada is resource intensive. For example, the compilers themselves generally consist of more than 300,000 lines of complex source code. In addition, because of the strict inter-module checking, Ada compilers do not currently support program overlays and so applications generally require a large amount of memory. But, the future is bright for Ada in terms of machines which will be able to support Ada development. Systems are now beginning to appear that use the new 80386 microprocessor. These processors support up to 4 Gigabytes of physical memory and 64 Terabytes of virtual memory[2]. Moreover, the 80386 executes at four Million Instructions Per Second (MIPS), and this equates to the power of a Digital Equipment Corporation VAX 11/780 minicomputer. This is indeed a dramatic increase in power over the older Personal Computers (PCs). For example, the IBM PC XT executes at only 333,000 instructions per second and is generally limited to 640 Kilobytes of physical memory.

The flexibility, power, and cost of PC workstations make them good candidates for Ada development. In the future, these workstations will play a large role in Ada production environments. By effective networking, together with a system of distributed source and library databases, micros can dramatically improve development throughput. Individual workstations can support a number of general purpose and Ada specific productivity tools. A large number of these tools already exist for PC environments and demand is encouraging the development of many more. Ultimately, these tools will be integrated into a very powerful Ada development environment together with the compiler.

As more micro compilers become available, compiler requirements analysis and selection will become an important step in the procurement and development processes. In addition to ensuring that an Ada compiler can produce good quality code and compile programs quickly, the prospective user must have some knowledge of the compiler's targeting capabilities, how well the compiler will integrate into the development environment, and the effectiveness of the tasking model implementation. To assess quality and performance, benchmarking and benchmark reports are an important part of requirements analysis and selection. Effective requirements analysis and selection of an Ada compiler play an important part in determining the success of Ada projects.

1.3. PURPOSE

The purpose of this paper is to:

- Provide information on the benefits of using microcomputers for Ada software development.
- Discuss important considerations for choosing microcomputer compilers.
- Provide details on currently available microcomputer compilers.
- Highlight some of the limitations, performance issues, and problems that have been experienced with microcomputer compilers.
- Provide information on what to look for in future compilers.

1.4. SCOPE AND CONSTRAINTS

This paper discusses various aspects of microcomputer compilers; however, more emphasis is given to Personal Computer (PC) environments such as the IBM PC AT. This has been done for a number of reasons. First, the SARAH team has received a large

number of enquiries regarding PC compilers and so we feel that a discussion of this type would seem appropriate at this time. Secondly, there is a large range of computers, loosely termed microcomputers, that host Ada compilers. A discussion of each of these environments and their respective compilers would be beyond the scope of this type of paper. Readers should refer to the various benchmarking reports[3] to gain additional information on these systems. Third, we see the PC workstation as an important tool for Ada development because of its low cost, flexibility, and versatility. As such we encourage the use of PCs and the development of Ada PC tools.

2. MICROCOMPUTERS FOR ADA DEVELOPMENT

Microcomputers can provide cheap and versatile Ada workstations. Also, an Ada development environment consisting of microcomputers is more survivable than an environment hosted on a single machine. For example, problems with one workstation will not severely affect overall production. SARAH team members have found that working with individual workstations has improved their productivity because they have flexibility in choosing and configuring the development tools for their particular system and they have the full power of the compiler when it is needed.

2.1. ENHANCED PRODUCTIVITY

Development productivity can be enhanced through the use of workstation compilers. One of the major problems with current Ada compilers is that they require a significant amount of computer resources. For example, on a VAX11/780, current Ada compilers can support a maximum of three to five concurrent compilations without severe performance degradation (i.e. compilation rates drop below usable limits). Some compilers, for example the SofTech Ada Language System (ALS), severely degradate with more than two simultaneous compilations. By using a network of microcomputers, the compilation load can be shared by a number of machines. Moreover, if the project team expands, additional machines can be introduced at a relatively low cost so that team productivity can be maintained.

2.2. TOOLS FOR PRODUCTIVITY

The PC compiler is only one tool which can enhance software development: the PC environment will support many general and Ada specific tools to help in the production of high quality Ada software. The SARAH team uses several general purpose tools to aid in Ada software production. For example, various text editors and tools such as Borland's Sidekick and SuperKey are used to support development. In addition to the general purpose tools, there are now several Ada specific tools available for PCs. For example, Xinotech Research, INC. markets the Xinotech Program Composer which is an extremely powerful syntactical Ada text editor. Other tools include: an on-line Ada encyclopedia from Tacyon; the AdaGraph design and development system from The Analytic Sciences Corporation; and Sketcher, an interactive Graphical Ada software design tool from SYSCON. The PC provides the basis for an extremely powerful Ada development environment but the full benefits will not be realized unless the tools work together in an integrated manner.

2.3. DISTRIBUTED DEVELOPMENT ENVIRONMENTS

Networking is required if microcomputers are to be used effectively in a development environment. Careful consideration must be given to how micros are used for software development, otherwise serious configuration control and productivity problems will arise. For example, without a network, team members are more inclined to maintain their own versions of the modules they are developing rather than constantly updating a central project library. To use micros effectively, a distributed environment which maintains a distributed source data base and distributed library system should be installed. The file server should host configuration management tools and allow for target code production if embedded applications are being developed.

The development environment used by Alsys Inc. of Waltham MA provides a case study of how micros can be used for Ada software development. Alsys uses a scheme where each engineer on the compiler project has a PC AT and these are connected to a VAX 11/750 via ethernet[4]. The network has an effective data transmission rate of about one megabyte per minute. Tools such as the VAX DEC/Code Management System (CMS) provide functions such as configuration management. The VAX also provides a repository of Ada source code. The network cost Alsys approximately $6000 for VAX enhancements and $1000 for each PC. The proliferation of networks for PCs has reduced costs significantly and as prices continue to fall, the total cost of installing this type of network will be far less than the quoted figures.

Another source of information on using micros for Ada software production is the Software Engineering Institute (SEI). The SEI will provide information on distributed development environments as a result of their Showcase Environment project[5]. Their goal is to implement a software development environment to support the construction of software systems consisting of up to one million lines of source code. The SEI intends to exploit the capabilities of workstations with high resolution graphics displays. These workstations will be supported by local area networks. The results of the work being done by the SEI on the Showcase Environment could prove beneficial for organizations who are contemplating installing a distributed Ada development environment.

CCSO attempted to implement an Ada development environment consisting of a number of PCs and a VAX 11/780: however, the result was not particularly successful. One of the major reasons was that no local area networking facilities were available. As such, the time required to upload and download data was excessive. In addition, the communications package being used was not particularly friendly and so team members were reluctant to use the system. Another problem was that the SofTech Ada Language System (ALS) was to provide the automated configuration management support for the project. The tools provided by the ALS, were good in concept, but some were unreliable and not

particularly easy to use. If PCs are to be used in an Ada development environment, then thought needs to be given to networking and effective tools need to be secured to aid in source code management.

3. CHOOSING A MICRO COMPILER

An effective requirements analysis for the selection of micro Ada compilers will become ever more important as the number of available compilers increases. This section discusses selection criteria for Ada compilers. Ada compilers are very complex pieces of software and are intended to be used in an Ada Programming Support Environment (APSE). As such, many parameters must be considered. In addition to assessing compiler requirements, prospective users must assess quality and effectiveness for Ada software production. Several methods can be used to assess these criteria. For example, this section discusses the use of benchmarking, benchmark test results, hands on experience, and discussions with users to aid in compiler selection. Several other important considerations such as runtime/executive licenses and vendor supplied packages are also discussed. Appropriate selection of Ada compilers is an important step for successful development of Ada software products.

3.1. SELECTION CRITERIA

The selection criteria for Ada compilers is more complex than that generally applied to the selection of compilers for older languages. Wallis and Wichmann[5] indicate that "..ideally, a compiler should compile the language defined in the language standard, produce good quality code, compile programs quickly and give good diagnostics when needed. These are indeed requirements for a good compiler, be it Pascal, Fortran, or Ada. But, for Ada, several other criteria also need to be considered. For example, one also needs to know which targets are supported, how well the compiler fits into the development environment, and how effectively the compiler supports tasking and low level features.

A list of essential parameters for compiler selection has been generated by an Ada-Europe working group[6]. This guide is written from the point of view of someone wishing to know something about an existing compiler and indicates what information may be needed from the compiler supplier. A check list is provided for each criteria and the guide is very easy to follow. Anyone considering the selection of an Ada compiler would be well advised to look at this guide.

An important criteria for the selection of an Ada compiler is: How well will the compiler integrate into the working environment? There are many things that need to be considered. For example, does the compiler produce intermediate program representations (e.g. DIANA (Descriptive Intermediate Attributed Notation for Ada) code). Is there a human readable form of this code? Will other tools such as debuggers be able to use this representation? Can other general purpose tools be invoked when the compiler is loaded? The SARAH team experienced the integration problem first hand. When the Alsys compiler (Version

1.0) was procured for the PC ATs, everyone was dismayed by the fact that tools such as Sidekick and Super Key would not run when the compiler was loaded. Team members had been using these tools previously and found that they significantly improved the effectiveness of the workstations. Thankfully Alsys was attune to our needs and when Version 1.2 of the compiler was released, we were once again able to use our favorite tools. The problems of incompatibility will be more pronounced when more Ada specific tools are released. Effective integration of Ada tools for microcomputers is an important consideration.

Target support is another criteria that must be considered when selecting a micro Ada compiler. For example, the Alsys compiler targets to the PC AT and PC XT (and selected compatible computers) running under the MSDOS operating system. The OASYS compiler is also hosted on the PC AT but it will not target code to run under MSDOS. The two compilers run on the same host but are completely different and are used for different reasons: the Alsys produces MSDOS applications software, whereas the OASYS will be used for producing embedded software applications. Care must be taken to select the right compiler for the job.

3.2. VALIDATION

Validation does not ensure quality! This is perhaps one of the most important things to remember when procuring an Ada compiler. Validation simply ensures that the compiler conforms to the language definition and is only a small part the overall compiler specification[3]. Even though a compiler is validated, it may have unacceptable compilation rates, be unreliable, and may produce inefficient code. The Ada Validation Office (AVO)[7] checks to see if the compiler conforms to the language standard but does not attempt to provide any information on quality or effectiveness. As such, the prospective user must resort to other methods, such as benchmarking, to ensure that the compiler will indeed meet requirements.

3.3. BENCHMARKING

Benchmarking plays a big part in determining the quality and effectiveness of compilers. As discussed, validation ensures that the compiler conforms to the language standard but says nothing about quality or efficiency. Benchmarking and benchmark test results can provide comparative data that can be used to assess the performance of compilers.

Copies of benchmark test results are now becoming available and can be extremely useful for compiler selection. A big benefit of getting benchmark test results is that the testing is generally done in conjunction with a test methodology and so comparative studies are generally more accurate than if the tests were applied by inexperienced personnel. The SEI has recently published benchmarks for several environments[8]. This document also outlines the methodology that was used for the tests.

Hopefully, the SEI will continue to produce these high quality reports. The Special Interest Group on Ada (SIGAda) Performance Issues Working Group is also compiling benchmark test results. These results will be made available to SIGAda members.

The Ada Compiler Evaluation Capability (ACEC) will be a very effective set of benchmarking tools for determining the quality and effectiveness of various compilers. The prototype ACEC became available from the Ada Validation Facility in January 1986. The benchmark tests were collected and organized into this prototype suite by the Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) team. To get a copy of the prototype test suite a request can be submitted to:

The Ada Validation Facility
ADS/SIOL
Wright Patterson AFB
OH 45433-6503

The request must be on a company letterhead and should be accompanied by a 2400 foot magnetic tape (tape format information must be provided). Additional tests and analysis tools will be available in the ACEC when it becomes available.

Benchmarking gives you a chance to get "hands-on" experience with the compiler before purchase. This is important since with "hands-on" the user interface and general usability criteria can be assessed. Even if test results are used to assess whether a compiler will meet project requirements, organizations should attempt to use the compiler before purchase. The benchmark programs can be used for gaining this "hands-on" experience. By using the compiler, prospective users can get a good idea of reliability, the quality of user documentation, and general ease of use.

3.4. DISCUSSIONS WITH USERS

One of the best ways to determine how well a particular compiler performs or whether it is suitable for your needs is to talk to experienced users. The SIGAda meetings provide an excellent forum to engage in these discussions. Three national SIGAda meetings are held each year and local groups generally have regular meetings. Most of the larger vendors also support users' groups for their products. Attendance at these meetings provides a good insight into the quality and effectiveness of particular products. For more information on SIGAda meetings contact:

Ada Information Clearinghouse
[illegible] (1211 S. Fern, C-107)
The Pentagon
Washington D.C. 20301-3081
Ph. (703) 685-1477

The Ada Information Clearinghouse can also provide information on currently validated compilers and other Ada related information.

3.5. RUNTIME/EXECUTIVE LICENSES

When selecting a compiler remember to check the license agreements and contracts carefully. Some compiler vendors are charging royalties on the runtime package. For example, Alsys provides 10 free executive licenses with each compiler. This means that if the application being developed will be used on more than 10 systems or if the software is to be sold to more than 10 people, then license fees must be paid to Alsys. The fees vary with the number of times the runtime software will be used. For numbers between 11 and 1000 Alsys must be paid $50 per copy. This reduces to $5 per copy for numbers ranging from 1001 through 100,000. As more compilers become available, organizations would be well advised to stay clear of compilers that have this type of licensing scheme.

3.6. VENDOR SUPPLIED PACKAGES

A major benefit of Ada is that the language is standardized and strictly controlled by validation and is not tied to individual vendors or machines But, is this entirely correct? Several vendors are providing additional packages that aid the software developer but these packages take the form of an Ada pre-defined package (i.e. no source code is provided). Alsys provides two such packages: 'Unsigned' and 'DOS'. The DOS package provides a number of functions that make calls to the MSDOS operating system. Features such as buffered keyboard input, file input/output, and absolute disk access are provided. Without doubt, these features are necessary in many applications and the Alsys package can save development time. However, since source code is not provided, the applications software cannot be compiled with another Ada compiler unless a DOS package similar to the Alsys package is developed. In a sense, if a vendor's pre-defined packages are used, the applications developer is tying the application to a particular compiler vendor. As stated earlier this can be dangerous.

4. CURRENTLY AVAILABLE MICRO COMPILERS

The number of validated Ada compilers is increasing at a fast rate. One of the major reasons for the rapid increase is that Ada compilers are easily rehosted and so a "base" compiler can run on a large number of different machines. As such, many of the compilers that run on larger machines will also run on micros. This section discusses some of the currently available compilers, provides references for obtaining more comprehensive and up-to-date lists of validated compilers, and provides information on some of the PC compilers currently under development.

4.1. REHOSTING

Since most Ada compilers are themselves written in Ada, rehosting to other environments has not been as difficult as has been the case with older languages. As such, many of the compilers that were developed on larger machines have been rehosted to microcomputers. For example, several compilers and environments have been rehosted to the MicroVAX. Some of the compilers currently available for the MicroVAX are the DEC VAXAda, the SofTech ALS, and TELESOFT's TeleGEN2. Another very popular micro based workstation is the Sun Microsystem. Organizations such as Telesoft, Verdix, Alsys, TeleLOGIC, and New York University have rehosted their compilers to this environment.

4.2. COMPILER INFORMATION

The Ada Information Clearinghouse (AdaIC) is one of the best sources to find out which compilers are currently validated and which vendors have indicated their intention to validate. Current validation lists are provided in each copy of the AdaIC newsletter. The lists provide information on the vendor and compiler, the host machine, and target machines. In addition AdaIC maintains an Ada Language Implementations Matrix[9] which shows all validated compilers as well as compilers that are under development or waiting validation.

4.3. PC COMPILERS

There is still not a large range of validated compilers for PCs. One of the major problems has been that the PC environments have not been powerful enough to accommodate an Ada compiler. For example, most PC systems have been limited to 640 Kilobytes of memory and execution rates have typically been less than 1 MIPS. The IBM PC AT now provides an adequate environment to support Ada development, and future machines, based on the 80386 microprocessor or equivalent, will allow vendors to effectively rehost their compilers or develop specific PC compilers. Since PCs are cheap and provide an environment that has become an

industry standard, PC compilers will be an important and lucrative market for vendors.

Two compilers that have been validated on the IBM PC AT and compatibles are the Alsys AlsyCOMP_003 and the OASYS PC Platform. These products are very different in concept. The Alsys compiler runs under the MSDOS operating system and directly uses the PC hardware. To run the compiler, the PC AT memory must be expanded to 4 Megabytes. Alsys markets their compiler with a 4 Megabyte memory board. Object code generated by the Alsys compiler must run under MSDOS and can use the protected memory mode of the 80286 microprocessor in the PC AT. Applications are therefore limited only to the maximum memory that the 80286 can support (16 Megabytes). Object code can also be targeted to the 8088 based PC XT but the application must not exceed 640 kilobytes.

The OASYS PC Platform does not exclusively use the PC hardware for operation. To use the OASYS system, a processor card must be installed and this card hosts the compiler. The platform card contains a 32 bit NS32032 microprocessor operating at 12.5 Mega Hertz and up to 16 Megabytes of memory. As such, the PC itself is only used as an input/output processor for the installed board which gains control of the entire system. The compiler that is hosted by the PC platform is the Verdix Ada Development System (VADS) and runs under the UNIX V.2 operating system. A major difference between the Alsys and OASYS compiler is that the OASYS does not target to the MSDOS environment and will not produce code to run on the host machine. One of the advantages of the OASYS compiler is that all the VADS embedded targets will be supported. This will include target support for the 1750A, 68000, NS32032, and a range of Intel processors. In summary, the OASYS system does not currently target applications to run under MSDOS in the host configuration and uses a separate processor board for operation. The Alsys compiler is a true PC AT compiler but does not support targets other than the PC AT and PC XT; applications must run under MSDOS.

Other organizations that have indicated their intention to validate on PCs are ARTEK, JANUS, General Transformations, New York University, and General Systems [9]. Many organizations attempting to develop low cost Ada compilers for PCs have failed because they have underestimated the complexity of the language and the computer resources necessary to run the compilers. For example, the Alsys PC AT compilers consists of approximately 300,000 lines of source code and requires 6,161 Kilobytes of disk storage to load. The requirements for inter-module checking, run-time error checking, and concurrent processing place a heavy burden on developers and host architectures. However, continued compiler research, more experience, and more powerful PCs will result in many more Ada compilers for PCs. As more compilers become available, costs should also decrease.

5. COMPILER LIMITATIONS AND PROBLEMS

Many of the limitations and problems discussed in this section are not peculiar to micro compilers. Indeed, the problems affect a very large number of currently available Ada compilers. A discussion of problems and limitations are provided in this paper to increase general awareness and to help in successful compiler requirements analysis and selection. Some of the limitations and problems discussed in this section are: the lack of incremental compilation, the need for integrated PC environments, the lack of target support, the need for distributed library systems, and support for low level features.

5.1. INCREMENTAL COMPILATION

Incremental compilation allows the user to make changes to a module without having to recompile all the dependent modules. The lack of incremental compilation can result in a huge recompilation overhead. This is because the dependencies built into Ada for inter-module consistency checking can cause even insignificant changes to result in a propagation of recompilations. Ada compilers need to support interactive changes. One way to support interactive changes without incurring excessive delays is to incorporate the changes in some rich data object such as a DIANA tree which preserves syntactic and semantic information rather than in simple ASCII text files.

Of the compilers currently validated, only the Rational supports incremental compilation. Hopefully, as the state-of-the-practice for Ada compiler design improves, necessary features such as incremental compilation will become common-place, even on microcomputers.

5.2. INTEGRATED ENVIRONMENTS FOR PCs

The lack of an integrated toolset/environment is a serious limitation to the Ada developer. A great deal of effort has been expended in defining Ada Programming Support Environments (APSE)[10] and tool interfaces (Common APSE Interface Set)[11] but the recommendations may not be entirely sufficient for distributed Ada development environments consisting of a large number of micros. Indeed, organizations producing Ada tools for PCs (including compiler vendors) are paying little attention to any standard tool interfaces.

Although there are several Ada specific tools now available for the PC and many more currently under development, little thought has been given to integration. If the tools are to be truly effective in a PC environment, then they must be integrated. For example, users should be able to move easily between the editor, compiler and debug facilities. This should be accomplished through some modern user interface (perhaps consisting of a system of icons and windows). Also, standard interfaces should

be provided for distributed source databases and compilation libraries. Interfaces for local area network communications should also be well defined so that the power of networking can be fully realized. Powerful Ada development tools for PCs are beginning to emerge; however, a standard method of integrating the tools is yet to be defined.

5.3. LIMITED TARGETS

The Ada software industry is drastically in need of compilers which target to a range of embedded computers. The Ada language was primarily designed to support the development of software for embedded systems. Yet, to date there are very few compilers that target to a range of embedded computers. Microcomputers such as the DEC MicroVAX support a number of products that will target to embedded systems; however, the range is extremely limited. For example, until recently, if an embedded application was required for an 8086 based embedded system, then the only compiler that could be used for development was the ALS.

If a PC network is to be used for development of embedded applications then some thought must be given to how the targeting will be accomplished. Compilers such as the one produced by Alsys are excellent for developing source code and general debugging, but do not target to a range of embedded systems. To provide the target code, a machine must be selected which will host a compiler that targets to the embedded system under development. For example, the Navy standard embedded architecture is the UYK 43 and the Navy specifies the ALS/N will be used to target to this system. As such, one machine in the development environment must support the ALS/N (perhaps a MicroVAX). PCs (using PC compilers) could be used in a networked environment to increase productivity during the development and debugging of source code, but ultimately the machine hosting the embedded target would be required for the production of target code.

5.4. DISTRIBUTED LIBRARY SYSTEMS

A serious limitation associated with using PCs for development is the lack of a distributed library system. A distributed library allows team members to compile parts of the system on their respective computers and to then test these elements without having to re-compile on a central computer. Individual libraries containing parts of the overall system can therefore reside on separate machines and the compilation overhead is more evenly distributed.

Without a distributed library, a central library must be maintained for integration and test. Completed source modules must be copied to the central computer, and compiled into a central library. The benefits of using a LAN without some type of distributed library are eroded because of the large amount of

file transferring that must take place.

5.5. LOW LEVEL FEATURES

Current compilers lack many of the low level features which are an integral part of the Ada language. These features are listed in Chapter 13 of the Reference Manual for the Ada Programming Language[7]. Some of the features that are generally missing are: support for interrupts, representational clauses, and address clauses.

Lack of low level features is a major problem because developers must resort to using assembly language to provide the machine interface. Even though these routines consist of only a few lines of code, they create problems for ongoing maintenance and configuration management. For example, in SARAH, a number of small assembly routines are required for functions such as the the low level communications driver and keyboard driver. This has created a problem for a number of reasons. First, the SARAH team, although well versed in Ada, does not have very much experience with 8088/80286 assembly language. Second, integrating the routines into the software complicates the integration process.

Each compiler vendor must provide an Appendix F to the Reference Manual for the Ada Programming Language which must specify all implementation dependent characteristics of the compiler. The Appendix F will indicate whether or not the low level features are supported.

6. EXPERIENCES AND PERFORMANCE ISSUES

The experiences reported in this section relate mainly to the Alsys PC AT compiler (AlsyCOMP_003) Versions 1.0 and 1.2. Issues such as code efficiency, the size of the runtime support code, and the effectiveness of the task model implementation are discussed.

6.1. CODE EFFICIENCY

The Alsys compiler can produce code that is as efficient, if not more efficient, as that produced by the better PC compilers for other languages. Benchmark tests[4] compared the Alsys compiler to the Lattice C (Revision 2.15) compiler and Turbo Pascal Version 2.00. The results are very encouraging and show that Ada compilers can in fact produce efficient code. In many tests the Ada compiler outperformed the other compilers. The SARAH team was particularly interested to see how well the Alsys compiler implemented procedure calls. SARAH was designed using modern software engineering practices and so consists of many small functions and procedures (or tools). If the application was to operate effectively, then the overhead associated with procedure calls needed to be minimal. The Ackermann function[12] is a good indicator of how effectively calls are implemented; the benchmark tests showed that the Alsys implementation is considerably more efficient than either Turbo Pascal or Lattice C.

6.2. SIZE OF RUNTIME SUPPORT

The runtime support module produced by the Ada compiler is large; however, with care, the size of the runtime can be minimized for a particular application. Experiments were done to see how large the overhead would be when the Alsys compiler was used in different modes and when different predefined packages were used. Compiling a simple procedure consisting of a single "null" statement created an executable file of 15,313 bytes. If tasking was added to the runtime, the file size increased to 39,953 bytes. Compiling the same simple procedure, without tasking, but specifying protected (or extended) memory mode produced a file of 40,545 bytes. These results show that the overhead for tasking and protected memory can be quite large, so only those modes that are required should be specified.

In addition to the runtime overhead, use of predefined packages can add to the size of executable files. For example, when the simple procedure was compiled with TEXT_IO (a package providing standard text input/output routines), the executable file increased in size from 15,313 to 41,953 bytes. If only one or two functions are to be used from TEXT_IO, then this is a large price to pay, particularly if the target machine has a constrained memory size. Since SARAH is to be targeted to a Zenith Z150 microcomputer with only 640 Kilobytes of memory, a decision was

made not to use TEXT_IO. Other predefined packages add considerably to the size of the executable file but nowhere near as much as TEXT_IO. For example, SEQUENTIAL_IO increased the size of the executable file from 15,313 to 22,177 bytes while DIRECT_IO increases the size to 23,249 bytes.

If memory size is constrained then care must be taken to minimize the runtime and pre-defined package overhead. To see how large the overhead could become for a typical application, the simple "null" procedure was compiled with TEXT_IO in the tasking and protected modes. The size of the executable file was 87,377 bytes. That is a big memory overhead just to execute a "null" statement. By adding a package which instantiated DIRECT_IO and SEQUENTIAL_IO, the size of the executable rose to 94,657 bytes. With Ada, the executable file can be quite large, even if the actual work element is small. Care must be taken to tailor compilation and use of predefined packages to meet the need of the application, or an unnecessary overhead will result.

6.3. RENDEZVOUS TIMES

Since rendezvous times for Ada task communications are still in the low millisecond range, designers must be careful not to incur too much of an overhead because of task rendezvous. Exact rendezvous figures are not available for the Alsys compiler; however, discussions with Alsys and other users indicate that that times are between three and 10 milliseconds. Until rendezvous times become acceptable (at least within the mid microsecond range) designers should look towards using procedures instead of tasks wherever possible. The Process Abstraction Methodology for Embedded Large Applications (PAMELA)[13] has specific criteria for deciding when to use tasks and procedures. Several other schemes can be used to reduce the effect of poor rendezvous times. For example, the Modular Approach for Software Construction, Operation and Testing (MASCOT3)[14] uses control queues and pools to get around the rendezvous problem.

6.4. TASK SCHEDULING

Currently, the Alsys compiler does not have timeslicing implemented for task scheduling. As such, an application which has tasks may not execute as expected. For example, Figure 6.1 contains a simple tasking application where both tasks are of equal priority. One would think that when the program executed, we would get a few lines of "Task one active.." followed by a few lines of "Task two active.." and so on, depending on the duration of the time slice. What in fact happens with the Alsys compiler is that only one of the messages will be printed over and over again. The problem is that there is no timeslicing implemented for task scheduling and so the compiler performs scheduling only at task synchronization points (i.e. at rendezvous points, at the end of a delay, and at task activation). As such, one task will execute until it becomes unrunable, reaches a synchronization

point, or a task with a higher priority becomes runable. Normal priority rules are followed for pre-emption and the priority values are in the range 1..10.

6.5. QUALITY

Many of the Ada products currently available are of dubious quality; however, this certainly does not apply to the ALSYS PC AT compiler. The Alsys compiler is a very reliable and robust piece of software. Very few problems have been found with the compiler and the standard of documentation is extremely high. The compiler is very easy to use and comprehensive on-line help is available. The user interface follows the conventional compile, bind, and execute cycle and, although easy to use, becomes a little tedious after a while. Alsys should look at providing a more integrated approach such as that provided by Borland with Turbo Pascal. The diagnostics provided by the compiler are effective and help pinpoint many problems. In summary, the Alsys PC AT compiler is an excellent product.

This does not mean that the compiler cannot be used for concurrent applications. However, developers need to be cognizant of this limitation so that unexpected problems do not adversely affect the development schedule.

```
with TEXT_IO;
use TEXT_IO;

procedure TASK_TEST is -- a procedure to check for task scheduling

   task Task_One;

   task Task_Two;


   task body Task_One is

      begin

         loop  --forever
            Put_Line ("Task one active..");
         end loop;

   end Task_One;


   task body Task_Two is

      begin

         loop  --forever
            Put_Line ("Task two active..");
         end loop;

   end Task_Two;


   begin --activate tasks

      null;

end Task_Test;
```

FIGURE 6.1 Task Scheduling Example

7. FUTURE DIRECTIONS

The micro compilers that are currently available are sufficient for developing Ada software. As illustrated, there are still many areas where improvements are required. This section provides information on the features that will be available in future generations of Ada compilers.

The next generation of Ada compilers will provide faster compilation. But how fast? Dr Robert Dewar from New York University (NYU) indicated during a session at the Pittsburgh SIGAda meeting (held in July 1986) that NYU had developed software for a PC AT which will perform syntax checks at about 120,000 lines of Ada source code per minute. From this result and other research, he believes that Ada compilers should be able to compile at a rate of 20,000 lines per minute. Indeed, this is big improvement over current compilers which compile, on average, at rates of less than 1000 lines per minute.

Vendors are now concentrating on code quality. Many of the current compilers do not support optimization. Compiler vendors have found that the task of designing and developing Ada compilers is enormous. Now that their compilers are operational, vendors are resorting to "fine tuning". For example, the next version of the Alsys compiler will incorporate a high level optimizer. Plans are also underway to introduce a low level optimizer. Code produced by future Ada compilers will be far more efficient than that produced by current compilers. Other performance improvements will include faster task rendezvous times and better task scheduling implementations.

Future compilers will be more user-friendly and operate within integrated development environments. As more Ada tools become available for PCs, vendors will need to specify common interfaces so that the tools can work together. The traditional compile, bind and execute cycle will give way to a more interactive environment which will include syntax directed editors, smart debugging tools, and distributed networks.

Users will ultimately work with a family of compilers. During initial development, users will be able to eliminate syntax errors by using fast syntax parsers (perhaps integrated with the editor). To ensure the code will compile correctly, a high speed compiler will be used. This compiler will allow for incremental and partial compilation. After the system has been successfully compiled and executed, an optimizing compiler will produce the efficient code needed for the the final product. These various levels of compilation may form an integral part of an Ada development environment.

8. SUMMARY AND RECOMMENDATIONS

8.1. SUMMARY

The basic tools are available to develop Ada software with microcomputers: however, there is enormous room for improvement. Microcomputers, particularly PCs, are becoming more powerful and so are in a better position to effectively host Ada compilers and Ada development tools. As the number of Ada projects increases, there will be a proliferation of compilers and tools. Microcomputers will play a large part in Ada software development. Networking, distributed source data bases and libraries, and low costs will make them attractive for organizations engaged in developing Ada software products.

Until recently, the Ada community has lacked the tools to effectively develop Ada software applications. Considering that the language was standardized in 1983, vendors have been slow to provide the required compilers and tools. The complexity of the language, a previosly limited market, and complicated validation procedures have often been blamed for the slow response. However, there are now sufficient compilers for most applications and the number of new products is increasing exponentially. Armed with current knowledge, continued research, and a huge Ada software market, compiler vendors will continue to improve their products and strive to provide features which further aid the Ada software developer.

As the number of available Ada compilers for micros increases, effective requirements analysis and selection will be an important step in the development process. Selecting an Ada compiler can be far more difficult than selecting a compiler for older languages because Ada compilers are more easily related, they possess features that allow integration with other tools, and they are generally far more complex. Failure to correctly specify and procure the right compiler could jeopardize the success of the development project. Since validation simply ensures that the compiler conforms to the language standard, the prospective users will have to resort to other techniques such as benchmarking to test for quality and effectiveness.

Experiences with the Alsys PC AT compiler has shown that Ada code can be as efficient as that produced by compilers for other languages. However, the size of the runtime support software produced by the compiler will cause problems for applications targeted to machines with limited memory if compilation and the use of predefined packages is not properly managed. Another area of concern for designers is task communications. Because of inefficient task rendezvous, designers need to carefully assess task usage.

8.2. RECOMMENDATIONS

Recommendations are:

- Consideration should be given to using networked PCs in Ada development environments.
- Vendor supplied pre-defined packages should be used with discretion.
- Ensure that you are familiar with the licensing agreements before deciding on a compiler.
- Perform an effective requirements analysis before selecting a compiler.
- Benchmark and test compilers before buying.
- When selecting a compiler, ask experienced users for their view of your choice.
- Ada software technology is fast moving and so it is extremely important to keep abreast of developments.

A. REFERENCES

[1] "SARAH Operational Concept Document", Command and Control Systems Office, US Air Force, 5 September 1986.

[2] MARBACH W. D., "Technology at New Plateaus", Personal Computing, October 1986, pp 172-177.

[3] WALLIS P.J.L., WICHMAN B.A., "Requirements Analysis For Ada Compilers", Communications of the ACM Vol 24 No 1, pp 37-41.

[4] BROGSOL B., AVAKIAN A.S., GART M.B., "Alsys Ada Compiler for the IBM PC AT.

[5] BARBACCI M.R., HABERMANN N., SHAW M., "The Software Engineering Institute: Bridging Practice and Potential", IEEE Software, Nov 1986, pp 4-21.

[6] NISSEN J.C.D., WALLIS P.J.L., WICHMANN B.A., et al, Ada-Europe Guidelines for the Selection and Specification of Ada Compilers", ACM Ada Letters Vol III No 1 (July-Aug 1983) pp 27-50.

[7] U.S. Department of Defense, "Reference Manual for the Ada Programming Language", ANSI/MIL-STD 1815A. Jan 1983.

[8] WEIDERMAN N., HABERMANN N., et al, "Evaluation of Ada Environments: Executive Summary Chapter 1 Chapter 2", Software Engineering Institute, August 1986.

[9] "Ada Language Implementations Matrix", Ada Letters, Vol VI No 6 (November-December 1986).

[10] "Requirements for the Programming Environment for the Common High Order Language", STONEMAN, Department of Defense, Washington, D.C., November 1979.

[11] "MIL-STD Common Ada Interface Set (CAIS)", National Technical Information Service (NTIS), accession number AD A157-589.

[12] WICHMANN B.A., "Ackermann's Function in Ada", ACM Ada Letters Vol VI No 3 (May, June 1986), pp 65-67.

[13] CHERRY G.W., "The PAMELA Designer's Handbook", Thought Tools, Reston VA.

[14] "Special Issue on MASCOT", Software Engineering Journal, IEE Savoy Place London, Vol 1 No 3, May 1986.

B. DEFINITIONS AND ACRONYMS

Ada Information Clearinghouse (AdaIC): A section of the AJPO which provides information on Ada matters.

Ada Joint Program Office (AJPO): A U.S. Department of Defense office responsible for the Ada programming language.

Ada Language System (ALS): An APSE developed by SofTech Inc, Waltham MA.

Ada Programming Support Environment (APSE): An integrated set of software development tools for Ada software development.

Ada Validation Office (AVO): An office of the U.S. Department of Defense responsible for the validation of Ada compilers.

Automatic Digital Network (AUTODIN): Communications network for U.S. Department of Defense.

Descriptive Intermediate Attributed Notation for Ada (DIANA): An intermediate data representation for Ada compilers.

Micro: Short for microcomputer. A computer in which the Central Processing Unit is made up of one or more microprocessors.

Millions of Instructions Per Second (MIPS): A measurement of computer execution speed.

Personal Computer (PC): A low cost desk-top microprocessor based computer.

Personal Computer-Advanced Technology (PC AT): A computer that is comatible with the IBM PC AT. This computer is based on the INTEL 80286 microprocessor and can address up to 16 Megabytes of memory in protected mode.

Personal Computer-EXtended Technology (PC XT): A computer that is compatible with the IBM PC XT. This computer is based on the INTEL 8088 microprocessor and is generally limited to 640 Kilobytes of phsical memory.

Sidekick: An on-line utility program from Borland that provides features such as a calendar, notepad, calculator, ASCII table, and phone directory.

Standard Automated Remote to AUTODIN Host (SARAH): A standard intelligent terminal for AUTODIN users.

SuperKey: An on-line utility program from Borland that acts as a keyboard enhancer.

Zenith Z-150: An IBM PC XT compatible computer sold by Zenith Data Systems.