Chapter 3 Programming Languages

3.1 Overview

Sun supports all popular languages, with added functionality for Fortran, Pascal, Modula-2, and C

Sun provides high-performance compilers for languages including C, FORTRAN, Pascal, Modula-2, and Common Lisp. Sun’s C compiler, used for systems development at Sun, is a highly evolved implementation of the C programming language. Sun FORTRAN“ is GSA-certified FORTRAN 77 with VAX/VMS FORTRAN 4.0 compiler extensions. Sun Pascal“ is enhanced Berkeley Pascal with many popular extensions. Sun Modula-2“ is based on the ETH, Zurich Modula-2 compiler with extensions for SunOS. Sun Common Lisp“ is an optimizing compiler and environment for artificial intelligence applications. In addition to these compiler languages, Sun also offers a command level language, *the shell programming language.*

Optimizers improve performance for all languages

Sun’s languages share a common language base, composed of Sun’s optimizers, code generator, and libraries (see Figure 3-1). Sun’s leading-edge optimizing compiler technology increases execution performance on Sun-3 and Sun386*i* systems by 50 percent. The Sun-4 optimizers and code generators take maximum advantage of Sun’s Scalable Processor ARChitecture (SPARC“) technology, increasing application speed up to three and a half times versus unoptimized code. Sun has enhanced the quality and speed of floating-point processing for high-precision computations by offering a choice of floating-point options. All Sun languages have access to this technology, which adheres to the ANSI/IEEE 754-1985 floating-point standard. Floating-point intensive applications further benefit from Sun’s optimization strategies. With interlanguage calling, based on common calling conventions, programs written in any of these languages can access each other’s libraries. This integration furnishes a complete software development solution with full access to Sun’s user interface, graphics environment, and networking facilities.

Sun offers languages, and support tools that create a programming environment

Sun offers a variety of programming support tools, such as *dbxtool*, a window-based symbolic debugger, the *make* utility, the *Source Code Control System (SCCS),* and *execution time profiling analysis*. Integrated with Sun’s languages and individual programming support tools, Sun’s Network Software Environment (NSE“) dramatically improves team productivity. The NSE is a comprehensive software development platform that improves software quality, development productivity, and programming performance. Please refer to Chapter 4, "Software Development Tools", for detailed descriptions of other supporting tools.

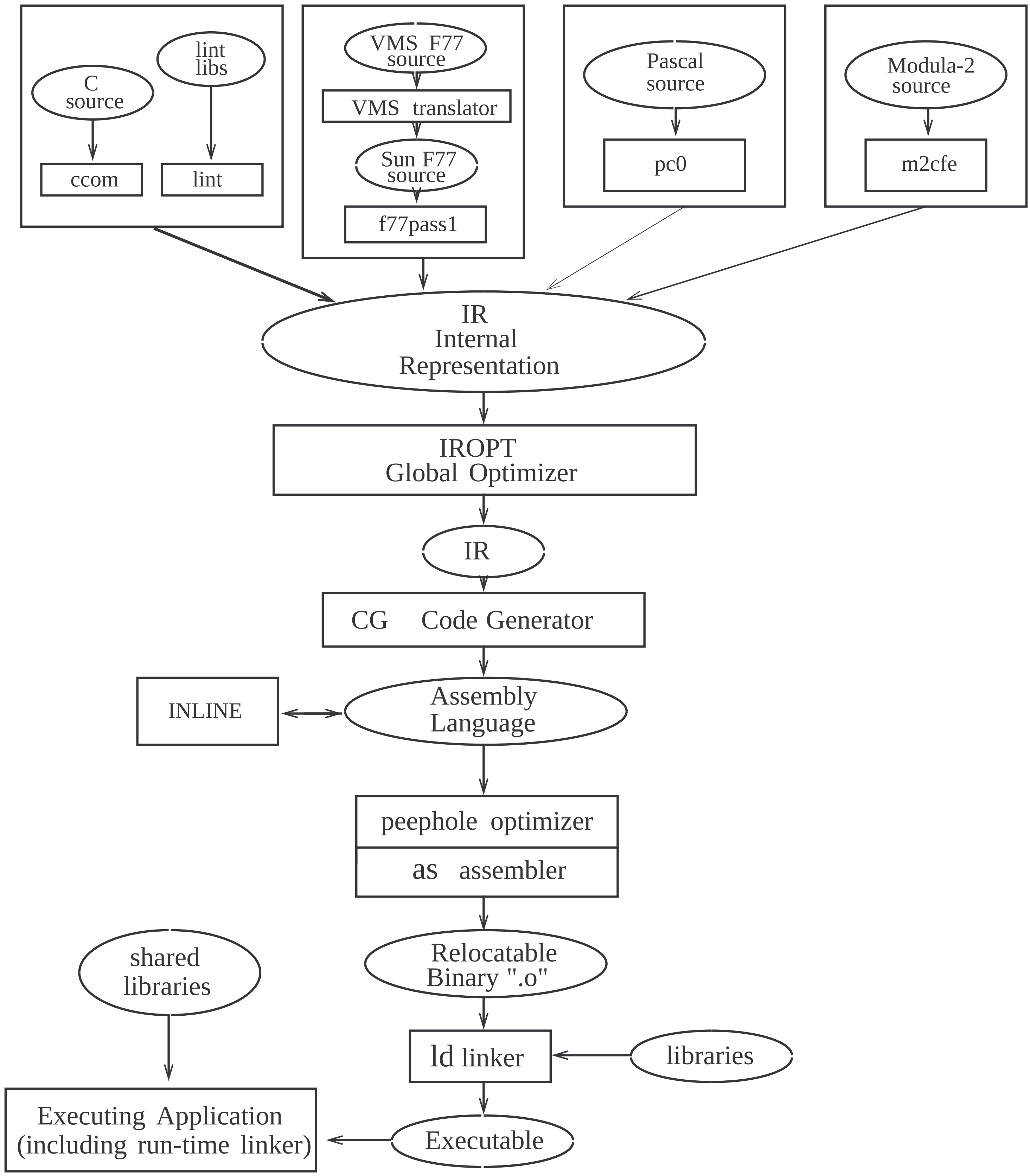

Figure 3-1. Compiler Flow for Sun Languages

3.2 C

Sun C is an enhanced C compiler

C is the language most commonly used by developers on Sun workstations. Sun’s C compiler is highly evolved with enhancements in the areas of code generation, interlanguage calling, optimization, and debugging. Sun’s C compiler balances syntactic and semantic extensions to the C language with standard C features, promoting program portability. Sun’s C compiler provides the interface between other Sun languages and the basic machine facilities. Two significant enhancements to Sun’s C compiler are the single-and double-precision compiler options, which permit more FORTRAN-like floating-point expression handling, allowing compute-intensive applications to be written in C.

Some of the features of Sun’s C compiler include:

- Definable data types including enumerated types
- Arbitrary-length variable names
- Machine-independent pointer manipulation
- General initialization, block structure, long integers, unions, and explicit-type conversions
- Access to all system facilities through the C runtime library

For further information on Sun’s C compiler, please refer to the *C Programmer’s Guide for the Sun Workstation.*

3.3 FORTRAN

Sun FORTRAN is an extension to the GSA-certified ANSI standard FORTRAN 77. The extensions are compatible with DEC FORTRAN 4.0 compiler. Some of these added features include records and pointer types and encode/decode statements. These features not only facilitate porting between DEC and Sun systems but also allow users to share the same code on both systems for products such as SunLink DNI. For deviations from the standard and a complete list of extended features available on Sun systems, please refer to the *FORTRAN Programmer’s Guide*.

ANSI Fortran with VMS extensions reduces porting efforts

Sun FORTRAN also provides an ANSI/IEEE-standard floating-point package that enhances the code fix when used with the Sun floating-point accelerator options. Sun FORTRAN floating-point routines can produce switched-object code for determining the fastest available floating-point hardware at runtime and select library routines for optimal hardware performance. Further performance gains are achieved with global and peephole optimizers, which help produce FORTRAN applications that execute up to 80 percent faster and reduce code size by an average of 10 percent. For the various options available to the Sun FORTRAN compiler, please refer to the reference manual page for the FORTRAN compiler. Other features of Sun FORTRAN include:

Sun Fortran features aid in portability and development

- ¥ Support for preprocessor directives such as #ifdef and #ifndef
- ¥ Access to command line arguments using *getarg* system call
- ¥ Namelist-directed I/O
- ¥ Do/End, Do, and Do/While statements
- ¥ Byte data type
- ¥ Logical operators on integers and arithmetic operations on logicals
- ¥ Identifier names up to 32 characters long
- ¥ Identifiers containing _ and $
- ¥ Source lines up to 132 characters long
- ¥ Record, Structure, and Union statements
- ¥ Type and Accept statements
- ¥ Pointer data type for compatibility with vectorized FORTRAN
- ¥ Recursive subroutine and function calls
- ¥ Relaxed input format

Unsupported VAX/VMS 4.0 extensions include:

- ¥ DEFINE FILE statement
- ¥ DELETE, FIND, REWRITE, and UNLOCK statements
- ¥ Nonstandard INQUIRE and OPEN options
- ¥ KEYID and key specifiers in READ statements
- ¥ Variables of type REAL*16 (quadruple precision)
- ¥ DISPOSE = specifier in CLOSE statement

3.4 Pascal

Sun Pascal is an extension to ISO level 1 standard Pascal

Sun Pascal is an enhanced version of Pascal that supports ISO level 1 standard Pascal with extensions that include interlanguage calling, conformant arrays, a variable-length string type, and external C and FORTRAN declarations. Sun Pascal global optimization generates code that executes up to 50 percent faster than the unoptimized code. Sun Pascal offers a high degree of portability between Sun and other environments. Options for the Sun Pascal compiler include presence of a floating-point processor, generation of code for symbolic debugger, and definitions of names to the preprocessor. Some of the features of Sun Pascal include:

- ¥ Underscores _ and dollar $ allowed in identifiers
- ¥ Conformant array parameters

¥ Otherwise clause in case statements

¥ *sizeof* operator

¥ *shortreal* and *longreal* types

¥ External FORTRAN and C declarations

¥ Bit operations on integral types

¥ Preprocessor facilities

¥ PUBLIC and PRIVATE declarations

Sun Pascal deviates from the ISO Pascal standard in the following ways:

Sun Pascal deviates from ISO Pascal

¥ Operands of binary set of operators {*, +, -} are required to have identical types. The standard permits different types as long as the base types are compatible.

¥ According to the standard, the expression [maxint...-maxint] should be equivalent to [ ]. The Sun Pascal compiler does not evaluate sets with elements larger than those indicated by the definition of type intset (predefined as 0...127).

¥ Sun Pascal treats files declared as file of ÔcharÕ the same as files declared as Ôtext.Õ In ISO Pascal the two types are distinct.

¥ The value of *m mod n* is not computed incorrectly for negative values of *m*. According to the standard, the divisor must be positive, and the result must be negative. However, there is a documented workaround for this problem.

For further information on SunÕs Pascal compiler, please refer to the *Pascal ProgrammerÕs Guide for the Sun Workstation.*

3.5 Modula-2

Sun Modula-2 implementation is based on WirthÕs Programming in Modula-2

Sun Modula-2 is a high-level language that offers the type-checking and control of Pascal and a modular framework for large-system design and low-level system access. Sun Modula-2 has a high degree of source portability. With system dependencies isolated in library modules, little effort is required to port a program to another environment. A built-in control system assures that only the current versions of object files build the complete program. Sun Modula-2 can access the C library or any other C routine, including those with variable length argument lists. The compiler also has special modules that provide complete access to the UNIX system and many other routines. Other Sun Modula-2 extensions include optimization and automatic compiler-allocated register variables. Other Modula-2 features include:

¥ Complete implementation based on *WirthÕs Programming in Modula-2* , 2nd and 3rd editions

¥ Identifiers containing _

¥ shortint, shortcard, and longreal data types

¥ Subranges preceded by optional base; identifiers such as Shortcard [10..20].

¥ Automatic compiler allocated register variables

¥ Augmented identification module syntax

¥ The Size function

¥ Functions that can return Array and Record data types

¥ Compile-time evaluation of floating-point constant expressions, including ABS, Float, and Trunc.

¥ Operations on binary files

¥ Variety of data type conversions

¥ Operations on files, directories, and general I/O

¥ Memory management and UNIX system support facilities

¥ Dependency analyzer program that generates in tabular form the import/export

For further information on Sun’s Modula-2 compiler, please refer to *Using Modula-2 on the Sun Workstation.*

3.6 Assembly Languages

Assembly Language programming is allowed, but not recommended

Because Sun promotes portability and open systems, programming at the assembly language level is inefficient. Also, *Sun’s* compilers generate optimized code at different levels, which makes assembly language programming unnecessary. Those who want to program in assembly language may do so, mixing calls with other high-level languages. (Sun supports three different assembly languages: Motorola 680X0, Sun’s SPARC, and Intel 80386.) However, applications developed using assembly language is not portable within the Sun environment.

Sun supplies reference manuals for its own SPARC, as well as Motorola and the Intel 80386. Please refer to the *Assembler Language Reference Manual for the Sun Workstation, Sun-4 Assembly Language Reference Manual,* and the *Sun386i Developer’s Guide (814-1009).*

3.7 Shell Programming

UNIX has one of the most powerful command languages. A shell program may be written using either the C-shell or the Bourne shell. The features supported by these two shells are very similar; the difference is the syntax. A major strength of these two languages is that both have central structures similar to a high-level programming language such as C or Pascal. However, since these programs are interpreted, they execute slower than the compiled programs. Shell programs are used widely for a variety of purposes. One of the main uses is for system administration. Another typical use is to parse a command line and invoke the appropriate program. For more information, please refer to the Commands Reference Manual for csh(1) and sh(1).

3.8 Cross-Compilers

Cross-compilers allow generation of code for all Sun platforms from a single workstation

With the introduction of heterogeneous networks, it has become necessary to provide software developers with tools that reduce the time and cost of producing software for multiple computer architectures. The Sun cross-compiler products allow programmers working in Sun C, FORTRAN, or Pascal to produce executable binaries for all Sun architectures from a single Sun system. Because it runs PC applications, the Sun386*i* workstation family offers software developers a unique cross-development environment. Source-code compatibility lets users simply recompile existing source code to use on any Sun hardware architecture. Figure 3-2 is a summary of Sun’s cross-compiler availability.

Host	Target	Languages	OS
Sun-2	Sun-3,-4	F77, C, Pascal	3.X
Sun-3	Sun-2,-4	F77, C, Pascal	3.X
Sun-4	Sun-2,-3	F77, C, Pascal	3.X
Sun-2,-3,-4	Sun-2,-3,-4	F77, C, Pascal	4.X*
Sun-3,-4	Sun386*i*	F77, C, Pascal	4.X*

* supported in the future

Figure 3-2. Cross-Compilers Supported by Sun

3.9 Calling across Languages

Sun supports multilanguage code in one application. Even though this support exists for all languages in general, there may be some restrictions.

Interaction of FORTRAN with other languages is different from others. When a FORTRAN program calls a C function, all the C entries are appended with an underscore "_" character because the FORTRAN linker appends this character to all external names. The symbol is undefined to the linker. It is also important to note that some operating systems may allow identical calling sequences from multiple languages. On Sun systems, some of the system calls are inherently tied to the C language. For instance, calls such as *ioctl* and *malloc* do not work from a FORTRAN program but function properly if invoked from a C program. If you need such functions, you can build a library of C functions providing these facilities, which the FORTRAN programs can call. The FORTRAN library does work with a variety of system calls, for example getuid, signal. For additional information on Sun FORTRAN, please refer to the *FORTRAN Programmer's Guide* .

Figure 3-3 is a summary of interlanguage calling supported by Sun.

Calling Language	Called Language	Comments
C	FORTRAN	Implemented
C	Modula-2	Implemented
C	Pascal	Not Recommended
C	Lisp	Implemented
FORTRAN	C	Implemented
FORTRAN	Pascal	Not Recommended
FORTRAN	Modula-2	Not Implemented
FORTRAN	LISP	Implemented
Pascal	C	Implemented
Pascal	FORTRAN	Implemented
Pascal	Modula-2	Not Implemented
Pascal	Lisp	Implemented
Modula-2	C	Implemented
Modula-2	FORTRAN	Not Implemented
Modula-2	Pascal	Not Implemented
Modula-2	Lisp	Not Implemented
Lisp	C	Implemented
Lisp	FORTRAN	Implemented
Lisp	Pascal	Implemented
Lisp	Modula-2	Not Implemented

Figure 3-3. Sun's Interlanguage Calling Sequences

3.10 Common Lisp

Sun Lisp is based on the Common Lisp standard, with added functionality

Sun Common Lisp is the optimal Lisp program for both development and delivery of symbolic computing applications, such as knowledge-based or expert systems, natural language processing, and vision and speech recognition systems. *1* is a complete implementation of the Common Lisp standard as defined by the Common Lisp Committee and includes a host of additional features.

Typically recognized strengths of the Lisp language are threefold: first, its symbolic computing capabilities for ease in representing data with qualitative descriptions as well as numerical; second, its inherently powerful debugging facilities; and third, its rapid prototyping capabilities to quickly develop an application without the limitations of completely defining an algorithm and data structures prior to coding, facilitating a more natural and faster development cycle.

Sun Common Lisp extends the Common Lisp standard with development tools including an Emacs-style editor that is knowledgeable about Lisp syntax and allows for easy customization of the editing environment for greater productivity. Debugging features include a data inspector, full function breaking, tracing, and single-stepping. The Pitman Error Handler, another Common Lisp Committee standard, is provided as a functional interface to the Lisp error handling system to customize procedures for managing errors.

Sun Common Lisp provides a set of tools for the programmer

To assist in porting, Common Lisp Object System (CLOS), a widely accepted object-oriented programming system for Lisp machine compatibility, and Flavors, the emerging object-oriented system, are provided. Two compilers are available: a development compiler for convenient, rapid compilation speeds and a production compiler, which produces fast, efficient code to be used for later, more stable versions of an application. A complete window toolkit enables fast, flexible window construction and management and features 2-D drawing primitives, pop-up menus, mouse-sensitive regions, and multiple viewports and fonts.

The Ephemeral Garbage Collection (EGC) system is a useful feature for developers who are faced with applications in which large garbage collections interruptions can effect performance. EGC features brief, bounded garbage collection intervals that are virtually imperceptible to users, improving both development and delivery runtime Lisp performance.

Sun Common Lisp's powerful Foreign Function Interface extends Lisp to take full advantage of code written in Sun's C, FORTRAN, and Pascal, enabling both calls to and from foreign functions. The Foreign Function Interface provides dynamic access to foreign structures and data sharing, allowing users to take full advantage of the graphics, networking, and host of Sun software solutions.

For more information on Sun Common Lisp, the following documentation may be of interest:

- *Sun Common Lisp Data Sheet*
- *Sun Common Lisp Reference Manual*
- *Sun Common Lisp User's Guide*
- *Sun Common Lisp Release Notes*

3.10.1 Symbolic Programming Environment

Sun's Symbolic Programming Environment (SPE™), which is sold separately from Sun Common Lisp, is a complete and integrated development environment that complements Sun Common Lisp with window-based debuggers, data inspectors, a source code manager, and other program analysis tools.

The Window-based Debugger allows users to examine and modify the runtime state of a program and more closely follow its execution to trace errors and bugs.

In a similar fashion, the Window-based Inspector gives users access to the internals of their complex data structures and variables. The user interface allows programmers to examine and set values within these structures so they can more easily debug and analyze their programs.

SPE provides a variety of tools to aid programmers in developing Lisp-based applications

The SPE Application Manager is a window-oriented facility that manages large multifile applications by keeping track of the order in which files must be loaded and the dependencies held within the files. The Application Manager also keeps track of which files have been modified, by whom, and whether recompilation is necessary.

The Editor is a specialized Lisp-sensitive editor with numerous commands for handling Lisp syntax, programs, and operations.

With the Source Code Finder, users can immediately access any source-code definition for functions or variables by entering the name of the symbol. The Source Code Finder then locates the source-code version of the symbol, whether or not it is loaded in the current session or is loaded only in compiled form. This facility makes debugging and understanding large applications manageable.

The Cross-Reference Analyzer analyzes a user application and can graphically display the relationships within a program, such as which functions are called from where and which special variables are used by what functions.

These tools make the SPE a powerful environment for developing Lisp-based applications. The SPE, used with Sun Common Lisp, has been used to develop a wide range of AI applications, from expert systems to vision and natural language applications.

3.11 Ada

Ada is one of the most recent programming languages to attract a mandated set of followers. The United States Department of Defense was active in the development and standardization of the language before 1983. The DoD continues today in the maintenance of the language via its Ada Joint Program Office (AJPO) and now requires that all new mission-critical computer applications be written in Ada. The use of the language is also growing in foreign markets, especially in Europe, and it has become an international standard. Its use is also growing in commercial markets both here and abroad.

The Ada language is ideal for modern software engineering methodologies. It supports such principles as object-oriented design, rapid prototyping, and programming-in-the-large concepts. Some of the language's most powerful characteristics include: capabilities for information hiding, abstract code representations, strong type checking, exception handling, and management of compilation order dependency rules. Information about what a particular program unit does can be selectively "hidden" from the user of the program unit by judicious use of specification and code segments. The language also supports "generics," which provides the general abstract coding of particular functions that are applicable to a broad class of objects. Strong type checking is rigidly enforced and exception handling is designed into the language definition. Ada requires the existence of a "library manager," which assures that dependency information between program units and specification-body pairs are kept in proper sequence. In other words, required recompilation sequences are automatically detected whenever any part of a program unit is recompiled.

There are many Ada compiler products available for Sun workstations. The three most popular, validated Ada compilers are supplied by TeleSoft, Verdix, and Alsys. For other vendors and further details, please refer to the *Catalyst* catalog. These products meet all of the Ada language requirements briefly outlined above. Each has some unique advantages to consider when porting an Ada application to a Sun platform. Each implements differently such things as interfaces to other languages (usually C and FORTRAN), standard representations for variable types, multitasking, interrupt handling, and operating system support. All of these three companies typically offer their same Ada technology on other platforms. As a general rule, when porting an

Ada application it is easiest to stay with the compiler that was used with the original Ada application.

TeleSoft® Ada technology (TeleGen2“) seems to focus on large Ada applications. At least one major aerospace company is using it for a very large Ada application (over 1,000,000 lines of code). In fact, the product itself is written in Ada (over 500,000 lines of code) and is self-compiled. The primary development platforms that TeleSoft® products operate on are VAX/VMS, IBM mainframes, and Sun. Some results have shown that if the existing Ada application is running with the DECAda compiler (VMS) or the Rational compiler, it has a reasonably good chance of being ported to Sun with the least amount of work using the TeleSoft product.

Porting Ada applications from other vendor platforms to Sun workstations

Verdix® Ada technology (VADS) is used extensively in the general UNIX marketplace. They have written much of their product in C and hence are frequently identified as the premier vendor of Ada products for the UNIX platform. If an Ada application is already running with UNIX on a non-Sun platform, it may be easier to use VADS to port it to Sun.

Vendors porting Ada software from the IBM-PC should refer to the *Catalyst* catalog for a compatible compiler on the Sun platform.

The degree of difficulty in the port of an Ada application depends on how many system-specific features are used in the original hardware/software configuration. Furthermore, each of the vendors mentioned have products in many of the same common areas and strict compartmentalization is not to be suggested by the above guidelines. The best rule is: If possible, use the same Ada compiler technology in porting from the original environment to the Sun workstation.

3.11.1 Ada Programming Support Environment (APSE)

APSE is an environment for Ada programmers that increases productivity

At the same time the Ada language was established, the concept of a companion programming support environment was defined. It was even outlined in terms of a kernel APSE content (KAPSE) and a minimal APSE content (MAPSE) in the original Stoneman documents. There have been continuing standardization efforts in this area and currently out for review is a description of a Common APSE Interface Specification (CAIS.)

The three Ada compiler products from TeleSoft, Verdix, and Alsys are accompanied by APSE tools on the Sun workstation. As a minimum, the library manager and linker/loader are provided with any validated Ada compiler product. However, all of the above supply additional tools to help the developer in the application coding and testing phases. They all supply dependency listers and source debuggers. Library maintenance tools and code optimization options are also included. Some of the products also include other tools for execution profiling/tuning and automatic recompilation of dependent objects.

The exact specifics of the tool complement that each company offers is continually being enhanced. Check with each vendor for the exact content of their APSE when beginning a port of an Ada application. Some of the companies have also expanded their tools interface to include workstation "look and feel" (window/mouse-based interaction).

Ada bindings to graphic packages like GKS, PHIGS, and window systems, like X11

In addition to APSE availability, Ada bindings to other products are increasingly available on Sun workstations. Today, TeleSoft offers Ada bindings to SunOS and SunView so that the application developer can easily access system or window calls from the Ada source code without having to use the Ada pragma interface construct to interface to C or other language bindings to these same functions. Ada bindings also exist for GKS and the newly emerging PHIGS graphics standards. In windowing standards, there are public domain versions of Ada bindings to X11 available on Sun.

Much work is underway to expand APSE tools availability. Not only are the Ada compiler vendors expanding this set, but also third-party companies are writing tools that can be interfaced to any of the available Ada compiler products. One company, Xinotech, offers a syntax-directed editor that can help prompt a user through the editing phases of developing Ada applications. As CAIS becomes more widely accepted, it is likely that the richness of Ada tools will expand dramatically. Thus, the porting and development of Ada applications will be made increasingly easier.

3.12 Future Directions

Sun is committed to supporting programming language standards. In the future, as standards develop for the different programming languages, Sun will be actively involved in both specification and implementation of these standards. Areas in which standards are currently being developed include FORTRAN 8X, ANSI C, and C++. In addition, Sun is committed to supporting efforts by third-party language vendors to provide programming languages that conform to standards. These languages include Ada, BASIC, and COBOL.