The DSN Standard Real-Time Language

R. L. Schwartz, G. L. Fisher, and R. C. Tausworthe
DSN Data Systems Section

A set of requirements for the Deep Space Network Standard Real-Time Language has been recently drafted. This language will be a modern high-order programming language well-suited to the special needs of real-time programs developed for use in the Deep Space Stations and Network Operations and Control Center. Nearly all DSN real-time programming has, in the past, been done using assembly language. The implementation of a standard high-order language is being planned in order to promote the development of real-time programs with higher reliability, increased programmer productivity, language commonality, flexibility, and re-use potential, and to provide a means for reducing the current life-cycle costs of DSN software.

I. Introduction

The DSN real-time software includes tracking and data acquisition software and mission support software. These programs, typically driven by high-speed data interrupts, have rigid input/output format requirements and must interface with special-purpose external hardware.

Because of the serious time constraints imposed by the high-speed interrupts, and the strong dependencies on external data formats, these programs have in the past been written, for the most part, in assembly language. This practice has produced software with a high development and maintenance cost. As the hardware base has evolved over a period of years, and as different missions have imposed their differing constraints on the software, large existing programs have had to be almost completely redeveloped without substantial support from the earlier software.

The DSN system environment has in the past included a number of different small- to medium-size computers, with widely divergent characteristics. This diversified type of environment is expected to extend also into the foreseeable future. Future DSN capabilities, for example, are planned to include the use of single-chip microprocessors as part of the standard DSN set of Control and Computation Modules (CCMs).

A standard higher-order language specifically suited to the type of real-time programs developed by the DSN is needed to promote higher reliability, increased programmer productivity, language commonality, flexibility, and provide a means for reducing the current life-cycle cost of DSN software. A set of requirements for such a Deep Space Network standard real-time language has been recently drafted. This article summarizes that draft material. Work is continuing on refinements to these requirements, and completion of the language specification is scheduled for the end of this fiscal year.

II. Brief History of Language Design for Reliable Software

The development of higher-level computer programming languages is motivated by the desire to increase programmer productivity and make the process of programming independent of the particular machine used. The rapidly changing computer hardware, together with a rapidly expanding domain of problems to be handled by means of a digital computer, necessitates the viewing of a programming language as a vehicle for the expression of the problem in abstract terms, obviating a detailed machine-level specification of the algorithm. Thus, a programming language forms an interface between its users, who are concerned primarily with the ease of expression of a problem in abstract terms, and the instruction set on a particular real machine.

As the class of problems to be solved increases in scope and complexity, likewise does the necessary complexity of expression within the programming language. The ability of a user to understand and accurately communicate his intentions through the programming language necessarily becomes more difficult, yet at the same time becomes more crucial. The programming language must accept the burden of providing a vehicle for the expression of the problem in a conceptually clear manner in order to promote programmer understanding of what he has created.

This need has been recognized within the last decade, with various approaches being taken. After an early period, in which the development of extended assembly languages, such as FORTRAN, (Ref. 1) paralleled the development of computer architecture, the need for greater abstractions from the basic machine was recognized. The introduction of such languages as ALGOL 60 (Ref. 2) and LISP (Ref. 3) signalled the beginning of an age of higher-level languages primarily oriented toward the user. An attempt was made to formally express the semantics of these languages without reference to the concrete machine. This attempt at formalization of the semantics of a programming language was carried further by PL/1 (Ref. 4) and ALGOL 68 (Ref. 5) in very different manners. Both languages sought to increase the flexibility and universality of the language. ALGOL 68 chose to follow a course of greater orthogonality (using as few rules as possible, minimizing the number of special cases) and generalization of structure, leading to almost unbounded power being given to the bewildered programmer. PL/1 chose to increase the power of the language through a large set of constructs and specialized rules that the bewildered programmer needed to be aware of in order to express his or her problem.

Soon, a realization of the high cost of software production led to a methodology emphasizing a structured approach to the development of computer programs. This methodology initiated, among its supporters, a shift away from languages such as ALGOL 68 and PL/1, in favor of simpler languages with features encouraging more disciplined programming.

The programming language PASCAL (Ref. 6) was the earliest and perhaps the most successful of the languages to be designed at this time. PASCAL contains a minimum number of flow-of-control constructs, no default declarations or automatic type conversions, and a clear, concise syntax. PASCAL sought to treat language semantics in a manner that would allow both a precise implementation specification and a description of a formal (logical) system in which properties of the program could be derived (Ref. 7).

Various other languages, such as ALPHARD (Ref. 8), EUCLID (Ref. 9), GYPSY (Ref. 10), MADCAP-S (Ref. 11), STRUGGLE[1] (Ref. 12), and ALPO[1] (Ref. 13) have generally followed the same ideas of semantic description used by PASCAL, but each has proposed somewhat different sets of language features to increase program reliability, provability, power, etc.

The proliferation of programming languages and the resulting program incompatibilities which have occurred in the last five to ten years have spurred various large users of software toward the development of a standard programming language (cf. Ref. 14). The British government adopted the CORAL 66 (Ref. 15) programming language as its standard in 1975, the German government is now completing their standardization on the language PEARL (Ref. 16), and the French government is currently working on defining a French standard language. The U.S. Department of Defense has recently completed a study to define the requirements of a DOD standard real-time higher-order language (Refs. 17, 18), to be used in all defense system applications. The language, to be called DOD-1 (with PASCAL as its base language), is scheduled to be rigorously defined and a test translator for it implemented by early 1979. The DSN has patterned its draft requirements, summarized here, after the DOD requirements.

III. Goals for the DSN Standard Real-Time Language

This section lists and discusses a set of general goals to be met by the DSN standard real-time language. These are presented in their approximate order of importance.

A. Life Cycle Cost Effectiveness

The overriding goal of using the DSN standard higher-order language is to reduce the life cycle cost of DSN software,

[1]These languages have not been implemented.

including initial development cost, maintenance and operation cost over the system lifetime, and costs associated with retiring a program (de-implementation), if any.

A great deal of each new DSN software system development is a repetition of programming processes that have been written before. Yet, many DSN systems are started from scratch with little benefit from this previous work, other than what an individual programmer might remember having been a part of the previous project. Higher-order languages are preferred, of course, and used when possible, but the assembly language of the host machine is usually necessary at present. Even with the "same" higher-order language, differing dialects and local computer system alterations make it almost impossible to sustain a truly powerful production environment.

The DSN standard higher-order real-time language will remain stable (except under DSN Engineering Change Control Board direction) to ensure efficient reutilization of once-written software. Such stability will also tend to minimize the cost of sustaining an effective production environment.

B. Reliability

Real-time programs in the DSN control critical real-time processes in which there is a potentially severe penalty for faulty operation. DSN Software therefore must strive toward total reliability. The DSN standard real-time programming language will support a programming environment which fosters the creation of well-structured, readable, understandable, testable, manageable programs. Such characteristics in a program are known to promote program reliability through fewer design faults, decreased scope of error, and increased ease in fault detection, location, and repair.

C. Efficiency

For programs that demand it, the object code generated by the language translator must be efficient at run-time, mainly in terms of speed of execution, but also in terms of storage requirements. The handling of high-speed telemetry interrupts, for example, necessitates rapid response and processing speeds. The DSN real-time language must therefore provide users a means to optimize time-critical portions of their programs when necessary. Since such optimization is likely to be very machine-dependent, the real-time language must permit users to access low level features of the host machine in a way not conflicting with other goals of the language.

D. Maintainability

DSN programs written in any language must be maintainable. Recent industry studies (Ref. 19) and DSN qualitative studies have shown that program maintenance accounts for some 60% of the life-cycle costs of large programs.

Programs are made maintainable by modular construction, functional organization, localized scoping of data connections, simple control structures, the absence of special default features, easily understandable higher-level language constructs, and a stable, controlled programming environment. Additionally, maintainability relates to the availability of diagnostic tools and measurement aids which support the calibration, alteration, fault detection, fault isolation and location, and repair of the program. The DSN real-time language will form the kernel of a unified total programming system which will accommodate all of these needs.

E. Stability

The programming environment within which software is developed needs to be stable in order to provide reasonable assurances of the long-term reliability, validity, maintainability, and reusability of DSN Software. The programming system must have evolutionary potentials and avenues for change, lest it become inadequate for future missions. However, such evolution must be at a controlled, cost-effective pace.

The DSN standard real time language and its processor designs will be owned, implemented, and maintained by the DSN. Alterations will only be permitted as directed by DSN Engineering Change Control Board[2] action. Such changes will invariably require assessment of impacts, costs, and benefits prior to the implementation of new or altered features.

F. Training

The time required to learn the DSN standard real time language should be relatively short. In relation to this goal, the DSN standard real time language is required to be similar in certain ways with other languages being used in the DSN (principally MBASIC^{tm} (Refs. 20, 21)). The features of the DSN language are meant to be English-like and self-descriptive. Nevertheless, the features will be well documented, with an accurate description of the language semantics oriented toward the understanding level of its users.

G. Transportability

Due to the range and diversity of programmable hardware in the DSN, reusability of software items relies on portability of program parts and transferability of personnel skills among the various machines used. Even among real-time programs with device and machine dependencies, there are a great number of operations, human and program, common among different software efforts on the different machines. The standard language will provide a means to isolate those portions of real-time programs which do and do not have an inherent

[2]Or other JPL or NASA responsible authority.

machine dependence, so that significant portions of programs may be reused as needed in other DSN applications.

The standard language will also increase the transportability of programmer knowledge and skill. The DSN standard real-time language is intended to allow each programmer to apply his or her skills among all machines and other tasks being done in the DSN.

IV. General Philosophy of Requirements

The DSN standard real-time language is intended to be a small, efficient, real-time-oriented, state-of-the-art, modern programming language for the DSN which can be adapted, via its extensible nature, to be useful for the entire spectrum of DSN real-time applications over the wide range of hardware characteristics projected for future use. The "smallness" of the language is meant to accommodate generality of expression while creating efficient programs using an affordable compiler (a few man-years per implementation).

The DSN real-time operational environment differs in important ways from machine to machine, and the needed features of DSN programs vary, depending on the particular machine and subsystem. Because of this, those facilities that can be better programmed as subsystem-dependent or hardware-dependent subroutines or macros have not been included in the real-time language requirements. The language therefore will have only very basic built-in input/output facilities, no specific mechanisms for parallel processing (other than interlocks), no co-routine syntax, and no built-in data types or constructs to handle the special and diversified hardware characteristics found in the present DSN hardware. All of these special characteristics will, however, be accommodatable by macros and subroutines written in the language which, by means of standard libraries, can be shared by groups of users.

A general principle that has been used in the formulation of the requirements is that the choice of syntax and semantics should follow the principle of "least astonishment." That is, there is probably a flaw in the language design if the "average programmer" is astonished to learn about some aspect of the language. The language is meant to appear sensible both to the average skilled programmer and the language specialist alike.

V. Language Requirements

The specific requirements of the language have been assembled in keeping with the general goals and requirements philosophy outlined above.

Briefly, the overall requirements to be satisfied by the language are:

(1) Statement-oriented.

(2) Supports top-down structured development of programs

(3) Strongly typed with full compile-time type checking

(4) Extensible, supporting data abstraction by means of a data definition facility

(5) Multi-layered language with several levels of definitional power

(6) Able to control non-standard external devices

(7) Allows separate compilation of program segments with full interface verification prior to execution

(8) Part of a total programming environment

(9) Compilable on DSN standard minicomputers

(10) Generates code for DSN standard minicomputers and CCM's

In this section we describe the salient features of the language which revolve around the data definition facility: its role in supporting data abstraction, its expanded definitional power, and its use in controlling real-time processes. We also briefly introduce the concept of a total programming environment for language support. Subsequent DSN Progress Report articles will describe these and other aspects of the language in more detail.

A. User Data Definition Facility

The DSN standard real-time language supports data abstraction by allowing the user to define new data types and operations within programs. New data types are defined by specifying the class of data objects comprising the type and the set of operations applicable to that class. The definition of the class of objects and operations on the objects is done by using the composition of previously defined data types and control structures, which, if required, can be augmented by the use of a medium-level, closer-to-machine-language layer described in section B below.

Once defined, these abstract data types may be used as one would use a built-in type; one may declare identifiers ranging over the data type, then operate on the data by means of the defined operations. By allowing access to the new object only by means of the pre-defined operators, an *encapsulation* of the data definition supporting the abstraction occurs, thereby guaranteeing the integrity of the object being defined; that is, no unauthorized modification of the representation may occur.

The number of specialized capabilities needed for a common language for all DSN real-time programming tasks is large and diverse. In many cases, there is no consensus as to the form these capabilities should take in a programming language. No higher-order language can build in all the features useful to a broad spectrum of applications, or which would anticipate future applications and requirements, or even provide a universally "best" capability in support of a single application area. Assembly language has had to serve as the main language tool for most existing DSN (and other) real-time programs because no higher order language has been available that can accommodate both the very low-level programming constructs as well as higher-order abstractions.

To require that a language be implemented which has only those primitives in the language required to handle current DSN applications is to build in obsolescence. Rather, the DSN real-time language must be robust in order to survive evolving hardware and software requirements. It must contain all the power necessary to satisfy current applications, yet, at the same time, contain the ability to extend that power to new and different application tasks.

A language with facilities for defining data and operations restricted to that data offers the capability to add new application-oriented structures and to use new programming techniques and mechanisms through descriptions written entirely within the language. It accommodates communication with non-standard external devices and diverse operating environments. The DSN standard real-time language definitions will have the appearance and costs of features which are built into the language while actually being catalogued as accessible application packages.

No single programming language can fulfill all the goals set forth earlier; but the DSN language, with its data and operation definition facilities, can adapt to meet changing requirements in a variety of areas while yet remaining very stable itself, thereby promoting efficiency, maintainability, reusability, etc.

B. Multi-Layered Language

The implementation of an abstract data type via the DSN standard real-time language data definition facility will be permitted to have more expressive power than will be allowed in the rest of the user program. Inside a data definition structure, the pointer data type may be used in order to implement the representation of linked structures or other data objects requiring the use of a pointer. Additional lower-level language features available only within the data definition facility include 1) a medium-level language layer which is closer to the machine language of the host computer, 2) calls to routines coded in other supported languages, and 3) operating system calls. Each of these capabilities is particularly important to support efficient control of real-time processes.

The medium-level, closer-to-the-machine-language layer will provide a machine-dependent method of accommodating those portions of DSN real-time programs which must absolutely have access to specific machine architecture or bit and byte level manipulations. The layer will consist of the normal high-level control constructs, a set of special machine-level data types (such as bit and byte), and a series of built-in functions which provide a one-to-one correspondence to the instructions present on the base machine. This medium-level language layer will thus provide machine-level access together with the high-level structuring tools present in high-level languages. A special machine-specific medium-level language will be required for each implementation.

Such a medium-level language concept is not new; it began with the medium-level language PL/360 (Ref. 22) developed by Niklaus Wirth for the IBM 360 in 1972. Since then, similar languages have been developed for many machines, following the style of PL/360.

The layered-language capability can be used, for example, to define a DSN standard (but not built-in) library data type called, say HIGH_SPEED_DATA_BLOCK, with data-accessing operations coded using the medium-level layer to unpack and retrieve the information as well as allocate and deallocate blocks as necessary.

C. Real-time Input/Output Control

The DSN standard real-time language will support control of real-time processes, both through generalized input/output operations and the use of its data definition facilities. The language will also provide an interlock construct as a primitive mechanism for programming data definitions used for synchronization of parallel and concurrent processes.

A large part of many DSN real-time programs is directly concerned with awkward or application-tailored input and output equipment, which have curious input/output instructions and special status words, and which deal with data in elaborate non-standard, machine-dependent formats, none of which can be clothed in the comfortable abstractions of a high-level programming language. As mentioned earlier, it is not possible for a single standard higher-level real-time language to contain efficient real-time constructs which are sufficiently general to be able to handle all, or even part, of the peripheral device interfaces likely to be seen in the DSN real-time environment.

Accordingly, any communication with external devices which is not possible via the built-in input/output constructs in the DSN standard real-time language can be handled by means of the data definition facility. The programmer will be able to define an abstract data type and operations which handle the necessary protocols and perform all the necessary communications. Inside the data definition structure, the programmer will have the use of the pointer data type, the medium-level language layer, calls to routines coded in other supported languages, and operating system requests.

D. The Total Programming Environment

The DSN standard real-time language will not necessarily require that the object machine have an operating system. The language implementations initially will be suitable for programming both the DSN standard minicomputer and the Control and Computation Modules (microprocessors). The language translator is currently planned for implementation on the DSN standard minicomputer, with a cross-compilation capability for compiling programs targeted for the Control and Computation Modules. There is no requirement for self-hosting on the CCMS.

The mere availability of a particular language processor by itself is not sufficient to satisfy the goals which have been set forth earlier. The time is past in which a modern language can be considered apart from the programming system in which it resides. The presence of synergistic tools to support program development is an essential factor in the cost-effective fabrication of programs of the complexity found in DSN tasks. For these reasons, and following the precedents of ECL (Ref. 23) and MBASIC^{tm}, the DSN real-time language will require a programming system designed specifically for the language. Among the elements of this programming system are (1) an intelligent text-editor, with features designed to aid in the coding of programs written in the language; (2) an intelligent linkage-editor (collection program) with facilities for resolving more of the module interfaces than just addresses; (3) a run-time support system that includes an intelligent, interactive test-probe capability; (4) an interactive mode of program development testing, with some emulation of the real-time environment; (5) a monitor capability which will provide performance measurement information, such as the total number of object code instructions, the machine cycle times required to support each of the statements in the standard language, the statement execution frequency counts, and the time-sampling statistics collected at run-time (either in interpretive or compiled-mode operation). Such performance measurements lead to programmer awareness of the time and storage characteristics of his code, and can substantially improve or optimize source code as required to meet applications constraints.

VI. Conclusion

The requirements reported in this article are, at this writing, in a preliminary state. Completion of the requirements and the endorsement of these by both programmatic and implementation organizations is required before the actual language and processor design activities begin. Liaison with institutional and flight-support computing efforts in their current quest for a JPL-wide standard real-time language may also be expected to influence the final set of requirements. (The current belief is that the DSN standard real-time language can form a kernel subset of the JPL standard real-time language, for the middle-to-large class of machines).

The authors do not mean to imply by this article that a new language must be invented to fulfill the requirements summarized here. However, since no higher-order languages currently implemented on DSN machines have been deemed adequate for DSN tasks (as borne out by the almost total reliance on assembly language), it seems fairly certain that whatever language is ultimately chosen (existing, adapted, or invented), the processor for that language will have to be designed, documented to DSN standards, and implemented. Current plans call for the completion of the DSN real-time language on the MODCOMP-II by the end of FY-80 and on the CCM's by the end of FY-81.

References

1. Engel, Jr., F., et al., "Draft Proposed ANS FORTRAN," *ACM SIGPLAN Notices*, 11(3), March 1976.
2. Naur, P. (ed.), "Revised Report on the Algorithmic Language ALGOL 60," *Comm. ACM 6*, pp. 1-17, 1963.
3. McCarthey, et al., *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, Mass., 1966.
4. IBM, "PL/1 Language Specification," Form GY33-6003-2, White Plains, New York.
5. van Wijngaarden, et al., "Revised Report on the Algorithmic Language ALGOL 68," *Acta Informatica* 5,1-3, January 1976 (also appeared in SIGPLAN Notices).
6. Wirth, N., "The Programming Language PASCAL," *Acta Informatica 1,* Vol. 1, pp. 35-63, 1971.
7. Hoare, C.A.R., N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL," *Acta Informatica* 2, pp. 335-355, 1973.
8. Wulf, W. A., R. L. London, M. Shaw, "Abstraction and Verification in ALPHARD: Introduction to Language and Methodology," Computer Science Department, Carnegie Mellon University, June 1976.
9. Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchel, G. V. Popek, "Report on the Programming Language Euclid," ACM *SIGPLAN Notices*, 12(2), February 1977.
10. Ambler, A. L., et al., "Gypsy: A Language for Specification and Implementation of Verifiable Programs," Report ICSCA-CMP-2, University of Texas at Austin, January 1977.
11. Lauterback, C., C. Lucena, B. Moszkowski, "On the Use of MADCAP for the Development of Correct Programs," Computer Science Department, Internal Memorandum 132, UCLA, September 1974.
12. Berry, D. M., L. Chirica, Z. Erlich, C. Lucena, E. Walton, "STRUGGLE" Structured Generalized Goto-less Language," Internal Memo 147, Computer Science Department, UCLA, 1974.
13. Dembinski, P., G. Inglas, R. Schwartz, J. Takemura, "ALPO: A Language That's Proof Oriented," in R. Uzgalis (ed.), *Four Languages: Experiments in Computer Language Design*, UCLA-ENG-7544, Computer Science Department, UCLA, July 1975.
14. American National Standards Institute Inc., reports of periodic meetings of the committee in charge of Programming languages for the Control of Industrial Processes, ISO/TC97/SC5/WG1.
15. *Official Definition of CORAL* 66, Inter Establishment Committee on Computer Applications, ISBN 0 11 4702217, London, 1970.
16. PEARL Status as of October 1976, ANSI Committee Report ISO/TC97/SC5/WG1 N21, October 1976.
17. Fisher, D. A., "A Common Programming Language for the Department of Defense, Background and Technical Requirements," Institute for Defense Analysis, NTIS Distribution 77-000275, June 1976.

18. "Department of Defense Requirements for High Order Computer Programming Languages – Revised Ironman," July 1977 (appearing in ACM SIGPLAN Notices 12(12), December 1977).

19. Ross, D., "Homilies for Humble Standards," *Comm. ACM 9(11)*, pp. 1-14, 1976.

20. *MBASIC Fundamentals*, Vol. 1, Jet Propulsion Laboratory Internal Document 900-752, February 1974.

21. *MBASIC, Vol. 2, Appendices*, Jet Propulsion Laboratory, Internal Document 900-752, February 1974.

22. Wirth, N., "PL360-A Programming Language for the 360 Computers," *Journal of the ACM*, Vol. 15, No. 1, pp. 37-74.

23. Wegbreit, B., et al., "The ECL Programming System," *Proc. AFIPS* 39, pp. 253-262, 1971.