Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

LANGUAGE ASSESSMENT CRITERIA FOR DISCRETE SIMULATION

James W. Hooper
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

Criteria are suggested for use in conducting comparative assessments of languages for use in discrete simulation. The criteria are grouped within the categories of simulation-specific criteria and general criteria. A discussion is provided concerning the significance the various assessment criteria have in modeling and simulation. Suggestions are offered concerning the use of the criteria in a language selection process.

1. INTRODUCTION

A bewildering array of languages is available for any project involving computers. Occasionally the choice of language is dictated for a project -- e.g., because the firm's contract specifies use of a certain language, or because only one language is available currently, and time doesn't permit obtaining another. But in general a real choice is available, and in this case the choice should be carefully considered.

In Section 2, criteria are considered that pertain specifically to simulation. In Section 3, criteria are presented that could profitably characterize all programming languages. The intent is that available languages may be assessed against these criteria, in order that the choice may effectively support the goals of a simulation study. In the final section (Section 4) some suggestions are given concerning use of the criteria in language selection.

2. SIMULATION-SPECIFIC CRITERIA

In a great many application areas, including simulation, a choice is available between general purpose languages and application-specific languages. The choice between general purpose languages and simulation languages often comes down to weighing the benefits to be obtained from the built-in simulation-specific features against the often-greater generality and computational power of a general purpose language. This dichotomy has been addressed by some language designers in providing simulation languages with broad general-purpose computational features -- e.g., Simula (Dahl and Nygaard 1966) and SIMSCRIPT II.5 (Kiviat, Villanueva and Markowitz 1973), and by developing sets of simulation-specific procedures for use with a general purpose language -- e.g., GASP IV (Pritsker 1974), for use with FORTRAN, and SIMPAS (Bryant 1980), for use with Pascal.

There are a number of functions that must be performed with any simulation model, and which thus should be supported by the language used for model development. These necessary functions may be stated, in summary form, as:

(a) modeling the dynamics of a system,
(b) modeling a system's state,
(c) performance data recording and analysis.

It is important that a simulation language aid the modeler by providing "conceptual guidance" (Shannon 1975) -- i.e., providing a conceptual framework for development of the model. This provides a significant advantage over using a general purpose language, and having no such guidance for organizing and developing the model.

The exact nature of system dynamics modeling varies depending on whether the simulation is continuous, discrete, or combined. In the continuous case, for example, a system's dynamics may be achieved by integrating differential equations, and executing blocks of code as specified time values are reached. In the discrete case, dynamics are effected by moving the clock ahead to the time of the next event in the Event Set, and executing code to model the associated state transition -- frequently including generation of random variates. The exact approach depends on a language's strategy (or strategies), which may be one of (a) process interaction, (b) activity scanning, and (c) event scheduling (Kiviat 1971, Nance 1971, Nance 1981, Hooper 1986). A number of simulation studies can profit from use of combined discrete/continuous methods (e.g., as provided by SLAM II (Pritsker 1984) and SIMAN (Pegden 1984)).

A great deal is involved in modeling a system's state, but in the final analysis, the system state at any given time is represented by entries in data structures. Thus the ease with which system states may be expressed depends on the ease with which entities may be created and terminated, their attributes and relationships expressed, and entities grouped according to their attributes and relationships. To this end, it is desirable that a language provide convenient means for using linked lists and ordered sets, among other data structures; also, built-in features for creating, managing and terminating entities, and for representing their attributes and relationships, provide a significant advantage. A good range of functions for generating random variables is a basic requirement for discrete event simulation studies.

With regard to recording and analyzing data, such capabilities as automatic collection of certain types of performance data, and the means to easily generate graphs and charts, are very important. Interactive interfaces, including graphics, are especially attractive and helpful; the adage "a picture is worth a thousand words" is likely an understatement when attempting to assess system performance.

In somewhat more detail, we can thus expand the three functions listed above into the set of criteria for measuring simulation-specific features of a language, as shown in Figure 1.

3. GENERAL CRITERIA

To provide effective simulation support, a language should have considerable flexibility and computational power. For example, relative to modeling a system's dynamics, it is important to have a full range of features (control flow, arithmetic computation, data handling) as provided by a good general purpose language. There are numerous features that could profitably characterize all programming languages -- whether general purpose or simulation languages. These features may be divided into the categories of (a) design aspects and (b) environmental aspects. Figure 2 summarizes some important criteria against which all programming languages may be measured.

Software development has become an extremely complex and error-fraught endeavor, as software systems have become larger and more complex. The field of software engineering has been developed since 1968 to

MODELING A SYSTEM'S DYNAMICS
- Framework for Modeling and Simulation (providing support for conceptualizing/organizing a model)
- Time Management Features Supplied
  - Discrete Event Methods
    - Event Scheduling
    - Activity Scanning
    - Process Interaction
  - Combined Discrete/Continuous

MODELING A SYSTEM'S STATE
- Range of Data Structures and Operations Supplied (especially linked lists, ordered sets, entities with attributes and relationships)
- Random Variable Generation Features

PERFORMANCE DATA COLLECTION AND ANALYSIS FEATURES
- Features for Specifying and Collecting Performance Data
- Features for Obtaining Standard Reports
- Features for Designing Special Reports

Figure 1: Simulation-Specific Criteria for Evaluating Language Features

LANGUAGE DESIGN ASPECTS
- Understandable Language Concepts
- Understandable Language Syntax
- Computational/Representational Power and Flexibility
- Support for the Software Development Process

ENVIRONMMENTAL ASPECTS
- Availability
- Software Development Support Environment
- Computer Runtime
- Portability
- Standardization

Figure 2: General Criteria for Evaluating Language Features

attempt to solve the prevailing problems of late, over-budget software, characterized by numerous errors, and difficult to understand and modify. Large simulation systems are examples of very complex software. Thus to as great an extent as possible the proven approaches of software engineering should be employed in the development of simulation software, including emphasis on activities throughout the "software life cycle". This implies having languages that support top-down, modular development, that are readable, that help to reduce the creation of errors during software development, that ease the problem of isolating errors when they occur, and that have conceptually simple, yet powerful, features. The criterion of "Support for the Software Development Process" of Figure 2 is intended to summarize such aspects of languages. Wiener and Sincovec (1984) state the opinion that languages which offer support in the following areas provide the basis for constructing reliable and maintainable software: (1) readability, (2) modules for modular software construction, (3) separate compilation with strong cross-reference checking, (4) the control of side effects, (5) data hiding, (6) data abstraction, (7) structured control of flow, (8) dynamic memory management, (9) type consistency checking between various subprograms, and (10) runtime checking.

The emphasis on software engineering techniques has impacted language design. For example, strong typing characterizes a number of languages (e.g., Pascal, Ada); simulationists are seriously considering strong typing languages for simulation. In February 1984 the Conference on Simulation in Strongly Typed Languages was held, under sponsorship of the Society for Computer Simulation (see Bryant and Unger 1984). As may be seen from the papers in the proceedings, and references in those papers, considerable activity is underway in this area.

Software development environments are becoming more and more important, and relate directly to the goal of developing error-free software more quickly -- i.e., of making programmers more productive. In order to adequately assess a general purpose language or a simulation language, the quality of the support environment must be considered. Howden (1982) categorizes tools in a software development environment by software life cycle activities; in particular, by tools and techniques for (1) requirements, (2) design, (3) coding, (4) verification, and (5) management. The Ada program support environment (Defense Advanced Research Projects Agency 1980, Sommerville 1985) consists of such tools as a run-time support system, database primitives, peripheral device interfaces, a compiler for Ada, an editor, a loader, a debugger, and a configuration manager.

The complexity of simulation models, including parallelism, makes the availability of an integrated set of support tools especially important. Substantial research has been performed as to what should constitute an effective "modeling and simulation support environment" -- i.e., a software development environment especially for modeling and simulation. Nance (1984) suggests several aspects that should characterize such an environment, including extensions to software engineering environments especially for modeling, program generators (e.g., DRAFT and CAPS), system specification languages, and a modeling methodology -- such as Nance's Conical Methodology (see Nance 1984 for references for DRAFT, CAPS, and the Conical Methodology). Overstreet and Nance (1985) present a description of a language (called CS, for condition specification) for specifying models at a level between a conceptual model and an implementation of the model. The advantages suggested in using this intermediate-level specification include improved error detection and analysis, and automated production of some kinds of documentation. Balci (1986) presents a full set of requirements for a model development environment, based on the Ada program support environment.

Reese and Sheppard (1983) describe a prototype "simulation software development environment" (SSDE), based on a series of databases, a language, and supporting tools, which automates some aspects of phase-to-phase transitions in the model life cycle. Joyce, Birtwistle and Wyvill (1984) describe "ANDES", a simulation environment with strong emphasis on animation for model specification, debugging and validation. Henriksen (1983b) suggests what the characteristics of simulation tools of the future may be. Working from a common "knowledge base" would be a number of tools -- including a model editor, an input preparation aid, a statistics collection facility, an experimental design facility, and an output definition facility, along with program editor, compiler, and runtime support. The emphasis is on the integration of these tools with respect to the knowledge base. Current research progress suggests that Henriksen's expectations are reasonable.

The current trend toward modeling and simulation support environments has already resulted in commercially-available systems; they do not presently achieve all the functions mentioned by Henriksen (1983b) and others, but they are steadily progressing toward such goals. One such system is TESS, developed by Pritsker and Associates for use with SLAM II (Pritsker 1984). Standridge (1985) gives an overview of TESS; the support capabilities of TESS include (1) graphically building SLAM networks, (2) forms entry of control and data, (3) database management of user-defined model parameters, inputs, and simulation-generated data, (4) preparation of reports and graphs, (5) analysis of simulation results and (6) the animation of simulation runs. An agreement was recently announced between Pritsker and Asociates and Wolverine Software Corporation, whereby TESS will be made available to users of GPSS/H (Henriksen 1983a). Systems Modeling Corporation has developed around SIMAN

(Pegden 1984) a set of integrated tools which provide for graphical building of models, data collection during simulation runs, graphical preparation of simulation results, and simulation of concurrent animation.

Modeling and simulation support environments will no doubt be an increasingly-important criterion by which simulation languages are measured. Expert systems hold promise in easing the development of simulation models and interpretation of simulation results. Shannon, Mayer and Adelsberger (1985) provide a summary of the likely potential of expert systems with regard to simulation.

4. SUGGESTIONS FOR APPLYING THE CRITERIA

If we are considering various languages for use in a simulation study, we may assess their features against the lists of criteria in Figures 1 and 2. An important general criterion for programming languages is naturalness for a given application. Since our emphasis here is primarily on simulation studies, it follows that a language that measures up poorly against the simulation-specific criteria of Figure 1, will not be especially "natural" for this application; on the other hand, a language which exhibits most of the simulation-specific criteria, may rate poorly overall if some of the general criteria (of Figure 2) are not met.

A suggestion for use of the criteria in comparative language assessment is to create a table with the criteria of figures 1 and 2 placed vertically at the left side of the table, and with languages to be assessed constituting the "horizontal axis". Suggested descriptive words for use in assessments are: none, poor, fair, good, very good, and excellent. In the case of time management features, availability may be indicated by use of yes or no.

There is no best language, in any absolute sense. Thus in most instances one should not simply assign equal weights to all criteria, rate a set of languages, "add columns", and pick the language with the "high score". Each project has its own priorities and resource limitations (e.g., human skills, time, money), and thus each language selection activity gives rise to a set of "weighting factors" for each of the criteria of figures 1 and 2, determined specifically for the project. For example, availability of an effective modeling and simulation support environment likely deserves considerably more emphasis than computer runtime in most projects; yet in a few situations runtime may be a very critical criterion. And, as another example, lack of the combined discrete/continuous time management feature may be of no consequence whatever in some projects. Of course one may choose not to use numerical weights, per se. But in any case, the relative importance of the criteria should be carefully considered in assessing the comparative advantages of languages.

The reader who wishes to pursue further the ideas discussed here, may find a good discussion of desirable general language features in Horowitz (1984) and Pratt (1984). Shannon (1975), Banks and Carson (1984), Shub (1980), and Law and Kelton (1982) give good discussions of simulation-specific language features. The subject of language support to the software development process is treated at length by Weiner and Sincovec (1984) and by Ghezzi and Jazayeri (1982). Nance (1984) provides a good introduction to some of the issues underlying model development environments, along with a brief historical summary and a status summary. The implications of discrete event language strategy for model development and execution is considered in Hooper (1986).

REFERENCES

Balci, O. (1986). Requirements for Model Development Environments. Computers and Operations Research 13, 53-67.

Banks, J., and Carson, J.S.,II (1984). Discrete-Event System Simulation. Prentice-Hall, Englewood Cliffs, New Jersey.

Bryant, R.M. (1980). SIMPAS: A Simulation Language Based on Pascal. In: Proceedings of the 1980 Winter Simulation Conference (T.I. Oren, C.M. Shub, P.F. Roth, eds.), IEEE, New York, 25-40.

Bryant, R., and Unger, B.W. (1984). Simulation in Strongly Typed Languages: Ada, Pascal, Simula .... Society for Computer Simulation, San Diego, California.

Dahl, O.J., and Nygaard, K. (1966). SIMULA--An ALGOL-Based Simulation Language. Communications of the ACM 9, 671-678.

Defense Advanced Research Projects Agency (1980). Ada Reference Manual. United States Department of Defense, Washington, D.C.

Ghezzi, C., and Jazayeri, M. (1982). Programming Language Concepts. Wiley, New York.

Henriksen, J.O. (1983a). State-of-the-Art GPSS. In: Proceedings of the 1983 Summer Computer Simulation Conference. Society for Computer Simulation, San Diego, California, 918-933.

Henriksen, J.O. (1983b). The Integrated Simulation Environment (Simulation Software of the 1990's). Operations Research 31, 1053-1073.

Hooper, J.W. (1986). Strategy-Related Characteristics of Discrete-Event Languages and Models. Simulation 46, 153-159.

Horowitz, E. (1984). Fundamentals of Programming Languages, Second Edition. Computer Science Press, Rockville, Maryland.

Howden, W.E. (1982). Contemporary Software Development Environments. Communications of the ACM 25, 318-329.

Joyce, J., Birtwistle, G., and Wyvill, B. (1984). In: Proceedings of the 1984 UKSC Conference on Computer Simulation (D.J. Murray-Smith, ed.). Butterworths, Boston.

Kiviat, P.J. (1971). Simulation Languages. In: Computer Simulation Experiments with Models of Economic Systems (T.H. Naylor, ed.). Wiley, New York, 406-489.

Kiviat, P.J., Villanueva, R., and Markowitz, H.M. (1973). Simscript II.5 Programming Language, Second Edition. C.A.C.I., Los Angeles, California.

Law, A.M., and Kelton, W.D. (1982). Simulation Modeling and Analysis. McGraw-Hill, New York.

Nance, R.E. (1971). On Time Flow Mechanisms for Discrete System Simulation. Management Science 18, 59-73.

Nance, R.E. (1981). The Time and State Relationships in Simulation Modeling. Communications of the ACM 24, 173-179.

Nance, R.E. (1984). Model Development Revisited. In: Proceedings of the 1984 Winter Simulation Conference (S. Sheppard, U. Pooch, D. Pegden, eds.). IEEE, New York, 75-80.

Overstreet, C.M., and Nance, R.E. (1985). A Specification Language to Assist in Analysis of Discrete Event Simulation Models. Communications of the ACM 28, 190-201.

Pegden, C.D. (1984). Introduction to SIMAN. Systems Modeling Corporation, State College, Pennsylvania.

Pratt, T.W. (1984). Programming Languages, Second Edition. Prentice-Hall, Englewood Cliffs, New Jersey.

Pritsker, A.A.B. (1974). The GASP IV Simulation Language. Wiley, New York.

Pritsker, A.A.B. (1984). Introduction to Simulation and SLAM II, Second Editon. Halsted Press, New York.

Reese, R., and Sheppard, S. (1983). A Software Development Environment for Simulation Programming. In: Proceedings of the 1983 Winter Simulation Conference (S. Roberts, J. Banks, B. Schmeiser, eds.). IEEE, New York, 419-426.

Shannon, R.E. (1975). Systems Simulation: The Art and Science. Prentice-Hall, Englewood Cliffs, New Jersey.

Shannon, R.E., Mayer, R., and Adelsberger, H.H. (1985). Expert Systems and Simulation. Simulation 44, 275-284.

Shub, C.M. (1980). Discrete Event Simulation Languages. In: Proceedings of the 1980 Winter Simulation Conference (T.I. Oren, C.M. Shub, P.F. Roth, eds.),Volume 2. IEEE, New York, 107-124.

Sommerville, I. (1985). Software Engineering, Second Edition. Addison-Wesley, Reading, Massachusetts.

Standridge, C.R. (1985). Performing Simulation Projects with The Extended Simulation System (TESS). Simulation 45, 283-291.

Wiener, R., and Sincovec, R. (1984). Software Engineering with Modula-2 and Ada. Wiley, New York.

AUTHOR'S BIOGRAPHY

JAMES W. HOOPER is an associate professor of computer science at the University of Alabama in Huntsville (UAH). He teaches and conducts research in the areas of programming languages and compilers, software engineering, systems prototyping and discrete simulation. He is currently conducting sponsored research in prototyping and simulation approaches for distributed systems, for the U.S. Army Strategic Defense Command. Prior to joining UAH in 1980, he was employed by NASA Marshall Space Flight Center, where he conducted research in simulation approaches for data management. He has a B.S. in mathematics from the University of North Alabama, M.S. in mathematics from Auburn University, M.S. in Computer Science from the University of Missouri at Rolla, and Ph.D. in Computer and Information Sciences from the University of Alabama at Birmingham. He is a member of SCS, ACM, and the IEEE Computer Society.

James W. Hooper
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899
(205) 895-6515