SIMULATION LANGUAGE FEATURES IN 1976: EXISTING AND NEEDED

Louis W. Miller and Howard L. Morgan

Dept of Decision Sciences - Wharton School
University of PA, Philadelphia, PA 19174

Introduction

It is now 1976, more than 20 years since computer simulation was first recognized as an area which had a need for special purpose computer languages and languages features. Many of the signs of maturity are present in the field: Two annual simulation conferences, courses in most major universities, and a number of standard textbooks and bibliographies[1,6,4]. In addition, there have been several surveys of simulation languages, e.g., [7] and we have no wish to repeat such a survey here. Rather, we atempt to describe a set of features that one should be able to "expect" any simulation language to have, and to discuss a set of features which are still needed - especially in the growing worlds of interactive computing and computer graphics and data bases.

We assume that the reader is familiar with at least one of the major simulation language systems, SIMSCRIPT, GPSS, SIMULA, GASP, or the like. Also, we assume that the reader has an interest in the validity of his or her simulation results, and hence that statistical terms and methods will be referred to without detailed explanation.

Existing Features That We Expect

Discussed below are those features which the simulation programmer should be able to expect in any simluation language system.

1. Data Structures and Memory Management. A modern simulation language can be expected to provide data structures and routines for their manipulation in forms such that the user can invoke them in a language natural for describing the model. Normal array type structures are not generally sufficient for providing this. Some form of list processing is most often used to handle the memory management.

   Within this class of features, we require:
   Representation of objects (entities). Since most simulations deal with some physical objects, the language must provide a means of describing the various classes or types of objects with which the model deals. The programming system should take care of setting up the recordsto hold the descriptors of individual objects within each class. Normally, the user should be able to refer to individual descriptors through names rather than positional codes.

   All of the major systems distinguish between permanent and temporary objects. For transient objects, some means of allocating and freeing space is provided.

   Object grouping, ordering and relationships. Simulation models usually involve complex relationships among objects, making it essential to be able to group objects together with various rankings imposed on the group, and to be able to relate objects to the owning object. Operations performed by simulation languages tend to do more of this sort of manipulation automatically than do algebraic languages.

2. Time Management. The data structures noted above are used to describe the static state of the system being modeled. All discrete event simulations operate by having changes of state occur at discrete point in simulated time called events. A simulation language must provide means for describing various classes of events and a method for sequencing individual events. A user model must contain a timing routine that is responsible for choosing the

next event and causing execution of the subprogram associated with each particular class of event. One of three methods is generally used for event selection in modern simulation languages.

Event sequencing operates by maintaining a time ordered list of event notices, which are transient objects created at points where it is determined that a future event is to take place.
In activity scanning the emphasis is on searching clocks and sets of binary conditions to find the next event which is ready to take place. Apart from questions of computational control, the former tends to give the user more control, but the activity scanning method requires the user to describe conditions under which an event can take place only once in writing in the program. Process oriented languages combine both methods. For example, an end- of processing event may be sequenced by event sequencing, but when a job has to wait for a machine to become idle (in a job-shop simulation), the start event comes about by an activity scanning routine.

3. Sampling from Distributions. The vast majority of simulation models incorporate sampling from probability distributions by Monte Carlo methods. Necessary for this is a well designed and tested pseudorandom number generator capable of repeating random number streams in different runs. The repeatability requirement is necessary for debugging and proper experimental design. Without a good built in generator, users tend to resort to ad hoc methods that have serious shortcomings. Along with pseudorandom generators, we would expect routines that supply random observations from a variety of common distributions, as well as from empirical distributions supplied as data.

4. Computational Capability. A simulation programming language must either have an algebraic basis, or links to an algebraic compiler. The world being simulated is usually a complicated place -- no matter how well thought out the design of special purpose simulation programming systems are, it rarely takes long for users to call for computational capabilities which the designers have not allowed for. Usually this comes about in trying to define complex decision rules or functional behavior.

5. Data Collection and Statistics. Obviously a simulation model is useless without some means for observing it. There are a number of areas here where the modeler can be helped: collection of observations, data reduction and computation of statistics, and display of results. Data collection can be performed by programming in statements directly, but this adds complexity and opportunity for error. Therefore, one expects languages to permit specification that all events of a given class cause certain items to be collected. Another approach is to allow the programmer to make global declarations, separate from the procedural statements describing the model, of what he or she wants measured.

   The common statistics such as means, counts, sums, variances, and histograms should be provided, both for time dependent and time independent objects. Finally, a convenient means for organizing and displaying results should be provided. Special features for this can take the form of automatic standardized output, report generators, or simplified output systems.

In addition to the above features, which generally relate to the language, there are features which relate to the general programming environment that one now expects in a simulation system.

1. Debugging Facilities. Obtaining an error free program can be more challenging in simulation work that other types of programming because of the complex data structures involved and because of the dynamic, stochastic nature of the execution path of the program. It is thus crucial to include run-time diagnostics such as checks to see if an object is made a member of a set prior to its removal, and to provide a trace of the flow ofprogram control.

2. File access systems. More and more simulations require large amounts of parameter data to start up. Often this data is contained in files stored on the same machine. A simple I/O interface to the host machine file system is expected.

3. Partial Compilation. Since simulations often grow large and complex, the ability to change a single module without requiring recompilation of the entire system is importnat and usually provided.

The above set of features should be, and is found to varying degrees in the major simulation language systems. What we wish to direct our attention to now are those featuress which are not always found in these systems.

Features which are needed

We have discussed the need for better debugging facilities for simulation programs than is often provided with standard systems. Yet in the use of online interactive systems, simulations have been somewhat tardy in providing the user with the power of interactive debugging. WIth the notable exception of GPSS-Norden, one does not usually see source language interactions between the simulation modeler and the running program.

The second major area where improvements are needed is in the statistics area. For the past several years, there have benn repeated discussions in the literature of the troubles with the standard random number generators provided by certain manufacturers. There has been less discussion, however, of the troubles caused by the collection of data and the forming of statistics which are misleading and biased. Fishman[2] and Iglehart[3] have been reporting recently on better techniques for generating random numbers, and for gathering and analyzing simulation statistics. The language builders owe it to the simulation community to place these features within the reach of the average user.

Over the years, a number of techniques for variance reduction have been described. Again, the technique of antithetic variates is the only one which is available directly in a language system. This and other techniques, such as importance sampling, should be included as standard language features.

The major requirement for modern simulation language systems is an ability to integrate the simulation modeling effort with the rest of the organization's database. In those cases where the simulation output is needed to schedule a plant, or direct operations, cumbersome special purpose links between the simulation outputs and the normal organization's database input have had to be developed. Since many of the concepts of modern data base technology are quite similar to those of simulation data structures, it should be a simple matter to permit direct access to databases both for obtaining simulation paramters and for depositing results. Unfortunately, this is not yet so.

Finally, except for some simple uses in plotting data directly from simulation statistics, there has been little imaginative use of computer graphics as an input/output tool for simulations. Interfaces to graphics languages might permit modelers to provide more meaningful displays to the managers for whom the results are intended. For example, showing the actual buildup of queues in traffic simulations through a pictorial representation would have more impact than reading the tables of numbers, or even looking at a plot against time. Also, input via graphics has been grossly underestimated. The special purpose graphics input for traffic simulations[5] which one of us used in a batch mode 8 years ago would be a natural means of getting input with todays interactive graphics devices.

Summary and Conclusions

Simulation languages have come quite far over the past 20 years. If the goal of the language designers is to enable non-specialists to write efficient, effective simulations, with reasonable statistical properties, there is still work to be done. We hope that this work will succeed so that over the next few years our "great expectations" list can be fulfilled.

References

1. Fishman, G. Concepts and Methods in Discrete Event Digital Simulation. John Wiley and Sons, 1973.

2. Fishman, G. "Some Test Results on the SIMSCRIPT II.5 and SIMPL/I Psuedorandom Number Generators," Technical Report 76-11, University of North Carolina.

3. Iglehart, D. "Simulating stable stochastic systems VI: quantile estimation," Control Analysis Corporation Report 86-15.

4. Meier, R., Newell and Pazer, Simulation in Business and Economics, Prentice-Hall 1969.

5. Morgan, H. "UTS-I: A Macro System for Traffic Network Simulation," Proc. AFIPS SJCC 1970.

6. Naylor, T., Balintfy, et. al. Computer Simulation Techniques. John Wiley and Sons, 1966.

7. Teichroew, D. and J. Lubin, "Computer Simulation Discussion of the Technique and Comparison of Languages," Comm. ACM 9 (October 1966).