1981 Winter Simulation Conference Proceedings
T.I. Ören, C.M. Delfosse, C.M. Shub (Eds.)

DISCRETE EVENT SIMULATION ON MINI- AND MICROCOMPUTERS: SOME EXPERIMENTS WITH THE PASCAL LANGUAGE

Andrew F. Seila
Der-Fa Robert Chen
University of Georgia

ABSTRACT

Currently, mini- and microcomputers are finding application in many areas of business and scientific work. However, the use of small computers for simulation studies is virtually nonexistent. In this paper, we discuss the language requirements for discrete event simulation and present the features of the Pascal language that make it a natural language to use for writing simulation programs. Finally, we discuss our experiences in developing several nontrivial simulations using Pascal.

1. INTRODUCTION

Since the introduction of the microcomputer in 1975, computing has undergone a transformation so radical that one could call it a metamorphosis. The costs of mini- and microcomputers (small computers) have steadily decreased, while the capabilities, in terms of hardware capability and software availability have increased. As a result, small computers are finding convenient application in areas of production and inventory control, accounting, marketing management, and many other problem areas. In contrast, with the exception of the myriad of computer games available, small computers have had virtually no effect upon the use of simulation as a problem solving tool. In this presentation, we discuss the use of Pascal, a language that is widely available on small computers, for discrete event simulation.

2. DISCRETE EVENT SIMULATION AND SIMULATION LANGUAGES

Discrete event simulation involves the realization, within a computer, of a system model that changes state only at discrete points in time, called events. The model consists of a collection of entities which have attributes and may belong to sets. Changes of the state of the system involve changes in the number of entities in the system, their attributes, and/or their set membership. For example, in a queueing model, the system consists of two types of entities--customers and servers. Customers may have attributes that describe their priority or record information (e.g., arrival time); servers' attributes indicate their status (idle or busy) and may include specific information such as service rate. The queue in which customers wait for service is a set to which customers may or may not belong, depending upon whether a server is available upon arrival.

In order for a discrete event simulation to move through time, the program must maintain a list of events, ordered by time of occurrence. This is simply a special type of set which contains a special kind of entity: event notices. Event notices must have at least two attributes: the time of occurrence and the identity of the event that occurs at that time. For example, in the queueing simulation, there are two events which result in state changes for the system--the arrival of a customer and the completion of a service. For each type of event, the simulation must have a routine which makes the appropriate state changes in the system.

In order to realize a discrete event simulation, the language used must provide facilities for representing entities, attributes and sets, for manipulating the entities in sets (inserting and removing them, and searching through the set), and for doing scientific computations (floating point). The simulation languages SIMSCRIPT, SIMULA, GPSS, GASP, and SLAM, which are available only on large mainframe computers, provide these features.

Recently, Bryant (1981a, b) has developed a simulation language called SIMPAS which is based upon the Pascal language. SIMPAS consists of a preprocessor which converts source statements into Pascal statements. The Pascal program must then be compiled, linked and executed. SIMPAS has been implemented on a PDP VAX 11 mini-computer and a shorter version, called Micro-SIMPAS (Bryant, 1981c), has been implemented on a Terak 8510/a Graphics Computer. The only simulation that has been developed and reported using SIMPAS is that

81CH1709-5/81/0000-0041$00.75

of an M/M/1 queue.

3. THE PASCAL LANGUAGE AND DISCRETE EVENT SIMULATION

Pascal is a block structured programming language, similar to PL/1, that was developed in 1971 by Nicholas Wirth (Jansen and Wirth, 1974). The language is designed with two main objectives: to facilitate systematic programming, and to allow generation of efficient code on currently available computers. Much attention has been given to standardizing Pascal and to making the code thus generated portable. Currently, Pascal compilers are available for computers costing as little as $3,000-$4,000, and there is some reason to believe that Pascal might become the standard programming language for microcomputers in the future.

A review of all of the features of Pascal is beyond the scope of this short paper. We refer the interested reader to the textbooks that are currently avaáilable. The following is a list of the special features of Pascal that make it a natural language for discrete event simulation:

Record variables - A record is a collection of data elements which may be of different types. This provides a means for representing entities and their attributes, and event notices.

Pointer variables and dynamic data types - Pascal allows the dynamic creation and disposal of record variables. Pointer variables allow one to access the contents of dynamically created records. This provides a means for creating and destroying entities and event notices, and performing the activities necessary for entities and event notices to belong to sets.

Scalar data types - The programmer can define the "set of values" of a variable. This is primarily useful in making the simulation program self-documenting.

4. DEMONSTRATION PROJECT

The authors have conducted a project whose objective was to demonstrate the feasibility of using Pascal as a discrete event simulation language, and therefore, by implication, to demonstrate the feasibility of using small computers to perform nontrivial simulations. This project consisted of two efforts:

1. Development of several realistic discrete event simulations; and
2. Development of a library of routines (functions and procedures) commonly used in discrete event simulations.

All programs were developed on a Texas Instruments 990 Model 10 minicomputer using TI Pascal. The models simulated were:

a. A priority queueing system;
b. The oil tanker model in Pritsker (1974, p. 201).
c. A model of lumber yard and dry kiln operations;
d. A production-inventory system; and
e. A multi-item inventory system.

5. PROJECT RESULTS

The project showed that it is possible to develop discrete event simulations using Pascal with essentially the same level of effort that is required using a special purpose simulation language. This means that relatively sophisticated, realistic simulations are not only possible, but practical on microcomputers costing as little as $3,000-$4,000. The limitation on the possible size and complexity of the simulation is imposed primarily by the size of the computer memory and the programmer's imagination.

In fact, we have found that Pascal may have some advantages over discret event simulation languages. Since Pascal is not a simulation language, the responsibility for maintaining the list of scheduled events and operating the timing routine rests with the programmer. The pedagogic value of this in a simulation course is clear since these aspects are hidden from the programmer's view in a simulation language. Moreover, if the simulation is being developed outside the classroom, the programmers gain added confidence in the final product by being able to "see" everything that goes on. Finally, if efficiency were a consideration, it is likely that a program written "from the ground up" in Pascal would be somewhat more efficient than one written using a simulation language. This results from the fact that activities such as scheduling events or searching through sets are performed in simulation languages by general routines having options that frequently are not used. In Pascal, these options need not be included (if they aren't needed) and the program does not have to spend time checking for them.

We also found that Pascal has some noteworthy disadvantages, relative to simulation languages. Since the programmer is responsible for all activities in the simulation program, the danger exists that subtle bugs can be introduced. For example, in our first simulation program we had a bug that was caused by using the time attribute of the current event after it had been changed and scheduled as another event. This problem would not have been allowed by a simulation language. A certain amount of discipline on the part of the programmer can avoid most problems of this type. A second major disadvantage of using Pascal (and, for that matter, any general purpose language) is that routines for doing common activities such as scheduling events, cancelling events, searching through sets, etc., must be provided by the programmer. We have developed a small library of commonly used routines to help overcome this problem. Some languages, such as GASP, have built-in routines for displaying data in plots. Routines of this type are currently not available in Pascal; however, we hope they will be available in the

future.

Finally, we feel that the block structure of Pascal is a major advantage in writing simulation programs. First, the block structure, combined with scalar data types and the ability to use identifiers of any length, allows programs to be almost self-documenting. Since working simulation programs are quite complex it is very important that they be thoroughly documented, and Pascal makes much of this effort quite natural. Secondly, the block structure of the language allows a team to use a structured approach to program development. This approach simplifies the problem of parcelling out the programming assignments and helps assure that the parts will fit together correctly.

In summary, using Pascal, we have found that it is possible to develop discrete event simulations on mini- and microcomputers with approximately the same amount of effort required using a simulation language. This means that, if simulation practitioners choose to use Pascal, much simulation work can now be done on small computers. This fact, we feel, has the potential to greatly alter simulation practice by moving it from the large, centralized organizations with access to a large computer, to any enterprise with access to a small computer.

REFERENCES

Bryant, R. M. (1981a), SIMPAS User Manual, Technical Report No. 54, Academic Computing Center, The University of Wiscon, Madison.

Bryant, R. M. (1981b), SIMPAS: A Simulation Language Based on Pascal, Report No. 55, Academic Computing Center, The University of Wisconsin, Madison.

Bryant, R. M. (1981c), Micro-SIMPAS: A Microprocessor-Based Simulation Language, Paper presented at the Annual Simulation Symposium, Tampa.

Jensen, K. and N. Wirth, Pascal: User Manual and Report, Lecture Notes in Computer Science 18, Springer-Verlag, Berlin, New York.

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