A NEW LOOK AT PROCESS ORIENTED SIMULATION LANGUAGES

Jair M. Babad and Linus Schrage
Graduate School of Business, University of Chicago, Chicago, Illinois

1. Introduction

The role of simulation as a research and analysis tool is widely recognized, and many simulation programming languages have been developed to facilitate the translation of simulation models into executable simulation programs. These languages may be (broadly) classified according to the underlying modelling approach as continuous (e.g., CSMP/360,DYNAMO) or discrete. The discrete languages may be further classified as event-oriented (e.g., SIMSCRIPT, GASP), activity-oriented (e.g., ECSL), or process-oriented (e.g., GPSS, SIMULA, and recent versions of SIMSCRIPT II.5). Of these, the process orientation is probably the most widely used due to the popularity of GPSS, which is attributable in part to IBM's support, as well as to the naturalness of the process orientation for describing simulated systems. Even SIMSCRIPT, the most widely used event-oriented simulation language, has recently added process-oriented commands to its vocabulary. Due to this popularity of the process orientation, it seems appropriate to restrict attention to this set of language types.

All process simulation languages (PSL) essentially translate a simulation process flow-chart into a set of blocks, or subroutine calls, within the language. During the simulation transactions (or processes in computer science terminology) are generated and move through the process blocks, and various statistics are collected. The processing of transactions and statistics might involve various computations and reports. PSL's provide facilities and commands for all these processing requirements, but these provisions lag behind current developments in programming, simulation and statistics. In particular, PSL's do not take into account the time-series nature of the statistical phenomena generated by the simulation. Most PSL's are designed for a batch-processing environment, in contrast to the growing popularity of interactive computer processing. PSL's also seem to ignore the model construction process, as some of them include very crude and limited debugging facilities.

These deficiencies are undoubtedly attributed to the "archaic" nature of currently successful PSL's. GPSS, the most widely used PSL, was designed in the late fifties, and its main commands and capabilities remain as they were originally implemented. In particular, GPSS is virtually useless for ordinary computational and processing tasks — a most disturbing fact, as GPSS is mostly model, and not statistics, oriented. SIMSCRIPT, though a full-fledged programming language, has only recently added a limited vocabulary of process oriented commands. It is heavily commited to event-oriented simulation, a commitment which is reflected in the nature of simulation models in SIMSCRIPT. The process oriented user who would like to use SIMSCRIPT should therefore pay a high premium for facilities that he does not need, both in comprehension time and in processing costs. SIMULA is an extension of ALGOL and is powerful in some respects but still lacking in other respects.

These observations motivated our interest in a new look at process simulation languages. We first describe the modelling, statistics gathering and programming requirements that, we believe, should be accommodated by any PSL. This description leads us to a set of principles and guidelines for a proposed (new) PSL. These guidelines are directed at the capabilities of the language, rather than at grammatical and syntactical aspects of it. We hope that these guidelines will lead (eventually) to the implementation of such a language, and to an even wider use of process simulation.

2. Modelling and Programming in Process Simulation

Process simulation is based on a process flow chart, which follows the progress of a transaction through the simulated system, from its arrival event to its departure event. The interaction between a specific transaction, the various elements of the system, and other transactions, is described by a set of blocks. A PSL accepts a description of such blocks and flow charts, and simulates the system either interpretively, or through compiled code (of the accepted process flow charts). Generally, each block in a flow chart corresponding one PSL instruction — a subroutine call or a block macro. For simplicity, we will denote these statements as blocks in the sequel.

As is clear from this description, a PSL should contain blocks that describe the progress of a transaction through the system. These include generation, advancing and termination blocks. However, in contrast to event simulation, a PSL should take into account interaction among transactions, which may result in conditional advancement or waiting. This "conditional progress" requires a complicated and sophisticated "clock" — or an (underlying) list manipulation mechanism; clearly, an efficient operation of this mechanism is essential to effective simulation. In particular, a user who knows his simulated model, and who can manipulate the events list mechanism, might achieve considerable savings in computing time. Current PSL's, however, include very limited capabilities for user's manipulation of the events list. While this approach makes PSL design easier, it is less desirable from the user's point of view.

The capability to manipulate the clock mechanism is of particular importance in an interactive simulation, in which the user might decide to change the flow of events or skip some events. The advantages of such interaction are self-evident, and in accordance with the experimental nature of simulation. Further, such an interactive capability would tremendously enhance the debugging of simulation models. Debugging is an arduous task by itself; but when it is coupled with the random nature of simulation, the limited control of the user on the system's progress, and the very limited report generating capabilities of some PSL's, it becomes an almost impossible task. Interactive simulation is therefore very important for debugging complex simulation models.

On the other hand, it is very clear that interactive simulation might lead to deadlock situations and increased overhead, and therefore must be carefully designed. This is one area in which current PSL's are clearly deficient, and which requires much work by designers of any new PSL.

Another area in which current PSL's are deficient are limited computational capabilities. This is the case mainly with GPSS, because of its INTEGER inherent mode, the cumbersome FUNCTION and VARIABLE blocks, and the lack of standard functions. A possible solution is the use of a full-fledged programming language--e.g., to link a FORTRAN or PL/I module to a GPSS program, or to learn the whole SIMSCRIPT language. This solution is a hindrance to many users who would like to use simulation without overburdening themselves with many programming details.

Another programming aspect which is alien to the process simulation, but is necessitated by current PSL, is the declaration or preamble, section of the simulation program. In this section the user defines the entities to be included in the simulation and their interrelations. Historically, all the early programming languages required such a section in users' programs. But the experience of APL shows that many of the declarations in this introductory section might be eliminated. As this section is usually the source for many programming errors its (partial) elimination is clearly desirable.

In the same spirit, the structure of SIMSCRIPT programs is far from process oriented because of its need for a PREAMBLE and MAIN. As a result, a simulation program in one of these languages does not resemble the model as closely as a GPSS program does, and much of the process simulation appeal and naturalness are eliminated. On the other hand, GPSS is (too much) oriented toward assembly language, and its instructions are too rigid. In particular, they lack the "English-like" form of SIMSCRIPT.

3. Statistics Collection in Simulation

All current PSL's have single instructions or blocks which do global statistics collection. Examples of these are the QUEUE-DEPART and TABULATE blocks of GPSS, and the TALLY and ACCUMULATE of SIMSCRIPT. In addition, each of these languages has some words (or names) that are used solely for statistical purposes, like some of the SNA's in GPSS, and MEAN of SIMSCRIPT. However, the approach of PSL to statistics is not less archaic than some of their modelling and programming concepts, as they do not per se take into account the time series nature of the statistical phenomena that are generated by the simulation. In particular, present day built-in statistics collection instructions compute estimates of variance as if the individual observations were independent, whereas it is well known that the waiting times, say, of jobs in a shop tend to be highly serially correlated. A PSL should enable the user to easily collect time series statistics, and, at the minimum, serial correlations and covariance matrices.

Another statistical facility that is of value is the capability to aggregate, or group, individual observations (like job waiting times), into larger macro-observations. This aggregation tends to reduce the correlation among macro-observations. Another aggregation mechanism should be data-dependent—e.g., enabling the user to group observations along the lines of the "regeneration point" approach of Babad [1], Fishman [6], and Crane and Iglehart [4].

The user should also have the ability to "preload" the system, discard certain intervals of initial data, or save the system status for future use. All of these capabilities are essentially nonexistant in current PSL, or are cumbersome and very restrictive.

It should be noted that much work should still be done on the empirical evaluation of the various methods proposed for analyzing simulation data. The works of Law [8], Babad [1], and Duket and Pritsker [5] indicate that data analysis methods should be thoroughly tested on a well understood system, regardless of the apparent reasonableness of these methods.

4. Guidelines for a Process Simulation Language

The discussion above demonstrates the need for a new PSL, which would overcome many, if not all, of the deficiencies in current PSL. In this section we present our ideas for the structure of such a language. We believe that this language is long overdue, and hope to see it implemented in the near future. For ease of reference, let us denote it as NPSL, i.e., the New Process Simulation Language. We would like the statement syntax of NPSL to be much like that of SIMSCRIPT II.

The basic structural element of NPSL is a block, in the sense of a PL/I block; i.e., a block in NPSL is a sequence of one or more instructions, which are logically related and can be referenced as a unit. A block will thus be used to describe the progress of a transaction through the system, a computation to be done, a report to be printed, a declaration of an entity, and the like. In other words, a block is used both in the role of a SEGMENT in GPSS, and a FUNCTION in FORTRAN. The use of blocks imposes modularity on the user, and corresponds to the current trend in program development and structure. Each block will be identified by its name (or label) and by a key word which would describe its purpose; e.g., COMPUTE block, PROCESS block, and the like. A reference to the name in other parts of the simulation program will then automatically involve the named block.

The modelling of a system will involve transactions, their progress through the system, and their interactions. Three types of objects will be recognized by NPSL: transactions, resources and sets. A transaction is considered to be the element whose progress through the system is described by a process block. It is GENERATEd, may be a MEMBER of a set and/or OWN another object, and eventually its progress through the system is TERMINATEd. A transaction may have as many attributes as needed for and during the simulation; these attributes might be set automatically by the NPSL, e.g., when they have been declared, or might be set dynamically by the user. A transaction USE's a resource; this use automatically manages the involved resources. Finally, a transaction may be SCHEDULEd for some future time; in this case the transaction should be deactivated till this future time, when the NPSL will automatically activate the transaction. It should be noted that we envision the USE and SCHEDULE as very general operations, which may be conditioned on (essentially) any thing that happens and/or exists in the system. In other words, a transaction might USE a resource WHILE no other transaction with higher priority uses it, or it might USE resource A WHILE resource B is being USEd by a transaction with a given attribute, etc. Consequently, USE and SCHEDULE will accommodate simple FIFO queues, as well as complex priority and preemption situations. Note that we do not distinguish between priorities and other attributes; thus, for example, a transaction might have many priorities, each for the use of another resource.

Resources correspond to FACILITIES and STORAGES in GPSS. Each resource has a capacity, is associated with a rate (or distribution) of service, may be LOCKed (or shutoff), etc. Further, the USE of a given resource might be conditioned, e.g., resource A may be USEd by transactions with priority exceeding B, or only when transactions with attribute C are also using this resource.

Finally, sets might be used for automatic grouping of other entities—either for statistical and reporting purposes, or for control of the system's progress. Thus, the progress of a transaction might be delayed due to its inclusion in some set, etc. We allow very general set structures — both hierarchical and network type sets. For example, entities in a family might be described by a hierarchical set, in which there is a set of parents which owns a set of children. The parental set might include two elements—father and mother—without any imposed hierarchy. Further, a set might be defined as the last k entities of a certain type, or as a group of k elements. Thus, the system would allow, for example, for moving averages (by considering the average value for a set of the last k elements), or for group averages (of a set of k elements, which is then emptied and "restarted"). This grouping might also be conditioned on the occurrence of certain events, e.g., certain "regeneration points."

The language elements that have been discussed so far, are in our opinion sufficient for the modelling of any system. But, an NPSL clearly requires additional language elements. In particular, we should discuss briefly the (simulation) control, statistics collection, computational and programming aspects, and input/output facilities.

The control of the simulation should include both transactions progress control and run control. We already mentioned the activation/deactivation capabilities of SCHEDULE. We suggest including SCAN and MONITOR control blocks in the NPSL. The user should be able to MONITOR the system for the occurrence of a specific event, like a certain point in time, a queue that is filled to capacity, and so forth. When a MONITORed event occurs, the status of the system at this point will be saved; the user should thus be able to SCAN for this status, and re-run the system from this status on. Similarly, the user should be able to tell the NPSL that it should reSCAN the pending transactions list, that it should not SCAN certain transactions lists; or that it should SCAN to the next event; i.e., running the simulation "one event at a time," which is an especially useful debugging device.

Similarly, we propose to have for debugging purposes TRACE—UNTRACE and FLOW—UNFLOW blocks, which might be conditioned on the occurrence of certain events. Both should be activated, by the execution of desired blocks, as well as by the usage of requested names--either as receptors in assignments (i.e., when they appear on the left hand side of an equation) or elsewhere. Notice also that the MONITOR and SCAN capabilities, when combined, constitute a capability which is essential for debugging.

The MONITORing capability is of value when the system's overall control is considered. The user can reSTART the system from every MONITORed status, and thus can start his simulation at a preconceived status, discard initial, nonstable, data, and the like. The overall system's control will be performed by a CONTROL block. Note that parallel runs, which are used to compare alternative policies, might be easily performed; e.g., be STARTing with initial run, MONITORing the status at the end of the run, reSTARTing from this status for each alternative policy, and MONITORing the status at the end of each policy's run. Finally, a COMPUTE block could be used to compare the MONITORed policy's runs. In other words, we believe that the control elements specified above are powerful enough, and sufficiently flexible, to accommodate many simulations that are impossible, or cumbersome to perform in today's PSL.

For statistics collection, we propose automatic recording of attributes' changes, as is being done today by SIMSCRIPT (for elements which are defined in the PREAMBLE) or by GPSS (for SNA). However, as is true with any other capability of the NPSL, this automatic recording may be conditioned. Similarly, we envision automatic recording of entities' statistics, which include the standard measures like mean and variance. In addition, the user should be able to request automatic recording of covariances, serial correlations of certain orders, and similar statistics. Note that these capabilities, when applied to sets, allow the grouping of data which is as so desired for covariance reduction, as described in an earlier section. The NPSL would also automatically perform final statistical analysis, like confidence intervals, conversion of sums of squares to variances, etc.

Finally, a few words should be devoted to standard programming facilities. We would like to see a powerful report generator. This will be similar to the one used by SIMSCRIPT, or by advanced Management Information Systems. Each report will be defined by a block, which would be invoked when needed. The NPSL will also have standard computational facilities, including branching, looping (in a PL/I type Do ..WHILE.. style), etc. In particular, the NPSL should include powerful and reliable distribution generators among its standard computational functions.

5. Conclusion

In this paper we outlined our approach to process simulation languages. We started with a quick analysis of deficiencies in current process simulation languages, and then described the main features that (we believe) such a language should have. Our notions may be challenged by many, and we are sure that such a challenge will only improve the quality of such a proposed language, and enrich the whole field of process simulation.

REFERENCES

[1] Babad, J., "The IBM GPSS Random Number Generator," to appear in GPSS Application Cases, ed. by T. Schriber.

[2] Conway, R. W.; Johnson, B. M.; and Maxwell, W. L., "Some Problems of Digital Systems Simulation," Management Science, Vol. 6, No. 1, pp. 92-110, October 1959.

[3] Conway, R. W., "Some Tactical Problems in Digital Simulation," Management Science, Vol. 10, No. 1, pp. 47-61, October, 1963.

[4] Crane, M. A., and Iglehart, D. L., "Simulating Stable Stochastic Systems: II. Regenerative Processes and Discrete-Event Simulations," Operations Research, Vol. 23, No. 1, pp. 33-45, January, 1975.

[5] Duket, S. D., and Pritsker, A. A. B., "Spectral Methods of Simulation Output," Technical Report, Purdue University, 1975.

[6] Fishman, G. S., "Bias Considerations in Simulation Experiments," Operations Research, pp. 785-790, 1972.

[7] Fishman, G. S., and Kiviat, P. J., "The Analysis of Simulation Generated Time Series," Management Science, Vol. 13, pp. 525-557, March, 1967.

[8] Law, A., "A Comparison of Two Techniques for Determining the Accuracy of Simulation Output," Technical Report, Department of I. E., University of Wisconsin, June, 1975.

[9] Merrill, F., and Schrage, L., "Efficient Use of Jurors: A Field Study and Simulation Model of a Court System," Washington University Law Quarterly, Vol. 1969, No. 2, pp. 151-183, Spring, 1969.