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

IMPLEMENTATIONS OF TIME (PANEL)

Chair:
Douglas W. Jones
University of Iowa
Iowa City, Iowa

Panelists:
James O. Henriksen
C. Dennis Pegden
Robert G. Sargent
Robert M. O'Keefe
Brian W. Unger

INTRODUCTION

In conventional discrete event simulation systems, the flow of simulated time is controlled by a data structure that is variously called the event set, the pending event set, the future-event chain, or the sequencing-set. At any instant, this structure contains records of those events, processes or activities that are to be simulated at some future time as a result of events that have already been simulated.

There are two basic operations on the event set: Scheduling an item at some future time, and retrieving the next scheduled item. There is little agreement between different discrete event simulation languages about what is scheduled and about what additional operations should be provided. For example, Simula 67 schedules coroutine activations in its sequencing-set, while GPSS schedules event records in its future-event chain. Operations such as finding the successors or predecessors of items or deleting previously scheduled items are sometimes required. Distributed simulation and mixed discrete and continuous simulation pose new constraints. For example, in distributed simulation, the event set is frequently partitioned over a number of machines, raising the problem of keeping the event set processing on these machines synchronized.

Event-Set Implementations. Kingston (1984) proposed five criteria that should be met by a successful event set implementation:

1. Efficiency - Operations should be fast for a wide range of event set sizes, and very fast for small event sets.

2. Robustness - The implementation should be efficient for any scheduling distribution.

3. Adaptiveness - The implementation should be able to take special advantage of simple scheduling distributions.

4. Generality - The implementation should be stable and should efficiently support a broad variety of of the less common operations.

5. Simplicity - The code for the implementation should be short and easily understood.

In searching event set literature, one can find claims that almost every event set implementation other than the simple linear list is optimal and should be used universally. In fact, many of the implementations that have been proposed are optimal for some combination of scheduling distribution, machine, and event set size, but most fail to live up to one or more of the above criteria. A complete survey of event set implementations is not possible in this context; their variety is comparable to that of searching or sorting algorithms. Those interested are referred to Jones (1986), Kingston (1984) and McCormack and Sargent (1981) for summaries of and references to a variety of implementations.

Among the better implementations that were investigated in the above-cited references were the indexed list implementation developed by Henriksen and the splay tree implementation developed by Sleator and Tarjan. Both of these live up to Kingston's criteria reasonably well, although splay trees have a better worst case but more complex code than Henriksen's implementation. If the scheduling distribution or the expected size of the event set are known in advance, a special implementation can be selected that is adapted to that combination; thus, the general adaptiveness and robustness criteria become less important.

The simple linear list, and some simple variants of it are almost certainly the best choice if it is known that the event set will stay small, although these can be quite inefficient for larger event sets. The median pointer method of Davey and Vaucher (1980) is one of the better choices for lists smaller than about 200. If it is known in advance that the scheduling distribution will be relatively uniform, simple binary trees perform quite well, but techniques such as the indexed list of Davey and Vaucher (1980) will probably perform even better. Neither of these is robust; both are as bad as a linear list for some scheduling distributions.

The Cost of Generality. A significant number of studies of the event set have avoided comparisons with implicit heaps and other unstable implementations, nominally because these are not sufficiently general. Since it is natural to assume that stability has a run-time cost, this leads to the assumption that implicit heaps may be faster

than general implementations of the event set. In fact, this is not true! Both McCormack and Sargent (1981) and Jones (1986) have shown that Henriksen's implementation is faster than implicit heaps, and Jones (1986) showed that splay trees can be even faster than Henriksen's implementation.

Although stability may indeed have no inherent cost, the ability to efficiently delete arbitrary event notices may impose a significant performance penalty. For the splay tree implementation, the penalty was a factor of about 1/3 according to Jones (1986). Thus, it is interesting to ask whether the deletion of previously scheduled event notices is really needed, or whether simulation models that require deletion can always be transformed so that deletion is not required.

In addition to requiring that the event set be stable, many discrete event simulation languages impose other requirements on the processing of simultaneous events. For example, the three phase approach requires that all scheduled events at some time be simulated before any conditional events they might cause at the same time. The reason that this kind of logic appears to have been introduced into simulation languages is that it increases the repeatability of simulation experiments and reduces their sensitivity to small changes in the models being used. Unfortunately, this can mask real instabilities in the systems being modeled. Although this may have helped sell early simulation systems (Gordon 1978), and it certainly aids debugging, this appears to be a dangerous practice and is certainly not worth paying any performance penalty.

The Prospects for Improvement. Early simulation systems frequently used simple linear lists as an event set implementation; as a result, changes in the event set implementation frequently led to performance improvements of as much as a factor of 100 in total simulation time for large models. This experience has lead many to conclude that the choice of an appropriate event set implementation is crucial to the performance of simulation systems, and thus, that considerable effort should be devoted to developing good implementations tailored to the specific needs of each simulation problem.

In fact, although the penalty imposed by a bad choice of event set is huge, the gain to be made by selecting a particular, fine-tuned implementation over, say, Henriksen's implementation or splay trees is very small. To understand why this is the case, it is necessary to compare the costs of event set operations with other computations that are common in the context of simulation. Simple experiments on the VAX 11/780, in the testing environment described in Jones (1986), show that performing a hold operation on a 10 item event set costs about the same as 3 procedure calls and returns or one call to a uniform pseudo-random number generator, while a hold operation on a 1000 item event set costs less than one call to a negative exponential pseudo-random number generator. The difference between the times taken by the two random number generators is due to a single call to the natural log function.

Given the above information, it is clear that the time taken for event set management in many simulation models should be significantly less than half the overall simulation time! Thus, even if a new implementation were found that required no time at all to perform event set operations, a factor of two speedup would be unlikely. It may well be that finding faster ways to compute the natural log function will lead to a greater speedup in many simulation systems than any likely improvement in the event set. This leaves a serious question: Where should the effort that is currently being devoted to finding new implementations of the event set be directed?

Avoiding the event set. It is clear that many simulation methodologies result in over-use of the event set. An unfortunate number of simulation languages provide only one tool for modularizing the code of a simulation model, the event service routine. Users of such languages are faced with the choice of either replicating common code sequences throughout their simulation programs or gathering those sequences into event service routines that are used to simulate procedures. When the latter is done, what ought to be a procedure call is accomplished by scheduling these new event service routines at the same simulated time as their callers. Clearly this approach should be avoided, since it enlarges the event set and adds additional event scheduling traffic!

A more subtle way in which unneeded event scheduling is commonly done involves deterministic chains of events. If event A schedules event B, and event B always schedules event C, event B may frequently be eliminated from the model! Consider, for example, the simulation of a token-ring network with a message in transit: It would be possible, of course, to simulate the arrival of this message at each network station it passes on the way to its destination, but it is usually possible to compute the aggregate transmission delay to the destination and thus avoid scheduling these intermediate events.

Finally, even when events themselves cannot be eliminated, a careful use of the transitivity of the 'schedules' relationship between events can reduce the number of pending events. Consider the case where event A schedules events B and C. If event B always happens before event C, modifying the model so that event B schedules event C will reduce the size of the event set. Users of the three phase approach to simulation have known this for a long time; this is why simple minded event set implementations such as the median pointer method have frequently proven to be quite adequate for supporting three-phase models.

Although the individual developers of simulation models would be well advised to pay attention to the above considerations, it should be noted that re-working a simulation model for efficient simulation can obscure the model. Thus, as with programming language optimization, it would be preferable to move the responsibility for this from the programmer to the language implementation. Whenever possible, our simulation language processors should reduce the expected event set size and traffic by replacing event service routines with simple procedures, eliminating unneeded events, and reorganizing scheduling relationships. These kinds of model transformations should be relatively easy for high-level model specification languages, but they may prove to be difficult for models expressed in general purpose programming languages such as Simula 67.

Exploiting Parallelism. There are at least three different levels at which parallelism can be exploited in discrete event simulation. At the highest level, we can look forward to new simulation languages that are supported by parallel simulation algorithms. This approach is already starting to bear fruit in the area of distributed simulation for queuing networks, but, as with any new programming language, the advantages of a new simulation language must be quite compelling before it will attract much of a following. At an intermediate level, we can hope for attempts to use these new parallel simulation algorithms to support conventional simulation languages. This will almost surely be impossible for general-purpose languages such as Simula 67, but it may well be practical for high-level model specification languages.

At the lowest level, parallelism can be used in the implementation of the event set. This should allow all of the computationally intensive aspects of event set handling to be performed on one or more support processors while the primary simulation processor uses conventional simulation algorithms. From the point of view of the primary simulation processor, this would allow the time taken for event set handling to be effectively ignored. Skew heaps (Jones 1986) have been modified to allow this approach to concurrency (assuming a shared memory environment), but they are not stable, and arbitrary deletion is not supported. The development of a sufficiently general parallel implementation of the event set would allow parallelism to be easily exploited in existing simulation languages, although (as noted above) the expected speedup from this approach is limited.

AUTHOR'S BIOGRAPHY

DOUGLAS W. JONES is an assistant professor in the Department of Computer Science at the University of Iowa. He received a B.S. in physics from Carnegie-Mellon University in 1973, and M.S. and Ph.D. degrees in computer science from the University of Illinois in 1976 and 1980 respectively. He has developed a gate-level logic simulator that has been used for teaching digital systems since 1983; this lead to an investigation of event set implementations and parallel simulation. His other research interests include system programming languages and computer architecture. He is a member of ACM and AAAS.

Douglas W. Jones
Department of Computer Science
University of Iowa
Iowa City, IA 52242
(319) 353-7479
CSNET: jones@cs.uiowa.edu

POSITION STATEMENT

James O. Henriksen
Wolverine Software Corporation
Annandale, Virginia

Historical Perspective. The flow of articles, papers, and theses on the subject of event list algorithms began, in earnest, in 1977, and it continues, apparently unabated, in 1986. As one who has contributed to this literature, I am both surprised and delighted that the topic is still of interest. The paragraphs that follow present some personal observations on the subject.

New Algorithms - Diminishing Returns. I am surprised that new algorithms continue to be proposed. How can you beat an O(log n) algorithm?

Analysis of Complexity. Virtually every article, paper, or thesis about event list algorithms has included some analysis of algorithm complexity. Such analyses have ranged from informal observations gleaned from empirical data to very formal, mathematical or statistical analyses. While formal analyses of complexity are very interesting, they provide an incomplete picture of algorithm performance. For example, consider how it can be possible for an $O(\sqrt{n})$ algorithm such as Henriksen's to consistently outperform O(log n) algorithms. Clearly, there is more to performance than is revealed through complexity analysis.

For Henriksen's algorithm, one can fairly easily construct two extreme cases for algorithm performance. If event times increase monotonically, each insertion into the event list can be made after exactly two comparisons. On the other hand, a worst case can be constructed for which O(n) comparisons are required for an insertion. Knowing that complexity can range from a constant number of comparisons to O(n) comparisons is no use whatsoever. What we need to know is the relationship between event time distributions and algorithm performance. Determining that relationship analytically is an extremely difficult problem, although some progress has been made in this regard. In the absence of analytical relationships, empirically derived distributions of algorithm performance would be useful. For example, the best-case performance of Henriksen's algorithm is infrequently attained, but not beyond the realm of

possibility in real-world simulations. However, the worst-case performance can be obtained only through systematic exploitation of the algorithm, and such exploitation is virtually impossible in a real-world model.

Other Applications of Event List Algorithms. While management of simulated time is the obvious purpose of event list algorithms, such algorithms are applicable to a wider class of problems. For example, consider the problem of simulating the flow of objects on a conveyor. Assume that we are interested in the problem of placing objects of random sizes onto the conveyor at randomly chosen points. To insert an object at a given point, we must determine when a sufficiently large unoccupied portion of the conveyor will reach that point. Assume that we have, at our disposal, data structures that, for each object, depict the object's last known position, the time at which the object reached that position, and the object's velocity. The most straightforward approach to modeling an insertion would be to determine the current positions of all objects on the conveyor. By simply searching the up-to-date position data, one could easily determine the location of the next sufficiently large free space to pass by the insertion point. The search could be integrated into the algorithm for updating position data. If the search/update were performed in downstream-to-upstream order, the algorithm could terminate when the first adequate free space was found, saving the time that would be consumed performing unnecessary position updates.

If the conveyor contains many objects, the algorithm outlined above probably will perform very poorly, because it performs a linear search in the space domain. Improved event list algorithms were first devised as alternatives for linear search in the time domain. An improved event list algorithm could easily be adapted to operate in the space domain. If this were done, the search-update process would be reduced from an O(n) process to (almost) an O(log n) process.

Floating Point Clocks - a Neglected Subject. While great attention has been devoted to event list algorithms, other areas in which fruitful research could be conducted have been ignored. For example, I know of no published studies on the ramifications of using a floating point clock in a simulation language. Some floating point hardware rounds the results of floating point computations, while other hardware truncates results. IBM (and IBM-compatible) mainframe floating point hardware employs "guard digit" arithmetic, a unique approach to floating point computation. What is the impact of the mode of computation on simulation?

In most floating point hardware, single precision data contains roughly 24 bits of precision. It is relatively easy to demonstrate that many, if not most, simulation applications that employ a floating point clock require double precision arithmetic. For example, if the simulator clock is to register time values in excess of 16,777,215, 24 bits of precision is inadequate; e.g., adding 1.0 to 16777216.0 will yield a result of 16777216.0. For these reasons, a general purpose simulation language that includes a floating point clock must implement the clock in double precision. Unfortunately, one does not have to look far to find languages that use single precision floating point clocks.

Many simulation models contain entities that operate on a fixed cycle time, e.g., circular conveyor systems. When modeling such systems, it is often necessary to view a point in simulated time as the time displacement into the current cycle for a given piece of equipment. When a floating point clock is used, floating point modulus division (FMOD in Fortran) computations must be performed to determine displacements. Unfortunately, FMOD is notoriously computationally unstable. What is the impact on simulations that require the use of FMOD? This is an interesting question.

AUTHOR'S BIOGRAPHY

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation, located in Annandale, Virginia (a suburb of Washington, D.C.) Wolverine Software was founded in 1976 to develop and market GPSS/H, a state-of-the-art version of the GPSS language. Since its introduction in 1977, GPSS/H has gained wide acceptance in both industry and academia. Mr. Henriksen is an Adjunct Professor in the Computer Science Department of the Virginia Polytechnic Institute and State University. He teaches courses in simulation and compiler construction at the university's Northern Virginia Graduate Center. Prior to forming Wolverine Software, he worked for CACI, Inc., where he served as project manager for development of the Univac 1100 Series version of Simscript II.5. Mr. Henriksen is a frequent contributor to the literature on simulation. He has given invited presentations at the Winter Simulation Conference, and at the Annual Simulation Symposium. Mr. Henriksen is a member of ACM, SIGSIM, SCS, the IEEE Computer Society, ORSA, and SME.

James O. Henriksen
Wolverine Software Corporation
7630 Little River Turnpike, Suite 208
Annandale, Virginia 22003-2653
(703) 750-3910

POSITION STATEMENT

C. Dennis Pegden
Systems Modeling Corporation
State College, Pennsylvania

An often overlooked design issue in implementing the time advance mechanism in a simulation language is the method used for coordinating the interaction between scheduled and unscheduled events. Scheduled events are those that are scheduled to occur

at a specific point in time in the future and may therefore be maintained in a time-ordered event list. Unscheduled events are those which are defined based on the state of the system and therefore cannot be placed in the event-list. The mechanism which determines the occurance of an unscheduled event is sufficiently complex that the exact time at which the event will occur cannot be predicted. Examples include the movement of parts along a conveyor which is stopping and starting and the movement of an automatic guided vehicle through a congested track network. The method used for detecting and executing the unscheduled events can in many instances play a bigger role in determining simulation execution time than the algorithm used for maintaining a time ordered event list.

C. Dennis Pegden
Systems Modeling Corporation
248 Calder Way
State College, PA 16801
(814) 238-5919

SOME REMARKS ON IMPLEMENTATION OF TIME

Robert G. Sargent
Syracuse University
Syracuse, New York

In order for a simulation model to run, some mechanism must be used to move the model through time. Today, in discrete event simulation, this is normally accomplished by using a time flow mechanism that contains a future event set algorithm. However, with the new developments occurring in computer hardware and with the increasing demands being made on simulation, there is need for additional research into how to have a simulation model move through time.

First, let us discuss future event set (event list, event calendar, future-event chain) algorithms. Since McCormack's dissertation research (McCormack 1979, 1981), how event lists behave and how several event list algorithms perform have been known, including knowing some algorithms that have good performance. Thus, from the practical point of view, there is no need for further research on event list algorithms. It is necessary to use a good algorithm in order to have execution efficiency in a simulation model; in particular, when a large number of events occur on the event list (Henriksen 1983). Unfortunately, there is considerable misunderstanding about event list algorithms. As stated by Henriksen (1983) "the variance in quality of what has been written closely parallels the variance in performance of the algorithms themselves: everything from scholarly works to utter nonsense".

Second, let us discuss time flow mechanisms. Generally, two basic methods for advancing time are considered: fixed step (ΔT) and event-to-event. The event-to-event method is usually used in discrete event simulation, the ΔT method is used in continuous simulation, and combined simulation commonly uses both methods. There is need for research in time flow mechanisms. Nance (1971) has shown that there is really a continuum between the two basic time flow mechanisms for discrete event simulations. Overstreet (1982) discusses the interaction of world views with time flow mechanisms, and Nance (1981) discusses the movement of a simulation model from state-to-state, i.e., a "state-sequenced simulation", and its relationship to the time flow mechanism.

Third, there is a need to determine how to move a simulation model through time and keep events synchronized when there is more than one processor being used (distributed synchronization) or when submodels or portions of a simulation model have their own event lists. With the numerous types of computer architectures and computer memories being developed today, that can cost significantly less than a single large processor computer, we need to determine how to perform simulation on them. One of the open questions that needs research is how to handle the event list (single or multiple) and keep the events synchronized. Also, there is increasing interest in joining existing simulation models together; this requires a method to move the models through time (Sargent 1986).

there are other areas related to implementation of time that need research. Some of them are how to implement "hardware-in-the-loop" and "man-in-the-loop" in discrete simulation, and how to have real time simulations interact, and the relationship of message driven simulation to discrete event simulation, including how they may interact.

AUTHOR'S BIOGRAPHY

ROBERT G. SARGENT is a Professor of Industrial Engineering and Operations Research and a member of the Computer and Information Science faculty at Syracuse University. Dr. Sargent has served the Winter Simulation Conference in several capacities, including being a member of the Board of Directorys for ten years, General Chairman of the 1977 Conference, and Co-editor of the Proceedings of the 1976 and 1977 Conferences. For his service to the Winter Simulation Conference, the Board of Directors presented him with a plaque in 1984. Professor Sargent was Department Editor of Simulation Modeling and Statistical Computing for the Communications of the ACM for five years, has served as Chairman of the TIMS College on Simulation and Gaming, and has received service awards from ACM and IIE. He currently is an ACM National Lecturer, a member of the Executive Committee of the IEEE Computer Society Technical Committee on Simulation, and a Director-at-large of the Society for Computer Simulation. Dr. Sargent received his education at the University of Michigan. His current research interests include model validation, simulation methodology, simulation application, performance evaluation, and applied operations research. Professor Sar-

gent is a member of AIIM, New York Academy of Sciences, Sigma Xi, ACM, IIE, ORSA, SCS, and TIMS and is listed in Who's Who in America.

Robert G. Sargent
Industrial Engineering & Operations Research
Syracuse University
Syracuse, NY 13244-1240
(315) 423-4348

THE THREE-PHASE APPROACH

Robert M. O'Keefe
Virginia Polytechnic Institute
Blacksburg, Virginia

Introduction. The three-phase world view involves the recognition of two distinct types of event: Bound or scheduled events, whose occurrence is predictable and can thus be scheduled, and conditional or contingent events, whose occurrence is dependent upon certain conditions. Bound events represent time dependent changes; conditional events represent state dependent changes. Typically, conditional events schedule bound events. For instance, the start of an activity is a conditional event, and the end is a bound event.

Within the time flow mechanism, a calendar of bound events is maintained. The time advance then involves the three phases:

Time - advance the clock to the time of the next bound event.

B - execute all bound events events due to occur at this time.

C - scan all the conditional events.

These phases are cycled for the duration of the simulation.

Event scheduling and activity scanning can both be perceived as two-phase approaches, subsets of the three-phase approach. Most models of reasonable complexity include both time and state dependent changes - thus the three-phase approach is inherently more powerful and applicable than event scheduling or activity scanning.

Efficiency. Activity scanning, of which the three-phase approach is a descendant, was neglected in the States due to the perceived inefficiency of the scan of conditional events. (Or more likely, due to the "not invented here" mentality.) As the number of conditional events increases, the time to scan them increases disproportionately. However, when the simulation is largely composed of state dependent changes, the approach can be far more efficient than event scheduling.

When modeling state dependent changes using pure event scheduling, they must be forced into the model as bound events. This is normally done by arranging for primary events to schedule other events for execution at the present clock time, or by including code for conditional events within that for bound events. The first method is ungainly and can be highly inefficient, particularly when scheduling an event incurs the overhead of shuffling a heap or similar mechanism; the second method is bad programming practice, since it reduces the modularity of the simulation program.

Cellular Simulation. When a simulation includes many highly independent bound events, the three-phase method can be very inefficient. The execution of a bound event that has no effect upon the conditional events, such that none of the conditional events can be successfully executed, results in a wasted scan.

This inefficiency can be overcome by use of cellular simulation (Spinelli de Carvalho 1976). Here all conditional events are closed (or opened) at each advance of the clock, and must be explicitly opened (or closed) within bound events. Only conditional events that are open are scanned. Thus, the programmer can considerably reduce the size of the scan at each time beat by identifying conditional events that are dependent upon the occurrence of one or more bound events. (In effect, this is similar to the extension of event scheduling where dependent and independent events are separately identified.)

Often, cells of associated scheduled and conditional events can be identified and grouped together. This cellular approach to simulation can be used as a basis for decomposition. With regards to distributed simulation, each cell could be executed on a different processor.

Inference. The conditional events can be scanned in any order, although a simple top-down scan is normally used. Thus, the textual order of the conditional events determines a priority ordering. This can sometimes be a highly efficient and simple means of modeling priorities; in some cases, it can be highly constraining. Evans (1984) has suggested that inference mechanisms could be used by the simulation executive so as to facilitate efficient and/or appropriate ordering of execution of the conditional events, much as is done in AI with rule-based systems.

Visual Displays. In a paper elsewhere in these proceedings (O'Keefe and Davies 1986b), the author suggests that the three-phase approach is useful for simulation with visual displays (animation). This is because many pictorial changes are state-based, for example: "If the machine breaks down, change its color to red." Often, it is useful to introduce both bound and conditional events into the program that are solely concerned with updating the visual display and arranging for interaction with the user. Thus, the need for efficient management of the event set and conditional events can be more important when using visual displays, due to the increased number of events.

Event set Mechanisms. Efficient maintenance of the event set is important to the three-phase approach. However, it is not as crucial as in event scheduling (since there will typically be less additions and deletions to the event set), or as in some implementations of process description world views (for instance, in GPSS the current event list can become exceedingly large). The author has found that the median pointer method proposed by Davey and Vaucher (1980) can be effectively implemented, and is as good as any other method when the event set size does not exceed about 200 entries (O'Keefe 1985).

Obviously, the efficiency of the three-phase approach is dependent upon the number of conditional events, and the amount of scanning that needs to be done. Use of cellular simulation, combined with an effective event set mechanism, can result in highly efficient execution of medium sized models with approximately one hundred events (both bound and conditional) and a few hundred active entities.

To some extent, any discussion concerning world views and event set mechanisms is speculative, since no formal comparisons have been done comparing world views against event set mechanisms. (Most research has compared different mechanisms under the hold model.) For instance, it is fairly obvious that process interaction as implemented in Simula 67 is considerably more efficient than process description as implemented in GPSS, since passive objects are re-activated directly through a reference pointer maintained by the programmer, whereas in GPSS, a similar effect is achieved by the executive scanning the current events list.

Conclusion. No single event set mechanism is suitable for all world views and all applications. For a simulation that is highly state dependent, the three-phase approach implemented with a simple linear linked list for the event set may be more efficient than event scheduling or process interaction with a heap mechanism. Those requiring more information on the three-phase approach should look at Crookes (1982), O'Keefe and Davies (1986a) and Pidd (1985).

AUTHOR'S BIOGRAPHY

ROBERT M. O'KEEFE is a visiting assistant professor in the Department of Computer Science at Virginia Tech, on leave from the Board of Studies in Management Science at the University of Kent at Canterbury, England. He received a B.Sc. in Computer Studies and Operational Research from the University of Lancaster in 1979, and a Ph.D. in Operational Research from the University of Southampton in 1984. Major research interests include artificial intelligence and simulation, visual interactive simulation, and the application of expert systems. He is a member of SCS, TIMS, ORS, AAAI and BCS, and a director of Decision Computing Limited.

Robert M. O'Keefe
Department of Computer Science
562 McBryde Hall
Virginia Polytechnic Institute
Blacksburg, VA 24061
(703) 961-6931

OPTIMISTIC SYNCHRONIZATION

Brian W. Unger
University of Calgary
Calgary, Alberta

Optimistic synchronization algorithms offer the potential of significant parallelism in distributed simulation. Simulations implemented as a set of communicating processes on multiple processors without shared memory require synchronizing process execution so that events occur in the appropriate temporal order. Optimistic algorithms enable concurrent processes to compute forward in time in a way that enables recovery when unexpected events take place in the past of a process. Recent work in this area indicates that much greater parallelism can be achieved than is possible with pessimistic algorithms, i.e. those that require a process to wait until sufficient information is available to advance simulation time without risk of events occurring in the past of a process.

Brian W. Unger
Computer Science Department
2500 University Drive, N.W.
University of Calgary
Calgary, Alberta T2N1N4
(403) 220-6038

REFERENCES

Crookes, J. (1982). Simulation in 1981. Eur. J. Op. Res. 9, 1-7.

Davey, D., and Vaucher, J.G. (1980). Self-Optimizing Partitioned Sequencing Sets for Discrete Event Simulation. INFOR 18, 41-61.

Evans, J.B. (1980). Simulation and intelligence. Technical report TR-A5-84, Centre of Computer Studies and Applications, University of Hong Kong.

Gordon, G. (1978). The Development of the General-Purpose Simulation System (GPSS). SIGPLAN Notices 13, 8 (August) 183-198. Sections 3.5 and 4.6.

Henriksen, J.O. (1983). Event List Management - A Tutorial. In: Proceedings of the 1983 Winter Simulation Conference (S. Roberts, J. Banks, and B. Schmeiser, eds.). Arlington, VA, 543-551.

Jones, D.W. (1986). An Empirical Comparison of Priority-Queue and Event-Set Implementations. Communications of the ACM 29, 300-311.

Kingston, J. H. (1984). Analysis of Algorithms for the Simulation Event List. Ph.D. thesis, Basser Dept. of Computer Science, Univ. of Sydney, Australia.

McCormack, W.M. (1979). Analysis of Future Event Set Algorithms for Discrete Event Simulation. Ph.D. thesis, Syracuse University, Syracuse, NY 13244.

McCormack, W.M., and Sargent, R.G. (1981). Analysis of Future Event Set Algorithms for Discrete Event Simulation. Communications of the ACM 24, 801-812.

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

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

O'Keefe, R.M. (1985). Comment on Complexity Analyses of Event Set Algorithms. Computer Journal 28, 496-497.

O'Keefe, R.M. and Davies, R.M. (1986a). A Microcomputer System for Simulation Modelling. Eur. J. Op. Res. 24, 23-29.

O'Keefe, R.M. and Davies, R.M. (1986b). Discrete Visual Simulation with Pascal_SIM. In: Proceedings of the 1986 Winter Simulation Conference.

Overstreet, C.M. (1982). Model Specification and Analysis for Discrete Event Simulation. Ph.D. thesis, Virginia Tech, Blacksburg, Va 24061.

Pidd, M. (1985). Computer Simulation in Management Science. John Wiley, New York.

Sargent, R.G. (1986). Joining Existing Simulation Programs Together. In: Proceedings of the 1986 Wither Simulation Conference.

Spinelli de Carvalho, R (1976). Cellular Simulation. Ph.D. thesis, University of Lancaster, England.