Zero Lookahead in a Distributed Time-Stepped Simulation

Ernest H. Page

The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102
epage@mitre.org

Abstract

Conservative parallel and distributed synchronization mechanisms require a quantity known as lookahead to enable deadlock avoidance. Unfortunately, identifying sources of lookahead can be problematic when lookahead does not naturally exist in the specified simulation model, often forcing model respecification. The U.S. Defense Modeling and Simulation Office (DMSO) is sponsoring the development of the High Level Architecture (HLA) for modeling and simulation. One of the primary components of the HLA is a general-purpose distributed simulation infrastructure known as the HLA Runtime Infrastructure (RTI). A mechanism to support zero lookahead has been implemented in recent prototypes of the RTI. This mechanism is evaluated in terms of the classical activity-scanning conceptual framework. The mechanism is argued to be well suited to support zero lookahead in distributed time-stepped simulations such as the ALSP Joint Training Confederation.

Categories and Subject Descriptors: I.6.5 [**Simulation and Modeling**]: Model Development – *modeling methodologies;* I.6.8 [**Simulation and Modeling**]: Types of Simulation – *discrete event, distributed, parallel.*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Activity scanning; Aggregate Level Simulation Protocol; conceptual frameworks; High Level Architecture; time flow mechanisms.

1 Introduction

Conservative synchronization mechanisms for parallel discrete event simulation (PDES) date to [4, 6, 7]. These mechanisms are inherently prone to deadlock. The most common implementations of conservative synchronization *avoid* deadlock through the use of null messages and a quantity known as *lookahead.* This type of conservative synchronization with deadlock avoidance has been implemented within the infrastructure software for both the Aggregate Level Simulation Protocol (ALSP, see [24, 27]) and the new standard for U.S. Department of Defense (DoD) models and simulations, the High Level Architecture (HLA, see [10]).

The biggest problem with lookahead is that it must naturally *exist* within the simulation model.[1] Potential sources of lookahead are described in [13, 22]. In many circumstances, model formulations that support

[1]Another problem with lookahead is that, typically, large lookahead values are required to enable a distributed (conservative) simulation to outperform its sequential counterpart. In the DoD arena, however – where the primary motivation for distribution is *interoperability,* rather than *speedup* – this factor loses some of its significance.

lookahead are difficult – if not impossible – to construct. Given its charter as a general purpose distributed simulation infrastructure, the High Level Architecture should support these so-called *zero lookahead* systems. Fujimoto [14] describes a mechanism to support zero lookahead in the HLA. This algorithm, which defines "available" versions of the `nextEventRequest` and `timeAdvanceRequest` services, requires a federate to *temporarily* exhibit a non-zero lookahead. This paper illustrates the use of the HLA services to support a distributed time-stepped simulation. The algorithm implied is related to the classical activity scanning conceptual framework.

2 Background

A brief review of some relevant topics is given below.

Discrete event simulation. The universe of things that go by the term *simulation* is vast. Fully enumerating and properly classifying the elements in that set is an open problem however, and although worthy of study, we leave the task for others to accomplish. The focus of this presentation is a well-defined class of simulation known as *discrete event simulation* (DES).[2]

What is a discrete event simulation (DES) model? In the abstract, it is series of state changes. More precisely, a DES model is composed of a set of attributes that taken together describe the model's state. One of these attributes, generally denoted *time,* may be used as an indexing variable to order these state changes. Thus, the notional execution of a DES model can be described by a chain of ordered pairs $\langle (t_0, I), (t_1, S_1), \ldots, (t_T, S_T) \rangle$, where I is the initial state, S_i describes the attribute values at time t_i, and S_T describes the terminal state.[3] The states in this chain are *events.* The event is the fundamental element of a discrete event simulation – it binds time and state. All discrete event simulations describe a series of events [20].

Time flow mechanisms. The *implementation* of a DES model can take a variety of forms. One of the implementation details for a discrete event simulation is the mechanism through which simulation time is advanced. The earliest formal characterization of time flow for discrete event simulation is due to Kiviat [17, 18]. Two basic forms exist:

- *Fixed-time increment* (or time-stepped). Here the values assumed by the simulation clock are evenly spaced along an interval on the real numbers.
- *Variable-time increment* (or next event, event-stepped). Here the values assumed by the simulation clock are spaced arbitrarily apart.

[2] The uninitiated reader is encouraged to refer to [11, 12, 16, 19, 28] for introductions to discrete event simulation.
[3] See [28] for an example of a comprehensive formal model of discrete event simulation.

Regardless of the time flow mechanism utilized, the *event* remains the fundamental concept that binds time and state; one view (next-event) considers the passage of time to be predicated on events, the other (fixed-time) considers events to be predicated on the passage of time [20, p. 60]. Nance [20] identifies these two TFMs as poles on a continuum, and provides the first mathematical formulation of the relative efficiency of the TFMs along the pole. Nance also demonstrates that the most efficient TFM depends on the nature of the system being simulated. Prior to [20] it was widely believed that next-event was necessarily superior in performance to fixed-time.

Conceptual frameworks. Discrete event simulation models and their implementations may reflect any of a variety of organizing principles. For example, a DES may be oriented around events – as provided by languages like SIMSCRIPT. Or it may may be oriented around *processes* – these are the "lifetimes" of objects within the model. The organizing principle of the model is known as its *conceptual framework* (or *world view*). The conceptual frameworks (CFs) alluded to here are, respectively, *event scheduling* and *process interaction.*

A *correctness-preserving* DES implementation preserves the notional execution described by the state chain above. Event scheduling languages directly implement this execution scheme. In a process oriented language, such as MODSIM or SIMULA, the attribute value changes at time t are often dispursed among several objects. The event is harder to distinguish in a process oriented simulation language, since it is distributed among the object lifetimes. Still, the execution of a process oriented simulation preserves the notional DES execution. An event scheduling and process oriented implementation of the same model will produce exactly the same results.[4]

The activity scanning conceptual framework. Another organizing principle for DES, which is naturally supportive of the fixed-time TFM, is the *activity scanning* conceptual framework (CF) suggested by [5]. Most widely used within the United Kingdom, an activity scanning-based simulation model is organized around *conditions* and *actions* (attribute value changes) that should be taken when the associated condition is satisfied. Together, a condition and its associated actions form an event – which is the initiation or termination of an *activity* [21].

Worthy of mention is the fact that fixed-time implementations *do not* necessarily preserve notional the DES execution. That is, since the clock is updated by a fixed amount, rather than updated to the time of the most imminent event, the time value associated with the state changes in a fixed-time implementation may differ from the time value in a next event implementation.[5] The difference is a function of the timestep

[4] In the absence of real number rounding discrepancies and assuming appropriate correlation of the random number streams.

[5] Formally, time flow mechanisms (fixed-time, next event) and conceptual frameworks (event scheduling, activity scanning and process interaction) are *separate* artifacts. An event scheduling implementation, for example, could utilize a fixed-time method for incrementing the clock. Practically speaking, however, event scheduling and process interaction languages use the next-event time flow mechanism exclusively. Tocher [26] defines a next-event implementation of activity scanning known as the *three-phase approach.* Refer to Balci [3] for an excellent discussion of conceptual frameworks for discrete event simulation.

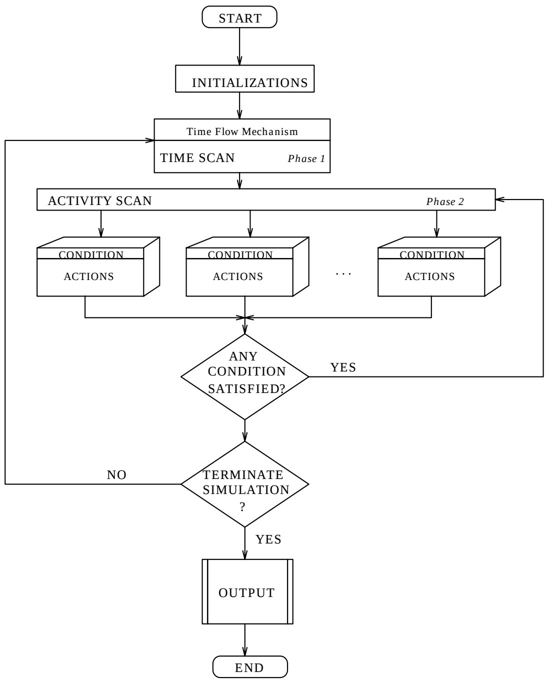

Figure 1: A Simulation Executive for the Activity Scanning Conceptual Framework.

size, typically denoted Δt, and this quantity is often also referred to as the single step *error term.*

An algorithm for activity scanning. A simulation executive for the activity scanning CF is illustrated in Figure 2 which originally appears in [3]. Execution begins with an initial assignment of model attribute values, and continues within a two-phase loop until the criteria for termination are satisfied. In *phase 1* the simulation clock is updated by a fixed amount, Δt. In *phase 2* each condition is evaluated, in turn, and if a condition is satisfied then the corresponding actions are performed. One evaluation of each condition represents a single scan. If any condition is satisfied, then the *complete* set of conditions must be re-scanned. Phase 2 continues until a single scan yields no satisfied conditions.

The need to re-scan the conditions arises from the fact that actions executed for condition j may enable condition i for some $i < j$. And, in the general case, there is no *a priori* order for the evaluation of conditions to prevent this from occurring. Circular dependencies are possible and generally speaking, a modeler should not be constrained to substantively restructure an intuitive design in order to accommodate the eccentricities of a particular implementation.[6]

Example. In the classical machine interference problem (see [9, 25]) a group of N semiautomatic machines $(1 < N < \infty)$ fail intermittently and must be repaired by an operator. Both failure and repair rates are distributed as Poisson random variables with parameters λ and μ respectively. The parameters λ and μ are

[6]Refer to [23] for arguments supporting this position.

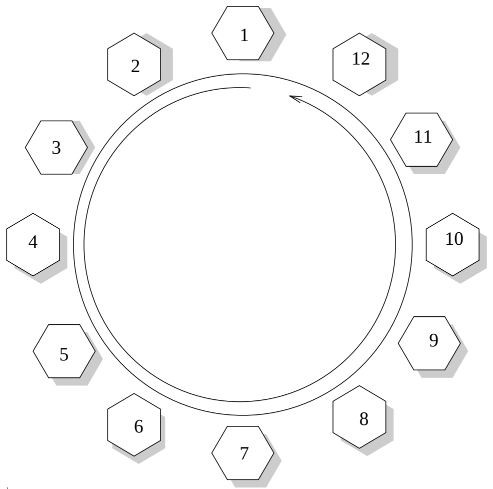

Figure 2: A Configuration for the Machine Interference Problem where $N = 12$.

assumed the same for each machine in the assignment. The model may be viewed as illustrated in Figure 2 which shows a system with $N = 12$.

There are many variations to this problem (representing various classes of queueing systems). In the variation implied by Figure 2 the technician patrols the perimeter of machine sites continuously, stopping as needed to repair failed machines. Typically, inspection is assumed to take zero time. That is, if the technician arrives at a machine at time t and the machine is operational then the technician proceeds immediately to the next machine in the circuit.

In an activity scanning formulation of this model, the condition that initiates the *repairing* activity for the technician may be defined as `if technician.location = m and m.status = failed.` The actions taken will include the assignment of a variable indicating the time of repair completion. This variable will be used in the condition that terminates the *repairing* activity. The condition that initiates the *failed* activity for a machine may be defined as `if m.failTime` $\leq$ `clock.` The actions taken will include setting the status of `m` to `failed`.

In an event scheduling or process interaction implementation of this model, *simultaneous* technician arrivals and machine failures present a challenge. To ensure *repeatability* across model executions, simultaneous events must be handled consistently. That is, either always process arrivals first and then failures or vice versa. In the former case, if an arrival and failure occur at time t, the technician perceives the machine as operational. In the latter case, the technician perceives the machine as failed. The ordering choice can, potentially, have significant impact on the statistical characteristics of the model. However, for this example, given the nature of the distributions governing failure and repair times, assuming a fixed travel time, and assuming a next event TFM, the probability of simultaneous arrivals and failures is very low. Therefore, the ordering choice has little practical significance. Consistency is still required, however, to ensure repeatability.

An activity scanning formulation of the machine interference problem must also provide a mechanism for the consistent handling of simultaneous events. The (priority-based) ordering of events within event scheduling and process interaction implementations as provided by most simulation programming languages does not extend directly to activity scanning implementations. Since the entire set of conditions is re-scanned when any condition is satisfied, modeler-supplied actions may also be needed to control the ordering of simultaneous events. Using the example given above, if we want to process arrivals before failures, then we need to do two things: (1) place the condition that initiates the *repairing* activity for the technician earlier in the scan list than the condition initiating the *failure* activity for the machine,[7] and (2) ensure that an action to set the `location` of the technician to some value other than `m` (e.g. `enRoute`) appears within the action set corresponding to the condition `if technician.location = m and m.status = operational.`

Generally speaking, the handling of simultaneous events is significantly simpler within activity scanning implementations than either event scheduling or process interaction implementations. With activity scanning, the actions corresponding to any condition are permitted to affect the truth value of any other condition(s) – and the set of affected conditions may be dynamic. The rescan mechanism inherent in activity scanning implementations assures that "transitive closure" over all conditions is achieved for any time t prior to advancing the simulation clock to $t + \Delta t$. Handling complex, and dynamic event dependencies within event scheduling and process interaction formulations can be more difficult.

3 Distributed Activity Scanning

The extension of the activity-scanning world view to a distributed fixed-time simulation is fairly straightforward. Simply view the activities as *logical processes* (a.k.a. *actors, federates*). And substitute the test "was any condition satisfied" with "did any logical process send output."

Essentially, the solution is a two-level barrier synchronization. Algorithms for both the logical process and the process scheduler (which may be centralized or distributed) are given in Figure 3. The scheduler algorithm assumes a distributed, consistent scheduler similar to the ALSP Common Module (ACM, see [27]).

In this algorithm, the logical process (LP) calculates its next state (for time t) and sends self-generated output (if any). The LP marks the end of this output (line 4) and requests to advance to time t. The LP then enters the main scanning loop (lines 9-19) within which the LP either receives a grant, which indicates that no LP in the federation sent output during the previous scan phase (line 10) or the LP receives input which it may respond to (lines 14-18).[8] When the grant is received, the LP updates it's local clock (line 21) and resumes from the top.

Each scheduler – one per LP – receives (and forwards as necessary) LP output, marker and requests to advance (lines 2-4). The scheduler also receives input for its LP and phase markers that are sent from the

[7] Simulation languages that support the activity scanning CF typically provide a priority-ordering mechanism for condition scanning.

[8] This presentation uses an ALSP-like message passing syntax. In HLA parlance, the federate honors and performs RTI service calls.

Simulation executive for logical process.

```
1   while true do
2     calculate state at $t$
3     send self-generated output for $t$
4     send output complete marker
5
6     request advance to $t$
7     scan $\leftarrow$ true
8
9     repeat
10      receive(msg)
11      if msg = grant_advance($t$)
12        scan $\leftarrow$ false
13      else
14        receive all messages
15        receive input complete marker
16        process input
17        send output in response to this new input
18        send output complete marker
19    until scan = false
20
21    $t \leftarrow t + \Delta t$
22
23  end
```

Simulation executive for scheduler.

```
1   while true do
2     receive and fwd output from my LP
3     receive and fwd marker from my LP
4     receive request to advance from my LP
5     receive input and remaining markers from fed
6     while all markers <> no_output_sent
7       if input waiting for my LP
8       /* release for scan */
9         send input to LP
10        send input complete marker
11        receive and fwd output from my LP
12        receive and fwd marker from my LP
13      else
14        mark my LP as no_output_sent
15      receive input and remaining markers from fed
16    end
17    grant time advance to my LP
18  end
```

Figure 3: Algorithms for Logical Process and Scheduler.

other schedulers in the federation (line 5). If any LP sent output, the scheduler enters its scanning loop (lines 6-16). Within the scanning loop, if any input is waiting for its LP, the scheduler sends the input to the LP and marks its completion (lines 9-10). Then the scheduler receives LP-generated output (if any) and completion marker (lines 11-12) and again receives input from the other schedulers within the federation (line 15). Note that if the LP has no input during a scan, the scheduler simply forwards a "no output" marker to the rest of the federation (line 14) and awaits the results of the scan (line 15).

The application of this approach within a zero-lookahead environment is clear. Time does not advance as long as federates are generating output, and each federate is guaranteed the opportunity to receive, process and respond to any externally-generated output during each scan. A federate may respond (or not respond) to any input it receives without requiring the passage of simulation time.

In reality, this mechanism to support zero lookahead is simply a deadlock detection and recovery mechanism. If viewed in the context of the Chandy-Misra UNITY model [8], during each timestep the computation reaches a *fixed point* and a superposed time flow mechanism detects the fixed point and alleviates it [1, 2].

4 Distributed Activity Scanning in the HLA

Although perhaps not as intuitive as it could be, support for zero lookahead in a distributed time-stepped environment is available in HLA RTI prototypes.[9] Subsequent to joining, registering subscription and publication information, setting regulation/constraint to on, and setting lookahead to zero, the federate enters its main loop, as follows:

```
1   while true do
2     calculate state at $t$
3     scan $\leftarrow$ true
4     repeat
5       send output timestamped at $t$ (if any)
6       nextEventRequestAvailable($t + \epsilon$)
7       receive update/interactions
8       receive timeAdvanceGrant
9       if time of grant is = $t + \epsilon$
10        scan $\leftarrow$ false
11        $t \leftarrow t + \Delta t$
12    until scan = false
13  end while
```

The value of ϵ should be somewhere between 0 and the timestep size, Δt. Generally, a value that is not subject to rounding error should be chosen, e.g. $\Delta t/2$. RTI service calls take the place of the markers in Figure 3. Specifically, the call to `nextEventRequestAvailable` serves to mark the end of the federate's output to the RTI and the `timeAdvanceGrant` marks the end of the RTI's input to the federate. In this implementation, a federate receives a grant after each scan phase in which the federate receives updates/interactions from the federation. The value of the grant is the timestamp of the last message sent to the federate. When all federates are blocked on a `nextEventRequestAvaliable(`$t + \epsilon$`)` and there are no messages in the network, the RTI detects this condition and each federate is granted a time advance to $t + \epsilon$ which signals a federate that it is safe to proceed to the next timestep.

Interestingly, this approach does not work with the fixed-time `timeAdvanceRequestAvailable` service. If all federates send output (line 5) and perform a `timeAdvanceRequestAvaliable(`$t + \epsilon$`)`, the RTI grants each an advance to $t + \epsilon$.

5 Conclusions

An algorithm for distributed time-stepped simulation based on the classical activity scanning conceptual framework is presented. The algorithm inherently supports zero lookahead and is described in terms of HLA services. An open problem is the application of this approach within the context of mixed or dynamic time-steps. The presentation here assumes a singular, fixed time-step size. The efficiency of any algorithm is

[9] The approach documented here was tested using RTI Version 1.0 Release 11.

an important consideration. As a matter of practice, however, it is a question of "how much is enough?" For many applications of parallel discrete event simulation (PDES) the algorithm presented would be inadequate. The objective of most of these applications is a *speedup* of parallel/distributed model execution compared to a sequential counterpart. Efficiency is measured in milliseconds and less. Often, synchronization costs dominate the total cost of the computation – this is often the case, for example, with queueing models. On the other hand, within the DoD arena the motivation for distributed computation is often *interoperability* and not speedup. A good example of this is ALSP Joint Training Confederation (JTC). The JTC is a collection of wargames that supports several large-scale command post exercises annually. Since it supports interactive training, the JTC is paced by real time. But unlike applications that use the distributed interactive simulation (DIS) protocol, when the computational burden is too high to maintain the desired rate, simulation time takes precedence over real time. That is, ALSP time can, and does, fluctuate around real time. The JTC currently runs at one-minute time steps (although investigations with smaller timesteps have been conducted and more are planned). The time required to calculate state and refresh the various databases and workstations for some of the larger wargames in the JTC is non-trivial, and can occupy a significant portion of the one minute time step. In this type of environment, the efficiency of a distributed time flow mechanism, even if measured in seconds, is almost a non-factor.

The proposed algorithm would certainly seem beneficial to any JTC-like HLA federations, and could be useful within the context of the JTC itself. Current plans for the JTC are to make use of the HLA Runtime Infrastructure in 1998 [15]. Despite several potential uses for zero lookahead in within the JTC, ALSP imposes an artificial lookahead on each federate. End-to-end interactions that could (and should) be accomplished within the context of a single value of simulation time require multiple timesteps. The opportunity to utilize a set of services that provide zero lookahead for distributed time-stepped simulations within the context of the HLA is welcome.

Acknowledgments

The assistance of Sean Griffin in setting up the experiments that support the algorithms in this paper is greatfully acknowledged.

References

[1] Abrams, M., Page, E.H. and Nance, R.E. (1991). "Linking Simulation Model Specification and Parallel Execution Through UNITY," In: *Proceedings of the 1991 Winter Simulation Conference,* pp. 223-232, Phoenix, AZ, December 8-11.

[2] Abrams, M., Page, E.H. and Nance, R.E. (1991). "Simulation Program Development by Stepwise Refinement in UNITY," In: *Proceedings of the 1991 Winter Simulation Conference,* pp. 233-242, Phoenix, AZ, December 8-11.

[3] Balci, O. (1988). "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages," In: *Proceedings of the 1988 Winter Simulation Conference,* pp. 287-295, San Diego, CA, December 12-14.

[4] Bryant, R.E. (1977). "Simulation of Packet Communications Architecture Computer Systems," Technical Report MIT-LCS-TR-188, Massachusetts Institute of Technology.

[5] Buxton, J.N. and Laski, J.G. (1962). "Control and Simulation Language," *The Computer Journal,* **5**, pp. 194-199.

[6] Chandy, K.M. and Misra, J. (1979). "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* **SE-5**(5), pp. 440-452, September.

[7] Chandy, K.M. and Misra, J. (1981). "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations," *Communications of the ACM,* **24**(11), pp. 198-206, November.

[8] Chandy, K.M. and Misra J. (1988). *Parallel Program Design: A Foundation,* Addison Wesley, Reading, MA.

[9] Cox, D.R. and Smith, W.L. (1961). *Queues*, Methuen and Company, Ltd.

[10] Defense Modeling and Simulation Office. (1995). *High Level Architecture for Modeling and Simulation Management Plan,* Version 1.6.

[11] Fishman, G.S. (1973). *Concepts and Methods in Discrete Event Digital Simulation,* John Wiley and Sons, New York.

[12] Franta, W.R. (1977). *Process View of Simulation,* American Elsevier.

[13] Fujimoto, R.M. (1990). "Parallel Discrete Event Simulation," *Communications of the ACM*, **33**(10), October, pp. 31-53.

[14] Fujimoto, R.M. (1997). "Zero Lookahead and Repeatability in the High Level Architecture," In: *Proceedings of the 1997 Spring Simulation Interoperability Workshop,* Orlando, FL, 3-7 March.

[15] Griffin, S.P., Page, E.H., Furness, C.Z. and Fischer, M.C. (1997). "Providing Uninterrupted Training to the Joint Training Confederation (JTC) Audience During Transition to the High Level Architecture (HLA)," In: *Proceedings of the 1997 Simulation Technology and Training Conference,* pp. 197-201, Canberra, Australia, 17-20 March.

[16] Gordon, G. (1978). *System Simulation,* Second Edition, Prentice-Hall, Englewood Cliffs, NJ.

[17] Kiviat, P.J. (1967). "Digital Computer Simulation: Modeling Concepts," RAND Memo RM-5378-PR, RAND Corporation, Santa Monica, CA, January.

[18] Kiviat, P.J. (1969). "Digital Computer Simulation: Computer Programming Languages," RAND Corp. memorandum RM-5883-PR, Santa Monica, CA, January.

[19] Law, A.M. and Kelton, W.D. (1991). *Simulation Modeling and Analysis,* Second Edition, McGraw-Hill, New York.

[20] Nance, R.E. (1971). "On Time Flow Mechanisms for Discrete Event Simulations," *Management Science,* **18**(1), pp. 59-73, September.

[21] Nance, R.E. (1981). "The Time and State Relationships in Simulation Modeling," *Communications of the ACM,* **24**(4), pp. 173-179, April.

[22] Nicol, D.M. and Fujimoto, R. (1994). "Parallel Simulation Today," *Annals of Operations Research,* **53**, Special Volume on Simulation and Modeling, O. Balci, Ed., pp. 249-285, November.

[23] Page, E.H. and Nance, R.E. (1994). "Parallel Discrete Event Simulation: A Modeling Methodological Perspective," In: *Proceedings of the 8th Workshop on Parallel and Distributed Simulation,* pp. 88-93, Edinburgh, Scotland, July 6-8.

[24] Page, E.H., Canova, B.S. and Tufarolo, J.A. (1997). "A Case Study of Verification, Validation and Accreditation for Advanced Distributed Simulation," *ACM Transactions on Modeling and Computer Simulation,* **7**(3), July, to appear.

[25] Palm, D.C. (1947). "The Distribution of Repairmen in Servicing Automatic Machines," *Industritidningen Norden 175,* p. 75.

[26] Tocher, K.D. (1963). *The Art of Simulation,* English Universities Press, London.

[27] Weatherly, R.M., Wilson, A.L., Canova, B.S., Page, E.H., Zabek, A.A. and Fischer, M.C. (1996). "Advanced Distributed Simulation Through the Aggregate Level Simulation Protocol," In: *Proceedings of the 29th Hawaii International Conference on Systems Sciences,* Vol. 1, pp. 407-415, Wailea, HI, 3-6 January.

[28] Zeigler, B.P. (1976). *Theory of Modelling and Simulation,* John Wiley and Sons, New York, NY.