Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

A FRAMEWORK FOR ADAPTIVE SYNCHRONIZATION OF DISTRIBUTED SIMULATIONS

Bertan Altuntas
Richard A. Wysk

Department of Industrial and Manufacturing Engineering
The Pennsylvania State University
310 Leonhard Building
University Park, PA 16802, U.S.A.

ABSTRACT

Increased complexity of simulation models and the related modeling needs for global supply chains have necessitated the execution of simulations on multiple processors. While distributed simulation promises reduced complexity (as the result of decomposition), increased parallelism and convenient analysis of geographically distributed systems, it poses a challenging problem: synchronizing the distributed simulation federates. This paper discusses a new discrete event distributed simulation framework, which is designed with two goals in mind: (1) easy and fast development of distributed simulations and (2) efficient adaptive synchronization of simulation processes. This research uses state machine models for the automated synthesis of so called 'local synchronization agents' and an adaptive synchronization algorithm has been developed based on pacing of simulation processes using real-time. Upon completion, this scalable framework is expected to shorten the lead-time to develop distributed simulation systems with reasonable performance characteristics.

1 INTRODUCTION

Synchronization of simulation processes has been one of the main problems researched in the area of parallel and distributed simulation (PADS). Lack of proper synchronization among simulation federates can create ambiguities during runtime which result in incorrect ordering of events eventually yielding useless simulation output. The research has been shaped around two main streams: conservative and optimistic techniques. General information on PADS and synchronization techniques can be found in Fujimoto (2000).

Synchronization is a key issue in PADS research and has direct effect on the efficiency/performance of distributed simulations. Another key issue as important as synchronization is the ease of development of distributed simulation systems (DSS).

Development usually requires extensive amount of programming by experts in the area and therefore distributed simulation is not an "available" tool for commercial applications yet. There are a few software libraries and tools available through academic research labs, however currently there is no commercial quality software that can be used to easily create distributed simulations.

Prominent researchers in the area have stated the importance of creating user friendly and transparent DSS development environments that would enable simulation developers widely adopt distributed simulation for commercial and research purposes. A nice discussion on this issue can be found in Page et al. (1999).

This paper introduces preliminary developments for ongoing research at The Pennsylvania State University's Computer Integrated Manufacturing Laboratory (CIMLAB) that aims to eventually develop a scalable distributed simulation framework addressing the two problems introduced above: (1) efficient synchronization and (2) easy and fast development of DSS.

The remainder of the paper is organized as follows. In Section 2, the synchronization methodology is introduced and the main idea of time-scaled synchronization is discussed. Section 3 introduces the structural models that form the heart of the synchronization mechanism and discusses how they are used to predict interactions. In Section 4, we discuss the structure of the synchronization agents and how they can be automatically generated. Finally, the paper concludes with the remarks in Section 5.

2 SYNCHRONIZATION METHODOLOGY

Research in CIMLAB has been focused on simulation based shop-floor control for more than a decade and a hierarchical control framework called "Rapid-CIM" has been developed. This framework includes software tools for automatically generating controller software at the manufacturing execution level and utilizes real-time discrete event simulation processes at the decision making level. Information on Rapid-CIM can be found in Joshi et al. (1995).

Research on real-time simulation has led us to develop a distributed simulation framework based on the idea of pacing simulation models using scaled real-time. In a real-time simulation process, the simulation clock advances at the same rate with the wallclock (wallclock time = real-time). Scaling real-time is a way of consistently speeding up (or slowing down) the simulation execution (the rate of advance of the simulation/virtual clock).

Our experience in the manufacturing domain has showed that if we create a distributed simulation system using real-time simulation processes and if we use commonly available networking technology (100 Mbit/s fast-Ethernet), under normal circumstances (reliable communication and nominal network load) we do not see any synchronization problems. This is due to the simple fact that communication is not a bottleneck in such a system and all simulation processes execute at the same speed (real-time). Durations of manufacturing processes (seconds/minutes) are several magnitudes of order longer compared to message transmission periods (milliseconds). Therefore, it is possible to avoid any synchronization problem in the system by using proper communication protocols.

This observation actually corresponds to two well-known issues in the synchronization research: (1) time ambiguities occur because simulation processes do not execute in a coordinated manner and (2) communication becomes the bottleneck in a distributed simulation system which is intended for as-fast-as-possible execution. In fact, safe-event mechanisms in conservative synchronization and roll-backs in optimistic synchronization both address the first issue, coordinated advance of simulation clocks.

Although it might look like optimistic synchronization methods do not employ a direct control on the advance of simulation clocks, the inherent coupling of simulation processes and event dynamics force optimistic execution to perform roll-backs when simulation clocks dangerously outrun each other. This is the reason that optimistic synchronization suffers from performance loss and extensive memory requirements when a tightly coupled set of physical processes is simulated.

This clearly indicates that dynamics of physical processes being simulated determine how corresponding logical processes should be synchronized. This observation gave rise to a new stream in synchronization research: adaptive synchronization. Adaptive synchronization targets exploiting parallelism when it is possible by changing its behavior during runtime. A survey of adaptive synchronization techniques can be found in Das (2000).

We developed an adaptive synchronization algorithm that is based on coordinating the advance of simulation processes using real-time. High resolution hardware clocks are used to pace the virtual clock of simulation processes. In stead of event driven, we utilize time-stepped simulation execution to be able to continuously control the advance of the simulation clocks with high precision. Hardware clocks are synchronized so that they run at approximately the same rate. Therefore, if all simulation processes start at the same wallclock time and all hardware clocks are synchronized, we expect the simulation clocks to advance at the same rate. This mechanism solves the first problem, the problem of coordinated advance of simulation clocks.

The solution mechanism we developed for the second problem is what makes this technique adaptive. Even if the advance rates of virtual clocks are synchronized, it is possible to observe a time ambiguity when speed of communication is close to (or less than) the rate of advance of simulation clocks. Consider the following example:

Example 1. Let LP_1 and LP_2 be two logical processes simulating the physical processes PP_1 and PP_2. Assume that their virtual clocks are running in perfect synchrony (i.e. virtual clock values are always the same). Assume that LP_1 sends a message to LP_2 at virtual time T_1 with a time-stamp $T_1 + \Delta T$. Here, ΔT represents the minimum amount of time it takes PP_1 to interact with PP_2. We know that when the message is sent out from LP_1, LP_2's virtual clock is also T_1. Assume that it takes ΔC wall-clock time units for the message to reach LP_2. In this case, for LP_2 to be able to execute the message at virtual time $T_1 + \Delta T$, it should receive the message before its virtual clock reaches the value $T_1 + \Delta T$. Thus, the amount of wallclock time it takes LP_2 to execute from virtual clock values T_1 to $T_1 + \Delta T$ should be less than ΔC, i.e. $\Delta T \cdot K > \Delta C$. Here, K is the time-scaling factor (K wallclock time units = 1 virtual time unit), $K = 1$ means the simulation is advancing at real-time pace. One can easily see that for this system to work, we should have:

$$K > \frac{\Delta C}{\Delta T} \tag{1}$$

Notice that K^{-1} is the speed of execution of the simulation process with respect to the real-time. Equation (1) simply shows us that the speed of execution has an upper bound (or the scaling factor has a lower bound) imposed by the speed of communication (or message transmission time) and the "physical interaction lag" (PIL) between physical processes.

Intuitively, one can say that, the ratio of the interaction lag in the physical process (by passing real entities) to the interaction lag in the logical process (by passing virtual entities such as messages) defines the upper bound on the ratio of the physical time to virtual time. One cannot speed up a simulation system beyond this limit without facing the possibility of creating time ambiguities.

The synchronization algorithm introduced in this paper, controls the speed factor K, by predicting potential messages and their PILs.

All logical processes are coupled with a Local Synchronization Agent (LSA), which is automatically generated from the simulation model. Each LSA has direct control over the speed of its coupled logical process. LSAs predict output messages of logical processes using state automata based structural models and broadcast this information among themselves. Thus, every LSA has a global picture of all potential messages. LSAs use this information to calculate the necessary speed limit for each potential message and change the speed accordingly at the earliest possible time of message transmission.

The system has a default speed defined by statically analyzing the simulation models and this is the fastest speed the system can run when there are no interactions. Once an interaction is over, LSAs switch back to the nominal speed. Speed changes are executed at agreed upon checkpoints in wallclock time so that synchronization is not lost during speed changes.

In short, the whole distributed simulation process dynamically changes its speed (through distributed consensus among LSAs) to accommodate interactions among logical processes.

Obviously, for this algorithm to work, we should have a well-defined networking environment such that we can calculate an upper bound on the communication delays. Therefore, we assume that the communication network is reliable and the maximum communication delay is known.

Although it is hard to know the maximum communication delay in advance in a general wide area network (it is a factor of network load), we consider a dedicated local area network with abundant bandwidth to solve this problem. Therefore, in cases where the network behavior cannot be estimated with reasonable tolerance (such as the Internet), this algorithm cannot guarantee synchronization.

3 STRUCTURAL MODELS AND PREDICTION OF OUTBOUND MESSAGES

Local synchronization agents has a key role in the distributed simulation framework developed in this research. In the heart of an LSA is a set of state automata that represents the flow structure of the coupled logical process. The structural model is a reduced replica of the simulation model. It does not include information regarding the details of the resources, schedules and most importantly the decision logic.

Generally, decision logic is the most complicated part of a simulation model from both development and representation sides. Therefore, leaving the decision logic behind provides us with a reduced version of the simulation model which only contains flow information (i.e. all possible routes entities can take through the system resources). Flow information or structural models (as referred to in this study), can be unambiguously represented using state automata based models. A detailed description of state automata can be found in Hopcroft, Motwani, and Ullman (2001).

LSAs use structural models to keep track of entity states in the logical processes. More importantly, the structural model contains crucial information about which events send messages to other processes and which events receive incoming messages.

Let M be a simulation model defined by the partition (S, P, R, E) where S is a state automaton representing the structural model (entity states, events/transitions and messaging), P is a decision function representing the decision logic (and data embedded within), R is the resource set defining resources, their states and their attributes, and E is the entity set representing the types of entities and their attributes.

The structural model S is defined by the 6-tuple $S = (\mathrm{X}, \Sigma, \Gamma, \Psi, \overline{\mathrm{X}}, B)$ where the components are defined as follows:

- X is a countable state space, such that $\mathrm{X} \neq \varnothing$
- Σ is a countable event set, such that $\Sigma \neq \varnothing$
- $\Gamma(\varepsilon): \mathrm{X} \to \Sigma$ is the set of enabled events for entity type ε at a state, defined for all entity types $\varepsilon \in \mathrm{E}$ and all states $\chi \in \mathrm{X}$.
- $\Psi: \mathrm{X} \times \Sigma \to \mathrm{X}$ is the transition mapping defined for all states in X and all enabled events in Σ.
- $\overline{\mathrm{X}} \subseteq \mathrm{X}$ is the set of initial states where an entity starts its flow.
- $B: \mathrm{X} \to Z^{+} \times Z^{+}$ is a bound-map that simply associates two virtual time values, a lower bound l_χ and an upper bound u_χ with each state $\chi \in \mathrm{X}$, where $0 \leq l_\chi \leq u_\chi < \infty$.

The decision function is the mapping $P: \mathrm{E} \times R \times \mathrm{X} \times \Gamma \to \Sigma$ based on the status of resources, from the set of enabled events at a specified state of an entity to a single selected event (selected transition). This function basically decides among alternative transitions of an entity. For example, if the simulation model is a highway system, the decision function would be all the traffic regulations and the structural model would be the map of the system with limited information about accessibility of roads by different vehicle types. So, the structural model is a template of flow of entities in the simulation process with some information about the accessibility of entities (accessibility is defined by Γ and Ψ).

Figure 1 shows a graphical depiction of an example structural model. Vertices correspond to states which represents locations of entities (resources) in the physical process and arcs correspond to events which define the transition function. This graph is for a two server queuing system.

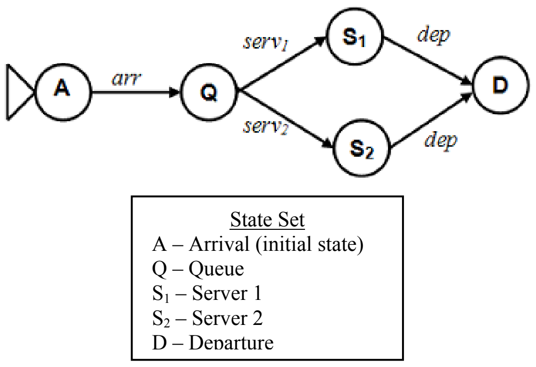

Figure 1: Directed Graph Representation of the Structural Model

The general structural model S can be cloned to create "entity-specific structural models", that are defined for each entity type. An entity-specific structural model is a reduced form of the general structural model that only has necessary states and transitions for a particular entity type. This information is provided by the mapping Γ that defines enabled events for all types of entities.

Let $S_\varepsilon = (\mathrm{X}_\varepsilon, \Sigma_\varepsilon, \Psi_\varepsilon, \overline{\mathrm{X}}_\varepsilon, B_\varepsilon)$ represent an entity-specific structural model of M defined for all entity types $\varepsilon \in \mathrm{E}$, where the components are defined as follows:

- $\bigcup_{\forall \varepsilon \in \mathrm{E}} \mathrm{X}_\varepsilon = \mathrm{X}$
- $\bigcup_{\forall \varepsilon \in \mathrm{E}} \Sigma_\varepsilon = \Sigma$
- $\bigcup_{\forall \varepsilon \in \mathrm{E}} \Psi_\varepsilon(\chi, \sigma) = \Psi(\chi, \sigma)$,$\forall \chi \in \mathrm{X}$, $\forall \sigma \in \Gamma(\chi)$
- $\bigcup_{\forall \varepsilon \in \mathrm{E}} \overline{\mathrm{X}}_\varepsilon = \overline{\mathrm{X}}$
- $B_\varepsilon \subseteq B$ such that B_ε is defined for all $\chi \in \mathrm{X}_\varepsilon \subseteq \mathrm{X}$

From the above definition, we can say that an entity-specific structural model is a specialized version of the general structural model, which only contains flow information relevant to a particular entity.

Entity-specific structural models play a central role in the synchronization mechanism, especially in the prediction of potential messages. Local synchronization agents use entity-specific structural models to track the entity states and periodically calculate minimum time to start of transmission of outbound messages.

Events are partitioned into two main types: internal events and external events. External events are also of two types: input events and output events. Input events are the ones that receive a message from another simulation process and external events are the ones that send a message to another simulation process. Internal events are not associated with any interactions and therefore they are only included for the completeness of the models.

External events in simulation processes model interactions between physical processes. Each output event has a matching input event that represents an entity leaving a source process and arriving at a destination process. This implies that each instance of an entity-specific structural model can generate at most one output message, although the model itself might have several alternative output events.

In this respect, outbound messages can be predicted by keeping a list of accessible output events and updating this list as the entity moves to new states. It is possible to generate a list of accessible output events for each state in the entity-specific structural model along with minimum time to reach each of these events from the current state. This information can be generated before run-time and can be saved along with the entity-specific model.

During run-time, when an entity moves to a new state, the LSA will perform simple table look-up operations to update the list of potential output events. We call these lists "entity-specific predicted outbound message" (ePOM) lists. In the next section, we explain how LSAs use information from ePOM lists to manage potential messages.

The look-up tables for each entity-specific model can be generated automatically by calculating the shortest path (using the lower bounds l_χ as the cost of visiting states) to all output events from each state in the model. Since this process is done prior to the run-time, it does not have any negative affect on the performance of the synchronization mechanism and can be considered as a setup process.

It is obvious that the performance of the prediction mechanism is the key factor that would affect overall performance of this synchronization mechanism. The prediction mechanism relies on the lower-bounds of state delays and quality of these lower-bound values will determine the performance of the prediction mechanism. In a stochastic simulation process where the probability distributions from which the delay values are sampled do not have bounds, this system cannot be used. Therefore, this sort of simulation models must be modified so that unbounded probability distribution are replaced with their truncated versions. On the other side, we believe that this system will perform good when deterministic simulation models (or stochastic models with low variability) are used.

4 SYNCHRONIZATION AGENTS

Local synchronization agents are composed of two main parts:

- Speed control engine (SCE)
- Message prediction and management engine (MPME).

Speed control engine calculates the speed-limit for the upcoming interaction in the distributed simulation system and re-adjusts the real-time scaling factor of the coupled simulation process until the interaction successfully takes place. Once the interaction is over, SCE goes back to the nominal simulation speed agreed in the DSS. SCE takes direct input from the message prediction and management engine.

Message prediction and management engine interacts with the coupled simulation process by reading the state of simulation entities periodically and updating the state of embedded entity-specific structural model instances. MPME creates an instance of the entity-specific structural models for each live entity in the simulation process and terminates instances when entities are disposed from the simulation process.

MPMEs of LSAs communicate among each other via special messages called "s-messages" (short for synchronization messages - to distinguish them from regular messages sent/received by simulation processes). These s-messages inform other LSAs about the potential messages and their timing. Each MPME use two lists to manage information about local and system-wide potential messages. These are:

1. Predicted Outbound Messages (POM) list
2. Predicted Inbound Messages (PIM) list.

POM is a locally-populated list which contains the potential outbound messages of the local simulation process. MPME samples potential messages from all local ePOM lists and populate the POM list. An LSA periodically broadcasts new information on the POM list to other LSAs using s-messages.

PIM is a remotely and locally populated list which contains potential inbound messages for all LPs in the DSS. SCE directly accesses the PIM list for timely adjustment of the simulation process speed. When MPME sends or receives an s-message, it copies the related information (unique entity id and predicted minimum time to transmission) to the PIM. Therefore, every PIM in the DSS contains the same information which tells what messages are potentially going to be transmitted and the earliest virtual time these transmissions can begin.

Broadcast of s-messages and use of the PIM lists enable distributed synchronous control of the simulation speeds for all logical processes. Figure 2 depicts the structure of LSAs and the DSS architecture.

Local synchronization agents can be automatically customized by using an interpreter that analyzes simulation models and generates all of the necessary structural models along with the look-up tables. Figure 3 depicts the process of creation of the structural models from simulation models. Obviously, the interpreter should be developed for the specific simulation package/language used to create simulation models. However, once an interpreter is developed it will reduce the development time of DSS significantly with this framework.

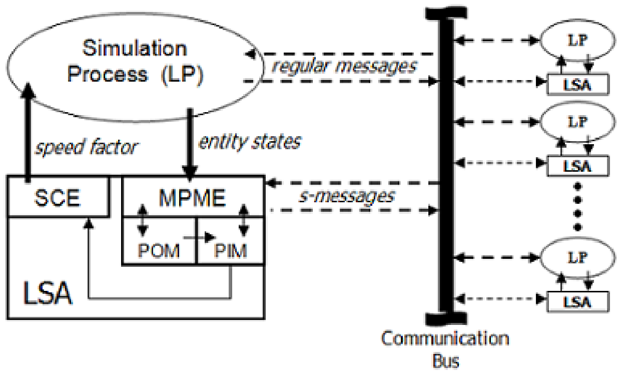

Figure 2: LSA Structure and DSS Architecture

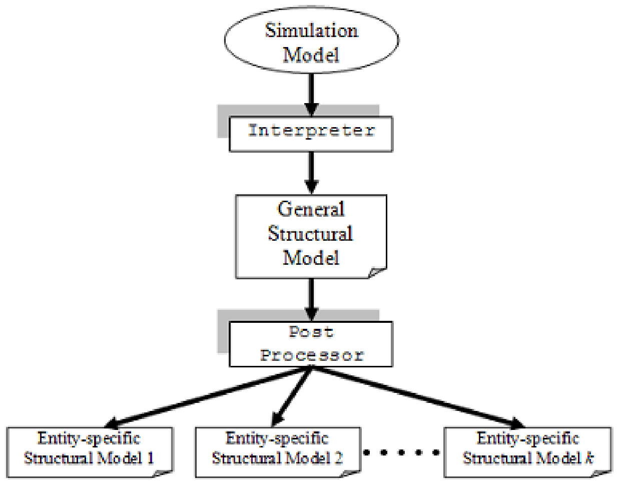
Simulation Model
Interpreter
General Structural Model
Post Processor
Entity-specific Structural Model 1
Entity-specific Structural Model 2
Entity-specific Structural Model k

Figure 3: Synthesis of Structural Models

Use of synchronization agents is a modular approach that eliminates the need to create custom simulation models for synchronization purposes. Simulation processes still need the functionality to communicate and interact in a distributed setting (such as compliance with HLA or some sort of client/server type communication capability). However, synchronization is totally isolated from the simulation process and it is the job of the synchronization agent, which runs as a separate process on the same computer with the simulation process.

One other functionality required of the simulation process is that it should be capable of paced execution synchronized with a hardware clock (i.e. system clock). This would require some sort of customization or outside control of the simulation kernel, unless the specific simulation kernel has built-in time scaled execution capability.

5 CONCLUSIONS

In this paper, we provided an overview of the adaptive synchronization framework being developed at the Penn State CIMLAB. This framework aims easy and rapid development of distributed simulation systems by automatically synthesizing synchronization agents from simulation models.

Synchronization agents control the pace of execution of simulation processes while processing run-time information to predict interactions between them. Interactions are predicted before they happen and this information is continuously shared among all synchronization agents. Using the information on predicted interactions, agents calculate safe interaction speeds (slower than an agreed upon no-interaction speed) and they synchronously (through the use of synchronized hardware clocks) regulate speed of all simulation processes dynamically.

Although this algorithm looks quite simple in theory, there are several issues that can affect the performance of this system when implemented. First, the communication system used for messaging must have adequate bandwidth matching the degree of coupling between simulation processes. The degree of coupling can be calculated by examining the number of input/output events in structural models.

Second, variability in certain key measures such as message transmission delays (difference between actual delays and the maximum delay) and entity-state delays (actual delays versus the minimum delay) will certainly affect the performance of the algorithm.

Third, synchronization of the hardware clocks must be done with reasonable accuracy so that synchronization agents can switch to different execution speeds with minimal time discrepancy. Although one can use barrier synchronization methods to correct errors rooted by this problem, this would result in extra overhead.

Our initial studies on the hardware clocks also revealed another important problem regarding the accuracy of the algorithm. The resolution of the hardware clock used for pacing the simulation process is extremely important. In theory, we cannot speed-up a simulation process more than the resolution of the real-time clock that we use to time the unit-delays between successive virtual clock ticks.

We realized that although hardware clocks inside personal computers have very high resolution (limited by the CPU clock frequency), it is not possible to sample these clocks with an accuracy better than 0.001 seconds under ordinary multi-tasking operating systems (this is due to the fact that the programmer does not have direct control over the scheduling of program threads run by the operating system). This creates a problem in implementing reasonably fast distributed simulators. However, one can use external clock frequency providers such as GPS satellite receivers (synchronized within 1 microsecond of the universal coordinated time) along with third party real-time operating system add-on components to solve this problem.

Above mentioned issues related to the implementation of the framework may look like divergence from regular hardware and software to implement the system. However, we believe that this framework is very versatile and the benefits from rapid development and ease of deployment can easily justify the additional cost of the hardware/software required for clock synchronization.

REFERENCES

Das, S. R. 2000. Adaptive protocols for parallel discrete event simulation. *Journal of the Operational Research Society*. 51: 385-394.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. New York: Wiley.

Hopcroft, J. E., R. Motwani, and J. D. Ullman. 2001. *Introduction to automata theory, languages, and computation*. 2nd ed. Boston: Addison-Wesley.

Joshi, S. B., J. S. Smith, R. A. Wysk, B. Peters, and C. Pegden. 1995. Rapid-CIM: An approach to rapid development of control software for FMS control. *27th CIRP International Seminar on Manufacturing Systems*, Ann Arbor, MI.

Page, E. H., D. M. Nicol, O. Balci, R. M. Fujimoto, P. A. Fishwick, P. L'Ecuyer, and R. Smith. 1999. Panel: strategic directions in simulation research. *In Proceeding of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H. B. Nembhard, D. T. Sturrock, G. W. Evans, 1509-1520. Piscataway. New Jersey: Institute of Industrial Engineers.

AUTHOR BIOGRAPHIES

BERTAN ALTUNTAS is a doctoral candidate in the Department of Industrial and Manufacturing Engineering at The Pennsylvania State University. He received his B.S. degree in Industrial Engineering from The Middle East Technical University in Turkey (1999). One year after his graduation, he was awarded the Fulbright Fellowship to continue his graduate studies in the U.S. In 2002, he received his M.S. degree in Industrial Engineering and Operations Research from The Pennsylvania State University. He was awarded the Material Handling Education Foundation scholarship in 2002. His research interests include parallel and distributed discrete event simulation, evolutionary distributed systems, cellular automata, intelligent computer integrated manufacturing and control of discrete event systems. He is a student member of IEEE, INFORMS, IIE and SCS. His email address is `<bertan@psu.edu>` and his web address is `<www.bertanaltuntas.com>`.

RICHARD A. WYSK, Ph. D. holds the Leonhard Chair in Engineering at Penn State University. Prior to his current position, he was director of the Institute for Manufac-

turing Systems and holder of the Royce Wisenbaker Chair in Innovation at Texas A&M. Dr. Wysk also served on the faculty of Virginia Tech and worked in industry as a research analyst for the Caterpillar Tractor Company and as production control manager for General Electric. He is a decorated Vietnam veteran. He is the author of several textbooks. Honors recognizing his research include the Institute of Industrial Engineers, David F. Baker Distinguished Research Award, and the Society of Manufacturing Engineers Outstanding Young Manufacturing Engineer Award. He holds his Bachelor's and Master's degrees in Industrial Engineering and Operations Research from the University of Massachusetts and a Ph. D. in Industrial Engineering from Purdue University. His work focuses on computer integrated manufacturing, computer automated manufacturing, computer-aided process planning and concurrent engineering. His email address is `<rwysk@psu.edu>` and his web address is `<www.engr.psu.edu/cim>`.