Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

DATA-DRIVEN SIMULATION OF NETWORKS WITH MANUFACTURING BLOCKING

Gordon M. Clark
Charles R. Cash

Department of Industrial and Systems Engineering
The Ohio State University
Columbus, Ohio 43210, U.S.A.

ABSTRACT

This paper presents a data-driven simulation of a network using manufacturing blocking to control work-in-process, and this simulation is called BNS for Blocking Network Simulation. The model is object oriented and is constructed in C++ using an object-oriented simulation environment called HOSE. The structure of the data-driven network simulator is illustrated with examples of the underlying code. An important feature of the model is a sequential procedure for determining run length to achieve statistical precision in an output performance measure. An illustrative example compares the performance of BNS with a queueing network approximation. The principal conclusion is that the data-driven network simulation can be applied by users with minimal simulation knowledge as a rapid modeling tool.

1. INTRODUCTION

The knowledge and time requirements to create models limit the use of simulation because the individuals with problem-domain knowledge frequently lack the skills required to construct and use simulation models. If they have the aptitude to learn how to use simulation models, they may not have the available time. We call a model *data driven* when users can apply the model to different situations by changing input data that only requires problem-domain knowledge with a minimal modeling knowledge requirement. As a result, a number of software companies offer products such as ProModel (Harrell and Tumay, 1992) and WITNESS (Murgiano, 1990) that allow users to define simulation models based on input data which significantly reduces the modeling knowledge requirement. However, in an attempt to be useful in diverse situations, these products allow the user to augment their data-driven models with specialized logic to represent particular processes. Of course, use of these features increases the user's modeling knowledge requirement. Typically, these products do not incorporate procedures for performing basic statistical analyses such as calculating confidence intervals and adjusting run lengths to achieve a desired statistical precision. RESQ (Gordon, MacNair and Welch, 1986) is an example of a simulation language that does, reducing the simulation user's knowledge requirement.

Simulation modelers need to develop domain-specific models as opposed to generic products that are data driven and only require minimal simulation knowledge so long as the users apply these models in a specific domain of application. That is, the simulation user should be able to apply his/her domain knowledge without learning a simulation language or mathematical statistics required for run-length control. The philosophy proposed in this paper is that simulation modelers should create domain-specific data-driven models that represent the characteristics of a class of systems such as a system of manufacturing cells using manufacturing blocking to control work-in-proces (WIP). This paper illustrates that approach by describing the architecture of a network simulation of work stations with fixed capacity buffers that control work-in-process (WIP) by manufacturing blocking. This paper refers to the simulation as the **B**locking **N**etwork **S**imulation (**BNS**). Construction of BNS uses a **H**eirarchical **O**bject-oriented **S**imulation **E**nvironment (**HOSE**) written in C++. BNS exploits two features of HOSE, viz., its object-oriented nature and its ability to allocate memory to new objects at run time. This paper illustrates how concepts in HOSE can be used to develop domain-specific data-driven simulations.

BNS was originally written to study cellular operating policies for SPECO, a precision gear manufacturer in Springfield, Ohio. The version written for SPECO incorporates several features omitted from this paper so that the details will not obscure the basic principles presented. The purpose of BNS is to facilitate extensive simulation experimentation of systems with manufacturing blocking. One could use BNS to perform "rapid modeling" analyses (Suri and de Treville, 1991;

Suri and de Treville, 1992) in which models are created in hours and executed in minutes. Suri and de Treville contend that simulation model development typically requires days or weeks and takes hours to execute on an PC; thus, they recommend queueing theory models for this purpose. See Snowdon and Ammons (1988) for a review of queueing network packages. This paper shows that simulation can be a rapid modeling tool with respect to model development; however, the execution times on a PC can be long as Suri observes. The performance of BNS on a test problem is compared to the manufacturing blocking approximation developed by Lee and Pollock (1990). The advantage of a simulation as a rapid modeling tool is the simulation model's inherent ability to represent diverse system structures.

Section 2 of this paper gives an overview of HOSE. The use of HOSE in developing BNS is described in section 3, and section 4 presents results from BNS and compares it to Lee and Pollock's queueing approximation.

2. OVERVIEW OF HOSE

The unique characteristic of HOSE is its structure that integrates concepts common to many simulation network languages with the DEVS formalism proposed by Zeigler (1984 and 1990). These concepts are implemented in C++ which is object-oriented, permits run-time memory allocation and de-allocation, and is very efficient. Lomow and Baezner (1991) and Joines et al (1992) describe other simulators implemented in C++.

2.1 Concepts Common to Existing Simulation Network Languages

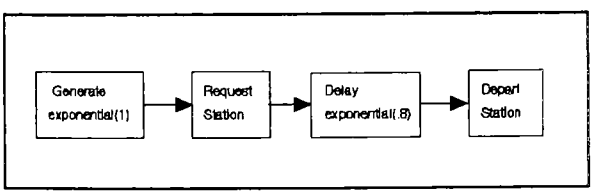

Figure 1 Single Station Queue

First, we will describe the particular interpretation in HOSE of concepts found in various simulation languages. Simulation entities are active objects in HOSE in that they are created, travel from node to node, and are destroyed when they no longer have nodes to visit. The simulation model can have resources which have a limited capacity. Entities can attempt to enter a resource at a request node at which the entity can request ϕ units of the resource capacity. If the available resource capacity is at least ϕ then the entity immediately enters the resource; otherwise, the requesting entity is placed in a queue list defined for each resource to wait for adequate capacity to be made available. Entities leave a resource at depart nodes at which they release ξ units of capacity causing the available resource capacity to be increased by ξ. Entities experience explicitly timed delays at delay nodes. No limit exists as to the number of entities at a particular delay node and the time at a delay node can either be a deterministic quantity or a sampled value for a probability distribution. Entities are created at create or generate nodes. Generate nodes produce a sequence of entities with a specified distribution of times between creation for the individual entities. Create nodes produce a single entity either at a scheduled time or when another entity arrives to a create node. Each type of entity follows a list of nodes. Figure 1 illustrates a sequence of nodes for a single station queueing system with exponentially distributed interarrival times and exponentially distributed service times. When an entity reaches the end of its node list, the entity's total life time in the simulation is automatically computed before the entity is destroyed. HOSE automatically produces output summaries that include:

- Queue length statistics for each resource
- Queue delay time statistics for each resource
- Utilization statistics for each resource
- Time in resource statistics for each resource
- Number of entities in the system for each entity type
- System time statistics for each entity type.

2.2 DEVS Concepts in HOSE

HOSE incorporates many concepts defined by Zeigler (1984 and 1990) to give it a very modular structure and the ability to represent hierarchical models. Zeigler defines an atomistic model using the Discrete EVent System (DEVS) formalism. These atomistic models are sufficiently modular so that they can be coupled together to form coupled models that are also DEVS models. A DEVS model includes adequate data to completely specify its state. DEVS models change state at event times. These events can be either external or internal events. An external event generates a state change caused by another DEVS model. An internal event is a state change caused by the DEVS model itself. The analogy in HOSE is that all nodes are DEVS models in the sense that they can have internal events, external events, and they manage their own data depicting their state. For example, an entity arriving to a delay node is an external event for the delay node. The delay node maintains a list of entities in the delay and their

scheduled times for departure from the delay. Departure of an entity from the delay is an internal event occurring without stimulus from another node. A generate node has an internal event when it produces a new entity. The data for a generate node includes an identifier for the particular node list that entities produced by the generate node will follow. The node also maintains a count of the number of entities created and destroyed by the node. The analogy between a DEVS model and resources in HOSE is less direct. Think of a resource and the collection of all request nodes and depart nodes refering to the resource as a DEVS model. The resource includes data depicting its state by maintaining a list of all entities waiting for the resource, i.e., the resource queue, and a list of all entities actually inside the resource. An arrival to a request node is an external event for the resource. The particular request node identifies the particular external event port defined by Zeigler for a DEVS model. HOSE routes a departure from the resource to the next node on the entity's node list. The depart node could be regarded as an output port for the resource. Also, the arrival of the entity to the next node is an external event for that node.

Zeigler (1990) defines a number of different messages sent between DEVS models to coordinate their external and internal events. HOSE has a much more simpler method for coordinating events. A sequencer object maintains a list of events, but each event on the list is an internal event for a node. When a node, call it the previous node, sends an entity to its next node causing an external arrival event, that arrival event is represented immediately after the state of the previous node has been updated. Thus, external events never appear on a sequencer's event list. Some loss in modularity occurs in HOSE because of this simplification, but HOSE also realizes a considerable increase in computational efficiency.

The structure of HOSE allows for a model node that has a sequencer object as a data member. The overall simulation model has its sequencer object which is called the base sequencer, and the overall simulation model can have multiple model nodes. This permits hierarchical models which have yet to be implemented.

2.3 Implementation in C++

A Class defines an object-oriented data structure in C++. This class structure defines data members and functions for the class. The C++ program can create individual instances of each class where each instance has its own particular data values. The class functions have access to those data values. The node class is an important class in HOSE. Each instance of a node class has an **arrival** function and a **doIt** function. The **arrival** function causes an entity arrival from another node. The **doIt** function causes the node to perform an internal event. A sequencer object calls a node's **doIt** function after removing the next event from the sequencer event list. All particular node types are derived from the node base class. That is, request, delay, depart, generate, and other nodes are derived from the node class so they inherit the functions of the node class, viz., the **arrival** and **doIt** functions.

Another important base class is the entity base class. Entities have attributes such as:

- Entity creation time
- Pointer to the entity node list
- Current node on the node list

Particular applications of HOSE may require more attributes for an entity than those listed above. For example, a cart entity may have a list of job entities on the cart. To allow for additional attributes such as a list of entities, the modeler derives a new class, e.g., class cart from the entity base class. All existing data members and functions defined for an entity are then inherited by instances of the cart class objects.

The new operator in C++ is very important to HOSE Using the new operator, class instances are created at run time using memory from the free store or memory not used by existing data and programs. For example, the statement

```
entity *pEnt = new
        entity("JOB",entityNum,route,time,pGen);
```

creates a new entity and stores a pointer to the entity in the variable pEnt. The function entity is called a constructor function in C++. The input arguments to the entity constructor function are:

"JOB"	a character string identifying the entity type on output traces
entityNum	an integer uniquely identifying the particular entity of type entityNum
route	a list of nodes for the entity to follow
time	the current simulation time or the entity creation time
pGen	a pointer to the generator node creating the entity

The statement

```
delete pEnt;
```

returns the memory allocated to entity whose pointer is pEnt to the free store. At that time, a destructor function

defined for the entity class is called which reports the destruction of the entity to its source generator node pointed to by the variable pGen.

3. Blocking Network Simulation (BNS)

The BNS represents the processing of jobs by a network of work stations having fixed capacity buffers using manufacturing blocking to control WIP. See Onvural and Perros (1986) for a description of different types of blocking. External arrivals can occur to work stations specified by the user. These arrivals may be determined either by an input job release schedule (from the Material Requirements Plan (MRP)) or by interarrival probability distributions. If the initial work station for an entering job is blocked, the job is placed in the shop backlog which has no limits on number of jobs. This section illustrates the structure of the BNS program, the input data for an example simulation, and describes the run length control algorithm to achieve a user-specified level of precision in an output performance measure.

3.1 Illustrative C++ Code

BNS represents manufacturing blocking by assigning two resources at each work station. One to represent the work station and one to represent the buffer. A job requests a buffer prior to departing from its previous work station; thus, a full buffer prevents the job from leaving the previous work station, blocking its flow. BNS reads input data specifying the stations in the network, and then creates resources for the buffers and work stations. This can be done by the C++ statements:

```
// rName = a character string giving the resource name
// bufName = a character string giving the buffer
//                    name
// rQty = number of machines in the work station
// bsize = buffer capacity
in >> rName >> rQty >> bsize;
// read from input data stream.
resource *station = new resource(rQty,rName);
// station is a pointer to the work station resource
bufName = textCat(rName, "_Buffer");
// concatenate _Buffer to the end of the rName
//       character string
resource *buffer = new
                              resource(bsize,bufName);
// buffer is a pointer to buffer resource
```

In C++, // tells the compiler to regard the remainder of the record as a comment.

After creating resources, BNS reads data specifying the route through the network for each job type. Similar to a process plan, the data for job type gives a sequence of operations. Assume that a step in the operation sequence specifies the work station pointed to by the resource pointer workStation and that the previous work station has resource pointer prevStation, then the C++ code is:

```
new request(1,*stationBuffer);
new depart(1,*prevStation);
new request(1,*workStation);
new delay(rName,lognormal,stream,mean,stdDev);
```

In BNS, each job is a lot that can have multiple parts. Each job requests one unit of the respective resources. The delay time in the example assumes a lognormal distribution for the operation time. In BNS, the delay time can include setup, a run time for each part in the lot, and a batch size capacity for the particular work station. Some work stations such as soaking pits can perform their operation in batches of parts. In that case, the lot is divided into several batches if the batch size is smaller than the lot size.

3.2 Illustrative Input Data

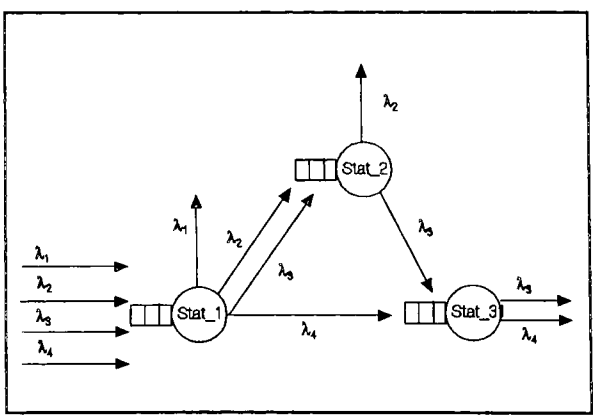

Figure 2 Three Station Network

A hypothetical example representing a simple manufacturing process illustrates the BNS program. In this example, three work stations (stat_1, stat_2, and stat_3) produce four job types (Type1, Type2, Type3, and Type4). Each work station consists of a buffer and a single machine. The buffer capacity is unlimited for the first station, but is restricted to 2 and 1 for work stations two and three. The order arrival processes are independent Poisson processes with rates λ_i = 0.16, 0.096, 0.224, and 0.32 for job types i=1,2,3 and 4, respectively. An order consists of a single part. Orders are processed by work stations according to the route

defined for a job type; however, the operation performed at work station one is the first sequence on all routes. Service times at a work station are exponential with rate 1 independent of the type of job being processed. Although, there are many performance measures of interest for this example, we are interested in estimating mean WIP. Figure 2 depicts a network diagram for this example.

To analyze this example using BNS, a user would simply create an ASCII text file illustrated below. The first record contains eight simulation model parameters. These parameters specify in order in which they appear: number of batches, initial deletion period, batch length, due-date assignment procedure, sample for arrival process or use an input job release schedule, confidence interval α-level, relative percent error (δ) of the sample mean WIP , and random number stream identifier. This example specifies an initial deletion period of 1000 time units, 20 batches with length 500 time units to estimate mean WIP within 10% of its true value with 90% confidence. An interarrival time distribution specifies order arrival times. A due-date assignment procedure is a feature that is omitted in this example. The following section discusses the remaining three parameters used to control of the simulation run length. The next record indicates the number of work stations to be modelled which is 3 in the example. Each work station record indicates the work station name, number of resources, and the capacity of the work station buffer (e.g., stat_2, 1 and 2). The unlimited capacity buffer for station one is represented by specifying a capacity which is not restrictive. The 4 on the next record indicates the number of job types to be modelled. This value is followed by the job name, number of sequences in its route, lot size, flow allowance time, interarrival time distribution, and distribution parameters (e.g. Type1, 1, 1, 2.0, exponential, 0.16). The number of sequences in the job types route specifies the number of following records containing data for the part's route. Each route sequence record specifies: the work station name, setup time mean, setup time coefficient of variation, mean cycle time, cycle time coefficient of variation, the operation's batch size processing capacity, and the operation time distribution. (e.g., stat_1, 0.0, 0.0, 1.0, 1.0, 1, exponential). The format for the job type data is repeated for three remaining jobs.

```
20  1000  500  0  0  0.10  0.10  1;
3;  // number of work stations
stat_1  1  1000;
stat_2  1  2;
stat_3  1  1;
4;  // number of job types
Type1  1  1  2.0  exponential  0.16;
stat_1  0.0  0.0  1.0  1.0  1  exponential;
Type2  2  1  4.0  exponential  0.096;
stat_1  0.0  0.0  1.0  1.0  1  exponential;
stat_2  0.0  0.0  1.0  1.0  1  exponential;
Type3  3  1  6.0  exponential  0.244;
stat_1  0.0  0.0  1.0  1.0  1  exponential;
stat_2  0.0  0.0  1.0  1.0  1  exponential;
stat_3  0.0  0.0  1.0  1.0  1  exponential;
Type4  2  1  4.0  exponential  0.32;
stat_1  0.0  0.0  1.0  1.0  1  exponential;
stat_3  0.0  0.0  1.0  1.0  1  exponential;
```

3.3 Run Length Control for Statistical Precision

BNS assumes "steady state" operation. If the user is representing a particular MRP, BNS assumes that the MRP is repeated a number of times where the state at the completion of a MRP becomes the input state for the next repetition of that MRP. Non-overlapping batch means is used to estimate confidence intervals for a single performance measure such as mean WIP or job throughput time. A batch is the period over which the MRP is defined. In studies for SPECO, a MRP for one calendar year was used and that MRP was repeated a number of times. The user inputs a stated precision level for the output performance measure expressed as a fraction, i.e., δ in the above example, of the mean value and a confidence level, i.e., $1-\alpha$, that the true mean value is within that fraction. The results reported in this paper used .10 for both δ and α. The run-length procedure used is a sequential procedure for achieving level of precision. We modified a procedure described by Law and Kelton (1991) for terminating simulations to be usable in the steady-state case using non-overlapping batch means.

The procedure requires the user to specify an initial transient period and a batch length, i.e., batchSize, measured in simulation time units. The value of batchSize in the above example is 500. The initial transient period is deleted from all statistical calculations. BNS always uses a fixed number of batches equal to 20. This value was selected based on the analysis performed by Schmeiser (1982) showing little advantage in using more than 30 batches. The procedure starts by simulating 20 batches of the specified length. If the confidence interval half width divided by the average of the 20 batch means is no more than δ then the simulation terminates reporting the average of the 20 batches. Otherwise, the simulation is continued for another 20 batches of length batchSize. Then, the batches are aggregated making 20 batches of length 2 * batchSize and the confidence interval half width is recomputed and checked again. More explicitly, the procedure is:

```
Simulate the transient period
j ← 0
halfWidth ← δ * xBar + 1
while (halfWidth > δ * xBar)
        Simulate 20 more batches of length batchSize
        Compute 20 batch means where each batch has
length j * batchSize
        halfWidth ← Confidence interval half-width
based on 20 batch means
        xBar ← Average of 20 batch means
end while
report xBar
```

4. Illustrative Results

Table I BNS Results with Exponential Service Times

	Average	Min	Max
Sim. Run Time	94,000	70,000	130,000
CPU Time (min)	29.71	24.80	40.75
Mean WIP Estimate	10.04	9.48	10.69

To indicate the performance of BNS, we simulated the three station system described in section 3.2. Also, we programmed and executed Lee and Pollock's (1990) approximation for that system. The mean WIP estimate obtained from Lee and Pollock's approximation is 9.9409 which is the same value reported by them. The exact value for that system is 9.9946 so Lee and Pollock's has an error of slightly more than .5%. The execution time for their approximation on a computer with an Intel 80486 cpu operating at 33 MHz was .0018 minutes. Table I presents the results from 10 replications of BNS with a 90% confidence interval half width requirement of 10% of the mean. The results from BNS verify the performance of the run length control procedure described in section 3.3. The results are clearly within 10% of the exact value; however, BNS required about a half hour to complete a simulation with the required precision. We note that increasing the required precision to 5% of the mean value will result in simulations that require about four times as long to complete.

Table II BNS Results with Uniform Service Times

	Average	Min	Max
Sim. Run Time	13,200	11,000	21,000
CPU Time (min)	1.07	0.77	1.48
Mean WIP Estimate	4.08	3.91	4.46

The above comparison matches a simulation with a queueing network approximation for a system configuration the approximation is specifically designed to represent. However, the simulation has the inherent ability to represent a wider range of system structures. The user of the queueing network approximation may have to employ a model of an inappropriate system structure. Table II shows results from BNS where the input data are modified to specify service times at all stations that are uniformly distributed over the interval (.5, 1.5). The mean WIP is now only 40% of its previous value, and the mean time for BNS to simulate this case is about 1 minute. Clearly, many users may prefer a simulation such as BNS as a rapid modeling tool when available queueing network approximations are inaccurate representations of system structure.

5. Conclusions

Based on the results presented in this paper, we offer the following conclusions:

1. Data-driven simulations can be constucted which require a minimal amount of modeling knowledge.

2. One can quickly initiate the execution of data-

driven simulations.

3. Data-driven simulations can be developed which represent specific system structures.

4. Queueing network approximations may give accurate results with very small computing times when the system structure is consistent with the assumptions inherent in the approximation.

5. Data-driven simulations may be the rapid modeling tool of choice when queueing network approximations of the desired system structure are unavailable.

6. The object-oriented nature and run-time memory allocation features of C++ are very desirable features for developing data-driven simulations

ACKNOWLEDGEMENTS

Development of BNS was partially supported by the IIT Research Institute under the Instrumented Factory (INFAC) for Gears Contract. Management of the SPECO Corporation inspired us to develop BNS by showing us the need for models to assist them in analyzing gear manufacturing operations. We appreciate the support of IIT Research Institute and SPECO.

REFERENCES

Gordon, Robert F., Edward A. MacNair, Peter Welch, Kurtis J. Gordon and James F. Kurose. 1986. Examples of Using the Research Queueing Package Modeling Environment. In *Proceedings of the 1986 Winter Simulation Conference*, ed. J. Wilson, J. Henriksen, and S. Roberts, 494-503, Institute of Electical and Electronic Engineers, San Francisco, California.

Harrell, Charles R. and Ken Tumay. 1992. ProModel Tutorial. In *Proceedings of the 1992 Winter Simulation Conference*, ed. by J.J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 405 - 410, Institute of Electical and Electronic Engineers, San Francisco, California.

Joines, J. A., K. A. Powell, Jr. and S. D. Roberts. 1992. Object-Oriented Modeling and Simulation with C++. In *Proceedings of the 1992 Winter Simulation Conference*, ed. by J J. Swain, D. Goldsman, R. C. Crain and J. R. Wilson, 145-153, Institute of Electical and Electronic Engineers, San Francisco, California.

Law, Averill M. and W. David Kelton. 1991. *Simulation Modeling and Analysis*. 2nd Edition, New York: McGraw-Hill, 536 - 537.

Lee, Hyo-Seong and Stephen M. Pollock. 1990. Approximation Analysis of Open Acyclic Queueing Networks with Blocking. *Operations Research*, 18:1123 - 1134.

Lomow, Greg and Dirk Baezner. 1991. A Tutorial Introduction to Object-Oriented Simulation and Sim++. In *Proceeding of the 1991 Winter Simulation Conference*, ed. B. L. Nelson, W. D. Kelton, and G. M. Clark, 157-163, Institute of Electical and Electronic Engineers, San Francisco, California.

Murgiano, Charles. 1990. A Tutorial on WITNESS. In *Proceedings of the 1990 Winter Simulation Conference*, ed. O. Balci, R. P. Sadowski, and R. E. Nance, 177-179, Institute of Electical and Electronic Engineers, San Francisco, California.

Onvural, R. and H. Perros. 1986. On Equivalencies of Blocking Mechanisms in Queueing Networks with Blocking. *Operations Research*, 34: 293-297.

Schmeiser, B. W. 1982. Batch Size Effects in the Analysis of Simulation Output. *Operations Research*, 30: 569 - 590.

Snowdon. Jane L. and Jane C. Ammons. 1988. A Survey of Queueing Network Packages for the Analysis of Manufacturing Systems. *Manufacturing Review*, 1:14-25.

Suri, Rajan and Suzanne de Treville. 1991. Full Speed Ahead. *OR/MS Today*, June 1991.

Suri, Rajan and Suzanne de Treville. 1992. Rapid Modeling: The Use of Queueing Models to Support Time Based Competitive Manufacturing. *Proceedings of the Germany/US Conference on Recent Developments in Operations Research*, ed. G. Fandel, Springer Verlag.

Zeigler, Bernard P. 1984. *Multifacetted Modeling and Discrete Event Simulation*. Orlando:Academic Press.

Zeigler, Bernard P. 1990. *Object-Oriented Simulation with Hierarchical , Modular Models: Intelligent Agents and Endomorphic Systems*. San Diego: Academic Press.

AUTHOR BIOGRAPHIES

GORDON M. CLARK is a professor in the department of Industrial and Systems Engineering at The Ohio State University. He received the B.I.E. degree from The Ohio State University in 1957, the M.Sc. in Industrial Engineering from The University of Southern California, and the Ph.D. degree from The Ohio State University in 1969. His current research interests include integrated decision support systems in a CIM environment and the design and analysis of efficient simulation experiments. He serves on the editorial board for the *ORSA Journal on Computing* and the *Journal of Operations Management*. In 1991 he served as program chair for the WSC.

CHARLES R. CASH is a Ph.D. student in the Department of Industrial and Systems Engineering at The Ohio State University. He received the B.S.I.S.E. degree in 1985 and the M.Sc. degree in 1986 from The Ohio State University. His research interests include simulation methodology, stochastic processes, and analysis of manufacturing systems. He is a member of IIE, SCS, and ORSA.