Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

SIMULATION OF ADVANCED MANUFACTURING SYSTEMS

Gerald W. Evans
William E. Biles

Department of Industrial Engineering
University of Louisville
Louisville, KY 40292

Michael W.Golway

SSOE, Inc.
1001 Madison Avenue
Toledo, OH 43624

ABSTRACT

This paper gives an overview of how simulation modeling techniques can be employed in the design and analysis of advanced manufacturing systems. The reasons for the complexities of these systems, as well as the uses of simulation software packages, are discussed. Finally, examples of simulation models of advanced manufacturing systems, as developed by the authors, are presented.

1 INTRODUCTION

Advanced manufacturing systems (AMSs) are those which manufacture parts with the material handling functions, machine operations, and machine tools under the control of a computer (Herald and Nof, 1978). The terms advanced manufacturing system, automated manufacturing system, and computerized manufacturing system have been used interchangeably in the literature (Gupta, 1990).

Examples of AMSs include flexible flow systems, flexible manufacturing systems, and flexible manufacturing cells. These systems can be composed of components such as robots, NC machining centers, automated guided vehicle systems, etc. The key characteristics of an AMS however include its flexibility (i.e., ability to produce a wide variety of parts with low set up times) and computer control.

The complexities of these systems basically result from their flexibility. These complexities make AMSs very difficult to design and operate. For example, Suri (1985) identified five phases of problems associated with the design and operation of flexible manufacturing systems: initial design; detailed design; installation; production planning, scheduling, and operation; and ongoing modifications. Each category of problems has its own particular set of design variables and performance measures; and, each set of decisions made with respect to one problem area affects the subsequent problem areas.

Examples of decisions which must be made in the design and operation of an AMS include the types of parts to be produced, the types and numbers of production machines and material handling equipment to include in the system, the layout of the system, the numbers of pallets/fixtures for each part type, the potential routings for each part type, local and global buffer capacities, lot sizes, sequencing rules, tool assignments, production rates, dispatching rules, etc. Some of these decisions (e.g., system layout) may be made only once every few years, while others may be made on a daily basis (e.g,. production schedules).

These decisions must be made while accounting for a variety of performance measures, including system cost and flexibility, due-date performance, quality of parts produced, production rates, inventory levels, machine utilizations, etc.

Simulation modeling can be a tremendous aid in the design process for an AMS. The purpose of this paper is to give the reader a brief overview of how this can be accomplished. Specifically, in the next section of the paper, we discuss the complexities associated with modeling AMSs. In the third section, we give a brief review of simulation software packages available for modeling AMSs. The fourth section of the paper contains several examples of the modeling of AMSs performed by the authors. Finally, the last section of the paper contains a summary and conclusions.

2 COMPLEXITIES ASSOCIATED WITH THE MODELING OF AMSs

In basic terms, advanced manufacturing systems are complex because

1) A wide variety of parts are typically produced by an AMS, and
2) A number of different resources must interact in a complex fashion in order for the system to operate efficiently.

The system may be able to produce various part

types simultaneously. Because of the typical AMS's flexibility, a particular type of part may follow any of several different routes. In addition to the complex routing decisions which may have to be modeled, decisions regarding the sequencing and scheduling of parts and resources must also be modeled. Again, this process is made more complex as a result of the variety of part types that can be produced by the system.

The types of resources associated with an AMS include pallets, fixtures, tools, robots, machines, conveyors, automated storage/retrieval systems, AGV's, AGV guidepaths, machine operators, maintenance personnel, inspection and testing equipment, etc. The simultaneous use of various resource types (e.g., a pallet, a machine, a conveyor, and a human operator) may have to be modeled. Hence, resource allocation decisions are not trivial.

Many researchers have recognized that the material handling subsystem of AMSs are typically very difficult to model (Chapter 13 of Law and Kelton, 1991). Again, this difficulty has to do with the complex interactions between and among material handling resources (e.g., AGV's, AGV guidepaths, conveyors, robots), production resources (e.g., machine tools), and parts.

Because of these modeling difficulties, some simulation languages have included special material handling modules. These modules can be viewed as simulators that can be employed within a larger system model constructed from the language. As an example, the SIMAN language allows an INTERSECTIONS element, a LINKS element, and a NETWORKS element, among other constructs for modeling AGV systems.

3 SIMULATION LANGUAGES AND SOFTWARE PACKAGES

Recent releases of simulation software packages have taken the modeling of AMS's to a new level. The previous clear cut boundaries between simulation languages and simulators of the past have slowly evaporated. Simulation packages such as Arena, ProModel, and SimEngine have introduced the ability to quickly develop models of complex systems without sacrificing the flexibility of using a simulation language. This new found speed with model development is primarily due to the use of object oriented modeling constructs and integrated graphical animations.

Object oriented modeling constructs have introduced the single most important advancement in simulation model building. The constructs combined with elaborate graphical user interfaces (GUI's) provide for simple high level programming. Holzner (1992) points out that the driving force behind an object is modularity. This means that we can think of the object in terms of its overall function and not concern ourselves with detailed internal data handling. Consequently, our programming effort has better structure and promotes the ability to make changes without disrupting existing code.

Joines, Powell, and Roberts (1993) note that object oriented simulation modeling has great appeal because its very easy to view the real-world as being composed of objects. AMS objects come in the form of robots, conveyors, and AGV's to name a few. Users of modern simulation languages/packages can build complex simulation models of an AMS through the integration of bitmap images (objects). Parameters for each of these objects are entered into easy-to-use dialog boxes. The modeling elements (e.g., resources, material handling equipment, etc...) are created and linked through a simple "point and click" mouse interface.

Arena's template concept provides a powerful approach to integrating the robust SIMAN V simulation language with industry specific applications (Collins and Watson, 1993). Each template provides the user with modeling constructs that combine common SIMAN V commands into a single object. For example, the SERVER construct is a general purpose object that is derived from server data, ENTER and LEAVE modules. Detailed constructs can also be created with System Modeling Template Developers kit. The template concept provides the capability of allowing simulation modeling to be used by a wide range of users.

Packages such as Arena, ProModel, and SimEngine also retain their conventional lower level programming language power. The user always has the capability to link external code or drop down into the simulation language to model special situations.

Graphical animations have moved from a traditional non-value added programming effort to powerful value added visual tools. This stems from the fact that developing graphical animations essentially coincide with model building. Defining parameters for operator walking paths to conveyor velocities are easily set through dialog boxes. Furthermore, detailed alterations of default bitmap images can be accomplished through the program's graphical editor.

Graphical animations provide one important advantage when modeling a complex AMS. This advantage is reduced verification time of the model. Visual depictions of a system greatly increase a user's confidence in the correctness of its representation. Take, for example, a combined continuous and discrete system that has a non-accumulating conveyor and drum filling station. The system's bottleneck is the fill rate into the drum. That is drums are "pulled" into the fill station. Individual empty drums are indexed into the fill

station one at a time where product is poured to a desired level. Once the level is reached, the drum is indexed out and a corresponding empty drum takes its place. Verifying this system's operation is easily accomplished with a graphical animation. In Arena, the combined discrete and continuous modeling of this system is quickly accomplished. After the differential equation has been set up to model product flow, a graphical representation in the form of product level (tank level object) and the instantaneous product level variable are accomplished in a total of four steps. When the user starts the simulation it is easy to verify that the drums are correctly indexing into the fill station. Moreover, the user can visually see the product fill into the drum, reach the desired level, and index out of the station. Verification of this system without animation would require faith in the output report's statistical averages. Non-graphical animations often force the user to plant "feelers" in the code to try and gain a better understanding of how the model is performing.

4 EXAMPLES

In this section of the paper, we detail several examples involving the modeling of AMS's. The purposes of these examples are, first, to illustrate some of the complexities associated with the modeling of AMSs and, second, to illustrate how to model these complexities with a specific language/system. The Arena and SLAMSYSTEM packages were used, but these models could have been constructed using any of several different languages. In fact, alternative approaches even within Arena or SLAMSYSTEM could have been used.

For other examples of simulation models of AMSs, see any of several textbooks on simulation modeling (e.g., Pegden, Shannon, and Sadowski, 1990) as well as the Proceedings of this Conference (e.g., Davis, 1986 and Jeyabalan and Otto, 1991).

4.1 Modeling of a High Speed Package Line

The purpose of this project was to develop a simulation model that would aid in the analysis of the relocation and equipment upgrading of a high speed packaging line. Specifically, management wanted to answer the following questions:

1) What is the effect on cycle time of adding dedicated docks to the product line?
2) How many docks should be added?
3) What are acceptable minimal raw material inventories?
4) What effect does upgrading specific equipment components have on the system's bottleneck(s) and cycle time?
5) What is the availability of the system based on existing component jams and machine failures?

Due to the proprietary nature of this project, specifics on product type, packaging components, and manufacturing methods can not be disclosed.

The system was modeled as a combined continuous/ discrete simulation using Arena. There are a total of 22 resources used in the model. The resources represent equipment in the package line, a single operator, and component jam location points. The equipment components are linked together with 10 conveyors. In addition, the system is fed with raw material components from a warehouse by a transporter.

The finished product fill operation is modeled with Arena's continuous modeling capabilities. Product is stored in a holding tank that's supplied by a separate mixing operation. Product is continuously pumped from the holding tanks into an in-line surge tank. The tank serves as buffer for the time required to index packaging components in and out of the fill station. From the surge tank, the product is discharged into the finished goods packaging components.

The pre-fill operation primarily involves automated component sorting. Raw material components are transferred from their shipping boxes into holding bins. The components then move through a series of conveyors that properly orient them for filling. The post mix operations involve batching and boxing, bar coding, and automated palletizing.

There were two particularly difficult modeling concepts associated with this simulation. In turned out that Arena easily handled both concepts through its high level programming interface. The modeling of this system was also accomplished with the standard Arena templates. The first modeling feature involved batching of entities between two conveyors while maintaining a "pull" production system. The second feature was the integration of the continuous fill operation with a discrete package component conveyor line. The third feature involved using an infinite loop to check raw material restock levels.

4.1.1 Pull System Batching Between Two Conveyors

This problem feature is very common in AMS's. The idea is that entities from conveyors are batched together in a specified quantity and transferred to a second conveyor. This event occurs in three separate places with the current example. The first instance occurs when individual components are batched in a group and loaded into the filling machine. A second

occurrence is when the filled units are batched again for carton packaging. Finally, the conveyor batching occurs at the end of the production line when the palletizer batches boxes of product onto a pallet.

Arena provides a very useful BATCH module that conveniently batches entities according to a user defined quantity. In addition, there is a fair amount of flexibility in defining how attributes are handled and whether the batched entity is permanent or temporary. There are two problems with using a BATCH module in conjunction with the two conveyors while maintaining a "pull" system.

The first approach used was to ACCESS a cell on conveyor 2 before EXITing conveyor 1. The BATCH module was originally placed before the EXIT module. This caused the program to freeze the process of moving entities on the conveyors. The problem with this approach is that a new batched entity is not released until the user defined batch quantity is reached. Therefore, when the first entity successfully ACCESSes the second conveyor and moves into the BATCH node it will remain there until the batched quantity level is met. Consequently, the EXIT conveyor 1 module is never reached. This in turn causes the first conveyor to fill up to capacity a stall. Conveyor 1 cannot free up a cell until an entity reaches the EXIT module.

The second approach that can be used is to place the BATCH module after the EXIT module. In this case, the entities ACCESSing conveyor 1 are not inhibited by the BATCH module reaching its required batch size. This creates an additional problem, however, because the "pull" system control is lost. When entities reach the EXIT module they immediately go into an internal queue. This creates a condition where conveyors 1 and 2 operate separately from each other. Conveyor 2 may be filled to capacity or stopped (e.g., a fill station failure) while conveyor 1 continues to EXIT entities.

The solution to the problem was actually a modification of the second attempt. Arena provides a very simple if-then variable control with the CHOOSE module. A variable called Batch 1 was created to keep track of the number of entities EXITing conveyor 1. After an entity passes through the EXIT module it immediately goes into the CHOOSE module. The following conditions are then applied:

IF Batch 1 ≥ 5 THEN Stop 1
ELSE Assign 1

Stop 1 was a label name assigned for a STOP module. If Batch 1 reaches the required level then conveyor 1 is stopped. This entity is then routed to the Assign 1 label. Assign 1 is the label name given to the ASSIGN module. If Batch 1 is less than 5 then the entity is routed to a node where the variable Batch 1 is incremented by 1. The BATCH module follows the ASSIGN node and when the required number of entities have been batched a new entity is released and sent to a second ASSIGN model. Here Batch 1 is reset to zero. The new entity ACCESSes conveyor 2 and when this is successfully accomplished the entity passes through a START module. The start module restarts conveyor 1 and the whole process is repeated.

4.1.2 Combined Continuous/Discrete Modeling

The combined continuous and discrete modeling capabilities of Arena proved to be very useful with this project. The high level interface provided a convenient and quick method to model the flow of product from a holding tank to an in-line surge tank and finally into multiple containers. To model continuous flow in Arena required the used of LEVEL, RATE, DETECT, ASSIGN, and CONTINUOUS modules.

There were a total of three LEVELS used. The first LEVEL is the holding tank (Tank 1), the second LEVEL is the in-line surge tank (Tank 2), and the third level is a representation of the container being filled. VARIABLES in the for of Tank1Level, Tank2Level, and ContainerLevel, respectively, were assigned for each of these LEVELS. Animation of the tank levels were also easily accomplished by inserting a Level Status into the workspace region. The LEVEL variables were used as expressions for the Level Status. RATE variables were assigned to represent overall flow in or out of the tanks. The VARIABLES are Tank1Flow, Tank2Flow, and ContainerFlow, respectively.

DETECT modules were used to for maximum and minimum critical values in each of the tanks. When a level reaches either of these critical points an entity is created from the DETECT module and routed to Assign module. As the entity passes through the ASSIGN module, VARIABLES and OTHER assignment types are set. For example, if the minimum level is reached in the holding tank the VARIABLE Tank1In is set to 100. This symbolizes a valve opening to allow 100 units per minute of the product to enter the tank. This value is higher than Tank1Out which causes the holding tank to refill. The OTHER assignment type is used to redefine the RATE variables. For example, each time a critical value is reached in the hold tank, Tank1Flow is reassigned to Tank1In - Tank1Out. After the entity exits the ASSIGN module it is immediately DISPOSED. A second DETECT module is used for the holding tank to determine when the maximum level is reached. At this occurrence, the DETECT module will release a separate entity and again route it to an ASSIGN module. This time the ASSIGN module sets the VARIABLE

Tank1In to 0, which represents the valve to the tank closing. OTHER is used to reassign the RATE variable Tank1Flow to Tank1In-Tank1Out.

The CONTINUOUS module provides the user with a great deal of flexibility. Three differential equations were assigned to represent each of the respective tank material flows. Runge-Kutta-Fehlberg (RKF) was selected for the method of integration. This method was chosen because the rate equations do not remain constant between event times. Corresponding minimum and maximum step sizes, absolute and relative errors, severity, and severity warnings were also set.

To accurately model the product filling operation required a combination of continuous and discrete logic. The holding tank serves as a large storage queue from which product is continuously pumped. The product is piped from the holding tank to an in-line surge tank. This tank serves as a buffer for the time delay incurred when containers are indexed in and out of the filling machine. Its primary purpose is to minimize the starting and stopping of the holding tank pump motors. Material is then continuously pumped from the surge tank into the containers, at the instance the containers are positioned under the fill station. Consequently, the surge tank level is cyclical during steady-state operation.

The link between the continuous and discrete logic was accomplished through the START and STOP modules for conveyor movement. The filling machine, in this example, connects two separate conveyors. The first conveys containers into the filling machine (Infeed1). The second conveys filled containers to another workstation (Conveyor3). A DETECT node was used to monitor the product level in each container. Once the desired filling level is reached, the DETECT node creates an entity that is routed to a START module. Settings are then made to restart in Infeed1 conveyor. After the entity exits the START module it is routed to an ASSIGN module. In the ASSIGN module a variable is set to turn off the flow rate from the surge tank. OTHER assignments are made to reassign container flow rate and surge tank flow rate. In addition, the container LEVEL is reset to 0. The entity is routed to a DISPOSE module immediately following the ASSIGN module. In a separate section of the workspace, modules are used to model the conveyor STATIONs. When a container entity reaches the end of the Infeed1 conveyor it passes through an ASSIGN module. The ASSIGN module causes the valve on the surge tank to open. This is a corresponding fill rate for the machine. The container entity then proceeds to a STOP module for the Infeed1 conveyor. This prevents additional container entities from accessing and jamming the conveyor. The entity will remain in the STOP module until the START module is reached by the DETECT entity described above. When the container entity is released it moves to another STATION and attempts to ACCESS Conveyor3. If this is accomplished, the entity the proceeds to EXIT Infeed1 and convey to the next station. An important result of this logic is that the primary system bottleneck is accurately modeled. Containers are "pulled" into the filling machine and regulated by both the fill rate and conveyor index time.

Animation provided a quick method to verify that this part of the model was properly working. The PLOT module was used to view the surge tank level while the simulation was running. Fill rate variables were then quickly adjusted to balance the surge tank flow based on the results of the graph. Furthermore, visual confirmation of the tank levels and containers properly filling and indexing out of the station provided confidence in the model's representation of the "real world" system.

4.1.3 Infinite Loop Checking for Replenishment of Raw Material Inventories

A standard modeling concept present with any AMS involves the replenishment of raw materials at the point where it is used in the process. A typical scenario involves a resource (e.g., operator, fork truck, AGV, etc...) retrieving a unit load of raw materials from a warehouse. The material is transported from the warehouse to the processing line where it is used in the final product. The standard Arena Template provides an easy method to accomplish this activity.

A "loop" is first established to monitor the in-process raw material inventory level. An ARRIVE module is used to crate a single entity at the start of the simulation run. The entity immediately enters a CHOOSE module where an if-then decision is made. If the inventory level, which is defined by a single VARIABLE, is above the desired reorder point then the entity is routed to a DELAY module. Here, the entity is delayed for a predefined time interval and released back to the CHOOSE module. The same inventory check is again tested. If the inventory level is below the reorder point then the entity is routed to a SIGNAL module, and immediately routed back to the CHOOSE module. The SIGNAL module is used in concert with the WAIT module. It instructs the WAIT module to release a defined quantity of entities that have entered its module. These entities were created from a separate ARRIVE module that represents inventory arriving in the warehouse. After the entity is released from the WAIT module, it is sent to a REQUEST module. This module attempts to secure a defined quantity of a particular TRANSPORTER. When this is successfully accomplished the entity is routed to an UNSTORE

module where raw materials are removed from the warehouse inventory and transported to the processing area. Before the TRANSPORTER is released, delay times for unloading the inventory occur and the in-process inventory level is adjusted accordingly.

Two separate networks of modules exist with this modeling scenario. The first network uses a single entity control an event in a the second network. The second network models the actual raw material entities used in the process. The two networks are easily connected through the SIGNAL and WAIT modules. Consequently, a great deal of control is provided to the user with minimal use of Arena modules.

4.2 Modeling of a Semi-automated Assembly Line

Evans and Biles (1992) described the modeling and analysis of a semi-automated assembly line involving 29 work stations, followed by a 5-station test/repair facility.

The purpose of the project was to develop a simulation model that would allow for experimentation with various design scenarios, involving changes to the current system, with respect to the:

1) number of back-up machines at each station,
2) locations and sizes of external buffers,
3) cycle times at the stations,
4) elimination of machine downtime,
5) velocity of pallets on the conveyor, and
6) number of pallets.

The product came in two different types of frames; the first frame type had four different types of models, while the second frame type came in five different types of models. Hence, nine different types of assemblies were produced by the system. The cycle times at the stations did not change from one type of assembly to another; however, there were set-up times involved at some stations when production was switched from one type of assembly to another.

The assemblies moved sequentially from one station to the next through the use of an accumulating (queueing) conveyor. Each assembly was contained on two different types of pallets, labeled A and B, as it moved through the system.

Most of the stations contained an "internal buffer" of size 1--i.e., one assembly at a time was processed at the station. However, a few of the stations contained an internal buffer with a capacity greater than one. For example, the sixth station was a washer that could accommodate 39 assemblies simultaneously.

The SLAMSYSTEM package was used to model the system. Each run of the model required the processing of three SLAMSYSTEM files: a control file, a network file, and a user insert file (consisting of FORTRAN code). Some 31 different types of resources were modeled -- one type for each of the 29 stations, and one type each for type A pallets and type B pallets.

The completed assembly produced in the AMS described above then entered a test and repair line where it was tested at each of four stations and, if defects were found, repaired at yet a fifth station. The physical system and process flow presented a far less complex situation than did the assembly line, so that the test and repair line was entirely modeled using the network features of SLAM-II. The advantage of the network modeling approach was that the network flow almost exactly mimicked the digital control logic of the system.

In numerous instances in the test repair model, the simulation model behaved exactly like the programmable logical controller (PLC) which monitored and controlled that segment of the system. For example, an entity arriving at a GOON node represented a pallet arriving at a processing station and tripping a switch. The PLC logic was as follows:

1) If the processing station (an AWAIT node representing that resource) was open (FREE), the test pallet (entity) containing the assembly was placed in the processing station and the operation (service activity) commenced;
2) If the processing station was busy, an activity was initiated which rerouted control back to the GOON node with a delay of 1 second, just as the PLC would do.

This "logic loop" was repeated every 1 second until the processing station (AWAIT node) was available (FREE).

The ultimate advantage of the test/repair line simulation model was that the manufacturing engineer in charge of the line could evaluate changes in equipment layout as well as PLC control logic in a matter of minutes. For example, a change in logic sequence was as simple as inverting two statements in the SLAM-II network model, while relocating the repair station away from the other test equipment involved nothing more than changing the destination (LABEL) of an entity in an ACTIVITY statement. Adding a new machine to the line was somewhat more involved, and required as much as 10 minutes to effect the required alteration to the SLAM-II simulation model.

5 SUMMARY AND CONCLUSIONS

Having a valid simulation model available gave the manufacturing engineers in the above examples much

greater confidence in making alterations in the existing production system. For example, for just a few thousand dollars, the manufacturing engineer associated with the second example above, was able to purchase a PC version of SLAMSYSTEM, engage consultants to show him how to model the production system for which he was responsible, and evaluate investments in new production equipment costing in excess of $1 million. He was then able to show his plant manager just how to achieve the corporate production goals for a new product line.

The key to this manufacturing engineer's success was that, through simulation, he was able to develop very detailed and accurate models of the AMS. He saw the need to have a tool that allowed him to evaluate changes in material flow, equipment placement, operating policies, and PLC logic before experimenting with those changes in the actual system. The fact that the model predicted the effect of proposed line changes within 2 percent gave him the confidence to employ simulation for more involved and expensive alterations in production system design.

The main advantages of modern simulation software packages is their abilities to help the simulationist model complex systems such as AMS's more easily. Sophisticated animations allow simulation methodology to be more easily "sold" to management; animations also can result in quicker verification of simulation models.

Another advantage of modern simulation software packages is their abilities to aid in the management of the entire simulation project through their file management system.

A drawback still associated with simulation software packages concerns their deficiencies in optimization analysis of the simulation output. The methodologies required in these areas (e.g., design of experiments, variance reduction techniques, response surface methodologies, nonlinear optimization, and multicriteria optimization) are, in general, unfamiliar to most simulationists in industry. Yet, many of the modern software packages still offer little help in this area.

ACKNOWLEDGEMENT

This paper represents an updated version of a paper presented at the 1992 Winter Simulation Conference (Evans and Biles, 1992).

REFERENCES

Banks, J. 1991. Selecting simulation software. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 15-20, The Society for Computer Simulation, San Diego, CA.

Banks, J., E. Aviles, J.R. McLaughlin, and R.C. Yuan. 1991. The simulator: new member of the simulation family. *Interfaces*. **21**, 76-86.

Collins, N., and Watson, C.M. 1993. Introduction to Arena. *Proceedings of the 1993 Winter Simulation Conference*, eds., G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles, 205-212, The Society for Computer Simulation, San Diego, CA.

Davis, D.A. 1986. Modeling AGV systems. *Proceedings of the 1986 Winter Simulation Conference*, eds., J. Wilson, J. Henriksen, and S. Roberts, 568-574, The Society for Computer Simulation, San Diego, CA.

Directory of Simulation Software. 1991. Elliot Estrine (Ed.), Society for Computer Simulation, **2**.

Evans, G.W. and W.E. Biles. 1992. Simulation of advanced manufacturing systems. *Proceedings of the 1992 Winter Simulation Conference*, eds., J. Swain and D. Goldsman, 163-169, The Society for Computer Simulation, San Diego, CA.

Goble, J. 1991. Introduction to SIMFACTORY II.5. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 77-80, The Society for Computer Simulation, San Diego, CA.

Gupta, M. 1990. An evaluation of operations planning and scheduling problem in advanced manufacturing systems. Ph.D. Dissertation, Department of Industrial Engineering, University of Louisville, Louisville, KY.

Harrel, C.R., and K. Tumay. 1991. ProModel tutorial. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 101-105, The Society for Computer Simulation, San Diego, CA.

Herald, M.J. and S.Y. Nof. 1978. The optimal planning of computerized manufacturing system. Report No. 11, School of Industrial Engineering, Purdue University.

Holzner, S. 1992. Borland C++ Programming. Brady Books, New York, NY.

Jeyabalan, V.J., and N.C. Otto. 1991. Simulation models of material delivery system. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 356-364, The Society for Computer Simulation, San Diego, CA.

Joines, J.A., Powell, K.A., Roberts, S.D. 1993. Building object-oriented simulations with C++. *Proceedings of the 1993 Winter Simulation Conference*, eds., G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles, 79-88, The Society for

Computer Simulation, San Diego, CA.

Krahl, D. 1991. Tutorial: scheduling manufacturing systems with FACTOR. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 128-131, The Society for Computer Simulation, San Diego, CA.

Law, A.M., and S.W. Haider. 1989. Selecting simulation software for manufacturing applications. *Proceedings of the 1989 Winter Simulation Conference*, eds., E.A. MacNair, K.J. Musselman, P. Heidelberger, 29-31, The Society for Computer Simulation, San Diego, CA.

Law, A.M. and W.D. Kelton. 1991. *Simulation Modeling and Analysis*, Second Edition. New York: McGraw-Hill.

Law, A.M. and M.G. McComas. 1991. Secrets of successful simulation studies. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark, 21-27, The Society for Computer Simulation San Diego, CA.

Pegden, C.D., R.E. Shannon, and R.P. Sadowski. 1990. *Introduction to simulation using SIMAN.* Highstown, New Jersey: McGraw-Hill, Inc.

Pritsker, A.A.B. 1986. *Introduction to Simulation and SLAM II*, Third Edition. New York: John Wiley and Sons.

Suri, R. 1985. An overview of evaluative models for flexible manufacturing systems. *Annals of Operations Research*. **3**, 13-21.

AUTHOR BIOGRAPHIES

GERALD W. EVANS is a Professor in the Department of Industrial Engineering at the University of Louisville. He received a B.S. in Mathematics, an M.S. in Industrial Engineering and a Ph.D. in Industrial Engineering, all from Purdue University. Prior to his current position, he was an assistant professor in the School of Business at the University of Louisville (1981-1983), a senior research engineer at General Motors Research Laboratories, and an industrial engineer for Rock Island Arsenal. His research interests includes multicriteria optimization and simulation modeling, especially as applied to problems in manufacturing system design and operation, engineering management, and the service industries. He was an Editor of the **1993 Winter Simulation Conference Proceedings**. He is an active member of IIE, TIMS, ORSA, and DSI.

WILLIAM E. BILES is the Edward R. Clark Professor of Computer-Aided Engineering in the Department of Industrial Engineering of the University of Louisville in Louisville, KY. He is currently involved in research in two principle areas: (1) simulation modeling of automated manufacturing systems, and (2) computer-integrated manufacturing of plastics. He is Associate Editor of the journal **Computers and Industrial Engineering**. Dr. Biles received the BS in Chemical Engineering from Auburn, the MSIE from the University of Alabama-Huntsville, and the Ph.D. in IEOR from Virginia Polytechnic Institute and State University. He has held industrial positions with Union Carbide and Morton Thiokol, and faculty positions at Notre Dame, Penn State, and LSU. He is a Fellow of IIE and chairman of the GRE engineering exam committee for Educational Testing Service.

MICHAEL W. GOLWAY is an Industrial Engineer with SSOE, Inc., a consulting firm located in Toledo, Ohio. He received the BES in Industrial Engineering from the University of Louisville. He was formerly a Senior Engineer with Biles & Associates, Inc., in Louisville, KY where he was primarily involved in the application of simulation to facility layout and design. Mr. Golway is a member of IIE.