Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A SIMULATION TESTBED FOR COMPARING THE PERFORMANCE OF ALTERNATIVE CONTROL ARCHITECTURES

Paul Rogers
Robert W. Brennan

Division of Manufacturing Engineering
Department of Mechanical Engineering
University of Calgary
2500 University Drive NW
Calgary, Alberta T2N 1N4, CANADA

ABSTRACT

This paper addresses the issue of appropriate control architectures for automated manufacturing systems. An experimental testbed for the evaluation of alternative control architectures is described that integrates discrete-event simulation (implemented with Arena) and a modular object-oriented state/control development environment (implemented in C++). As well, manufacturing system performance results for three test control architectures are presented.

1 INTRODUCTION

The appropriate choice of a control architecture or, decision-making structure, is a central issue in manufacturing systems control research. Various types of distributed control systems have been proposed ranging from hierarchical control architectures to non-hierarchical or heterarchical control architectures. Although a considerable amount of research has been done in this area, very little work has been done on quantitative comparisons between different control architectures. In particular, there is a lack of research that compares the performance of alternative control architectures designed to control the same manufacturing system.

In this paper, we will present an experimental testbed that is being used to investigate the performance of alternative control architectures for a given manufacturing system. A generic structure will be presented that allows any type of manufacturing system, simulated using the Arena discrete-event simulation software, to be controlled by various decision-making structures developed in an object-oriented programming environment.

The next section will provide some background on the manufacturing control problem by first providing a brief review of the research that has been done in this area, then by discussing several unresolved issues that follow from the literature. In §3, the details of the experimental testbed will be discussed. In order to allow various types of manufacturing systems and control architectures to be investigated, a modular approach has been used.

An example of a series of experiments performed using the testbed is provided in §4. Three basic control architecture types are investigated in this section that are intended to be representative of the range of control architectures that have been investigated in the literature.

Finally, we conclude with a discussion of the current work that is being performed with the manufacturing control architecture experimental testbed that is intended to bridge the gap between the work on theoretical control architectures and the actual application of these control architectures to real manufacturing control problems.

2 BACKGROUND

2.1 Manufacturing Control Architectures

Recently, there has been a considerable amount of work done in the area of manufacturing systems control. This work has focused on getting away from centralized forms of control (*i.e.*, a single control computer) and has moved towards more decentralized forms of control (*i.e.*, involving multiple decision-makers which can be arranged in various architectural forms).

A primary issue that relates to the decentralized approach to control is the issue of appropriate control architectures for these systems: what decision-making structures result in a modifiable, reliable, and fault-tolerant system? Experience has shown that traditional, centralized architectures can be quite inflexible to change and provide little fault tolerance. This has led industrial and academic researchers to the development of a spectrum of decentralized control architectures.

At one end of the spectrum lie the hierarchical control architectures with theoretical foundations in organizational theory and large-scale system control theory (Mesarović *et al.* 1970; Singh 1980). At the other end lie non-hierarchical, or "heterarchical", structures arising from more recent developments in distributed data processing systems (DDPS) (Enslow 1978) as well as advances in artificial intelligence (AI), and object-oriented programming (O-OP). Falling between these two extremes are systems similar to those described by Jones and Saleh (1990) that incorporate the peer-to-peer relationships of non-hierarchical control and the task decomposition of hierarchical control.

The review of the literature in this area has shown that existing industrial and academic research on manufacturing system control has focused on qualitative comparisons of alternative structures and, although this approach has proven the concept of heterarchical control, it is agreed among researchers that quantitative comparisons of the various control architectures are required (Dilts *et al.* 1991; Duffie and Piper 1986; Duffie *et al.* 1988).

2.2 Unresolved Issues

A number of questions follow from the research that has been conducted in the area of manufacturing control architectures. Of primary importance is the fundamental question concerning the choice of control architecture: *i.e.*, is it possible to determine whether a specific control architecture is appropriate for solving a given manufacturing system control problem? In other words, can methods be generated to determine the "best" control architecture for a given problem?

In order to gain insights into this question, it is important that quantitative and objective comparisons between alternative control architectures are made. With this in mind, this paper is concerned with providing a description of the important first step that was taken to achieve the objective of providing an objective and quantitative comparison of alternative control architectures: *i.e.*, the development of an experimental testbed that can be used to investigate the relative performance of any variety of manufacturing control architectures with any type of manufacturing system.

In the sections that follow, a description of this experimental testbed will be given as well as an example of experiments with a single manufacturing system using this approach.

3 DEVELOPING AN EXPERIMENTAL TESTBED

A modular experimental testbed has been developed to conduct the experiments as is shown in Figure 1.

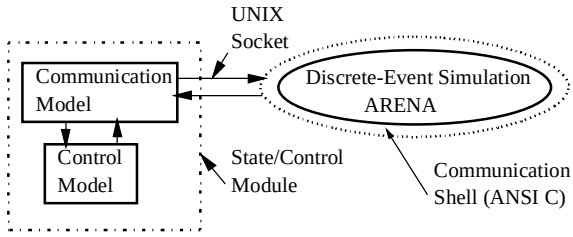

Figure 1: The Experimental Testbed

This testbed consists of two main modules:

i) a discrete-event simulation model and communication shell, intended to emulate the operation of the real manufacturing system, and
ii) a state/control module, used to implement alternative decision-making schemes.

The simulation model is written in the Arena programming language (Pegden *et al.* 1995), and can be modified relatively easily to represent alternative manufacturing system configurations. The Arena statistics collection facilities make it easy to collect and analyze a wide range of performance information on the emulated system while the animation capability is useful for visualization of system behavior.

The Arena software package allows application solution templates to be developed which can be thought of as software "containers" that encapsulate program logic that may be re-used in various simulations. For this application, the simulation logic that is associated with inter-process communications is implemented using an Arena application solution template. Custom message interface modules were developed that make it easier to: (i) communicate with the communication shell, and (ii) collect and display statistics on communications. The Arena application solution template was developed to encapsulate communications logic in order to increase the modularity of the experimental testbed. As a result, this logic is relatively easy to implement in a simulation and the logic is portable to other manufacturing system simulations in Arena.

The Arena model is augmented with additional ANSI C routines which allow it to communicate with the separate control module through input/output streams (implemented using UNIX or INET sockets). This allows any important state changes in the emulated system to be reported to the control module and reactive control decisions to be communicated back to the system.

The state/control module, which is used to implement the test control architectures, is implemented in the C++ programming language (Stroustrup 1993). The modular nature of this experimental testbed has resulted

in a system that has the ability to deal with any type of manufacturing system (*e.g.*, by changing the Arena model) and any type of control architecture (*e.g.*, by changing the control model). Figure 1 illustrates the basic structure of the testbed. In the following sections, the elements of the experimental testbed will be described in more detail and examples of the testbed operation will be given.

3.1 The Simulation Model

Since the majority of the control architecture work described in the literature is concerned with part scheduling in flexible manufacturing systems (FMS) (Kimemia and Gershwin 1983; Gershwin 1989) and cellular manufacturing systems (Jones and Saleh 1990; Davis *et al.* 1991; Duffie and Prabhu 1994), the system that is investigated for this research is concerned with the scheduling of parts through a simple manufacturing cell. The simulated manufacturing cell is similar in structure to the manufacturing cell that is investigated by Duffie *et al.* (1988).

The system that is investigated here consists of a number of automatic machines connected by a material handling robot as illustrated in Figure 2. Each of the machines in the system is capable of performing various operations depending upon its tool setup, which is limited by the number of tools that can be held at the machine's local tool storage area. As well, each machine in the investigated system is prone to failure. The parts that are introduced to the manufacturing cell require that all of their processing should be performed within the manufacturing cell. Since each machine is capable of performing a number of operations, there is a degree of routing flexibility that is available to the part. Additionally, since machines may fail, and their local tool setup may change, this routing flexibility is dependent upon the machine status and setup of the manufacturing cell.

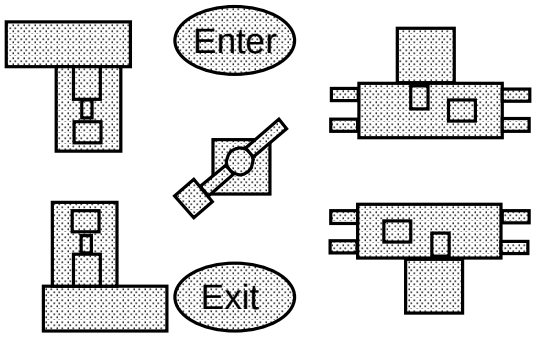

Figure 2: The Simulated Manufacturing System

In Figure 2, the two machines shown on the left represent automatic milling stations, and the two machines on the right represent automatic boring stations. A material handling robot is shown at the center of the figure.

Raw parts arrive at the input area of the manufacturing cell (*i.e.*, the area labeled "Enter" in Figure 2) and wait to be moved by the material handling robot to a machine for processing. For the simulation logic, parts are moved to a material handling robot queue to model this process. Part processing is then simulated for the cell as follows: (i) The part is moved to the input queue of a machine by the material handling robot where a first-in-first-out queuing discipline is followed. (ii) Once the part reaches the machine, the machine starts processing each of the part's operations that are scheduled for this visit. (iii) Once the machine has finished processing the operations, the part is moved to the material handling robot queue to wait to be moved to the next machine. (iv) If the part's process plan has been completed, the robot moves the part to the output area, labeled "Exit" in Figure 2; if more operations are required, the part is moved to another machine for processing (*i.e.*, we return to step i)).

Figure 2 provides an overview of the basic components of the simulated system, but does not give any details concerning the logic of the simulation. In order to understand the manufacturing cell's simulation logic, the logic can be can be thought of in terms of a number of software segments shown in Figure 3.

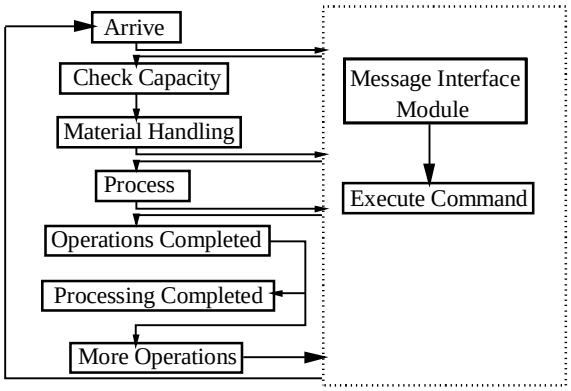

Figure 3: Manufacturing Cell Simulation Logic

New part arrivals are handled in the "arrive" segment. This segment contains the software that is used to model the part arrival process as well as the software used to initialize the attributes of the part.

Since the simulation program next requires information on the part's routing, the external control program is consulted at this point. Whenever simulation status information is reported to the control program or control commands are passed to the simulation program the communications application template is used. This

software module allows a message to be passed to the control software that indicates the simulation's state and a message to be received from the control software that contains a control command. These messages are passed between the two software modules shown in Figure 1 using a UNIX or INET socket as was described at the beginning of this section.

The "execute command" software segment of the simulation program is integral to the inter-process communication process since it represents the segment of the simulation software where the control command received from the control architecture is interpreted. Whenever the message interface module is executed in the simulation model, the next logic is the "execute command" segment as is illustrated in Figure 3. For our part processing example, this logic would indicate that the new part should be sent to the "check capacity" segment which is used to handle the situation that occurs when the number of parts in process exceeds the number of parts that the control program can control.

The next two steps involve the simulation of the material handling and part processing. At this point, the part attributes that indicate where the part's next processing will occur are checked, and the part is transported to the appropriate workstation for processing. Once the processing at the workstation has been completed, the part enters the "operations completed" segment. This segment is used to make the decision concerning where the part is next routed. To accomplish this, the control program is consulted as indicated in Figure 3. If all of the part's operations have been completed, the part exits the system in the "processing completed" segment where final statistics concerning the part are collected. Otherwise, the part continues to its next operation by first entering the "more operations" segment where a decision is made concerning where the part should have its next operations completed. Once again, the "more operations" segment relies on the communications template and the "execute command" segment for this routing decision before the part is sent to the "material handling" segment.

3.2 The State/Control Model

As was noted previously, the test control architectures are implemented in the state/control module in a manner that makes the control software transparent to the other elements of the experimental testbed described in Figure 1. To accomplish this, the state/control module provides the same interface and follows the same general procedures for each of the test control architectures.

This is accomplished in the state/control module with two separate models: the communication model and the control model. The communication model handles message parsing and the interface to the discrete-event simulation software; the control model is responsible for initializing the control software and for control program execution. Before describing these two models and the state/control module's interface to the other elements of the experimental testbed, the general procedure that is used by the state/control module will be described.

Each of the test control architectures follow the same general procedure to achieve control: (i) instantiate control system decision-makers, and (ii) make control decisions dynamically, as and when requested by the system being controlled.

The first step occurs when the control software is started, and involves preparing the decision-makers that comprise a given architecture for the process of controlling a specific manufacturing system. Once the control system is initialized, the control program starts the process of monitoring the controlled system and executing the control commands required to control the manufacturing system.

In order to instantiate the control system decision-makers, an expert system has been developed that utilizes a forward reasoning process (Brennan and Rogers 1997). The expert system relies on a set of rule bases that describe the capabilities of the workcenters in the manufacturing cell and the detailed processing requirements for the parts that are to be introduced to the system. When the appropriate rule bases are read by the state/control module, forward reasoning is used to generate information that can be used by the control program.

Once the decision-making instances are instantiated, the state/control module is ready to execute the control process. This is accomplished by each of the test control architectures through the same basic control functions. These common control functions, which are implemented in the control model, provide a consistent interface between the control software and the controlled system.

Although this basic structure is consistent between each of the test control architectures, the details of how these control functions are performed is dependent upon the choice of control architecture. The differences between each test control architecture can be seen in the differences between the control models.

As is shown in Figure 1, the communication model interfaces with the control model, where the methods for the specific test control architecture reside. For example, when a control function is requested by the discrete-event simulation software, the control model handles the implementation of the control function. Similarly, when the control model is ready to issue a control command, it uses the communication model to pass its control command to the discrete-event simulation software.

Within the state/control module, the communication model acts as an interface between the control model and the controlled system (*i.e.*, the emulated manufacturing system). Through this interface, requests for decisions by

the control architecture can be made, and the control architecture can obtain status updates.

3.3 Inter-Process Communications

In the previous sections, some of the details of the inter-process communications that are used in the experimental testbed have been introduced. This section will be concerned with all of the elements of the experimental testbed shown in Figure 1 that allow messages to be passed between the state/control module and the simulation module.

Inter-process communications are handled in the experimental testbed in three locations:

i) the state/control module's communication model,
ii) the simulation module's communication shell, and
iii) the discrete-event simulation's message interface module.

All of these elements of the experimental testbed work together to allow control decisions to be implemented in the emulated manufacturing system. In order to act as an interface between the control model and the controlled system, the state/control module's communication model must setup the state/control module's UNIX socket (*i.e.*, communications on a single computer) or INET socket (*i.e.*, communications between remote computers over the internet).

During the control program initialization process described in §3.2, the communication socket is constructed. In terms of the control program's world view, this is equivalent to setting up a file to which messages can be written and from which messages can be read. When the control program begins executing, a five step process is continuously performed by the communication model:

i) wait for a connection with the simulation module,
ii) read the simulation module's message from the socket's input stream,
iii) parse the message into a format that can be used by the control program,
iv) parse the control model's response into a control message, and
v) write the control message to the output stream.

The communication shell serves the same purpose for the discrete-event simulation as does the communication model for the control model. In order to allow messages to be passed to and from the simulation module, the communication shell accomplishes the following tasks: message parsing, communication socket management, and simulation event management.

When the Arena simulation software is run, the communication shell immediately constructs a socket with the same network addresses as the socket constructed by the communication model. Just as in the case of the communication model, the communication shell performs a five step process while the discrete-event simulation is executing:

i) connect with the state/control module when required by the emulated manufacturing cell,
ii) parse the simulation state information into a state message,
iii) write the state message to the socket's output stream,
iv) read the control command from the socket's input stream, and
v) parse the control command into a format that can be used by the discrete-event simulation.

This process is continued until the simulation run ends. At this point a final state message is sent to the state/control module announcing the end of the simulation run.

The last software component that allows inter-process communications to occur has already been introduced in §3.1: the Arena custom application template for communications. The primary purpose of this portion of the discrete-event simulation software is to allow communications between the simulated system and the control program to be initiated by conditions that occur in the simulated system. As was discussed in §3.1, whenever a control decision is required, the message interface module is activated to allow state information to be communicated to the control software.

4 EXPERIMENTS WITH A SIMULATED TEST MANUFACTURING CELL

The results reported in this section are intended to provide an indication of how the experimental testbed can be used to evaluate the performance of alternative control architectures. Three different test control architectures are evaluated that are intended to be representative of the full spectrum of control architectures discussed in §2 from hierarchical to non-hierarchical control:

i) Constrained hierarchical (CH),
ii) Unconstrained hierarchical (UH) ,
iii) Non-hierarchical (NH).

As the name implies, the constrained hierarchical (CH) control architecture is intended to represent control architectures firmly at the hierarchical end of the control architecture spectrum. The reason for using the word

"constrained" to describe this test control architecture is related to both the communication paths and the type of coordination that is used by this control architecture. Only communications between infimal and supremal decision-makers are possible in this control architecture. As a result, if a failure occurs with one of the decision-makers, this could result in other decision-makers being isolated from the rest of the control systems.

The design of the unconstrained hierarchical (UH) control architecture is intended to remove some of the constraints placed on the CH control architecture. In particular, two main constraints are removed: (a) fixed communication paths, and (b) rigid machine agent coordination.

The non-hierarchical (NH) control architecture is designed in a similar manner to the non-hierarchical, or heterarchical, control architectures discussed in §2. The main characteristic of this type of architecture is that problem solving is distributed among a number of cooperatively autonomous agents. For the test control system that are used in the experiments, distributed problem solving will be in the form of part-oriented scheduling which is the method that has been used by a number of researchers who have investigated non-hierarchical control (Lin and Solberg 1992; Duffie and Prabhu 1994).

4.1 A Description of the Experiments

As was discussed in §3, the experimental testbed is designed to allow various changes to be made to the emulated manufacturing system and the control software in order to allow each of the test control architectures to be evaluated under various operating conditions. In this section the different operating scenarios that are used for the simulation experiments will be presented. Each of the control architectures is tested under the same set of operating conditions in order to provide an objective comparison of the resulting manufacturing system performance as well as the relative control architecture performance. Figure 4 provides a detailed summary of the basic operating conditions that were used to evaluate each of the test control architectures.

The manufacturing system under these circumstances operates in a near-deterministic fashion that, although unrealistic, should provide a basis for comparison with the other test scenarios. This base test scenario is intended to be used to provide a basis of comparison between a deterministic, or near-deterministic test scenario and two test scenarios that introduce uncertainty to the manufacturing environment. Since one of the primary design goals for manufacturing control systems is achieving a system that is capable of coping with changes that occur on the shopfloor, introducing uncertainty to this environment should allow the relative effectiveness of each control architecture at achieving this goal to be determined.

Parameter	**Description**
Service Time	Deterministic: based on the proc. time estimate
Time between arrivals	4.0 minutes
No. of operations	8
Tool availability	Both milling stations and both boring stations have the same tool setup
Machine failures	none
Material handling time	Exponential (mean = 0.5 minutes)

Figure 4: Base Scenario

Three different test scenarios are being investigated that allow a determinstic base case to be compared with two stochastic scenarios: (i) Scenario #1: the base set of experiments, (ii) Scenario #2: stochastic (*i.e.*, this scenario introduces exponential demand and unreliable workstations), (iii) Scenario #3: stochastic with tooling constraints (*i.e.*, this scenario builds on scenario #2).

4.2 Manufacturing System Performance

The relative performance of each of the three test control architectures for the first three test scenarios is shown in Figure 5(a-b). This figure shows the test control architectures' flow time and tardiness performance respectively.

Based on point estimates from the simulation runs, the average WIP for scenario #1 is 7.5 for the CH control architecture, 3.7 for the UH control architecture, and 5.6 for the NH control architecture.

For the three test scenarios, tardiness is evaluated by assigning an arbitrary flow time allowance to the parts. A time of 20 minutes has been assigned for these experiments in order to create a situation where all of the control architectures would have some tardy jobs. By using mean tardiness as a measure we are assuming that early jobs bring no rewards and that there are penalties for late jobs (French 1982).

As can be seen in Figure 5, the UH control architecture appears to provide the best flow time and tardiness performance of all three test architectures for each of the test scenarios. The CH control architecture results in the highest flow times and tardiness values.

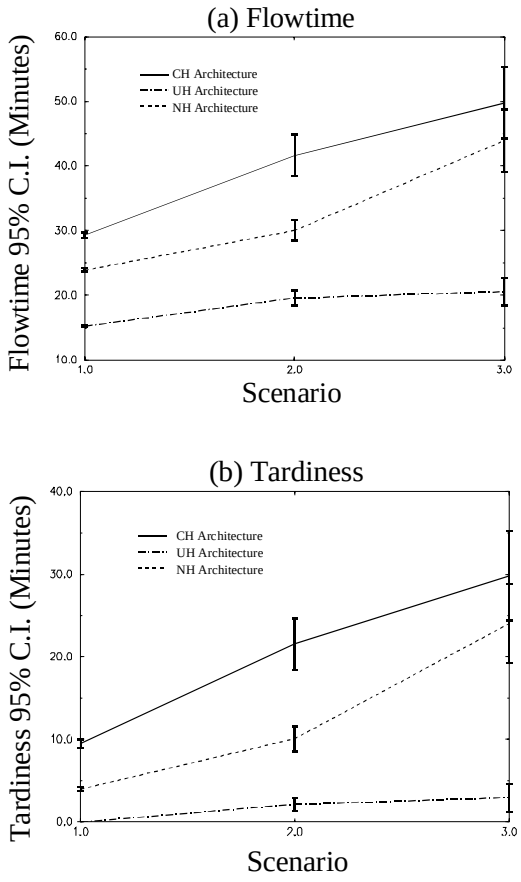

Figure 5: Test Control Architecture Performance

When we consider machine failures and an exponentially distributed time between arrivals in the second test scenario, the CH and NH control architectures' flow times are 112% and 53% higher respectively than the flow time for UH control architecture.

The NH control architecture's flow time performance does not appear to worsen with respect to the UH control architecture, but the CH control architecture shows a relative increase of 19% in its flowtime relative to the UH control architecture. This result seems to indicate that the two distributed control architectures (*i.e.*, the UH and NH control architectures) are more capable of handling disruptions such as machine failures and variation in the part arrivals than the CH control architecture.

When tooling constraints are placed on the machines (*i.e.*, scenario #3) the disparity between the CH and NH control architecture flow times and the UH control architecture flow time increases even more.

5 CONCLUSIONS

An interesting result follows from the experimental analysis of the three test control architectures described in the previous section: it appears that control architectures that contain properties of both hierarchical and non-hierarchical control architectures show better flow time and tardiness performance than architectures that sit on either extreme of the control architecture spectrum. In order to determine why this is the case for the scenarios described in §4, it is important to ask the following questions: (i) What factors characterize a control architecture?, and (ii) How do various control architectures perform for various constraints?

Work is currently being conducted by the authors that is focused on identifying key parameters of the manufacturing system control problem that can be used to characterize alternative control architectures. Larger control systems than the test systems studied here will most likely be hybrids of these smaller architectures, composed of clusters of hierarchies and heterarchies that are dynamically reconfigurable (*i.e.*, both the control architecture's organizational structure and coordination modes should be reconfigurable). If these types of systems are to be realized, it is important that that the underlying properties of alternative control architectures are first identified in order to allow decisions to be made concerning the most appropriate type of control architecture for a given manufacturing control problem. The experimental testbed reported in this paper will play a central role in this type of investigation by allowing insights into the second question posed above to be obtained.

ACKNOWLEDGMENTS

The authors wish to thank the Natural Sciences and Engineering Research Council of Canada for their generous support of this research under grants OGP-012-1522 and OGP-019-7339.

REFERENCES

Brennan, R.W. and P. Rogers. 1997. Applying a knowledge-based system to a control architecture experimental testbed. *In IASTED International Conference on Artificial Intelligence and Soft Computing Proceedings*.

Davis, W., A. Jones, and A. Saleh. 1991. A generic architecture for intelligent control systems. *National Institute of Standards and Technology Report NISTIR 4521*.

Dilts, D.M., N.P. Boyd, and H.H. Whorms. 1991. The evolution of control architectures for automated

manufacturing systems. *Journal of Manufacturing Systems*, 10:79-93.

Duffie, N.A. and R.S. Piper. 1986. Non-hierarchical control of a flexible manufacturing cell. *Proceedings of the International Conference on Intelligent Manufacturing Systems*, 51-54.

Duffie, N.A., R. Chitturi, and Jong-I Mou. 1988. Fault-tolerant heterarchical control of heterogenous manufacturing system entities. *Journal of Manufacturing Systems*, 7:315-328.

Duffie, N.A. and V.V. Prabhu. 1994. Real-time distributed scheduling of heterarchical manufacturing systems. *Journal of Manufacturing Systems*, 13:94-107.

Enslow, P.H. Jr. 1978. What is a 'distributed' data processing system? *Computer*, 13-21.

French, S. 1982. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood Limited.

Gershwin, S.B. 1989. Hierarchical flow control: a framework for scheduling and planning discrete events in manufacturing systems. *Proceedings of the IEEE*, 77:195-209.

Jones, A. and A. Saleh. 1990. A multi-level/multi-layer architecture for intelligent shopfloor control. *International Journal of Computer Integrated Manufacturing*, 3:60-70.

Kimemia, J. and S.B. Gershwin. 1983. An algorithm for the computer control of a flexible manufacturing system. *IIE Transactions*, 15:353-362.

Lin, G.Y. and J.J. Solberg. 1992. Integrated shop floor control using autonomous agents. *IIE Transactions*, 24:57-71.

Mesarović, M.D., D. Macko, and Y. Takahara. 1970. *Theory of Hierarchical, Multilevel, Systems*. Academic Press.

Pegden,C.D., R.E. Shannon, and R.P. Sadowski. 1995. *Introduction to Simulation Using SIMAN*. McGraw Hill.

Singh, M.G. 1980. *Dynamic Hierarchical Control, Revised Edition*. North-Holland.

Stroustrup, B. 1993. *The C++ Programming Language, Second Edition*. Addison-Wesley.

AUTHOR BIOGRAPHIES

PAUL ROGERS is an Associate Professor and Head of the Division of Manufacturing Engineering at the University of Calgary. His research interests include discrete-event simulation, production planning and control systems, object-oriented modeling for intelligent manufacturing, and models for the analysis of manufacturing systems. He is a Professional Engineer and a member of IIE, INFORMS, SME, and SCS and serves on the Editorial Board of the *International Journal of Computer Integrated Manufacturing*. He holds Ph.D. and M.Eng. degrees from Cambridge University in England.

ROBERT W. BRENNAN is an Assistant Professor in the Division of Manufacturing Engineering at the University of Calgary. His research interests include control architectures for manufacturing systems, optimization of discrete-event simulation, and models for the analysis of manufacturing systems. He has over seven years of industrial experience in project management and control systems and is a Professional Engineer and a member of IIE, INFORMS, and IASTED. He holds Ph.D. and B.Sc. degrees from the University of Calgary.