DESIGN AND IMPLEMENTATION OF FMS CONTROL SOFTWARE

Jeffrey S. Smith

Texas A&M University

ABSTRACT: This paper describes a methodology for the design and implementation of fully functional control software for flexible manufacturing systems (FMS). In a typical FMS implementation, the control software determines the ultimate flexibility of the system. Traditionally, control software has been custom-written for each FMS implementation and, hence, has been extremely costly. More significantly, the software often requires significant modifications in order to respond to changing production requirements for the FMS. In response to these problems, this paper describes a methodology and the associated software tools for the development of fully functional, distributed FMS control software. The objective of the methodology is to simplify and shorten the development cycle for control software. The methodology described in this paper partitions the control problem into a decision making component and an execution component. A well-defined interface between the two components allows the independent development of each component. In addition, the decision making module can be modified whenever necessary as dictated by the production requirements. An existing resource description model is adapted for use in the target environment and existing execution software generation techniques are used to automatically generate a significant portion of the execution software. Specific laboratory experience is described. In the example implementation, the decision making functions are performed by a discrete event simulation that communicates directly with a shop floor execution system.

INTRODUCTION

This paper describes a methodology and the associated software tools for the development of fully functional control software for flexible manufacturing systems (FMS). The control systems described in this paper are designed to work with mostly automated systems, but also facilitate the use of human operators. Figure 1 shows the example FMS that will be used to describe the control software development methodology. This system is located within the Texas A&M Computer Aided Manufacturing (TAMCAM) lab. The system consists of three CNC milling machines, one CNC turning center, two industrial robots, and an automated cart-based conveyor system. In addition, human operators are used to load and unload some machines and perform inspection tasks.

The shop floor control system in this environment is responsible converting the production requirements into specific instructions for the individual pieces of equipment and interacting with the equipment to implement the instructions [Wysk and Smith, 1995]. The production requirements are provided by the higher level planning system (e.g., the MRP system) and are in the form of part process plans. Production orders can be input to the shop floor control system at any time by

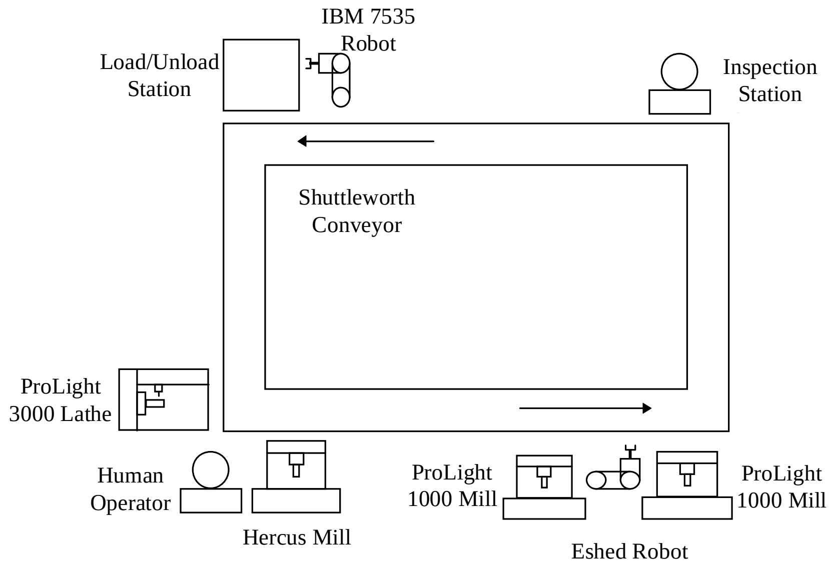


Figure 1. Texas A&M Computer Aided Manufacturing lab.

specifying a part type (identifying the process plan), quantity, and due date for the order. Equipment instructions are in the form of numerical (NC) instructions or the equivalent instructions that can be interpreted by the vendor-supplied machine controllers.. The objective of the shop floor control system is to identify a "good" sequence of equipment instructions in terms of the system performance. Evaluation criteria include typical performance measures such as batch makespan, cycletime, and due date-based measures. The specific decision making logic is not the primary concern of this paper. Instead, this paper describes the implementation of a control system that facilitates a variety of decision making methods in flexible manufacturing environments.

A key concept for development of flexible control systems is the explicit separation of *decision making* functions from *execution* functions in the control system [Jones and Saleh, 1990; Smith *et al.*, 1994]. Under this paradigm, the execution functions provide the device-level control of the equipment, whereas the decision making functions schedule the tasks of the execution functions such that the production requirements are met. As a result, the execution functions are dependent only on the physical configuration of the equipment and not on the specific parts and production volumes currently required. Similarly, the decision making functions are not affected by the details required to exercise low-level control over the physical equipment. The goal is to provide a well defined interface between the decision making and execution systems so that they can be developed independently and "plugged" together to meet current needs.

FMS CONTROL

Simulation-based Control

Smith *et al.*(1994) describe an FMS control system that uses discrete event simulation as the decision making function. In this control system, the explicit separation is provided by a message queue mechanism between the simulation and the execution system. The simulation-based decision maker sends task request messages to the execution system through the message queue. Once the execution system has completed the task, it returns a completion message to the decision maker through the message queue. Under this paradigm, neither system knows how the other works (or even whether it is manual or automatic). That is, the decision maker doesn't know or care how execution tasks are performed, and the execution system doesn't know or care how decisions are made as to task sequencing. As a result, either system can be modified or even replaced without affecting the other.

The simulation-based control system was implemented using a modified version of the SIMAN simulation language. The modification allows delay-type blocks to function either in *simulation mode* or in *real-time control mode*. In simulation mode, entities coming into the delay block are held for the specified delay time. In real-time control mode, the task message associated with the delay block is sent to the execution function and the entity waits at the delay block until a completion message is received from the execution function. The ability to run in either simulation model or real-time mode allows the same simulation to be used as a predictive analysis tool and a real-time controller (Smith *et al.*, 1994). The SIMAN modifications for implementing real-time control have now been included in the commercial versions of the Arena simulation package.

Drake (1996) has expanded the view of simulation-based planning, scheduling, and control of flexible manufacturing systems to simplify the development and use of the simulation-based decision maker. He points out that there has been considerable interest in "intelligent" shop floor control systems which use simulation technology to predict the future impact of operating decisions on manufacturing performance. While many of these efforts have defined the requirements, responsibilities, and pertinent issues of on-line simulation for CIM and FMS, they do not develop a detailed design of the simulation-based shop floor control system. A central theme to Drake's simulation-based control system is the partitioning of the simulation into separate *frames* for the physical characteristics and the decision-making functions. Consequently, it is easier to develop a "generic" simulation model that can be exercised without recompiling the model [Drake, 1996].

Factory Resource Model

The explicit separation of decision making from execution is only possible if both components have a consistent description of the underlying system and a formal, well-defined interface between the modules. Wysk *et al.* (1995) describe a factory *resource model* for describing the interactions between the system components. The resource model consists of two primary components: a resource classification, and a graph-based model of the resource interactions. The resource classification is based on the shop floor controller class described by Smith and Joshi (1995). The controller class identifies common equipment behavior and control requirements and describes a

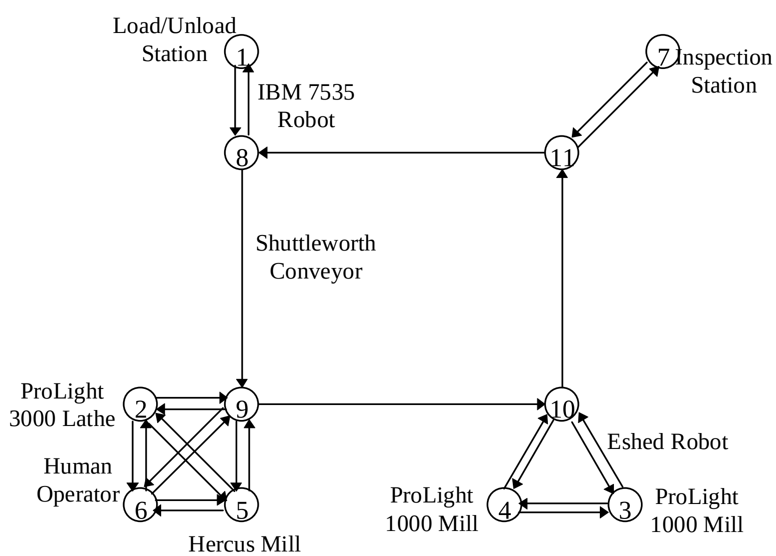


Figure 2. Resource model connectivity graph for the TAMCAM lab

Table 1. Equipment and physical configuration of the TAMCAM lab.

Name	Type (ID)	Ports	Locatations
Eshed Robot	MH (MH1)	-	-
IBM 7535	MH (MH2)	-	-
Load/Unload	MP (MP1)	1	1,2,3,4,5
ProLight Lathe	MP (MP2)	2	6
ProLight Mill 1	MP (MP3)	3	7
ProLight Mill 2	MP (MP4)	4	8
Hercus Mill	MP (MP5)	5	9
Human 1	MH, MP, MT (H1)	6	10
Inspector	MH, MP, MT (H2)	7	11
Shuttleworth	MT (MT1)	8, 9, 10, 11	12, 13, 14, 15

corresponding C++ class hierarchy for use in controller development. Although the equipment classification provides the information necessary to operate individual pieces of equipment, the classification does not describe how the individual pieces of equipment interact with one another within the control system. For example, although the ability of an industrial robot to move components between different points in space is captured in the Material Handling class, the use of the robot to load and unload a specific NC machine in the facility is not described in either class. These types of interactions are described by using a graph-based connectivity model (Wysk *et al.*, 1995). The resource model graph for the TAMCAM facility is illustrated in Figure 2.

The nodes in this graph represent *ports*, and the arcs represent the ability to move parts between locations represented by ports. Ports are basically locations in which parts can be stored and/or processed. In the example system, each of the NC machines is represented by a port, each loading station on the cart-based conveyor system is represented by a port, and the human operators are represented by ports. Table 1 shows each piece of equipment along with its type (according to the equipment classification), and the ports that it owns. Ownership of a port is an important concept associated with equipment interactions. Each port has a single *owner* and can have multiple *clients*. The owner of a port is responsible for granting access to the port to the clients of that port. Since the resource model instance will be used to generate the execution system, access control is necessary for collision avoidance. For example, the Eshed robot in the example system is used to load and unload parts from the two ProLight milling machines. Therefore, the two milling machines and the cart station on the conveyor are represented by connected ports (nodes 3, 4, and 10, respectively) in the connectivity graph.

Arcs in the connectivity graph also describe the task interactions between equipment level components in the control system. Analogous to port ownership, each arc has an associated *facilitator*, that moves the part from the location associated with the tail node to the location associated with the head node. As such, the facilitator must be a client of both the tail and head node ports. Consider a part that needs to be processed on the Pro Light mill associated with port 4 in Figure 2. Assuming that the part is currently at the load unload station, the arcs on a path through the connectivity graph from port 1 to port 4 describe the tasks required to move the part. Specifically, the IBM 7535 robot moves the part from the Load/Unload station (port 1) to the cart station on the Shuttleworth conveyor (port 2), the Shuttleworth conveyor moves the part to port 10 (through port 9), the Eshed robot moves the part from port 10 to the ProLight mill (port 4) where the part is processed. The return path to the Load/Unload station similarly defines the tasks associated with transporting the part back to the Load/Unload station upon completion of the processing.

DECISION MAKER

The decision maker is responsible for routing parts through the system and setting priorities between individual parts at the processing stations. In the example system, these decisions translate to prioritizing the loading of parts from the load/unload station onto the Shuttleworth conveyor system and setting the queue priorities for each of the ProLights, the Hercus, and the inspector. The Shuttleworth conveyor itself is responsible for tracking empty pallets, avoiding congestion, and determining the specific routes that parts take from a source port to a destination port. The structure of the material transport controller is described by Edlabdakar (1995). As such, the decision maker is does not make decisions regarding specific part routing on the conveyor. Instead, it simply instructs the Shuttleworth controller to move parts from a source port to a destination port, and the conveyor controller decides on the path and sets priorities between carts already on the system.

An alternative design would be to have the decision maker explicitly track all pallet movement between ports on the conveyor system. However, this would also mandate that the decision maker direct the flow of empty pallets. In the example system, this presents an unnecessary burden on the decision maker. However, in a more complex system, this would be a natural application for hierarchical simulation. Using hierarchical simulation, the equipment level material transport controller is also based on simulation, and can be used to predict specific transport times.

The Arena simulation is constructed directly from the resource model instance. Each material processor (MP) in the system is represented by a station submodel and two resources. The first resource represents the machine itself, and the second resource represents a WIP control and is used to determine when to release parts to the machine. The capacity of the machine resource is simply the capacity of the machine itself (usually 1). The capacity of the WIP control resource determines how many parts heading for the associated machine are allowed in the material transport system. Therefore, if a part is on a machine and the WIP control resource for the subsequent machine is not available, the part will not be allowed to request transportation, and the current machine will be blocked. Without the WIP control resource, either a single part or all parts destined for a specific machine could be in the transport system. Depending on the ratio of processing time to transport time, both of these options could be unduly restrictive. The optimal capacities of the WIP control resources are system-specific and must be determined through experimentation.

The material transporter (MT) is represented in the model by a resource. The capacity of the resource determines how many parts are allowed to be in transit concurrently. For the example system, this capacity is equal to the number of transport pallets available. Modeling the material transporter as a resource rather than as a network is an artifact of the autonomy given the material transport controller. In cases where the decision maker must make all decisions about pallet movement, the network model can be used to represent the material transport system. However, as described above, in real-time control mode, the simulation must explicitly instruct the material transport controller as to how to handle idle pallets. In the traditional simulation environments, the time delays associated with blocking are modeled, but the specific events leading to unblocking are not typically visible to the developer. As a result, it is difficult to convert these events into specific instructions for the execution system.

Each part routing is defined by an Arena sequence. Each station in the sequence corresponds to a machine in the processing sequence of the part. The processing times are not fixed, but, instead, are input as part of the production orders. The production orders are read from an external data file. This file includes the number of parts of each type in the order as well as the processing times at each step in the processing sequence. The production orders data file is read whenever there are new production orders. Specific execution tasks are determined by solving the shortest path in the resource model instance graph. That is, the source and destination nodes are identified by the IS attribute in the simulation and the arcs along the shortest path determine the required execution tasks.

EXECUTION SYSTEM

The execution system is also generated from the resource model instance. The execution system is based on the three-level hierarchical control architecture described by Smith *et al.* (1994). The example implementation uses only the bottom two levels in the hierarchy. Each piece of equipment (MP, MH, and MT) has an associated equipment level controller. These controllers provide the interface between the shop floor control system and the device-specific controller provided by the equipment vendor. The second level, or workstation controller, is responsible for coordinating the activities between the equipment level controllers. For example, the decision maker in the example system might request that the Eshed robot move a part from the Shuttleworth conveyor port 10 to the ProLight Mill (represented by the arc between nodes 10 and 4 in the graph shown in Figure 2). The workstation level controller would receive the request and synchronize the actions of the Shuttleworth, Eshed, and ProLight required to implement this task.

The equipment and workstation level controllers are based on the message-based part state graph (MPSG) model described by Smith and Joshi (1993). The MPSG model provides a formal language with which to describe the processing protocol between distributed controllers in a shop floor control system. The processing protocol is essentially the sequence of messages that are passed between controllers and the tasks that individual pieces of equipment perform to implement higher level tasks. A significant advantage of using the MPSG model is that a significant portion of the execution module for a controller can be automatically generated from a textual representation of the MPSG.

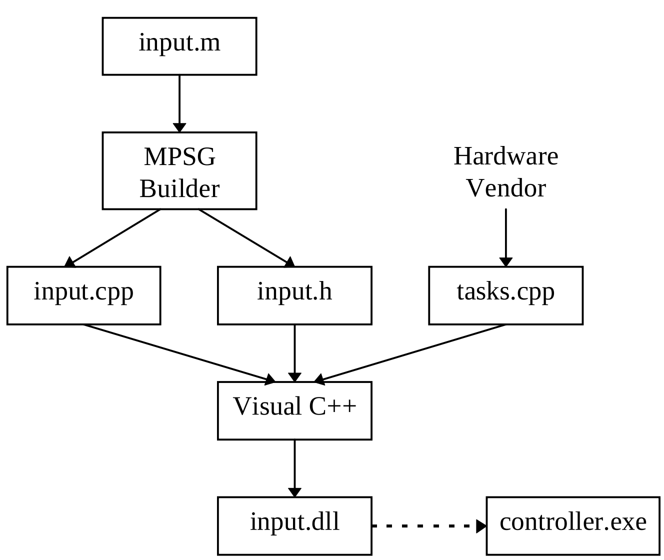


Figure 3. Execution module development.

Figure 3 shows a flowchart describing the creating of MPSG-based controllers. The input file *input.m* is the textual representation of the MPSG. This file is generated based on the shop floor controller class described by Smith and Joshi (1995) and the machine type. The input file is processed by the MPSG Builder which crates the controller-specific C++ code to implement the processing

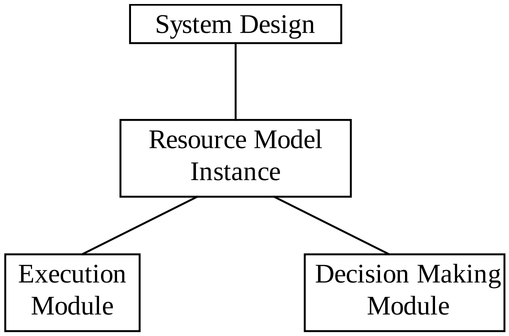


Figure 4. System design methodology

protocol (input.cpp and input.h). The task action file (tasks.cpp) is the machine-specific code to implement the machine actions (e.g., downloading and executing an NC file, moving a robot arm, opening and closing a robot's gripper, etc.). The majority of this code must be supplied by the equipment vendor or developed based on vendor-supplied specifications. These source files are compiled to create a dynamic link library (input.dll) with the processing protocol interpreter. The dynamic link library is used with the generic controller (controller.exe) to form an operational controller. The code required for data communications between controllers is part of the base controller class and is implemented in the generic controller. This class includes code for serial communication with the machine and TCP/IP-based communication with the other controllers in the control system.

DEVELOPMENT METHODOLOGY

Figure 4 presents a flowchart describing the control system development methodology. The system design stage involves selecting the equipment to be used in the system and specifying the system layout. There is a tremendous body of existing research on the FMS system design phase, and it will not be discussed further in this paper. Once the design has been finalized, a resource model instance is created to provide a formal description of the system. The resource model instance is then used as the primary input to the decision making and execution module development, as described above. In the example system, the decision making and execution modules communicate through a TCP/IP socket-based connection. This type of connection allows the components to run on different systems (potentially in different locations).

Figure 5 illustrates the control system components developed for the TAMCAM lab. In this implementation, each equipment level controller runs on a separate personal computer running Microsoft Windows for Workgroups. The equipment level controllers are connected to the vendor-supplied device controllers through a serial interface. The workstation controller and the Arena

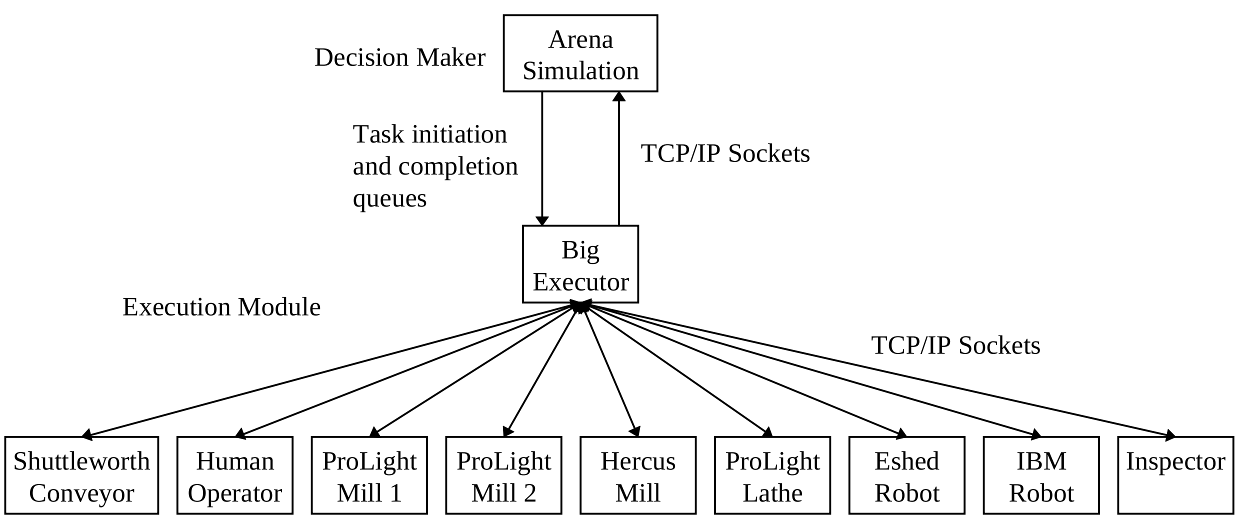
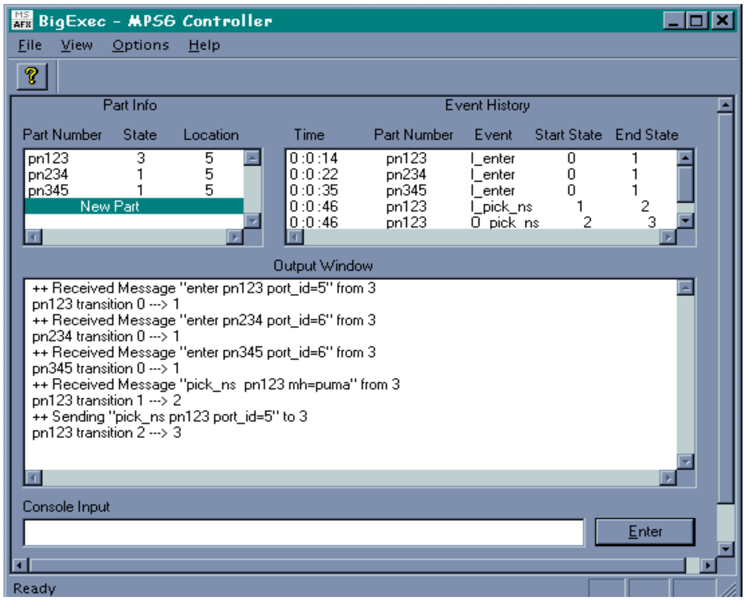


Figure 5. Control system implementation



Execution interface.

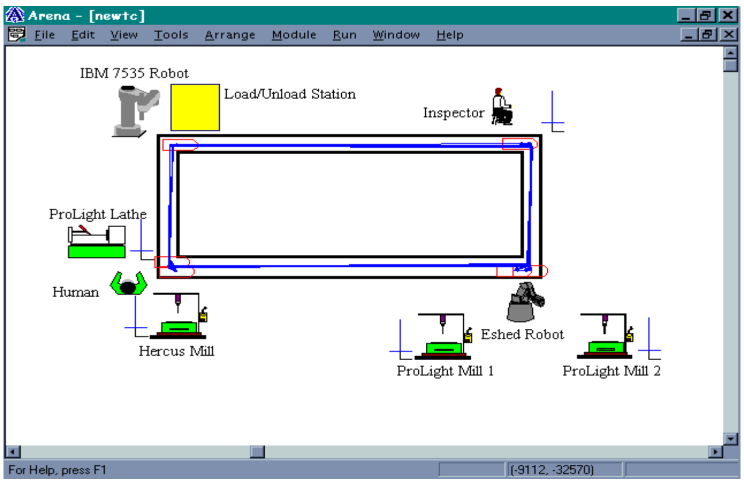


Figure 6. Decision making and execution modules.

simulation run on a personal computer running Microsoft Windows NT. All of the communications between components use TCP/IP sockets and go through a software router running on the same computer as the workstation controller.

Figure 6 shows the interfaces to the decision making simulation and execution modules in the example system. All of the code for the example implementation has been developed using standard software tools (Microsoft Visual C++ and the Microsoft Foundation Classes). The control system is described in more detail and the software is available through the World Wide Web home page at http://www-tamcam.tamu.edu.

CONCLUSIONS

This paper has presented a methodology for the design and implementation of fully functional control software for flexible manufacturing systems. The shop floor control functions are explicitly separated into decision making and execution modules, which allows the modules to be developed and modified independently. The methodology relies on the resource model instance, which is a formal description of the FMS, to serve as the primary input for the development of the decision making and execution modules for the control system. Limited experience has shown that the use of the described methodology and tools significantly decreases the time required to implement fully functional control software.

ACKNOWLEDGMENTS

This work was supported by ARPA Project 8881, “Rapid Prototyping of Shop-floor Control Systems for CIM.”

REFERENCES

Drake, G. R., “Concepts for On-line Simulation Systems,” Master’s Thesis, Department of Industrial Engineering, Texas A&M University, College Station, TX, 1996.

Edlabadakar, A. "Framework for a Flexible, Real-time Controller for Automated Material Transport Systems," M.S. Thesis, Industrial Engineering Dept., Texas A&M University, 1995.

Jones, A. T. and Saleh, A., A Multi-level/Multi-layer Architecture for Intelligent Shop Floor Control, *International Journal of Computer Integrated Manufacturing*, Vol. 3, No. 1, 1990, pp. 60-70.

Smith, J.S. and Joshi, S.B., "A Shop Floor Controller Class for Computer Integrated Manufacturing," *International Journal of Computer Integrated Manufacturing*, Vol. 8, No. 5, September-October 1995 .

Smith, J. S., Wysk, R. A., Sturrock, D., Ramaswamy, S., Smith, G., and Joshi S. B., "Discrete Event Simulation for Shop Floor Control," *Proceedings of the 1994 Winter Simulation Conference*, December, 1994a, Lake Buena Vista, FL, pp. 962-969.

Smith, J.S., Hoberecht, W.C., and Joshi, S.B., "A Shop Floor Control Architecture for Computer Integrated Manufacturing," *Journal of Design and Manufacturing*, (accepted) April 1994b.

Smith, J. S. and Joshi, S. B., "Message-Based Part State Graphs (MPSG) for Shop Floor Control," *Proceedings of the 2nd IIE Research Conference*, Los Angels, CA, May 26-28, 1993.

Wysk, R. A., Peters, B. A., and Smith, J. S., "A Formal Process Planning Schema for Shop Floor Control," Engineering Design and Automation Journal, Vol. 1. No. 1, Spring 1995, pp.3-19.

Wysk, R. A. and Smith, J. S., "A Formal Functional Characterization of Shop Floor Control," *Computers in Industrial Engineering*, Vol. 28, No. 3, 1995, pp. 631-644.