

Virtual Factories: An Object-Oriented Simulation-Based Framework for Real-Time FMS Control

Douglas A. Bodner and Spiridon A. Reveliotis

School of Industrial and Systems Engineering

Georgia Institute of Technology

Atlanta, GA 30332-0205 USA

[bodner, spyros]@isye.gatech.edu

Abstract – Increasingly, automation and computer-based control play key roles in the operation of manufacturing systems. These two factors are necessary to support trends towards greater product customization and increased emphasis on reduction of lead times. At the same time, though, automation and computer-based control can be enormously expensive to design and implement in a factory. These high costs motivate the need for modeling and analysis. Existing modeling and analysis tools are effective for high level modeling of manufacturing systems. However, they suffer from limitations which are important in detailed analysis of system performance. This research is motivated by the need to develop modeling tools which support rapid development of detailed simulation models to assess system performance. The fundamental idea is to develop such an environment (i.e., a “virtual factory”) which one may use to test different system configurations and control policies. We demonstrate our work thus far by presenting a case study involving deadlock avoidance in a semiconductor cluster tool.

I. INTRODUCTION

Flexible automation and computer-based control play key roles in supporting greater product customization and reduction of manufacturing lead times. At the same time, though, they can be enormously expensive to design and implement in a factory. For example, one fundamental problem is that the automation and control for each system must often be designed from scratch. Another problem is that the equipment needed to support manufacture of sophisticated products is typically expensive.

The high cost of flexible manufacturing system (FMS) development motivates the need for modeling and analysis. It is critical that one be able to determine accurate estimates for system performance of a given system design or reconfiguration before implementation. At the same time, it is important that system designs and reconfigurations be relatively straight-forward to implement. Existing modeling and analysis tools are effective for high level modeling of manufacturing systems (e.g., aggregate part flows). However, they suffer from limitations which are important in detailed analysis necessary for accurate system performance estimates and for capturing details necessary for implementation. For example, queueing networks typically omit important structural features of the system, such as blocking. Even with more detailed simulation models developed using commercial

simulation languages, it is difficult to represent the control system accurately [6]. Often, one must resort to coding in the underlying language from which the simulation language is compiled.

This is not a trivial task and makes simulation difficult, time-consuming and hard to validate. In fact, simulation technology is not used to its full potential because of these types of difficulties. This research is motivated by the need to develop modeling tools and methodologies which support rapid development of simulation models to assess system performance, especially investigating the role of real-time control. Our eventual goal is to provide an environment for use as a high fidelity testbed for manufacturing system design and control, in other words, to create a “virtual factory.”

In this paper, we present our research in this area to date. Our focus has been on (i) developing formal models of system structure and behavior to support control policy development and performance assessment, (ii) specifying the problem of real-time control of flexibly automated manufacturing systems, and (iii) developing a software environment which may be used to assess performance of control policies. We also present a case study involving deadlock avoidance in a cluster tool used for semiconductor manufacturing. This work is being conducted in the facilities of Virtual Factory Lab at the Georgia Institute of Technology.

II. FMS DOMAIN ANALYSIS

A variety of modeling approaches can be used to study manufacturing systems. To study the low-level phenomena needed for detailed analysis, though, simulation remains the most flexible and powerful [5]. Traditional simulation languages are very useful in developing relatively simple models. However, they are limited in their ability to represent complex behavior and manufacturing control [7], [11]. This has motivated a significant body of research in the development of object-oriented approaches to simulation [7]. The object-oriented paradigm facilitates modeling of complex behavior through natural mapping abstractions and modular code development. For our simulation environment, we have chosen such an object-oriented modeling architecture, OOSIM, developed at the Georgia Institute of Technology [3], [6]. OOSIM is implemented in C++ and runs on Unix

workstations. In this section, we briefly discuss a key feature of this architecture, namely the manufacturing *reference model* (i.e., a formalized and normative description of a class of systems) from which it is developed.

The reference model is being extended so that it synthesizes three key elements: (i) a representation of manufacturing systems needed to develop simulation abstractions for performance assessment, (ii) a representation needed to support control policy development, and (iii) a representation needed to support implementation of control software. This paper focuses on the first two representations. The reference model is obtained through *domain analysis*. Domain analysis is the process of organizing knowledge about a class of problems (i.e., a problem domain) for purposes of developing software abstractions [1].

Our domain analysis combines two decompositions of the domain of manufacturing systems: plant vs. control, and processing vs. logistics. The relevant entities in the reference model include the following.

- *Material* consists of a set of jobs categorized by job type. Each job has a unique identity, a job type, a space requirement and a process plan. The process plan consists of a set of operations which may have precedence relationships.
- A *location* abstracts the parts of manufacturing equipment which can hold or process material. Thus, a location is a building block for devices such as transporters and machines. It is characterized by capacity and capability (i.e., set of functions which it can perform). Hence, a location has two state variables: content state (i.e., set of material which occupies or has reserved it), and processing state (i.e., its current activity status, which may, for example, be “busy,” “failed” or “idle”).
- The control system consists of a set of *controllers*, each overseeing a specific set of devices and other controllers. This set of devices and other controllers constitutes a

controller domain. The entire manufacturing system may be decomposed into a set of controller domains. A domain has interface points called *shared locations* for purposes of material transfer to other controller domains. A controller maintains state information about the devices and other controllers in its domain and about entities with which it interacts outside of its domain, through a *client model* construct.

- The production sub-system consists of the set of machines which perform physical transformations or logical transformations (e.g., an inspection) on material. A machine is a device, or collection of processing locations and buffer locations. An *operation-location map* specifies the set of locations which may perform a particular operation in a process plan.
- The transportation sub-system moves material through the production facility. It consists of a set of constituent sub-systems which comprise a material handling network, and at a lower level, a set of transporter devices. A *domain map* specifies the set of transporters (or a transport controller) which may move a job from one machine location to another within a controller domain.

Taken together, these entities form the basis for the simulation modeling abstractions provided by OOSIM. These are implemented as a set of C++ classes. At the same time, a further abstraction of the plant (i.e., production and transportation sub-systems) forms the basis for a representation which enables the development of control policies. This abstracted representation of the plant is called a *resource allocation system* (RAS). The dynamic behavior of the RAS is modeled as a finite state automaton. This representation has been successfully used in the development of deadlock avoidance policies [10]. The fundamental elements of a resource allocation system are a set of resources and a set of jobs. Table 1 shows the mapping from the reference model to an RAS.

Table 1. Simulation and RAS representations

Simulation-based representation	RAS representation
Job	An RAS has a set J of job types which are active in the system at any time.
Location	Each location L_i is mapped to a resource type R_i with the same capacity.
Process Plan, Domain Map and Operation-Location Map	Each job type $JT^j \in J$ has a sequential set of process stages, each of which is characterized by a set of resources which the job must obtain before the process stage commences. In the current RAS model, the resources are uniquely defined and include transportation resources. Hence, flexibility is not considered, and the domain map and operation-location map are incorporated with the process plan.
System state consists of the assignment of job objects to location objects (content state), the current processing state of these location objects at any point in time, and the current operation of each job in its process plan.	System state is determined by the number of job instances executing at each process stage supported by the current system configuration. (Given the unique definition of resource requirements for each process stage, it follows that the system state also determines the current allocation of resources of resources to jobs in the system.)

The resource allocation system representation and finite state automaton model form the basis for development of algorithms which govern such control decisions as dispatching, job release and deadlock avoidance. These algorithms can be implemented within a controller object(s) in the simulation to assess their effect on system performance, and eventually in a real controller as software controlling activities on the shop floor.

III. THE PROBLEM OF REAL-TIME CONTROL OF MANUFACTURING SYSTEMS

In the past fifteen years, significant advances have been made in flexible automation. Yet real-time control of flexible automation remains a difficult task. Smith and Joshi [12] cite the lack of standard models for control system development. Additionally, system complexity is a real problem which results from a large variety of event types, stochastic system behavior and even from flexibility itself. There is a gap between control policy development and performance assessment. We describe our approach to the problem here.

A. Hierarchical Decomposition

A typical approach used to address the complexity of FMS control systems is hierarchical decomposition into sub-problems. Such a decomposition is organized as a set of layers, each associated with a different time-scale. A widely used decomposition features three levels [1], [13]. At the highest level, *strategic decisions* involve initial deployment and subsequent expansion of the production system. Descending the hierarchy, *tactical decisions* involve allocating production capacity to various product types to satisfy external demands. Finally, *operational decisions* coordinate shop floor activities such as job release, routing and dispatching to meet requirements of the tactical decision level.

This particular decomposition has effectively addressed performance-related issues. However, it does not account for lower-level issues involved in the real-time control of a manufacturing system. These types of issues can be classified as (i) ensuring stable and logically correct system behavior, and (ii) interfacing the control system with manufacturing equipment. In the first category, concerns focus on resolving deadlock situations and reacting to disruptions such as machine failures and job expediting. This type of control is called *structural control* [9]. In the second category, concerns focus on developing robust interfaces to handle the great variety of manufacturing equipment, standardized models to represent equipment behavior, and methods to facilitate control software portability in an environment of changing technological standards.

In this work, we use an extended hierarchical decomposition featuring the five levels shown in Fig. 1.

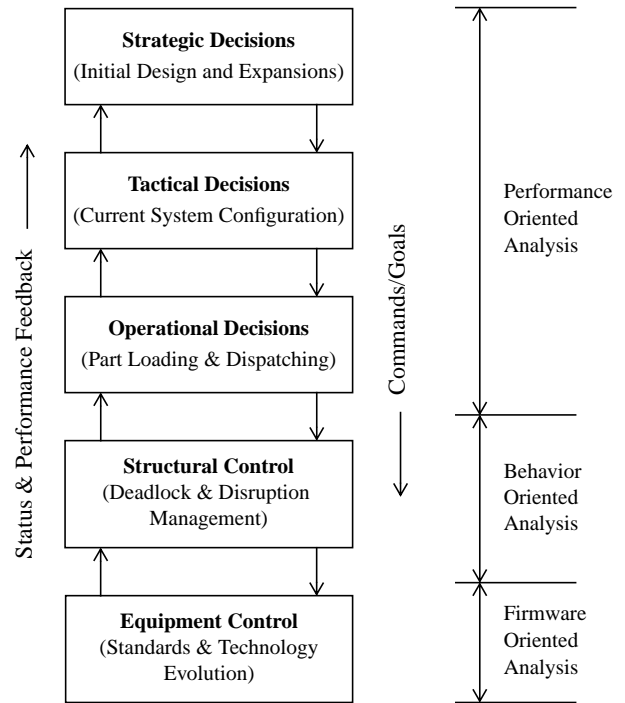


Fig. 1. Hierarchical framework for design and control of discrete-parts manufacturing systems

B. Implementation of a Controller Model

For the remainder of this paper, we concentrate on FMS operational control, as well as structural control associated with deadlock management, in particular *deadlock avoidance*. A deadlock is defined as a circular wait condition among resource allocation requests which causes all the requests to be persistently infeasible. Obviously, the outcome of a deadlock in an automated FMS prevents the smooth flow of jobs through the system, and it drives the utilization of the resources involved to zero. Deadlock avoidance is a control policy that uses on-line feedback about system state to control resource allocation so that a deadlock does not occur [1].

These types of decisions are made in an automated system by a controller. We view a controller as an event-driven and data-driven entity. That is, its decisions are invoked by events, or messages which it receives, and it has access to data about the system which it uses to make decisions. It then creates events (messages) which eventually translate to physical events on the shop floor.

In our simulation modeling environment, the controller is an *object*. It communicates with other objects via *message-passing*. The controller has a set of decision routines, or *controller scripts*, which are decision logic for operational and structural control. Execution of a controller script is triggered when the controller receives a message from another object in

the simulation. A particular type of message triggers a particular type of script. For example, consider an operational control decision. A machine has just finished processing a job in a manufacturing cell. The machine sends a message to the cell controller object, which invokes a controller script. This script results in a routing decision (i.e., to which machine should the job go next), a dispatching decision (i.e., should the job be taken there now), and a material handling dispatching decision (i.e., which transport device should move the job). These decisions may be sequential or may be dependent on one another, or they may be distributed over more than one controller. The point is that the controller script construct provides a structure to implement algorithms for these decisions within the simulation. If the script executes successfully (i.e., the decision is currently feasible), the controller issues a message. Otherwise, it creates a *pending work tag* for that particular task. Once the controller receives a message indicating feasibility of that task, the pending work tag is removed, and the script is executed.

Structural control is implemented in a script as a module called by operational control routines. It provides decisions which ensure that the operational decisions are keeping the system behavior stable and logically correct. In the previous situation, a deadlock avoidance module checks the job dispatching decision to determine whether it is admissible (i.e., does not lead to deadlock). It turns out that for the considered class of RAS, the *optimal* deadlock avoidance policy (DAP) is uniquely defined and effectively computable [9]. Optimal in this sense means minimal restrictiveness. However, in many cases, its complexity is superpolynomial. Practical considerations make us focus on deadlock avoidance policies which are *scalable* and *provably correct*.

Scalability implies that the policy implementation, especially its real-time execution, is of polynomial time with respect to system size, as defined by the number of system resource types and supported process stages. Correctness implies the rigorous establishment that the system will never deadlock when controlled by the policy (under normal operating conditions). Since the optimal policy is *NP-hard* in the general case [9], scalable and correct DAP's will be *sub-optimal*. A promising class in terms of efficiency is that known as Ω -DAP's. Furthermore, a sub-class of resource allocation systems with considerable practical significance has recently been shown to admit a scalable optimal DAP [10].

Fig. 2 shows the generic structure of the controller and the logical behavior of its scripts. The controller has a set of methods which enable it to communicate with other objects. The modeler specifies the messages, which have the format $\langle \text{sender, message_type, parameters} \rangle$. Each message is matched to a controller script based on *message-type*. The controller has a set C_o of scripts for operational control, and another set C_s of scripts for structural control. A specific

decision generated by a script in C_o is submitted to a script in C_s . For example, the script in C_s could be a deadlock avoidance script which determines whether the dispatching of a job to its next machine is currently admissible. Another possibility is that the controller employs a *deadlock detection and recovery* scheme (e.g., [14]). In this case, if the operational control script finds that a job dispatching request is currently infeasible because it is blocked, it submits the dispatching request to a deadlock detection script in C_s . If this script discovers that the system is currently in deadlock, it initiates a series of messages to result in an appropriate recovery scheme. Thus, structural control may result either in a restriction on actions proposed by operational control, or in additional tasks.

The controller maintains a variety of information. This information is contained in the *real-time control database* which the controller accesses to support execution of its controller scripts. Much of the information is *static* in that it represents the system structure. Process plans, operation-location maps and domain maps all represent structural information which does not change unless the system is somehow reconfigured. *Dynamic* information about the system state is captured in the client model database and in a set of pending work tags. Client models capture the content state and current processing state of each location in the controller's domain. Pending work tags are messages received previously which could not be acted upon because they were infeasible at the time. A status update message may directly result in an update to the real-time control database, or it may trigger execution of a task in the pending work list.

This controller primarily implements *dispatching rules* as a means of scheduling rather than *global optimization* scheduling. A more complex controller could have a set of

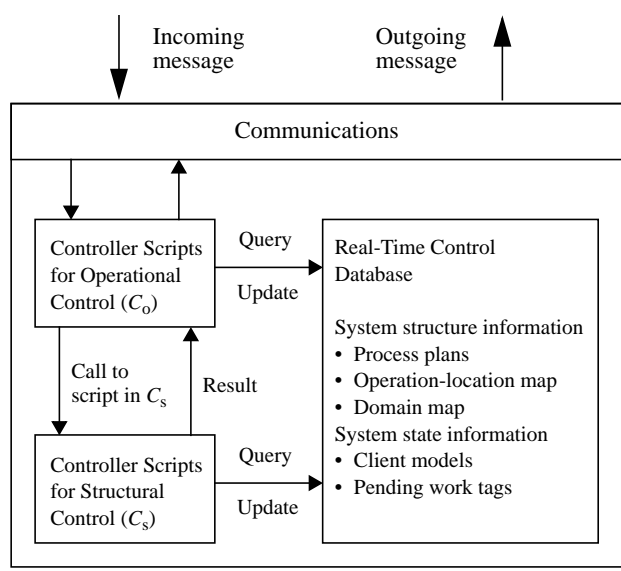


Fig. 2. Structure and behavior of FMS controller

controller scripts which constitute a scheduler. It would take input in the form of production goals and deadlines and then generate a schedule. This controller would also maintain as part of the real-time database a representation of the current schedule. The operational control scripts would then be responsible for coordination of material movement and processing activities so that the schedule is met.

The controller script provides a useful structure for implementing control policies within a simulation model for performance evaluation. The next section discusses a demonstration of the simulation-based framework.

IV. DEMONSTRATION

We briefly describe an example featuring a cluster tool used for semiconductor fabrication. Our goal is to demonstrate our proposed framework for simulation-based performance assessment of real-time control. The particular problem studied is deadlock avoidance during real-time control. Our goal is to determine the system performance of a policy in terms of throughput, which is a primary performance measure in the semiconductor industry [8].

A. System Description and Base Model

Consider a cluster tool (a highly automated device used in the fabrication of semiconductor wafers). It consists of a set of modular process chambers organized around a mainframe enclosing a controlled environment. Inside the mainframe is a robotic material handling device. An operator loads and unloads cassettes of wafers through loadlock chambers, and the wafers are processed and transported individually. Fig. 3 shows a schematic.

In the model, each chamber is represented as a processing device with a location for each unit of processing capability. The robot is represented as a transport device with one location. The loadlock chambers are represented by location objects with capacity of 100 wafers; wafers are represented as job objects. Process plans are sequential, and each process

location can perform a specified set of operations (operation-location map). Finally, the controller is represented as a controller object with a set of controller scripts.

In our model, we assume that there are no process chamber or robot failures, and that process and transport times are deterministic due to the automated system operation. Also, we consider only static routings. Hence, the process plan is strictly sequential, and the operation-location map provides a one-to-one mapping between process operations and chambers. For simplicity, we do not distinguish between the two. The main variability in this model is introduced by blocking and control policy restrictions designed to keep the tool from entering into a deadlock state.

B. Deadlock Avoidance in a Cluster Tool

Cluster tools are a new technology. A key question is how best to use the modular chamber design to promote system performance. Current practice uses a capacity of one wafer for each process chamber and fairly simple wafer routings. For example, the chambers may all be assigned the same operation so that they function in parallel. Likewise, wafers may be routed through the chambers in serial progression. It is not clear, however, that these configurations result in the best system performance. We are therefore investigating more complex wafer routings.

If a cluster tool is to process a set of wafers which exhibit a complex part flow, then deadlock is possible. In particular, a deadlock can occur if a cycle exists among the various possible wafer movements for a given wafer mix. Of course, one can prevent a deadlock simply by disallowing wafer mixes that exhibit a cycle. However, we are interested in part mixes with complex routings (e.g., a re-entrant flow or multiple wafer types). Therefore, we consider the possibility of deadlock.

Here, we consider a tool configuration in which each process chamber has a capacity of two wafers. In this configuration, an *optimal* deadlock avoidance policy may be implemented [10]. This is possible because this configuration does not admit “unsafe” states (i.e., non-deadlock states which are guaranteed to lead to a deadlock state). Thus, a proposed part movement can be evaluated via a one-step look-ahead procedure to detect whether it results in deadlock. This procedure is of polynomial complexity [9]. Given a part movement request, the look-ahead algorithm considers the system content state if the move is made. It determines whether a deadlock results through the following steps.

0. Mark all resources having a job. Label this resource set as \mathbf{M} . Set $s := \text{false}$.
1. If $(\mathbf{M} \neq \emptyset \text{ and } s = \text{false})$, set $s := \text{true}$. Else go to step 3.
2. Scan \mathbf{M} and eliminate any resources from \mathbf{M} which have excess capacity, or which have a job that will next exit the system or move to a resource not in \mathbf{M} . If any resources are eliminated from \mathbf{M} , set $s := \text{false}$. Go to step 1.

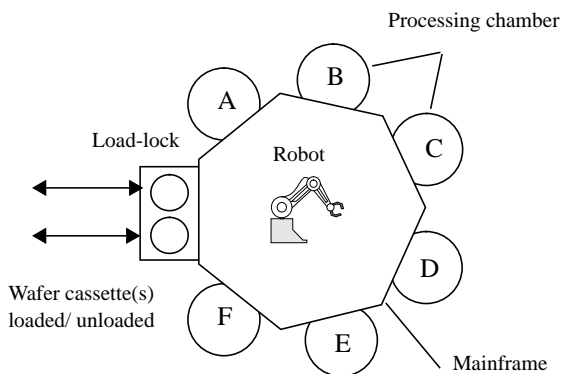


Fig. 3. Configuration of a cluster tool

3. If ($\mathbf{M} = \emptyset$), end with no deadlock. Else end with a deadlock consisting of the resources in \mathbf{M} .

This algorithm is implemented as a script in the structural control module. It is invoked by a dispatching script from the operational control module. For this example, we consider a simple dispatching rule for operational control, namely first-come-first served.

C. Demonstration

The data used for the demonstration are shown below in Table 2 and Table 3. This is not actual data, but it is representative. It is intended for illustrative purposes. Wafer type I features a re-entrant flow pattern through the chambers, while wafer type II features a simple serial flow pattern. A deadlock can occur due to the re-entrant flow.

Table 2. Wafer data

Wafer Type	Batch Size	Routing
I	80	A → B → C → D → E → B → C → F
II	40	A → B → C → D → E → F

Table 3. Processing time data

Chamber	Time (sec.)	Chamber	Time (sec.)
A	20	D	53
B	25	E	47
C	28	F	23

Robot travel times used in the model are on the order of 1.5 sec. to 6.0 sec., depending on distance being traveled between chambers. Travel time also depends on whether the robot is carrying a wafer. The robot travel times include loading and unloading of wafers from the chambers.

For simultaneous production of the two wafer batches, we obtain a production flowtime of 183.0 min. for the two batches of wafers. This translates to a combined throughput of 39.4 wafers per hour. We intend to test further scenarios using the basic cluster tool model, including the effect of different operational control policies under the structural control policy used here (e.g., last-buffer-first served), and the effect of different structural control policies and configurations.

V. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we have focused on the development of a simulation environment, or virtual factory, to assess system performance. Our initial work involves the integration of control policy development and simulation-based performance assessment. To date, we have developed a

reference model which supports both activities and provides the basis for their integration.

Future work includes (i) extending the underlying reference model to support implementation of control software, (ii) development of a set of generic controller scripts for operational and structural control, (iii) incorporating routing flexibility and failure handling into our framework, and (iv) extending our scope to cover tactical and strategic decision-making in manufacturing. In addition, we intend to validate our results over a range of case study applications.

VI. REFERENCES

- [1] G. Arango and R. Prieto-Diaz, Domain analysis: concepts and research directions, in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Eds., Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 9-33.
- [2] Z. A. Banaszak and B. H. Krogh, Deadlock avoidance in flexible manufacturing systems with concurrently competing process flow, *IEEE Trans. Robotics and Automation*, vol. 6, no. 6, 1990, pp. 724-734.
- [3] D. A. Bodner, *Real-Time Control Approaches to Deadlock Management in Automated Manufacturing Systems*, Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, 1996.
- [4] S. B. Gershwin, Hierarchical flow control: A framework for scheduling and planning discrete events in manufacturing systems, in *Proceedings of the IEEE*, vol. 77, 1989, pp. 195-209.
- [5] D. J. Miller, Simulation of a semiconductor manufacturing line, in *Commun. ACM*, vol. 33, no. 10, 1990, pp. 98-108.
- [6] S. Narayanan, D. A. Bodner, U. Srekanth, T. Govindaraj, L. F. McGinnis and C. M. Mitchell, An object-based, direct-image approach to the modeling and simulation of manufacturing systems, technical report MHRC-TR-94-04, Material Handling Research Center, Georgia Institute of Technology, Atlanta, GA, USA, 1994.
- [7] S. Narayanan, D. A. Bodner, U. Srekanth, T. Govindaraj, L. F. McGinnis and C. M. Mitchell, Research in object-oriented manufacturing simulations: An assessment of the state of the art, *IIE Trans.*, to appear.
- [8] T. L. Perkinson, P. K. McLarty, R. S. Gyurcsik and R. K. Cavin III, Single-wafer cluster tool performance: an analysis of throughput," *IEEE Trans. Semiconductor Manufacturing*, vol. 7, no. 3, 1994, pp. 369-373.
- [9] S. A. Reveliotis, *Structural Analysis and Control of Flexible Manufacturing Systems with a Performance Perspective*, Ph.D. dissertation, University of Illinois, Urbana, IL, USA, 1996.
- [10] S. A. Reveliotis and P. M. Ferreira, Deadlock avoidance policies for automated manufacturing cells, *IEEE Trans. Robotics and Automation*, vol. 12, no. 6, 1996, pp. 845-857.
- [11] S. Ruiz-Mier and J. Talavage, A hybrid paradigm for modeling of complex systems, in *Artificial Intelligence, Simulation and Modelling*, L. A. Widman, K. A. Loparo and N. Nielsen, Eds., New York: Wiley Interscience, 1989, pp. 381-395.
- [12] J. S. Smith and S. B. Joshi, Reusable software concepts applied to the development of FMS control software, *International Journal of Computer Integrated Manufacturing*, vol. 5, no. 3, 1992, pp. 182-196.
- [13] K. E. Stecke, Design, planning, scheduling and control problems of flexible manufacturing systems, in *Annals of Operations Research*, vol. 3, 1985.
- [14] R. A. Wysk, N. S. Yang and S. Joshi, Resolution of deadlocks in flexible manufacturing systems: avoidance and recovery approaches, *Journal of Manufacturing Systems*, vol. 13, no. 2, 1994, pp. 128-138.