

# Integration of Structural and Performance-Oriented Control in Flexibly Automated Manufacturing

Douglas A. Bodner, Jonghun Park, Spyros A. Reveliotis and Leon F. McGinnis  
Keck Virtual Factory Lab • School of Industrial and Systems Engineering  
Georgia Institute of Technology  
Atlanta, GA 30332-0205 USA  
dbodner@isye.gatech.edu

**Abstract** – Advanced mechatronic systems increasingly are finding application in modern manufacturing, as high-tech production requirements dictate high levels of machine precision, automation and integration. A high-tech factory typically contains a variety of flexibly automated processing and material handling systems. Working together, these systems are capable of executing complex operations to produce sophisticated end products. At the same time, such systems must operate in a logically correct and efficient manner to help justify their capital cost. This requirement points to the importance of system-level control to coordinate shop floor equipment and activities. In this paper, we present a generic control architecture designed to promote logically correct and efficient system behavior. The architecture and its associated control logic are based on a discrete-event systems representation. We also briefly outline future plans to use the architecture as the basis for simulation-based prototyping.

## I. INTRODUCTION

Flexibly automated equipment is increasingly found in high-tech manufacturing environments such as semiconductor fabrication, electronics assembly and biomedical production. This equipment consists of versatile and high precision devices for processing and material transport. The electronic, mechanical, and software control components of this equipment work together to perform the intricate processing and transport operations required to produce sophisticated end products and support product variety and customization. From a systems-level perspective, computer-integrated control enables individual pieces of automated equipment to be coordinated as an aggregate mechatronic system.

One important consideration in mechatronics design is flexibility [18]. Advances in electronic and mechanical hardware and in computing and network technology have enabled greater factory flexibility. In many cases, though, flexibility has not lived up to its original promise of enabling low-volume and high-variety production (e.g., [13, 15]). One fundamental stumbling block is that of control, due to *ad-hoc* approaches in development of factory control software [6]. Another important consideration relates to cost. A factory with high-tech automated systems can be an expensive proposition. For example, estimates place the capital cost of a new semiconductor fab at well over \$1 billion, most of which is equipment investment [16].

Two important conclusions can be drawn from this situation, both related to control. First, to realize more fully the benefits of flexibility, control software at the system

level must be based on generic and formal models of system structure and behavior (e.g., [6, 8, 17]). Second, to help justify equipment capital costs, flexibly automated systems must operate in an efficient and logically correct manner. In terms of efficiency, control of the factory must seek to “optimize” some performance metric (e.g., throughput). In terms of logically correct behavior, the state of the factory must not be allowed to reach an “unrecoverable” state. One example of such a state is a *deadlock*, which is defined as a set of pending tasks, each of which requires resources held by another task in the set such that a circular wait condition exists [1].

Due to its complexity, the control problem often is decomposed hierarchically into sets of more manageable control problems based on time-scale. For example, a well-accepted decomposition divides the problem, along decreasing time-scales, into strategic decisions (design), tactical decisions (configuration) and operational decisions (scheduling) [14]. This decomposition relates only to performance-based control. To ensure logically correct behavior and implementability, it should be extended by adding structural control and equipment interfacing and control at lower levels [2]. Structural control refers to control that guarantees logically consistent system behavior (e.g., deadlock-free behavior) [12].

There exists a wealth of literature devoted to performance-based control of flexibly automated systems (i.e., scheduling). In addition, recent years have seen many results in structural control, mainly in the area of deadlock avoidance. However, little work has considered the combination of the two, as is necessary for implementation of fully automated and truly flexible systems. In the remainder of this paper, we present a control architecture designed to integrate performance-based control with structural control. First, we describe the formal model used as the basis of this research. Then we present the architecture, the control logic and the communications between control entities. Finally, we conclude with plans to create an environment for system prototyping.

## II. FLEXIBLY AUTOMATED SYSTEMS

### A. System Model

The class of automated systems considered has several features: a set of workstations, a flexible material handling system that connects workstations, and an automated

storage/retrieval system that accommodates entering and leaving jobs. Each workstation has a set of processors and buffer locations. The material handling system has a set of transporters, a pre-specified transportation network, and pick-up/deposit points. For concreteness, we assume an automated guided vehicle (AGV) system. For purposes of control, the production sub-system is distinguished from the material handling sub-system, since they have different operational characteristics.

This research considers only discrete-part manufacturing systems, and focuses on their discrete-event behavioral characteristics. Hence, we rely heavily on the discrete-event dynamic system (DEDS) formalism [3] for modeling. In this formalism, events trigger discrete changes in system state. Events may trigger decisions by a control entity, i.e., controller, according to some pre-specified logic. Likewise, a control decision may result in an event.

The overall framework for the system model is based on an existing object-oriented representation of manufacturing systems [7]. In summary, this representation, used as the basis for discrete-event simulation modeling, characterizes an automated factory as consisting of:

- a set of jobs processed by the system, each with a job type and a corresponding process plan dictating a sequence of process operations to be performed;
- a production sub-system, consisting of one or more sets of workstations, or machines, each of which has processing and buffering capability;
- a material handling sub-system, consisting of one or more sets of transport devices, each of which moves jobs through the system; and
- a distributed control system that is partitioned into control domains, each of which has a controller overseeing a set of equipment and/or other controllers.

A control domain, for example, can be a cell or an AGV network. Control domains contain interfaces, or shared locations, for purposes of job transfer to other domains. A controller (i) maintains state information about its domain, (ii) has access to system information, (iii) has decision-making logic, and (iv) communicates with other entities in the system. State information corresponds to such data as current availability of equipment and locations of jobs (including a representation of a controller's pending tasks, e.g., a job that has finished and needs to be moved). System information includes:

- process plans;
- an operation-location map, or a mapping between operations and the equipment that can perform them; and
- a domain map that maps valid movements between locations in the domain to a material transporter (or controller overseeing multiple transporters).

## B. Resource Allocation Systems

For structural control logic purposes, the representation is abstracted to a *resource allocation system* (RAS), which characterizes a factory as a set of resources and entities that move through the resources according to a set of instructions [12]. This formalism provides a mathematical basis for developing control algorithms and for proving

algorithm properties such as tractability and correctness. The production sub-system is represented by a factory RAS, consisting of (i) a set of workstations  $W = \{W_i, i = 1, \dots, m\}$ , each with finite buffer capacity  $b(W_i)$  and finite number of processors  $p(W_i)$ , and (ii) a set of job types  $J = \{J_j, j = 1, \dots, n\}$ . Each job type  $J_j$  has a sequence of stages  $\langle J_{jk}, k = 1, \dots, l(J_j) \rangle$ , representing its process plan. If each job stage maps to only one  $W_i$ , the RAS is a *single-unit RAS* [12]. Otherwise, it is *disjunctive* and admits routing flexibility. At any time, the factory RAS state  $s_f$  consists of the allocation of job stages to buffers over all workstations.

Events in the factory RAS consist of the following:

$e_j^a \equiv$  arrival of a job of type  $J_j$  to the system

$e_{jk}^s \equiv$  start of job stage  $J_{jk}$

$e_{jk}^f \equiv$  finish of job stage  $J_{jk}$

$e_j^d \equiv$  departure of a job of type  $J_j$  from the system.

An event associated with a particular job instance  $x$  will be represented, for example, as  $e^a_x$ .

Similarly, the material handling sub-system is represented by a material handling RAS. We assume that a zone-based AGV system is employed to avoid transporter collisions, and that only one transporter may occupy a zone at a time. Additional assumptions include that a transporter is required to move a job between job stages, and that each transporter holds only one job. The material handling RAS is analogous to the factory RAS, except that transporters serve as the entities moving through the system, and zones in the network serve as the resources. A transporter moves through a set of zones in transporting a job. There may be more than one path of zones between any two workstations. Hence, the material handling RAS is disjunctive.

In moving a job between workstations, a transporter performs *inbound travel*, moving unloaded from its current position to the source workstation. Then it performs *outbound travel*, moving with the job from the source to the destination. After depositing the job, it may either move to a docking station or be dispatched to move another job. A transporter may be either TRAVELLING or PARKED. The transporter state at any time is characterized by its travel status and remaining milestones to be visited on its current transport (e.g., source, destination). If travel status is PARKED, it has no remaining nodes.

Zone connectivity is given by an undirected graph  $G = (N, A)$ , whose arcs represent zones and whose nodes represent intersections. The material handling RAS state  $s_m$  is a matrix of dimension  $|N| \times |N|$ , whose elements indicate whether a transporter occupies a zone. If the zone represented by arc  $(p, q)$  contains a transporter  $t$ , then  $s_m(p, q) \equiv t$ ; otherwise it equals zero. Note that this representation implies a direction in which the transporter is travelling in the zone (i.e., from intersection  $p$  to  $q$ ).

Events occurring in the material handling RAS consist of the following.

$e_{xk}^{is} \equiv$  inbound start of job  $x$  waiting for job stage  $k$

$e_{xk}^{if} \equiv$  inbound finish of job  $x$  waiting for job stage  $k$

$e_{xk}^{os} \equiv$  outbound start of job  $x$  waiting for job stage  $k$

$e_{xk}^{of} \equiv$  outbound finish of job  $x$  waiting for job stage  $k$

$e_{x,u}^s \equiv$  start of the  $u$ th zone for transport of job  $x$

$e_{x,u}^f \equiv$  finish of the  $u$ th zone for transport of job  $x$ .

Both RAS models provide state information used by a controller in making structural control decisions. In the production sub-system, a controller uses the RAS state to make decisions concerning the induction of jobs into the system, and the movement of jobs between workstations. The material handling sub-system receives movement requests from the production sub-system, and its controller uses the material handling RAS state to dispatch transporters and route them through zones in fulfilling these requests. Note that the state representation must also include time information (e.g., remaining process time, job due dates, etc.) to support performance-based scheduling.

### C. Structural Control of an RAS

Structural control is an important prerequisite to performance-based control, because it guarantees logically correct system behavior. Here, the focus is on deadlock avoidance. Structural control is implemented as a structural control policy (SCP) designed to avoid deadlocks. Previous research has established a number of results related to two required aspects of an SCP: correctness (i.e., guarantee of deadlock-free behavior) and tractability (i.e., polynomial-time execution of the SCP algorithm) (e.g., [1,4,5,10,12]).

To illustrate the need for structural control, consider a simple factory RAS with three workstations,  $W_1$ ,  $W_2$  and  $W_3$ , and two job types,  $J_1 = \langle W_1, W_2, W_3 \rangle$  and  $J_2 = \langle W_3, W_2, W_1 \rangle$ , as shown in Figure 1. Each workstation has one processor and one buffer (i.e., it can handle one job at a time). The RAS state  $s_f$  is represented by  $[s_{f1}, s_{f2}, s_{f3}]$ , where  $s_{fi}$  represents the content of  $W_i$  in terms of job stage. The state  $[J_{11}, J_{12}, J_{13}]$  is characterized as *safe*, since it does not lead to deadlock. It represents the unidirectional flow of job type  $J_1$  through the workstations. The state  $[J_{11}, \emptyset, J_{21}]$ , on the other hand, is characterized as *unsafe*, because it will lead eventually to a deadlock state of either  $[J_{11}, J_{22}, \emptyset]$  (if job type  $J_2$  is moved to  $W_2$  first) or  $[\emptyset, J_{12}, J_{21}]$  (if job type  $J_1$  is moved to  $W_2$  first). These are deadlocks because neither job can advance, since its next workstation is held by the other job, and there is a circular wait condition.

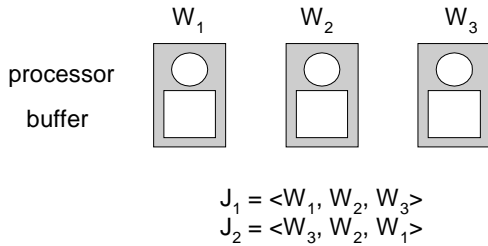


Figure 1. Example RAS

While this simplistic example is meant only to illustrate the concept, it should be noted that deadlock is a possibility in systems with limited buffer capacity and complex job flows. It is the responsibility of an SCP to disallow the system from entering an unsafe state. An SCP works either by enabling or by disabling events. A controller overseeing a factory RAS can control only the start of a job stage ( $e_{jk}^s$ ),

representing either the induction or advancement of a job. Thus, an SCP needs to determine whether a particular  $e_{xk}^s$  transitions the RAS into an unsafe state. The state space of an RAS is exponential; hence it is not represented directly. Rather, to meet the polynomial-time requirement, the SCP must rely on the satisfaction of some substitute condition that implies safety and that is tested in a one-step look-ahead fashion. That is, given a proposed action, the SCP checks the resulting RAS state to determine whether the movement is allowable, or *admissible*.

As a result, at any given an RAS state, a particular SCP divides the set of job start events into two sets: admissible and not admissible. An *optimal* SCP disables only those events that cause the system to enter an unsafe state. Hence, an optimal SCP is minimally restrictive, allowing a scheduling policy maximum latitude to select from different alternatives. An example of an optimal and polynomial-time SCP is presented by Reveliotis *et al.* [12] for a special class of RAS with  $b(W_i) \geq 2$  for all  $i = 1, \dots, m$ .

For many RAS cases, there is not a known optimal polynomial-time SCP. Hence, an SCP may be sub-optimal in the sense that it is overly conservative to guarantee deadlock-free behavior (i.e., it may disable a particular  $e_{jk}^s$  that does not cause an unsafe state).

Structural control for a material handling RAS is handled in a manner similar to that of the factory RAS. The SCP enables or disables zone advancement ( $e_{x,u}^s$ ) to ensure deadlock-free behavior. Structural control of an AGV network is discussed more fully in [10].

Our focus is not on development of new structural control policies, but rather on the integration of existing policies with scheduling policies. Hence, our architecture uses existing policies. The architecture is designed to be configurable with respect to SCP's in that one can select from among several policies for a particular production run.

## III. CONTROL ARCHITECTURE

### A. Control Architecture

Consistent with existing hierarchical control approaches, we utilize a hierarchy and define the factory supervisor (FS) as the controller responsible for managing the flow of jobs through the production sub-system. FS releases jobs to production, and dispatches jobs to required workstations. FS serves as supervisor to a set of workstation supervisors and material handling supervisors, each of which oversees a set of equipment resources. A workstation supervisor (WS) oversees a workstation, consisting of a set of processors and a set of buffer locations. A material handling supervisor (MS) oversees a set of transporters. The architecture is shown in Figure 2.

Each controller interacts with other controllers, sending commands and/or receiving state updates. The information flow is a critical consideration that affects how the whole system behaves. Hence, specification of communication events is important. FS sends commands to WS and MS, and WS and MS send status updates to FS. WS and MS supervise operation of the devices in their domains by communicating with the controllers of those devices.

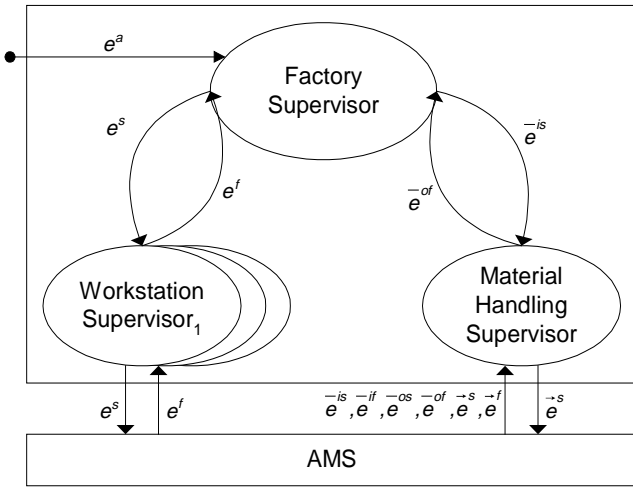


Figure 2. Control architecture

### B. Generic Controller Structure

A controller functions, in many ways, as a server in a client-server architecture [7]. It receives “requests” from clients and processes them by executing decision logic. For example, FS may receive a message from WS that a job has finished processing at WS (i.e.,  $e_{jk}^f$ ). FS then invokes its decision logic to determine the admissibility of the transition to the next workstation. After admissibility is granted, FS then sends a movement request. MS receives this message, and then invokes its decision logic to determine which transporter should perform the movement, and it dispatches the transporter. In turn, FS acts as a server to WS, and as a client to MS. MS acts as a server to FS.

A controller maintains a representation of its domain through a set of client models that embody the RAS state and timing information. A client model simply is the RAS information about each resource with which a controller interacts. In addition, the client models contain the set of valid messages exchanged between a controller and its clients. When a controller receives a particular message, it triggers a decision routine that may update the controller’s client models, or may cause the controller to initiate a shop floor action such as moving a job.

Controllers use three considerations related to moving a job to the next workstation or a transporter to the next zone. The generic logic filters the movement request through each of these in turn. This is shown in Figure 3.

1. *Feasible*. A movement is feasible if it is physically or logically possible in the RAS (e.g., is not blocked).
2. *Admissible*. A movement is admissible if it is allowed by the SCP that guarantees no deadlocking.
3. *Dispatched*. A movement is dispatched if it is selected by the scheduling policy in use. This decision includes a determination of whether the movement is *schedulable*. A movement is schedulable if it can immediately be implemented according to the scheduling policy. (Some scheduling policies are *idling* in the sense that they delay a movement that otherwise can be made, in the interest of improving a performance measure such as average job

time in system.) If a movement is feasible, admissible, and schedulable, it is *dispatchable*, and can be selected for implementation by the scheduling policy. Example scheduling policies include dispatching rules such as least slack and last-buffer-first-served.

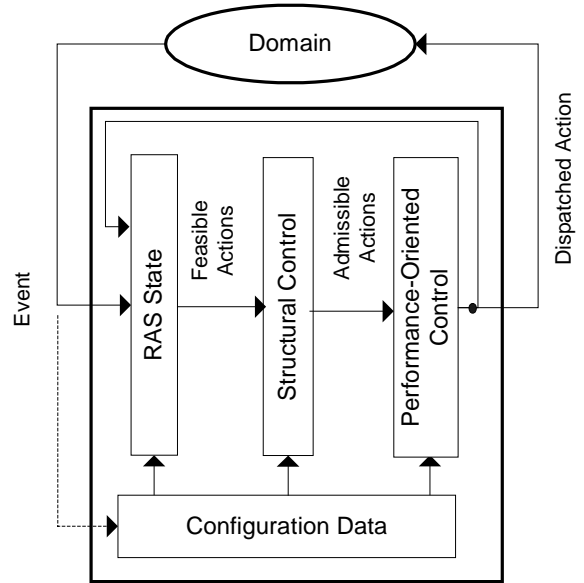


Figure 3. Control decisions

Thus, a controller’s logic integrates both performance-based and structural control considerations. A controller often receives requests that it cannot process since they are not feasible or admissible when received. For controller  $\psi$ , these requests are stored in a pending work queue  $Q_\psi$  (part of the state representation). When a resource becomes available, the controller checks  $Q_\psi$  and uses its decision logic to select which one, if any, to service.

## IV. CONTROL LOGIC

In this section, we describe control logic for the supervisors, mainly concentrating on the factory supervisor.

### A. The Factory Supervisor

Like other controllers, the control logic for FS acts as an event-handler, receiving message events and reacting to them. FS utilizes a factory RAS representing all workstations within its control domain. When it receives a message event  $e_j^a$  or  $e_{jk}^f$ , it must decide whether to induct or advance the job in question. Before doing this, it must decide whether the corresponding job induction or advancement request is dispatchable. Figure 4 shows the control logic for FS in flow chart form.

If event  $e_{sk}^s$  is selected by FS to be dispatched, it is set in motion when FS sends a command (representing  $e_{sk}^{is}$ ) to the appropriate MS. The actual command contains the source, the destination and the job identifier. MS implements the command by dispatching a transporter. When the transporter finishes moving the job, MS communicates  $e_{sk}^{of}$  to FS. Then, FS commands WS to start the job (i.e., event  $e_{sk}^s$ ). During this time, FS must track the current location of

job  $x$  and the space at the destination workstation reserved for  $x$ . This is critical in an automated system, to prevent a buffer space from being allocated to more than one job simultaneously. To do this, FS maintains a current factory RAS state  $s_f$  and a target factory RAS state  $s'_f$ . The target state represents the future resource allocation based on resource reservations made by FS. It is advanced each time FS sends a movement request to MS. When FS receives acknowledgment from MS that the job has been picked up from the source ( $e^{os}_{xk}$ ), FS updates the current state  $s_f$  to show the job at the destination and a free buffer at the source.

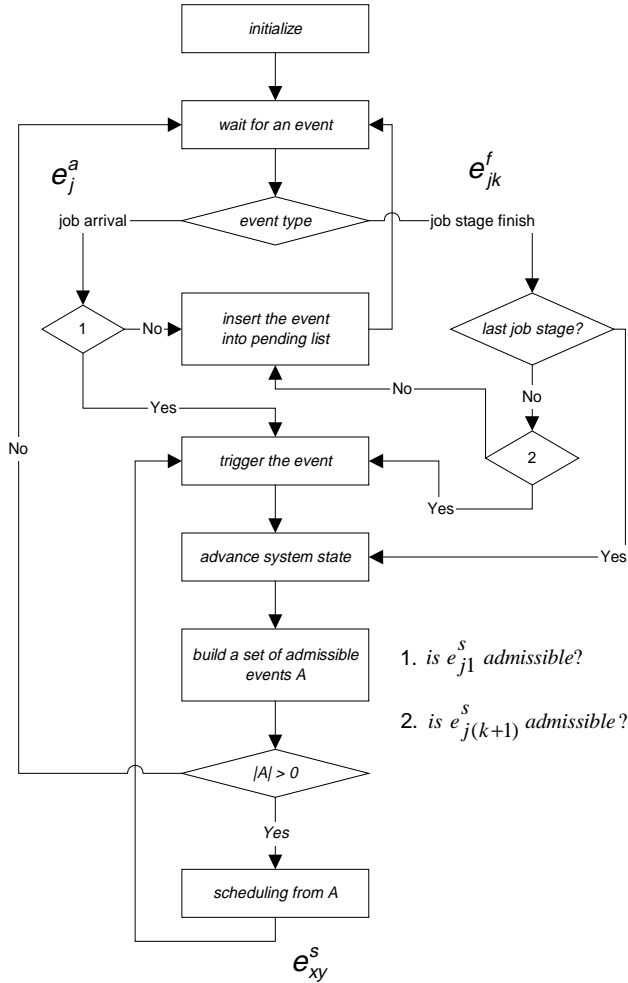


Figure 4. FS control logic

FS may have a number of movement requests outstanding at any given time. If MS is unable to handle all requests immediately, the pending requests are stored in its pending work queue  $Q_m$ . When there are multiple transporters, concurrency of material handling operations can be increased as the size of the queue becomes larger. This is done by planning the requests ahead instead of dispatching myopically. In this way, transporters can be coordinated in a more flexible way. However, we must verify whether logical correctness of the factory RAS can be maintained. Fortunately, under the protocols defined in this paper, we can show that, given any two admissible

factory RAS states, if one state results from the other by advancing some job stages by one step, then any feasible transition sequence leading from one state to the other is admissible [9]. Therefore, the pending requests in  $Q_m$  can be entirely shuffled without causing a deadlock, and an efficient transporter routing algorithm may be used, for example, to minimize the total travel distance.

### B. The Workstation Supervisor

In this control architecture, the main functions of the workstation supervisor are to assign jobs to processors, to start the processing of a job stage, and to communicate to FS the completion of a job stage. We assume that structural control is not needed for a workstation, since it consists of parallel processors. However, should a particular type of equipment need structural control, it can be implemented.

When  $WS_i$  receives a job start command  $e^s_{jk}$  from FS, it uses a scheduling rule to select a processor to process the job. This scheduling rule can account for differences in processors (e.g., processing time for an operation or quality). If no processor currently is available, the job start request is added to  $Q_{wi}$ . When  $WS_i$  receives notification from a processor that a job has finished, it communicates this to FS, so that FS can arrange transport of the job to its next workstation. In addition, if  $Q_{wi} \neq \emptyset$ ,  $WS_i$  invokes its logic to select the next job for processing. It should be noted that this control architecture does not address the specific operational control and interfacing for a particular piece of equipment; this type of control is at a lower level.

### C. The Material Handling Supervisor

The material handling supervisor MS is responsible for decomposing movement requests received from FS into commands to its transporters to implement the requests. It uses the material handling RAS, in a manner analogous to the way FS uses the factory RAS, to dispatch transporters to movement requests from FS, and to route transporters through zones. The material handling RAS allows the checking of feasibility (i.e., transporter and zone availability), admissibility (via the SCP), and performance-based routing and dispatching (via a scheduling policy that may seek to minimize vehicle travel time or vehicle congestion).

One difference is that MS maintains only the current state of the material handling RAS,  $s_m$ . It does not need to maintain a target state, since a transporter does not simultaneously hold its current zone and reserve its next. When a transporter requests its next zone, MS either grants the request, allowing the transporter to move immediately, or it defers the request, adding it to  $Q_m$ . For details, the reader is referred to [9].

### D. Implementation

The control architecture has been implemented in an existing robotic cell. This cell contains a track-mounted, six-axis F3™ robot from CRS Robotics, which serves as the material transporter, a set of simulated workstations, and a storage carousel. The control architecture is implemented in Java™. The FS, MS and WS applications

communicate with one another. This work includes the equipment interfacing and control aspects needed to operationalize the control architecture.

The control architecture is configurable in several ways, so that it can be used for different FMS configurations. The following items can be specified for a particular FMS configuration and a particular production run:

- number of workstations,
- number of processors and buffers at each workstation,
- job mix and process plans for each job type, and
- SCP used and scheduling policy used.

A variety of cell configurations have been tested successfully, up to a cell size of 16 workstations. Since the focus is on polynomial-time policies, the architecture can handle large cells. Currently, three SCP's have been implemented: two versions of the resource upstream neighborhood (RUN) policy [11] and the optimal policy [12].

## V. CONCLUSION AND FUTURE WORK

We have presented a control architecture that integrates performance-oriented and structural control in an effort to promote efficient and logically correct factory behavior. The architecture is generic in that it applies to a large class of flexibly automated systems. In addition, it provides a controller structure that applies both to production controllers and to material handling controllers.

The control architecture implementation has been interfaced with a simulation model of the robotic cell, implemented in IGRIP™, a CAD-based, 3D robotic system simulator from Deneb Robotics (see Figure 5).



Figure 5. Robotic cell model

At present, the primary emphasis is on the specification of the structural control aspects. Further research currently is underway to use this control architecture to create an experimental testbed using the IGRIP™ model, and to use this testbed for simulation-based testing of structurally controlled systems. Such testing will determine which performance-oriented scheduling policies perform well in structurally controlled systems. This is currently an open research issue.

This work will advance research being done in off-line systems programming, which seeks to program systems off-

line using detailed software models, in order to prototype behavior and performance. The goal is to create a highly configurable modeling and analysis environment (i.e., a virtual factory), in which the analyst can easily vary FMS configuration, production requirements and control policies, to assess the impact on system performance, and then be able to transfer the resulting specification to the real system.

## VI. ACKNOWLEDGMENTS

The authors would like to acknowledge the W. M. Keck Foundation, whose funding has provided the facilities necessary for this research.

## VII. REFERENCES

- [1] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows, *IEEE Trans. Robot. Automat.*, vol. 6, pp. 724-734, 1990.
- [2] D. A. Bodner and S. A. Reveliotis. Virtual factories: An object-oriented, simulation-based framework for real-time control, in *Proc. IEEE Int. Conf. Emerging Tech. and Factory Automat.*, Los Angeles, CA, 1997, pp. 208-213.
- [3] C. G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*, Boston: Aksen Associates, Inc., 1993.
- [4] J. Ezpeleta, J. M. Colom and J. Martinez. A petri net based deadlock prevention policy for flexible manufacturing systems, *IEEE Trans. Robot. Automat.*, vol. 10, pp. 173-184, 1995.
- [5] M. P. Fanti, B. Maione, S. Mascolo and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems, *IEEE Trans. Robot. Automat.*, vol. 13, pp. 347-363, 1997.
- [6] S. B. Joshi, E. G. Mettala, J. S. Smith and R. A. Wysk. Formal models for control of flexible manufacturing cells: Physical and system models, *IEEE Trans. Robot. Automat.*, vol. 11, pp. 558-570, 1995.
- [7] S. Narayanan, D. A. Bodner, U. Srekanth, T. Govindaraj, L. F. McGinnis and C. M. Mitchell. Modeling control decisions in manufacturing systems simulation using objects, in *Proc. IEEE Int. Conf. Syst., Man, Cybernet.*, San Antonio, 1994, pp. 1392-1397.
- [8] A. W. Naylor and R. A. Volz. Design of integrated manufacturing system control software, *IEEE Trans. Syst., Man, and Cybern.*, vol. 17, pp. 881-897, 1987.
- [9] J. Park, S. A. Reveliotis, D. A. Bodner and L. F. McGinnis. A distributed, event-based control architecture for flexibly automated manufacturing systems, working paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1999.
- [10] S. A. Reveliotis. Conflict resolution in AGV systems, technical report, School of Industrial and Systems Engineering, Georgia Institute of Technology, (submitted to *IIE Trans.*), 1998.
- [11] S. A. Reveliotis and P. M. Ferreira. Deadlock avoidance policies for automated manufacturing cells, *IEEE Trans. Robot. Automat.*, vol. 12, pp. 845-857, 1996.
- [12] S. A. Reveliotis, M. A. Lawley and P. M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems, *IEEE Trans. Automat. Contr.*, vol. 42, pp. 1344-1357, 1997.
- [13] P. Singer. The driving forces in cluster tool development, *Semiconductor International*, vol. 18, pp. 113-118, 1995.
- [14] K. E. Stecke. Design, planning, scheduling and control problems of flexible manufacturing systems, *Annals Oper. Res.*, vol. 3, 1985.
- [15] F. F. Suarez, M. A. Cusumano and C. H. Fine. An empirical study of manufacturing flexibility in printed circuit board assembly, *Oper. Res.*, vol. 44, pp. 223-240, 1997.
- [16] S. Tandon. Challenges for 300mm plasma etch system development, *Semiconductor International*, vol. 21, pp. 75-91, 1998.
- [17] T. M. Tirpak, S. M. Daniel, J. D. LaLonde and W. J. Davis. A note on a fractal architecture for modelling and controlling flexible manufacturing systems, *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 564-567, 1992.
- [18] D. Tompkinson and J. Horne. *Mechatronics Engineering*, New York: McGraw-Hill, 1996.