A Distributed, Event-Driven Control Architecture for Flexibly Automated Manufacturing Systems

Jonghun Park, Spyros A. Reveliotis, Douglas A. Bodner and Leon F. McGinnis
School of Industrial & Systems Engineering
Georgia Institute of Technology

Abstract

This paper presents a new distributed real-time control architecture for flexibly automated production systems. The modeling assumptions underlying the design are driven by and abstract the structure and operations of the emerging 300mm semiconductor manufacturing fab, one of the most extensively automated environments in contemporary manufacturing. The key element of the controller design itself, which differentiates it from past efforts, is the *distribution* of the control function to the constituent components of the system shop-floor architecture, while maintaining both the *logical correctness* and the *efficiency* of the system behavior. The architecture was designed to be easily implementable in the actual system shop-floor, and therefore, is aligned with and augments current practices in these environments. From a theoretical perspective, the proposed design has employed, integrated and extended a series of theoretical results from the field of Discrete Event Dynamical Systems. It is our expectation that the proposed architecture will also provide a formal framework for further analytical studies on the performance evaluation and performance-oriented control / scheduling of the considered class of manufacturing systems.

Keywords: Flexibly Automated Manufacturing System, Real-Time Control, Deadlock Avoidance, Structural Control, Scheduling

1 Introduction

Effective and efficient deployment and operation of large-scale flexibly automated production systems is of ever-increasing importance in today's competitive manufacturing. From a business perspective, flexible automation is a key enabler to the economies of scope and increased system responsiveness sought in contemporary manufacturing strategy. On the technology side, flexibly automated production systems are driven by applications in which the presence of the human operator implies high contamination risks for either the product or the operator. In some other cases, the part size is either too large or too small for the human operator, requiring the automation of material handling and storage functions.

As a case in point, consider the emerging 300mm semiconductor fab. An interesting discussion on the layouts contemplated for this environment, and their relative strengths and weaknesses, is presented in [10]. Three particular layouts are proposed, known as (i) *farm*, (ii) *hybrid*, and (iii) *modified hybrid* layout. In all three cases, the fab processing equipment is organized in a number of *bays*, stemming from a central aisle. This aisle contains the *interbay* material handling system, typically an overhead over-the-rail transportation system, transferring *cassettes of wafers* among the fab bays. The bays themselves are interfaced to the interbay material handling system through the bay *stocker*, i.e., an automated storage/retrieval system (AS/RS), which accommodates wafer cassettes waiting for processing in the bay area, or cassettes that have completed processing in the current bay, and request transferring to some other bay. Within the bay, process tools are also organized around another automated material handling system, typically an *Automated Guided Vehicle (AGV)*, *Rail Guided Vehicle (RGV)* or *Overhead Hoist Transport (OHT)* system, facilitating the *intrabay* part transfers. Depending on the specific layout adopted, process tools in a certain bay may be aggregated on the basis of a common functionality (farm layout), or the support of a certain processing sequence (modified hybrid layout). The third – i.e., *hybrid* – layout is obtained from the farm layout, by adding into each bay area the metrology tools required for evaluating the results of the corresponding processes. Each of the bay process and/or metrology tools also has a small buffer that accommodates cassettes requesting or having finished processing at that particular tool. The basic topology of such a manufacturing facility is presented in Figure 1.

Controlling the part flow in the manufacturing environment of Figure 1 is a complex task. At each decision-making point, the system controller is faced with a variety of options/control actions, and the applied control logic must ensure (i) the *logical correctness / consistency* of the system operation, as well as (ii) *satisfactory*, or in some cases, *optimized performance*. The extensive sequentialization and the re-entrant nature of the overall processing scheme make this a difficult control problem. The problem is further complicated by: (i) the coupling of the local process dynamics, introduced by the finite process buffering capacity; (ii) the multi-

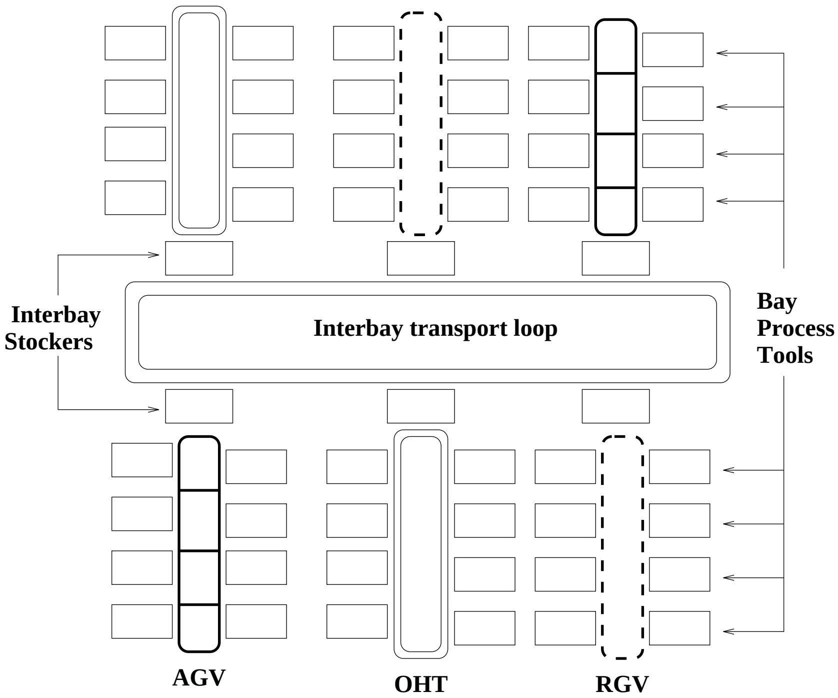

Figure 1: The typical layout for the emerging 300mm semiconductor fab

directional/arbitrary intra and inter-bay part flows; (iii) the drive for reduced cycle times; and (iv) the need to accommodate frequent engineering/experimental lots. The interfacing of the inter/intra-bay material flow control with the controller coordinating the traffic in the supporting automated material handling system is another open research issue. Finally, an integrated solution to the real-time part-flow control problem must also provide the exception-handling procedures for accommodating contingencies like the loss of some process units due to mechanical break-down and/or decalibration, and the expediting of "hot" jobs through the system.

Current semiconductor manufacturing practices have primarily addressed only performance-related issues, trying to maintain high resource utilization through (i) job-release (wafer-start) control, and (ii) the employment of heuristic/dispatching rules for controlling the part flow among the different bays and/or bay processes (e.g., [33, 39, 58]). Typically, the scheduling logic applied in a particular fab is a customized combination of a set of generic job-release and/or dispatching rules, which is shown to be efficient for the particular fab context through extensive simulation experiments. More involved scheduling schemes, attempting a more global view of the system operation, and more systematic performance optimization, have been proposed in the literature, e.g. [52, 8, 48, 37], but their effectiveness and implementability in an extensively automated production shop-floor is still an open issue.

Noticeably, there has been some academic research work on the development of an *integrated* solution to the production control problem. For instance, the works presented in [50, 25, 17, 3] develop hierarchical decompositions for, both, the production planning and control problem, as well as the shop-floor layout. However, although the proposed frameworks span from the production planning activity to real-time operation, they still focus on performance concerns, like the effective utilization of the system resources, throughput maximization, and/or maintaining minimal work-in-process (WIP). Logical/behavioral problems arising in the real-time control of the considered production environments have been largely ignored during the past design and development efforts, and they have been resolved mainly through external (human) intervention. More recently, there has been some effort to address these problems during the controller design phase, but the currently implemented policies seem to be completely customized to the particular fab operation, and have been developed in a rather ad-hoc fashion. Hence, the resulting control logic is not formally validated, and cannot be easily reconfigured under changing circumstances [26].

A series of academic research efforts focused more on the real-time operational details of the production activity and the formal characterization of the underlying system behaviors, can be found in [36, 4, 51, 9, 23, 26, 15]. A key contribution of these works has been shifting the emphasis from the strictly performance oriented issues in the manufacturing system related research, to issues concerning also the logical correctness and robustness of the system operation, a crucial factor for the automation of these environments. Hence, a whole new field, characterized as *logical / structural control* of flexibly automated manufacturing systems has been developed in the last decade, with some important results reported in, e.g., [1, 53, 57, 47, 14, 13]. This research, however, has still to address, both, (i) the implementation of the developed results in a distributed fashion that will accommodate the cellular structure of modern production environments – e.g., bay-oriented structure of the modern fabs – and (ii) the successful integration of the proposed logical control policies with the more traditional performance-oriented control / scheduling logic. Only recently, the work presented in [12] has initiated the effort of integrating some results obtained from the logical control paradigm in the broader real-time manufacturing system control framework.

Motivated by the above observations, this paper seeks to promote the development of an integrated real-time control framework for flexibly automated manufacturing systems, like the 300mm semiconductor fab, by augmenting existing manufacturing practices with new analytical results and insights regarding the behavior and control of these environments, provided by Discrete Event Dynamical Systems (DEDS) theory. Specifically, the proposed framework seeks:

1. to *distribute* the control function to the primary system constituents – i.e., the system processes/workstations, material handling system components, and the higher-level cel-

lular structures – in a way that maintains the logical correctness and robustness of the overall system behavior, and

2. to effectively *integrate* the logical/structural and performance related control at each system unit.

Regarding the first objective, we develop a generic hierarchical representation of the system structure, which, in turn, reflects in the adopted decomposition of the underlying control problem. This decomposition also allows us to employ well-developed structural and performance control policies regarding the internal part-flow for each of the recognized system modules, and reduces the main research issue to the development of the *inter-module communication protocols* that will ensure the consistent and efficient operation of the entire manufacturing system. Hence, our work is conceptually in line with the current computer integrated manufacturing (CIM) system modeling practice [22], that perceives the underlying (real-time) control problem as the development of a communication protocol that enables the orderly request and acknowledgment of a set of functional services among the set of interacting manufacturing components. The detailed definition of the manufacturing entities recognized by the proposed system representation, and their interaction, are motivated by the OOSIM modeling paradigm [35, 5, 6], and are discussed in the next section.

Regarding the second objective, our solution is based on a DEDS-theoretic representation of the production system entities and their internal control function, which is conceptually similar to that employed in [15]. However, instead of the entity-attribute-relationship paradigm [55, 56] used in [15], our work employs the Colored Petri net (CPN) [24] framework for the detailed modeling of the system component structures and their (controlled) behavior, and more importantly, it refines the "scheduling policy" entity of [15], by adding a distinct logical/structural control function to the original, more performance-oriented specification. It is also noted that, similar to [15], the proposed framework "seeks to reveal [and incorporate] basic common features for a wide class of control strategies for an arbitrary FMS, [rather than propose] specialized scheduling algorithms". Yet, an important future research issue is the investigation of the efficiency of currently adopted scheduling schemes in the context of the proposed control framework, and the development of potentially more efficient scheduling policies. We believe that the representation of the system operation developed in this paper, and the associated control architecture, provide the appropriate context for the analytical investigation of the performance optimizing control of flexibly automated production systems.

The detailed organization of the rest of the paper is as follows: Section 2 introduces the basic concepts and assumptions underlying the proposed modeling framework. It also outlines the overall architecture and the fundamental principles driving the development of the proposed control logic. The section concludes with the specification of a generic, real-time event-driven control scheme that can function as a template during the development of the detailed control logic for the various system functional units. This development is undertaken in Section 3, which addresses the modeling and control of each functional unit recognized by the proposed framework, providing a formal characterization of the unit uncontrolled behavior, and discussing how the arising structural and performance-oriented control requirements can be supported by results existing in the literature. It also specifies the communication protocol driving the interaction between each unit controller and its neighboring controllers in the proposed hierarchical framework. Section 4 provides a formal verification of the correctness of the proposed control logic, while Section 5 concludes the paper and proposes directions for future work.

2 A Flexibly Automated Manufacturing System Model and an Event-Driven Control Architecture

As mentioned in the previous section, there have been many different approaches to the modeling of manufacturing systems and their control architectures, each appropriate to some purpose and some set of criteria. Focusing on the logistics of material flow and process deployment, our purpose is to support assessment of control system robustness and performance, and easy development / reconfiguration of specific control system implementations. Criteria that have driven our work include:

1. the resulting modeling methodology should be as simple as possible, i.e., it should require the fewest number of entity types and relationship types possible,

2. the resulting modeling methodology should permit a high degree of model configuration and customization to accommodate a wide variety of applications,

3. the resulting models should map naturally onto real manufacturing systems, i.e., the abstraction gap between reality and the model should be as small as possible.

Manufacturing systems are often viewed as hierarchical organizations (e.g., [50, 3, 18]). For example, the 300mm fab layout illustrated in Figure 1 may be viewed as having the hierarchical structure displayed in Figure 2. Although this is not the only way to view such systems, currently it is the most conventional. As a result, it constitutes the basis of many existing control implementations, and is the view adopted in this work. Our modeling methodology represents each element in the hierarchy as a *"control domain"*, with a particular structure. Since a specific control domain may contain/supervise other control domains of similar structure, the resulting models can be characterized as *self-referencing* or *fractal* (e.g., similar to [51]).

More specifically, our work is strongly influenced by the OOSIM effort towards modeling discrete manufacturing systems [35, 5, 6]. In the context of the OOSIM framework, a manu-

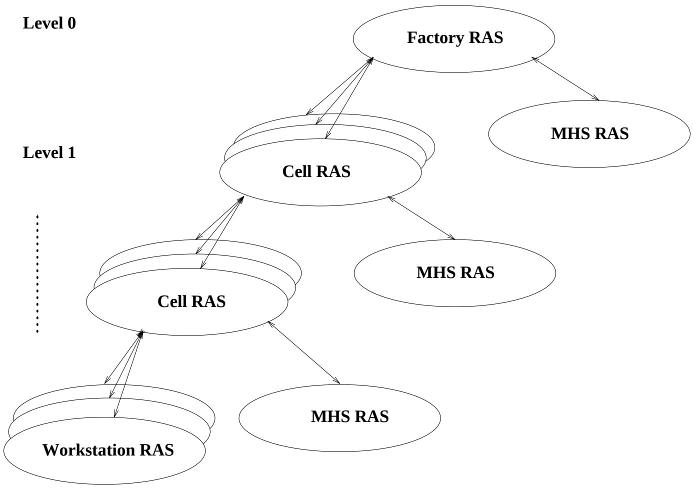

Figure 2: The overall control architecture

facturing system is viewed from two distinct perspectives: (i) *processing vs. logistics*, and (ii) *plant vs. control*. Describing the framework requires, both, definitions and assumptions about the domain of application.

The fundamental abstractions supporting the *"plant"* view of manufacturing are *material* and *location*, defined as follows:

Material is a discrete package, job, container, or other unit that individually, or with other units, is transported, stored and processed. Material has a unique identity and associated handling and processing requirements, which constitute the *process plan* associated with it.

Location is a specific, identifiable physical space where material may reside while it is being processed, transported or stored. A location has *capacity* (in terms of units or containers), and a *capability* (set of functions that it can perform, each of which may have associated with it some processing time requirements). A location has a *state*, determined by: (i) its current content (e.g., a job with a specific ID), and (ii) its status (e.g., idle, busy, failed, etc.)

Two assumptions inherent in the above definitions are that (i) the application domain is limited to situations where the key material flows may be abstracted as discrete unit movements, either individually or in batches of some sort, and (ii) there are only a finite number of locations that need to be considered. Furthermore, each location has a *"behavior"* that can be characterized analytically. Hence, a machine tool can be abstracted to a location with a particular set of behaviors, i.e., it transforms a job (material) from one state (prior to an operation) to another state (after an operation). An inspection station transforms material logically, by adding information about the condition of the material. Storage systems consist of locations that transform material in time. A robot or a conveyor is a location that may contain a number of units of material, and transforms them in terms of both spatial and temporal state.

In our approach to modeling, the *control system* responds to changes in the state of material and locations by initiating certain behaviors at locations. The control system, then, is naturally described in terms of collections of locations that are managed by a specific controller. We will refer to such collections as *cells*. A cell, in this context, typically will contain two types of locations, those that transform the physical or information properties of material, and those that transport or store the material. This distinction is captured in the *processing vs. logistics* view, in which a cell consists of two types of subsystems:

A Cell Production Subsystem, i.e., the set of locations performing physical or logical transformations of material. These locations have an associated *operation-location* map, which specifies the set of production locations that can perform a particular operation, or step in a process plan.

A Cell Material Handling Subsystem, i.e., the set of locations associated with storage of material and movement operations to transfer material between production locations, storage locations, and cell input/output locations. A *domain map* specifies the set of routes associated with each material handling device or location.

Interface points facilitating material transfer among the different controller domains (e.g., the cell production subsystem and the material handling subsystem) are characterized as *shared locations*. Hence, shared locations define the boundary of a controller domain; physically, they correspond to the AS/RS buffering capacity of the production units – i.e., cells and/or workstations – falling within the scope of the considered controller domain. From an information-flow/logical perspective, the problem of establishing effective communication / coordination among the different system controllers can be posed as the requirement for effective management of the shared locations interfacing their controller domains.

The primary controller domains recognized by the control architecture proposed in this paper, and their interaction, is depicted in Figure 2. Each of the identified cell structures is managed by a cell supervisor, which coordinates the part-flow within the cell production subsystem, i.e., the *induction* of new jobs into the cell, and their *dispatching* to the required cell production units. The physical domain of the cell supervisor is defined by the AS/RS's interfacing the cell production units to the intra-cell material handling system, as well as the AS/RS owned by the considered cell. On the other hand, the internal operation of each of the cell production units and its material handling system is controlled by dedicated cell and material handling system supervisors. These supervisors are in a subordinate relationship to the parent cell controller, in the sense that (i) their processed tasks/jobs are defined by commands generated by the parent cell supervisor, and (ii) they provide feedback information to it regarding the processing status of their task assignments. In turn, these lower-level controllers supervise the operation of the hardware devices and/or logical system components in their domain, by exchanging a series of communication signals with the controllers of these units.

Given the regularity characterizing the proposed command hierarchy, it can be concluded that the entire control architecture is completely defined by specifying

1. the generic internal control logic of
   (a) the system workstation controllers, that constitute the leaf (basic) entities in the proposed command hierarchy,
   (b) the intra-cell material handling system controller, and
   (c) the cell production system controller,
2. as well as the communication messages exchanged between the cell controller, and its subordinate production unit and material handling system controllers.

For discriminatory purposes, we shall refer to the controller supervising the entire shop-floor activity as the *factory* supervisor. It should be noticed, however, that from a functional standpoint, the control logic employed is identical to that applied to the cell supervision. Then, each of the requirements listed above is systematically addressed in the remaining sections of the paper. Moreover, the development of these generic control modules can be further systematized by noticing that the control problem underlying the operation of all three controller classes identified above can be described as the problem of supporting the concurrent execution of a sequence of externally specified tasks through logically correct and efficient allocation of the limited resources of the corresponding controller domain. Hence, for the purposes of the further analysis and controller design, it is pertinent to abstract each of the underlying controller domains to a *resource allocation system (RAS)* [47]. This abstraction facilitates the application in the undertaken controller design of more general methodological approaches proposed for DEDS analysis and control, and drives the discussion of the material developed in the subsequent sections.

According to the RAS theory developed in [47] (cf. also [15, 32]), a RAS model can be further represented as a controlled *finite state automaton (FSA)* [40]. In this FSA model, the RAS *state* is essentially the composite of the states of the locations contained in the RAS domain, while state transitions correspond to job initiations, terminations, and advancements between their processing stages, commanded by the domain controller to its subordinate controllers, which execute and confirm the corresponding task(s). Since the entire interaction of

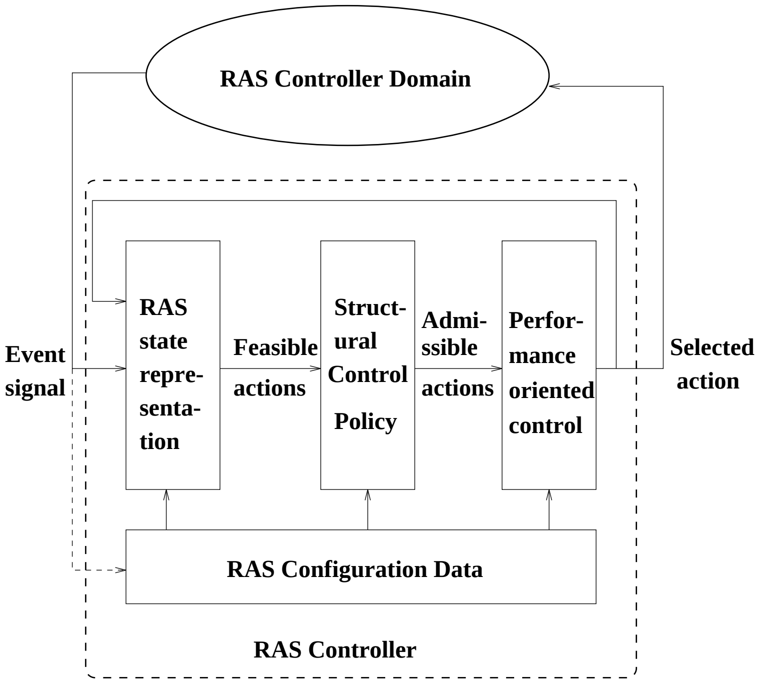

Figure 3: A generic real-time event-driven RAS control scheme

a controller with its physical environment is facilitated by a sequence of communication exchanges with its subordinate controllers, it follows that the primary events that drive the RAS models are essentially defined by these message sequences. In other words, similar to [22], the perception capabilities of a given controller regarding its underlying operational domain, its discriminatory power, and eventually, the set of its potential behaviors, are drastically affected by the information content of the signals exchanged between the controller and its subordinates, and by the applied communication protocol. As stated above, one of the main contributions of the work reported here is the definition of the messages to be exchanged between the various system controllers, and of a supporting communication protocol that will ensure logically correct and robust behavior for the entire flexibly automated manufacturing system of Figure 1.

The FSA-based representation of the operation of the controller domains defined in Figure 2 is also instrumental for structuring the underlying decision making/control process. Figure 3 illustrates the proposed modeling paradigm, in which, similar to [15, 32, 4], the controller decision-making process is triggered by the occurrence of (communication) events that either (i) command actions imposed by the unit controller on its domain, or (ii) inform the controller about changes taking place in its environment. Furthermore, all these events are reflected as changes in the *perceived* RAS state. However, contrary to the aforementioned works, in the proposed control framework, the reaction from the system controller induced by an event is structured as follows (cf. Figure 3): The event signal causes a change in the RAS state representation. The resulting state is subsequently evaluated in order to determine the action to be taken. Specifically, each RAS state is associated with a set of *feasible* actions, primarily determined by the physical capabilities of the corresponding controller domain. To ensure, however, logical correctness and robustness of the system operation, the set of feasible actions must be filtered through an applied *structural control policy (SCP)*. The SCP determines the set of *admissible* actions. The selection of a specific – if any – admissible action for immediate execution by the system domain, is the responsibility of the *performance-oriented control/scheduling policy*. This selection is reflected in the data-structure representing the system state, and the entire cycle repeats itself.

In the following section we provide detailed descriptions for the operation of the workstation, material-handling and cell/factory domains, and the associated control logic. To formalize the exposition of the behavior of each of those RAS units, we apply the formalism of *communicating Colored Petri net (CPN)* (similar to [9]). A brief introduction to CPN-based modeling and its relationship to the FSA framework mentioned above, is provided in the Appendix. We also reiterate that the emphasis of our developments is on the structural correctness and robustness of the resulting control framework. Regarding the performance-oriented control, we observe that most of the existing dispatching/scheduling policies (e.g., the dispatching rules of [38, 33], the distributed scheduling policies of [31], and the more computationally involved tracking policies of [48]) can be easily implemented in the context of the generic control logic depicted in Figure 3. However, some recently obtained results [42] indicate that the effectiveness and efficiency of these scheduling policies in the operational context and capabilities of the flexibly automated production systems is an open research issue. The resulting control problem is essentially equivalent to the notorious "job-shop scheduling" problem [34], reformulated in the operational context of the proposed framework, and will be addressed in a sequel paper.

3 The Fundamental Resource Allocation System Models and the Associated Control Logic

This section considers the operation of the three basic RAS models – workstation, material handling system (MHS), and cell – recognized by the proposed framework, and their interaction. More specifically, each of the following three subsections (i) describes the basic structure and the uncontrolled behavior of the corresponding RAS domain, (ii) provides a formal characterization of this behavior by means of the Colored Petri net modeling paradigm, (iii) discusses how the structural and performance-oriented control function of the generic scheme in Figure 3 can be supported by the existing literature results, and (iv) details the interaction between the RAS controller and its neighboring controllers in the hierarchical framework of Figure 2. For clarity purposes, we take a minimalist approach in the definition of the considered RAS structures, behavioral characteristics, and the underlying control logic. However, the proposed framework presents considerable modeling flexibility that would allow the introduction and/or modification of new behavioral aspects and control elements. In fact, part of our ongoing research effort is the customization of the generic architecture defined herein, to a control framework appropriate for Kempf's specification for the Intel mini-fab operations (cf. [27]).

3.1 The Workstation RAS

For the purposes of the work presented in this paper, the simplest configuration of the system workstation W_i is defined by an *array of buffers*, $B_i(k),\ k = 1, \ldots, C_i$, and a single *processor* $\mathcal{P}_i$, able to perform a set of *elementary operations* $\mathcal{O}_i = \{O_{i1}, \ldots, O_{i,q(i)}\}$. Moreover, in the proposed control scheme, the allocation of the workstation buffers $B_i(k)$ is essentially managed by the parent cell supervisor, while the workstation controller is only responsible for the operation of the processor $\mathcal{P}_i$. Specifically, the parent cell supervisor places – by means of the intra-cell MHS (c.f. Sections 3.2, 3.3 for details) – job instances, ζ_x, requiring the execution of some elementary operation, $O(\zeta_x) \in \mathcal{O}_i$, in the workstation free buffers, $B_i(k)$, and signals appropriately the workstation controller about the job arrival and the requested processing. The workstation controller updates its internal state representation, and if the workstation processor, $\mathcal{P}_i$, is currently idling, dispatches the job for processing. On the other hand, whenever the workstation processor, $\mathcal{P}_i$, completes the processing of a certain job instance, ζ_x, the workstation controller is notified, the message is also passed to the parent cell supervisor, and the internal system state representation is appropriately updated. Furthermore, if the set of currently unprocessed jobs is nonempty, another job instance, ζ_y, is selected for processing. Regarding the physical transfer of jobs during their processing, we can either assume that (i) the processing of job instance ζ_x, located in buffer $B_i(k)$, takes place in situ, or (ii) that ζ_x is transported to the working area of processor $\mathcal{P}_i$ through a simple material handling mechanism, and returned subsequently to buffer $B_i(k)$.

The above informal description of the workstation RAS is formalized by the Colored Petri net model of Figure 4. Notice that in in this figure, places holding tokens that correspond to messages exchanged between the workstation RAS and its neighboring – i.e. subordinate and/or parent – controllers are represented by rectangles. The place and transition interpretations, as well as the token color domains of the various places are also presented in the figure. In the basic scheme of Figure 4, the only information passed by the parent cell supervisor to the workstation controller is the job id, its location in the workstation buffer array (AS/RS), and the requested operation. In more general circumstances, this message could be extended to include the job due date and/or priority information. Naturally, in the initial net marking, the only marked place will be place P_3, with a single token indicating the processor availability.

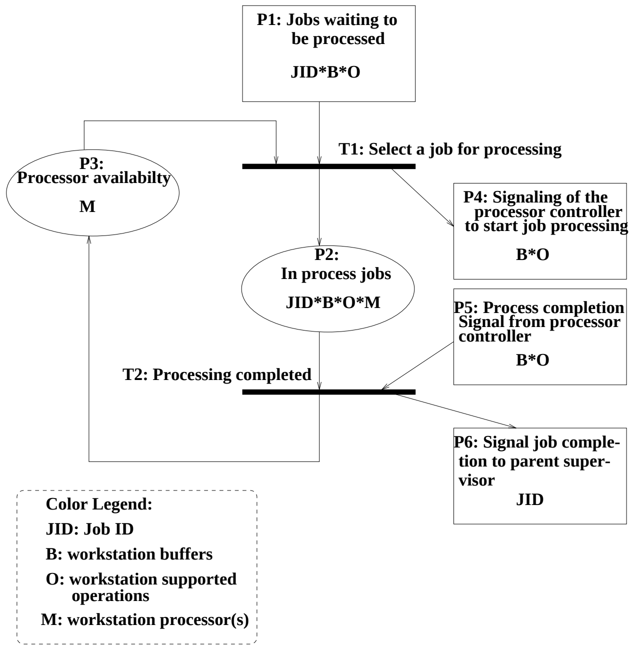

Figure 4: The Colored Petri net modeling the behavior of the workstation RAS

Regarding the required control logic, it can be seen that the resulting part flow within the workstation consists of a single processing step involving the acquisition, usage, and release of the station processor. Therefore, it is free of deadlocking problems, and the only arising control issues are performance related. Specifically, there is the need for some arbitration (scheduling) logic that will select the next job to be dispatched to the processor, when more than one job instances are waiting in the workstation task queue (place P_1). In the prevailing regime, some "single-machine scheduling" dispatching rule would be applied [34], and in the context of the CPN-modeling framework, it would be implemented as part of the firing logic for transition T_1. The employment of more complex globally optimizing scheduling algorithms for part sequencing at the workstation level, that are furthermore compatible with the event-driven nature of the proposed framework, is an interesting problem currently under investigation.

Finally, we notice that the model of Figure 4 can be easily extended to apply to workstations with more than one parallel processors, by redefining the domain of color M. Hence, for instance, this extended workstation RAS would constitute an appropriate model for the fab bays under a farm layout. Also, if the workstation parallel processors are not of identical capabilities, then, part of the underlying scheduling problem is the assignment of the processors to the executed job instances in a way that maximizes the workstation operational efficiency. For some further discussion on this problem, the reader is referred to [49].

3.2 The Material Handling System (MHS) RAS

In the proposed architecture, each material handling system (MHS) component is associated with a cell construct, and facilitates the part transfer among the cell processing units – i.e., workstations and/or higher level cell constructs – as well as the cell storage (AS/RS) facility. Hence, the shared locations interfacing a MHS component to its surrounding environment are the buffers of the production units and the cell storage facility that it supports. In the following, we shall denote the set of processing units supported by the MHS by $PU_i, \ i = 0, \ldots, n$, with the additional convention that PU_0 indicates the AS/RS facility of the parent cell. Each of these processing units is interfaced to the MHS through a set of randomly accessed buffers, to be denoted by $B_i(k), \ k = 1, \ldots, C_i$. Under this notation, the cell MHS facilitates part movements from location $B_i(k)$ to location $B_j(l)$, commanded to the MHS controller by the parent cell supervisor.

The complexity of the internal traffic control of these MHS components depends strongly on the number of transporters/vehicles available in the system and the configuration of the underlying transport network. Hence, we organize the development of this section in two parts: The first part models the *single transporter* MHS RAS, and defines the basic interaction among the MHS controller, the controller of the transport device, and the parent cell supervisor. The second part models the case of a *zone controlled Automated Guided Vehicle (AGV)* system, and also addresses issues arising from the complexity of the internal traffic control of such an environment.

3.2.1 Single-Transporter MHS RAS

This MHS RAS is appropriate for modeling the operation of the (i) material handling robot in a robotic manufacturing cell, or (ii) the vehicle in a single-loop, single-vehicle AGV system. In both cases, the MHS controller receives messages of the form $< JID, B_i(k), B_j(l) >$, requesting the transfer of job instance JID, from location $B_i(k)$, to location $B_j(l)$. If the system transporter V is idle, then, the MHS controller executes the part transfer. The transfer itself is organized in four stages: In the first stage, the transporter is commanded to travel to the source location $B_i(k)$; this stage is known as the transporter *inbound* trip. In the second stage, it is commanded to pick the part JID, situated in that location. In the third stage, the transporter is commanded to travel to the destination location, $B_j(l)$; this stage constitutes the transporter *outbound* trip. Finally, in the fourth stage, the transporter deposits the transferred material to the destination location. At the completion of this stage, the parent cell is notified that the requested transfer has been completed, and the transporter is released. Subsequently, the MHS supervisor considers if there are pending part transfer requests for which the destination buffer is already empty; pending transfers that meet this requirement are characterized as *admissible*,

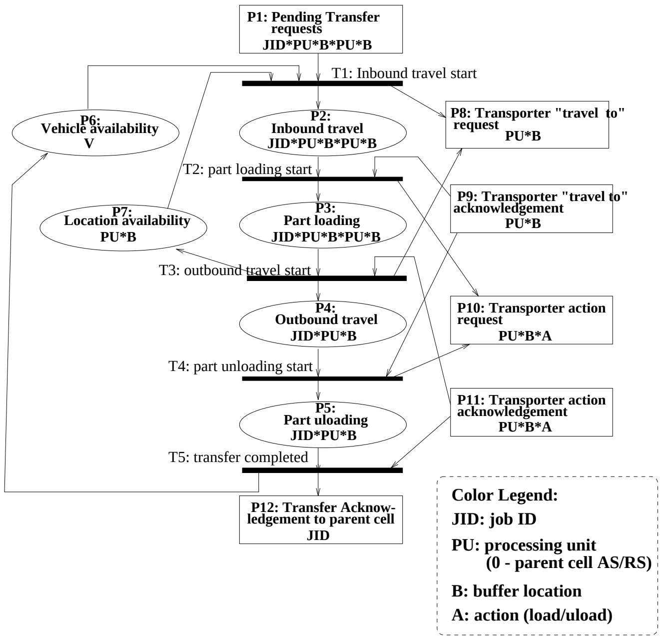

Figure 5: The Colored Petri net modeling the behavior of the Single-Transporter MHS RAS

and any of them can be selected by the MHS supervisor to be executed next. If the set of admissible transfer requests is empty, the transporter is set idle.

The operation of the single-transporter MHS RAS described above is formally represented by the Colored Petri net of Figure 5. The basic colors and the functionality of the network are self-explanatory. Similar to the net of Figure 4, places represented by rectangles are *communication* places, i.e., their tokens correspond to messages exchanged among the MHS controller, its parent cell and the transporter device controller. In the net initial marking, the only marked places are places P_6 and P_7, with the corresponding tokens denoting the transporter and location availabilities.

From a flow control standpoint, the only transition requiring additional control logic is T_1, i.e., the selection of the transfer request to be serviced next. Specifically, as stated above, in order to ensure deadlock-free operation of the MHS, the transporter must select a pending transfer for immediate service only if the destination location is free. Of course, such a policy does not address the possibility of an *induced* deadlock, which arises when the entire set of pending requests cannot meet the admissibility requirement. We shall characterize this kind of deadlocking situation as *type-1* MHS deadlock. In the proposed control framework, it is the responsibility of the parent cell supervisor to ensure that the operation of the MHS is type-1 deadlock-free; the issue is addressed in Section 3.3, which discusses the control logic underlying the operation of the cell RAS.

Notice that the proposed control scheme allows the execution of the part transfer requests, issued by the parent cell supervisor, in a sequence different from that in which they were communicated to the MHS RAS. The decoupling of these two sequences seeks to optimize the internal operation of the MHS by providing more latitude to the routing of the system transporter. However, the effective utilization of this operational flexibility requires the deployment of an additional type of control logic underlying the firing of transition T_1, i.e., a scheduling policy determining the transporter assignment to admissible pending transfer requests. In real-life environments, this control requirement is covered by some vehicle dispatching rule(s), e.g., cf. [11, 2, 16].

3.2.2 The Automated Guided Vehicle (AGV) RAS

In the context of the work presented in this paper, a zone-controlled AGV system can be defined as a set of unmanned vehicles (AGV's), $\mathcal{V}_j,\ j = 1, \ldots, v$, serving the transportation needs of a number of processing units, $PU_i,\ i = 0, 1, \ldots, n$, by traveling on a network of predetermined guidepaths. Motion on the different links of this network is *bidirectional*. However, AGV's themselves are *unidirectional*, i.e., they must do a U-turn in order to travel in the opposite direction. To avoid physical collision, *zone control* is applied, i.e., the entire network is divided into a number of areas (*zones*), and each zone can be occupied by only one AGV at a time. A natural segmentation associates one zone with each intersection and processing unit node, and each path link. In case that there are lengthy path links, they can be further segmented into a series of zones. Finally, it is typical that the system possesses one (or more) *docking* stations where idle vehicles park and potentially recharge their batteries.

The abstraction of the zone-controlled AGV operational environment to a resource allocation system has been carried out in [44]. Specifically, in the AGV RAS representation proposed in [44], the topological structure of the system guidepaths is modeled by the AGV *guidepath graph*, $G = (N, E)$. G is also assumed to be *undirected* and *strongly connected*. The arc set E of this graph is defined by the AGV zones corresponding to the path links of the network, and it constitutes the *resource set* of the underlying RAS model. The node set N corresponds to the intersections, processing units and the docking station of the actual AGV network, as well as the interconnecting points of artificially segmented links. The system processing units are modeled by a set of terminal nodes N_W connected to the rest of the network by a single arc. The presence of the AGV at a processing unit node is denoted by the occupancy of the corresponding linking arc. Furthermore, it is assumed that an AGV reaching a processing unit node $n_i \in N_W$ immediately performs a *U-turn* at that node; otherwise, the vehicle motion on the guidepath graph is unidirectional. Similarly, the docking station is modeled by another terminal node, d_s, connected to the rest of the network with a single arc. However, a vehicle entering this node, exits the guidepath network.

Processes in this RAS model correspond to transfer trips executed by vehicles $\mathcal{V}_i$, $i = 1, \ldots, v$. They are formally defined by triples, $< s,\ d,\ d_s >$, defining the trip milestone nodes, where s and d denote the *"source"* and the *"destination"* nodes, and d_s the node corresponding to the system docking station. The requirement that all vehicles can be eventually parked at the system docking station(s) is a technicality, introduced in order to establish the finiteness of the RAS processes, and the definition of a target (*"home"*) state for the system operation. However, vehicles can be assigned to a new transfer trip either while they are parked at the docking station, d_s, or while they are performing the last part of their current trip, i.e., returning from node d to node d_s. Apparently, at any point in time, the number of processes (vehicles) in the system is limited by v, i.e., the number of vehicles available in the system.

For the purposes of real-time control pursued in this paper, vehicles are assumed to always reside in one of the guidepath arcs $e \in E$ while waiting to be allocated the next leg/zone of their trip. In other words, vehicle occupation of the system intersections is only *transient*, corresponding to transitions from link to link of the system guidepath; vehicles cannot stay in an intersection while waiting for the acquisition of the next link. Hence, intersections are only facilitators of the motion from link to link, and do not constitute resources of the underlying RAS model. The only control needed for the allocation of the intersection zones, is the localized prioritization of the vehicles waiting to cross them, so that collisions in the intersections are avoided.

Requests for part transfer among the cell processing units are still initiated by the parent cell supervisor, and have the same format as in the single-transporter MHS case. Furthermore, all past remarks regarding the admissibility of the initiation of a new transfer by the MHS, the possibility of type-1 MHS deadlock, and its resolution at the parent cell RAS level, apply to this new environment. However, contrary to the single-transporter case, the assignment of vehicles to pending transfer requests, and their traveling in the system guidepath network requires additional coordination, in order (i) to avoid internal vehicle routing conflicts, and (ii) to maximize the system operational efficiencies. Hence, the augmented capabilities of the zone-controlled AGV system give rise to a new suite of resource allocation problems that must be internally resolved by the system supervisor.

From a logical control perspective, the new kind of problem arising in the AGV RAS is depicted in Figure 6. Notice that in the depicted case, all three vehicles are permanently blocked, since each of them can advance only through one of the two links occupied by the other vehicles. This situation is characterized as *type-2* MHS deadlock, and its avoidance is a logical control problem that must be addressed by the MHS supervisor. The problem of type-2 MHS deadlock avoidance has been addressed in [44], where a computationally efficient variation of *Banker's algorithm* [19], appropriate for the AGV RAS has been developed. Additionally, *performance-oriented control* problems that must be addressed in this environment concern:

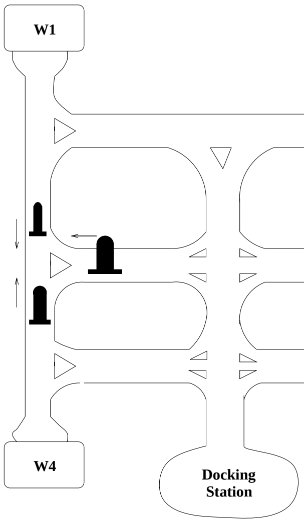

Figure 6: An example of an type-2 MHS deadlock

(i) the allocation of a free vehicle to a set of pending part transfer requests, (ii) the selection of a free vehicle to be dispatched to a pending part transfer request, (iii) the selection of the next zone to be allocated to a vehicle having completed its current travel leg, and (iv) the allocation of a freed zone to a set of vehicles contesting for it. The optimal resolution of these decisions seems to be an extremely complex problem. An introductory discussion of current practices and further literature on the issue can be found in, e.g., [20, 7, 28, 16].

Figure 7 provides a Colored Petri net-based representation of the operation of the AGV RAS discussed above, and its interaction with the parent cell and subordinate supervisors. Notice that the CPN of Figure 7 is a refined version of the CPN of Figure 5, i.e., both nets support the same functionality regarding the interfacing of the MHS controller to the parent cell supervisor, while the net of Figure 7 accounts also for the complexities arising from the internal vehicle traffic control. In order to assist with the interpretation of this net structure, we notice that transition T_1 models the initiation of new part transfers in the system, transitions T_2 and T_3 model the vehicle inbound trip, transitions T_4 and T_5 model the part loading stage, transitions T_6 and T_7 model the vehicle outbound trip, transitions T_8 and T_9 model the part unloading stage, and transitions T_{10} and T_{11} model the vehicle return trip to the docking station. As a result, the firing logic of transitions T_1, T_2, T_6 and T_{10} must be augmented by appropriate structural and performance-oriented control logic ensuring the safety and efficiency of the vehicle dispatching and routing; this logic can be derived from the references cited in the previous discussion. In the system initial marking, the only marked places are P_8, P_{10} and P_{11}, indicating the vehicle, zone and buffer availabilities, respectively.

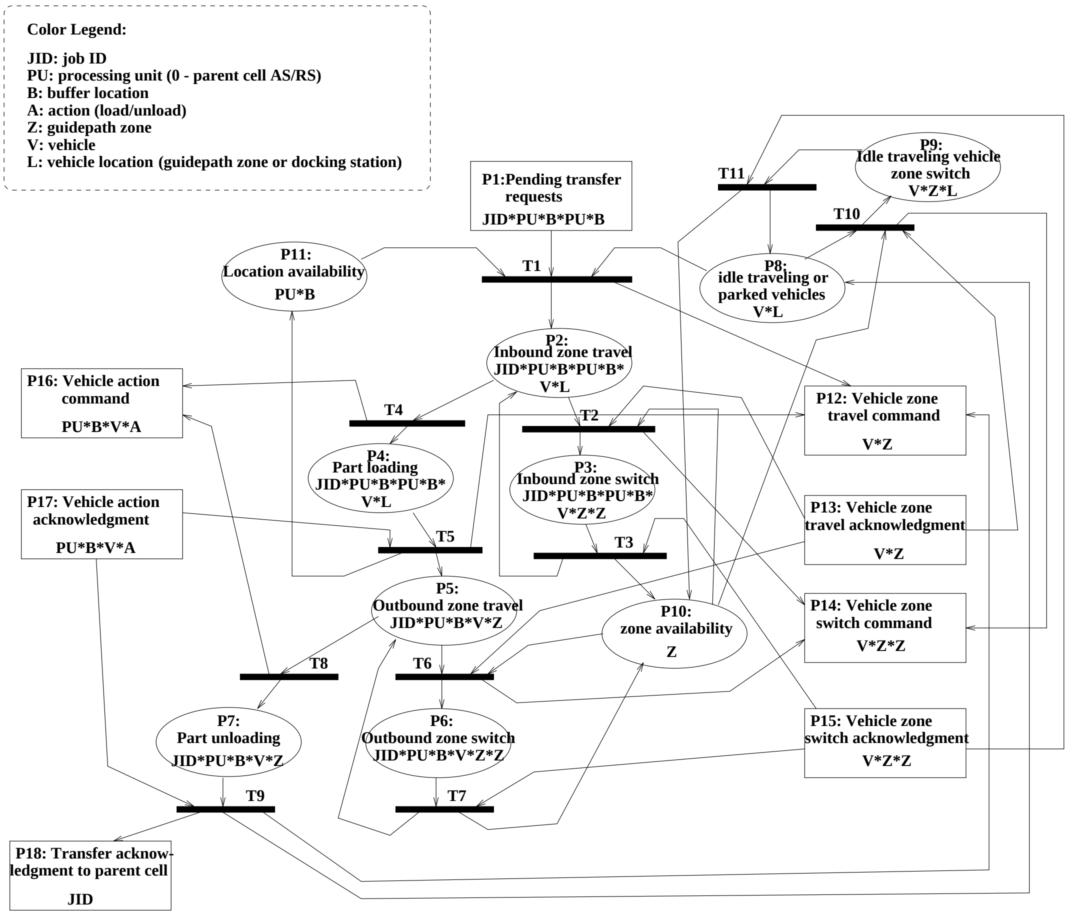

Figure 7: The Colored Petri net modeling the behavior of the AGV RAS

3.3 The Cell RAS

The basic cell structure in the proposed representation consists of an automated storage / retrieval system (AS/RS), a number of processing units able to support specific sets of physical or logical transformation operations to the parts processed through the cell, and a material handling system (MHS) facilitating the transfer of parts among the cell processing units and/or the cell AS/RS. Furthermore, it has been indicated that each cell processing unit can be a single workstation – constituting a terminal (leaf) node in the proposed architectural hierarchy – or it can be another cell construct itself. In both cases, these units are interfaced to the rest of the system through a finite random-access buffer array – a more formal abstraction of the AS/RS concept – which can be accessed by both the cell MHS and the internal MHS of the unit itself.

Parts are routed to the different processing units by matching their immediate processing requirements to the processing capabilities possessed by the different units. Specifically, each part to be processed through the cell has an associated *process plan*, i.e., a sequence of operations that must be executed for the successful completion of the part processing. In general, the operations involved in a process plan will be defined at a level of abstraction that reflects the processing capabilities of the cell processing units. Furthermore, to the extent that the processing capabilities of the various cell units might overlap, the translation of a process plan to a routing scheme will not be linear, but may allow for considerable routing flexibility, i.e., one or more operations can be supported by more than one cell processing unit. The effective exploitation of this routing flexibility is part of the performance control problem, and currently, it is an open research issue.

Controlling the part flow in a cell RAS running according to the above operational scheme, gives rise to both logical and performance related issues. In the context of the decomposition scheme proposed in this framework, the main focus is on the safe and efficient allocation of the interfacing buffering capacity of the cell processing units to the various parts processed through the cell. The management of the internal dynamics in each processing unit and the cell MHS component are transparent to the cell controller; these units are perceived only as facilitators of the required processing and transport steps. Under these assumptions, it is possible to apply the structural control theory developed in [47] in order to address the logical control requirements of this environment. Similarly, to support the cell performance oriented control, one can apply dispatching rules, like those developed in [29, 30]. Specifically, in the case that each cell processing unit is interfaced to the rest of the system through a rather large-sized buffer (e.g., a stocker or an AS/RS unit), these scheduling rules are expected to be quite efficient. If, however, each processing unit presents a rather limited buffering capacity (e.g., a robotic cell or a cluster tool), then, the expected performance of these policies needs further investigation [41, 45].

From a logical standpoint, the major concern to be addressed in the flow control of this environment, is the resolution of the type-1 MHS deadlock, discussed in Section 3.2.1. From the cell RAS viewpoint, a type-1 MHS deadlock arises every time that a subset of the cell processing units have their interfacing buffers allocated to capacity to jobs that require for their next processing step another unit in this subset. Currently, this situation is most typically resolved by transferring each job completing its current processing stage back to the cell AS/RS, before it is taken to the next required processing unit [10]. Although this approach is logically correct, it can lead to excessive (and unnecessary) material handling. An alternative approach is to try to reduce the amount of intermediate part transfers to the cell AS/RS by running an on-line *deadlock detection and recovery* algorithm. Under this approach, parts are always transferred to their next required resource, and an algorithm detecting deadlocks is run to the resulting RAS state (c.f. [46] for such an algorithm). If the algorithm indicates the occurrence of a deadlock, then one or more jobs involved in the deadlock are transferred back to the AS/RS in order to unblock the system. The selection of the job(s) to be temporarily unloaded should be based on the time or process criticality of those jobs, and also, the required material handling effort. The third deadlock resolution scheme advocated in the literature tries to prevent/avoid the occurrence of deadlocks by proactively controlling the job induction and advancement through the cell processing units. A series of provably correct *deadlock prevention/avoidance policies* that have appeared in the literature can be found in, e.g., [1, 53, 57, 47, 14, 13]. All these policies are immediately implementable in the cell RAS of the proposed control architecture. A naturally arising question is "which should be the preferable deadlock resolution strategy – i.e., detection and recovery or prevention/avoidance – for the cell RAS?" It seems that the answer strongly depends on the cost of the material handling transfers, and their relative time requirements versus the requested process times. Also, from a higher/strategic perspective, the avoidance approaches lead to a more measurable and predictable system behavior, and therefore, to a more responsive and consistent system. Detection and recovery approaches are more prone to system congestion, and the overhead of the deadlock recovery might be a significant unexpected degradation in the system performance. On the other hand, this approach might lead to higher utilization of the system resources since, in general, it allows for a higher level of concurrency in the system operation compared to the more conservative avoidance strategy. To the extent that the final answer to this question might be context specific, it remains to implement and test these different paradigms in a series of application environments. A discussion of some initial experimentation in this direction can be found in [54, 10] while a more analytical investigation is given in [43].

In the discussion below, we focus on the application of deadlock prevention / avoidance strategy to the structural control of the cell RAS, since this approach presents a greater theoretical interest regarding its support in the distributed environment of the proposed control architecture. More specifically, it was already pointed above that the cell control requirements with respect to deadlock avoidance can be supported by the immediate implementation of the structural control policies presented in, e.g., [1, 53, 57, 47, 14, 13], on the RAS defined by the interfacing buffers of the cell processing units. In this operational scheme, the cell MHS is a simple facilitator of the part transfers among the cell processing units and/or the cell AS/RS, requested by the cell controller. Hence, as long as the cell MHS executes the part transfer in the sequence commanded by the cell supervisor to the MHS controller, and the applied cell SCP is provably correct, no type-1 MHS deadlock will arise. However, this sequentialization of the part transfers fails to exploit the concurrency offered by a distributed MHS, like an AGV system, and even in the case of single-transporter MHS, it might lead to degraded performance by not allowing the optimization of the transporter routing. Hence, our methodology allows the *"reshuffling"* of the part transfer request list maintained by the MHS controller, through the implementation of appropriate scheduling logic.

From the cell controller standpoint, the proposed scheme implies that a buffer location is

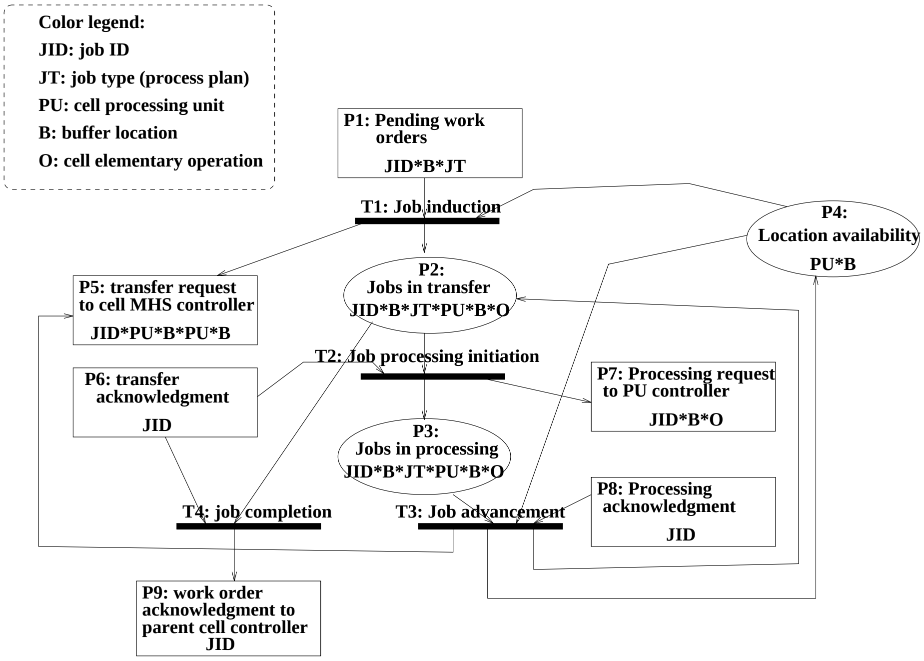

Figure 8: The Colored Petri net modeling the behavior of the cell RAS

perceived as released as long as the cell controller commands the transfer of the part currently occupying it to another location, i.e., before the MHS controller acknowledges the part transfer. As a result, this location is free to be used as the destination buffer for another part transfer, which, in fact, can be realized even before the original part transfer is completed. We characterize this effect as *structural planning*, since it allows the cell controller to determine and command an entire series of part transfers to its subordinate MHS controller. A naturally arising question is whether the control mechanism established by the proposed control logic for the cell and MHS RAS, their interacting communication protocol, and the underlying structural planning effect maintain type-1 deadlock-free operation. This question is systematically addressed in the next section.

The operation of the cell RAS described above can be formally represented by the Colored Petri net of Figure 8. The meaning of the various places and transitions, and of the basic net colors is self-explanatory. To understand the "job completion" part of the net, we notice that each process plan concludes with a fictitious operation taking place to the cell AS/RS. Hence, executing a part transfer to this unit is a signal to the cell controller that the part processing has been completed. The transitions that must be guarded by the underlying cell control logic are T_1 an T_3, respectively modeling the job induction and advancement through the cell units. Furthermore, notice that the idea of *structural planning*, discussed above, is implemented in the CPN-based representation by having transition T_3 release the resource currently held by a part upon the decision of the part advancement, i.e., before the acknowledgment of the part advancement during the corresponding firing of transition T_2. Finally, the only place marked in the initial net marking is P_4, indicating the availability of the interfacing buffers of the various cell processing units.

Concluding the discussion of the cell RAS, we notice that the cell modeling the basic structure of the shop-floor level is conventionally characterized as the *factory* RAS. From an operational perspective, the factory RAS controller presents the same functionality as any other cell controller in the RAS hierarchy. The only practical difference is that the part processing requirements will be commanded to this cell controller by a higher level production planning / demand management unit, in the company management information system (MIS) framework, rather than a parent cell supervisor (similar to [15, 32]).

4 Establishing the Logical Correctness of the Proposed Hierarchical Control Architecture

This section focuses on the issue of logical correctness of the proposed real-time shop-floor control architecture. In particular, we provide a structured argument for the following proposition:

Proposition 1 *The operation of a flexibly automated manufacturing system, controlled according to the control scheme developed in Sections 2 and 3, is* deadlock-free*.*

Proof: The validity of this proposition will be established through an *inductive* argument, with the induction *base* establishing the correctness of the internal operation of the RAS's constituting the leaf nodes in the proposed architectural structure – i.e., the workstation and the MHS RAS – and the induction *step* establishing the correctness of the operation of a cell construct, given the correctness of the internal operation of its constituent components – i.e., the cell production units and the cell MHS.

Regarding the establishment of the induction base, we simply notice the following:

1. According to Section 3.1, the uncontrolled behavior of the workstation RAS is deadlock-free, and therefore, there is no need for the application of any particular deadlock avoidance policy.

2. The internal deadlock-free operation of the MHS components is established by (i) the imposition of the requirement that a part transfer request is selected for service only if the destination location is free, and (ii) in the case of the AGV RAS, the deployment of some existing provably correct deadlock avoidance policy (e.g., [44]) that will ensure deadlock-free vehicle traffic, by controlling the vehicle dispatching and the zone allocation – i.e., the firing of transitions T_1, T_2, T_6 and T_{10} in the CPN of Figure 7.

To establish the correctness of the cell operation, given the correctness of the internal operation of its constituent components, we essentially must prove that, under the proposed control scheme, the cell operation is free of type-1 MHS deadlock. Clearly, this would be the case if the cell controller used one of the existing provably correct deadlock avoidance policies, (e.g., [47, 1, 57, 14, 13]) for the allocation of the buffering capacity interfacing the cell production units to the cell MHS, while using the cell MHS itself as a mere facilitator of the commanded part transfers. However, as it was noted in Section 3.3, the final behavior w.r.t. part flow observed in the cell environment can be more complicated due to *structural planning*, i.e., due to the fact that the cell MHS controller can "reshuffle" the pending part transfer list, in order to improve the efficiency of its internal operation. Hence, to complete the proof of Proposition 1, we must establish that the "reshuffling" incurred by the cell MHS controller will not give rise to any additional deadlock problems.

In order to prove this additional result, we first need a more formal characterization of the notion of structural planning, and the (pending transfer request) "reshuffling" concept. The situation can be systematically described as follows: Consider a cell RAS controlled by a correct deadlock avoidance policy, and with constituent components of correct internal behavior. Also, at a certain point in time, let S_1 denote the the cell RAS state w.r.t. the allocation of the buffering capacity of its processing units to the various job instances. At that point, the cell controller exercises structural planning, commanding the advancement of a job set $\mathcal{J} = \{j_1, \ldots, j_k\}$ to the processing units supporting the job next processing stages. Hence, the current state representation of the cell controller is state S_2, resulting from the advancement of each job $j_j \in \mathcal{J}$ to its new location. Apparently, since the cell controller employs a correct deadlock avoidance policy in its decision making, (i) there exists at least one feasible transfer sequence, $\mathcal{S} =< j_{[1]}, j_{[2]}, \ldots, j_{[k]} >$, taking the cell RAS from state S_1 to state S_2 – specifically, this is the sequence in which the part transfers were commanded to the MHS controller – and furthermore, (ii) state S_2 is safe, in the sense that there exists at least one job advancement sequence that can meet all the remaining job processing requirements without giving rise to a deadlock. However, under the "reshuffling" assumption, the MHS controller will attempt to execute the part transfer requirements in set $\mathcal{J}$ by selecting jobs from this set in any arbitrary order obeying its internal performance optimization logic, with the only additional restriction that a job can be selected for transfer only if its destination location is free. We claim that:

Lemma 1 *The operation of the cell MHS, under the reshuffling scheme described above, is type-1 deadlock-free.*

Proof of Lemma 1: Lemma 1 is proved by an additional inductive argument, based on the size of set $\mathcal{J}$. Specifically, for the base case of $|\mathcal{J}| = 1$, the only option of the MHS controller is sequence $\mathcal{S}$, and therefore, given the above discussion, the system operation is deadlock-free.

Suppose that Lemma 1 is correct for all job sets with $|\mathcal{J}| \leq k$. We shall show that it also holds true for all job sets with $|\mathcal{J}'| = k+1$. To see this, let $j_{j^*} \in \mathcal{J}'$ denote the first job selected for transfer by the MHS controller. According to the operational assumptions stated above, this transfer is feasible, in the sense that the destination location for job j_{i^*} is free; let S_3 denote the state resulting from S_1 by the advancement of job j_{i^*} to its destination location. Then, S_3 differs from S_2 only w.r.t. the positioning of jobs $j_j \in \mathcal{J}'' \equiv \mathcal{J}' \backslash \{j_{i^*}\}$. Notice that $|\mathcal{J}''| = k$. Furthermore, it is easy to see that the ordering $\mathcal{S}''$ of the job set $\mathcal{J}''$, obtained from the ordering $\mathcal{S}'$ of the original set $\mathcal{J}'$ by elimination of job instance j_{i^*}, is a feasible job advancement sequence leading from state S_3 to S_2. Hence, according to the induction hypothesis, any reshuffling of the job set $\mathcal{J}''$ according to the assumptions stated above, leads to (type-1) deadlock-free behavior. Since job j_{i^*} was selected arbitrarily, this establishes the deadlock-free property of the reshuffling of the entire job set $\mathcal{J}'$, and concludes the proof of Lemma 1, as well as the proof of Proposition 1. $\Box$

5 Conclusions and Future Work

This paper has developed a distributed real-time control architecture for contemporary flexibly automated production systems. The modeling assumptions regarding the system structure and functionality, underlying the design of the proposed control logic, have been driven by the structure of the emerging 300mm semiconductor manufacturing fab, one of the most extensively automated environments in contemporary manufacturing. The controller design itself meets a number of objectives that differentiate it from past similar efforts. Specifically, the major concern in the proposed design has been the distribution of the control function to the constituent components of the system shop-floor architecture, while maintaining the logical correctness and efficiency of the system behavior. An additional guideline observed by the undertaken work was that the resulting solution must be easily implementable in the actual system shop-floor, and therefore, it should be aligned to and augment current practices in these environments. From a theoretical perspective, the proposed design has employed, integrated and extended a series of theoretical results emerging in the burgeoning field of Discrete Event Dynamical Systems. Finally, in addition to providing a systematic structure for the effective deployment of real-time shop-floor controllers for flexibly automated environments, the proposed framework can also provide the basis for further analytical studies aiming at the performance evaluation of existing scheduling policies in the considered class of systems, as well as the eventual design of potentially more efficient performance control policies.

Indeed, the systematic study of the problem of efficient performance control in the context of the proposed architectural framework is part of our research agenda. An additional issue that needs further exploration is the development of the control logic that will allow the extension of the system nominal behavior described in this paper to systematically accommodate operational contingencies, like resource outages and the processing of expedient jobs. Some initial results on this problem regarding a single-level (non-hierarchical) RAS structure can be found in [42]. Finally, the system architecture and the underlying control logic developed in this paper have been implemented in the *Virtual Factory Lab* of the *School of Industrial & Systems Engineering*, at the *Georgia Institute of Technology*. The developed platform integrates table-top experimental manufacturing equipment, as well as virtual manufacturing units implemented in an array of simulation software. It functions as an experimental platform for the research program outlined above, but also as an educational component for the introduction of real-time manufacturing system control to senior-level undergraduate and graduate engineering students.

A Appendix: An Introduction to Colored Petri Net-based Modeling

Colored Petri nets (CPN's) have been proposed in the literature as an extension of the ordinary Petri net (OPN) modeling framework, leading to more powerful and compact system representations [24]. From a modeling perspective, both, CPN and OPN frameworks possess the same expressive power [24]. In this appendix, we present the fundamental concepts behind the CPN modeling framework that facilitate the interpretation of the CPN-based RAS models of Section 3. For a more extensive discussion on the CPN modeling framework, its applicability, and the available analysis techniques, the reader is referred to [24].

According to [24], a (non-hierarchical) *Colored Petri net* is formally defined by a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, where:

1. Σ is a finite set of non-empty types, called the *color sets*,
2. P is a finite set of *places*,
3. T is a finite set of *transitions*,
4. A is a finite set of *arcs*, such that $P \cap T = P \cap A = T \cap A = \emptyset$
5. N is a *node function*, defined from A into $P \times T \cup T \times P$,
6. C is a *color function*, defined from P into Σ,
7. G is a *guard function*, defined from T into expressions such that

$$\forall t \in T: \ [\mathtt{Type}(G(t)) = \mathtt{Boolean}] \wedge [\mathtt{Type}(\mathtt{Var}(G(t))) \subseteq \Sigma]$$

8. E is an *arc expression function*, defined from A into expressions such that

$$\forall a \in A: \ [\mathtt{Type}(E(a)) = C(p(a))_{MS}] \wedge [\mathtt{Type}(\mathtt{Var}(E(a))) \subseteq \Sigma]$$

and with $p(a)$ being a place of $N(a)$,

9. I is an *initialization function*, defined from P into closed expressions such that

$$\forall p \in P: \ \mathtt{Type}(I(p)) = C(p)_{MS}$$

Furthermore, in the above definition, $C(p)_{MS}$ denotes the *set of all multisets* defined over the set $C(p)$, the color set associated with place p. The concept of a *multiset* m over a non-empty set S is itself formally defined as a function $m \in [S \rightarrow \mathcal{N}]$, where $\mathcal{N}$ represents the set of natural numbers. Usually, m is represented by the formal sum $\sum_{s \in S} m(s)\,\grave{}\,s$.

The concepts of *places, transitions* and *arcs* in the above definition have the same significance, and they are represented by the same semantics, as in the ordinary Petri net framework. However, tokens occupying any CPN place, p, are characterized by/possess additional attributes, which are formally expressed by the *color set*, $C(p)$, associated with place p. Formally, a color set is a *type set*, and associates with the tokens of place p a type, in the same way that types are associated to variables in any structured programming language. In particular, types can be defined over any set of primitives, and, similar to the programming language case, composite types can be defined from simpler types; of particular interest to this work is the *product* operation, $T_1 * T_2$, over two (or more) type sets T_1 and T_2, which essentially defines a *record* type, with two *fields* of type T_1 and T_2. An additional extension of the CPN structure that significantly augments its modeling power compared to its OPN counterpart, is the potential association of a *guard* function, $G(t)$, to each net transition, $t \in T$. This function is essentially an additional logical condition (*predicate*) that must be satisfied in order for transition t to be fireable, and renders the CPN modeling and computational power equivalent to that of a *Turing Machine* [21].

More specifically, the logic behind the firing of a net transition $t \in T$ can be formally stated as follows: First, the concept of *token element* is defined as a pair (p, c), with $p \in P$ and $c \in C(p)$. The set of all token elements is denoted by TE. Then, a net *marking*, M, is defined as a multiset over set TE. In particular, the *initial net marking*, M_0, is obtained from function I, by evaluating the expressions:

$$\forall (p, c) \in TE: \ M_0(p, c) = (I(p))(c)$$

Furthermore, a *binding* b of transition t is defined as a function on the set of variables associated with transition t, $\mathtt{Var}(t)$, such that: (i) $\forall v \in \mathtt{Var}(t): \ b(v) \in \mathtt{Type}(v)$, and (ii) $G(t) < b >= \mathtt{TRUE}$.

Finally, in marking M, transition t is said to be *enabled* with respect to binding b, if and only if (iff):

$$\forall p \in P: \ E(p,t) < b > \leq \ M(p)$$

where $M(p)$ is the multiset indicating the token content of place p in marking M, and comparison ($\leq$) is applied elementwise.

When a transition is enabled, it can *fire* (or *occur*), changing the current marking M to marking M', defined by:

$$\forall p \in P: \ M'(p) = M(p) - E(p,t) < b > +E(t,p) < b >$$

In the CPN parlance, marking M' is said to be *directly reachable* from marking M by the occurrence of transition t, and it is denoted by: $M[t > M'$. A sequence of transition firings such that $M_1[t_1 > M_2[t_2 > M_3 \ldots M_{n-1}[t_{n-1} > M_n$ is characterized as a *finite occurrence sequence* between markings M_1 and M_n, and it is more concisely denoted by $M_1[t_1 t_2 \ldots t_{n-1} > M_n$. Furthermore, we say that marking M' is *reachable* from marking M iff there exists a finite occurrence sequence leading from M to M'.

Similar to the OPN case, the marking of a CPN provides a formal and concise representation of the system *state*, while the set of fireable transitions defines the set of *events/actions* that can be executed in that state. Specifically, in the RAS modeling framework developed in this paper, the *event feasibility* is expressed by the token availability requirements associated with each transition, expressed primarily through functions E, whereas the imposed *control logic* is implemented through the transition guard functions, G.

Finally, communication between the various system controllers is modeled in our distributed system representation through the employment of *communication* or *signal* places in the proposed CPN structures (similar to [9]). Specifically, communication places appear in pairs of *sender* and *receiver*, each contained in a separate net instantiation, and they model the operation of a *unidirectional* communication channel among the corresponding controllers. Hence, every time that a token is placed in the sender communication place, it is automatically consumed by being transferred to the corresponding receiver place. The color content of the token should be interpreted as the message content / parameters passed from the sender to the receiver controller.

Acknowledgments

The authors would like to acknowledge the W. M. Keck Foundation and Ford Motor Company for their support of the underlying research effort.

References

[1] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Trans. on Robotics and Automation*, 6:724–734, 1990.

[2] J. J. Bartholdi and L. K. Platzman. Decentralized control of automated guided vehicles on a simple loop. *IIE Trans.*, 21:76–81, 1989.

[3] A. Bauer, R. Bowden, J. Browne, J. Duggan, and G. Lyons. *Shop Floor Control Systems: From Design to Implementation.* Chapman & Hall, 1991.

[4] J. A. Behuniak, I. A. Ahamad, and A. M. Courtright. *Production Software That Works : A Guide to Concurrent Development of Realtime Manufacturing Systems.* Digital Press, Maynard, MA, 1992.

[5] D. A. Bodner. *Real-Time Control Approaches to Deadlock Management in Automated Manufacturing Systems.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1996.

[6] D. A. Bodner and S. A. Reveliotis. Virtual factories: An object-oriented simulation-based framework for real-time fms control. In *ETFA'97*, pages 208–213. IEEE, 1997.

[7] C. G. Co and J. M A. Tanchoco. A review of research on agvs vehicle management. Technical report, School of Industrial Eng., Purdue Univ., 1990.

[8] D. Connors, G. Feigin, and D. Yao. Scheduling semiconductor lines using a fluid network model. *IEEE Transactions on Robotics & Automation*, 10:88–98, 1994.

[9] R. Cossins and P. Ferreira. Celeritas: A coloured petri net approach to simulation and control of flexible manufacturing systems. *Int. J.Prod. Res.*, 30:1925–1956, 1992.

[10] C. Cunningham, R. Wright, K. Benhayoune, E. Campbell, V. Swaminathan, and R. White. 300mm factory layout and automated material handling system analysis. In *Proceedings of the Autosimulations Symposium '99*, pages –. Autosimulations, 1999.

[11] P. J. Egbelu and J. M. A. Tanchoco. Characterization of automatic guided vehicle dispatching rules. *IJPR*, 22:359–374, 1984.

[12] J. Ezpeleta and J. M. Colom. Automatic synthesis of colored petri nets for the control of fms. *IEEE Trans. on R&A*, 13:327–337, 1997.

[13] J. Ezpeleta, J. M. Colom, and J. Martinez. A petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics & Automation*, 11:173–184, 1995.

[14] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Transactions on Robotics & Automation*, 13:347–363, 1997.

[15] M. P. Fanti, B. Maione, G. Piscitelli, and B. Turchiano. System approach to design generic software for real-time control of flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 26(2):190–202, 1996.

[16] T. Ganesharajah, N. G. Hall, and C. Sriskandarajah. Design and operational issues in agv-served manufacturing systems. *Annals of OR*, 76:109–154, 1998.

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

[18] S. B. Gershwin. *Manufacturing Systems Engineering.* PTR Prentice Hall, Englewood Cliffs, N.J., 1994.

[19] A. N. Habermann. Prevention of system deadlocks. *Comm. ACM*, 12:373–377, 1969.

[20] T. J. Hodgson, R. E. King, S. K. Monteith, and S. R. Schultz. Developing control rules for an agvs using markov decision processes. *Material Flow*, 4:85–96, 1987.

[21] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

[22] Z. Idelmerfaa and J. Richard. Cim systems modeling for control system re-usability. *Int. J. Computer Integrated Manufacturing*, 11:195–204, 1998.

[23] M. A. Jafari. An architecture for a shop-floor controller using colored petri nets. *The Intl. Jrnl. of FMS*, 4:159–181, 1992.

[24] K. Jensen. *Coloured Petri Nets: Volume 1.* Springer, 2nd ed., NY,NY, 1997.

[25] A. T. Jones and C. R. McLean. A proposed hierarchical control model for automated manufacturing systems. *Journal of Manufacturing Systems*, 5:15–25, 1986.

[26] S. B. Joshi, E. G. Mettala, J. S. Smith, and R. A. Wysk. Formal models for control of flexible manufacturing cells : Physical and system model. *IEEE Transactions on Robotics & Automation*, 11(4):558–570, 1995.

[27] K. Kempf. Intel five-machine six step mini-fab description. Technical Report http://www.eas.asu.edu/~aar/research/intel/papers/fabspec.html, Intel, 1994.

[28] C. M. Klein and J. Kim. Agv dispatching. *IJPR*, 34:95–110, 1996.

[29] P. R. Kumar. Scheduling manufacturing systems of re-entrant lines. In D. D. Yao, editor, *Stochastic Modeling and Analysis of Manufacturing Systems*, pages 325–360. Springer-Verlag, 1994.

[30] P. R. Kumar. Scheduling semiconductor manufacturing plants. *IEEE Control Systems Magazine*, 14–6:33–40, 1994.

[31] S. H. Lu and P. R. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, 36:1406–1416, 1991.

[32] G. Maione and G. Piscitelli. Object-oriented design of the control software for a flexible manufacturing system. *Int. J. Computer Integrated Manufacturing*, 12:1–14, 1999.

[33] T. McCutchen and C-Y Lee. An analysis of dispatching rules in a semiconductor wafer fabrication environment. *Journal of Electronics Manufacturing*, 5:165–174, 1995.

[34] S. Nahmias. *Production and Operations Analysis – 3rd ed.* IRWIN, Chicago, IL, 1997.

[35] S. Narayanan, D. A. Bodner, U. Sreekanth, T. Govindaraj, L. F. McGinnis, and C. M. Mitchell. Modeling control decisions in manufacturing systems simulation using objects. In *Proceedings of the 1994 IEEE International Conference on Systems, Man, & Cybernetics*, volume 2, pages 1392–1397. IEEE, 1994.

[36] A. W. Naylor and R. A. Volz. Design of integrated manufacturing system control software. *IEEE Transactions on Systems, Man, & Cybernetics*, 17:881–897, 1987.

[37] I. M. Ovacik and R. Uzsoy. *Decomposition Methods for Complex Factory Scheduling Problems.* Kluwer Academic Publ., 1997.

[38] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25:45–61, 1977.

[39] N. Patel. Using simulation to evaluate complex control logic for lot delivery. In *Proc. of AutoSimulations Symposium '99.* AutoSimulations, 1999.

[40] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[41] S. A. Reveliotis. The instability of the lbfs scheduling policy in capacitated re-entrant lines. In *ACC'98.* AACC, 1998.

[42] S. A. Reveliotis. Accommodating fms operational contingencies through routing flexibility. *IEEE Trans. on R&A*, 15:3–19, 1999.

[43] S. A. Reveliotis. An analytical investigation of the deadlock avoidance vs. detection & recovery problem in buffer-space allocation of flexibly automated production systems. *IEEE Trans. on SMC: Part B*, 30:–, 2000.

[44] S. A. Reveliotis. Conflict resolution in agv systems. *IIE Trans.*, 32(7):647–659, 2000.

[45] S. A. Reveliotis. The destabilizing effect of blocking due to finite buffering capacity in multi-class queueing networks. *IEEE Transactions on Automatic Control*, 45:585–588, 2000.

[46] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Trans. on Automatic Control*, 42:1344–1357, 1997.

[47] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Structural control of large-scale flexibly automated manufacturing systems. In C. T. Leondes, editor, *Computer Aided and Integrated Manufacturing Systems: Techniques and Applications*. Gordon & Breach, 1998.

[48] A. Sharifnia. Stability and performance of a simple distributed tracking policy for production control of manufacturing systems. *IEEE Transactions on Automatic Control*, 40:1109–1113, 1995.

[49] A. Sharifnia. Instability of the join-the-shortest-queue and fcfs policies in queueing systems and their stabilization. *Operations Research*, 45:309–314, 1997.

[50] K. E. Stecke. Design, planning, scheduling and control problems of flexible manufacturing systems. *Annals of Operations Research*, 3, 1985.

[51] 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 Transactions on Systems, Man, & Cybernetics*, 22(3):564–567, 1992.

[52] R. Uzsoy, C-Y Lee, and L. A. Martin-Vega. A review of production planning and scheduling models in the semiconductor industry part ii: Shop-floor control. *IIE Trans.*, 26:44–55, 1994.

[53] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics & Automation*, 6:713–722, 1990.

[54] R. A. Wysk, N. S. Yang, and S. Joshi. Resolution of deadlocks in flexible manufacturing systems: Avoidance and recovery approaches. *Journal of Manufacturing Systems*, 13:128–138, 1994.

[55] B. P. Zeigler. *Theory of Modeling and Simulation.* Krieger, FL, 1976.

[56] B. P. Zeigler. *Multifaceted Modeling and Discrete Event Simulation.* Academic, London, UK, 1984.

[57] M. Zhou and F. Dicesare. Parallel and sequential mutual exclusions for petri net modeling of manufacturing systems with shared resources. *IEEE Transactions on Robotics & Automation*, 7:515–527, 1991.

[58] J. Zoghby. Improving planning decisions through autosched ap. In *Proc. of Autosimulations Symposium'99*, pages –. Autosimulations, 1999.