PRIAM: Polite Rescheduler for Intelligent Automated Manufacturing

Thomas K. Tsukada, *Member, IEEE,* and Kang G. Shin, *Fellow, IEEE*

Abstract—**This paper considers the problem of rescheduling in a decentralized manufacturing system. Flexible manufacturing systems must be able to respond to unexpected disruptions, including schedule disruptions. However, when a cell controller in a decentralized system responds to a disruption, it may disrupt some other cell, because the actions taken at one cell may have some consequence at another cell. In the approach we propose, a controller at a disrupted cell tries to respond in a way which is likely to be least disruptive to other cells, through negotiation with controllers at other cells. This approach, which we call "polite replanning," has the advantage of retaining much of any original distributed plan, while avoiding wide propagation of the disruption through the rest of the system. We apply this concept to the domain of distributed factory rescheduling, and describe PRIAM (*polite rescheduler for intelligent automated manufacturing*), a "polite" rescheduling architecture which is currently under development. Simulation results show that the use of negotiation in "polite" rescheduling prevents the wide propagation of disruption from an initial local disruption.**

I. Introduction

DECENTRALIZATION is an important concept and reality in computer-integrated manufacturing. There has been a great deal of research into new control models for manufacturing system organization, emphasizing organizational flexibility, and modularity and simplicity of design [10], [6], [20]. While decentralization offers many advantages, such as greater fault-tolerance and exploitation of parallelism, it also poses new problems, including the problem of coordination. When interacting components of a manufacturing system have different controllers, these controllers must coordinate their actions so that each is allowed to achieve its goals unhindered.

One problem in which coordination is important is that of recovery from disruptions. Manufacturing systems routinely experience disruptions (unexpected disruptive events), such as machine failures and resource unavailability. Flexible manufacturing systems must be able to recover from such disruptions efficiently. In a distributed manufacturing system, intelligent run-time coordination is necessary for such flexibility, because actions taken at one part of the system can adversely affect other parts of the system. If a disruption occurs at one cell of the system, that cell's controller must take some action. However, this action may result in a disruption at another cell. In such a way, a disruption at one cell may propagate through the whole system. A good example of this type of propagation of disruptions can be seen in the rescheduling (schedule revision) of a cellular manufacturing system. If one cell suffers a disruption, the jobs at that cell may have to be rescheduled. However, because of precedence constraints or resource sharing, this rescheduling may disrupt the schedules of other cells, by the late arrival of parts or resource unavailability.

In this domain of factory scheduling, we address the problem of recovering from a disruption in a distributed plan, specifically a problem of how an individual agent handles the recovery from such a disruption. The disruption of a plan may be costly, not only because of the recovery task itself. When a factory schedule is disrupted, for example, commitments based upon the original schedule, dealing with material transport or personnel, may have to be reorganized. At worst, guarantees made to a customer about delivery times may be violated. Thus, when unexpected events can occur, one goal is to handle disruptions with as little change to existing schedules as possible. In our approach, which we call "polite replanning," the affected agent attempts to solve locally the problem of finding a response to the disruption, in such a way that it will be least disruptive to other agents. This approach avoids the costs of making the local problem into a global problem, while it remains in a cooperative framework by attempting to isolate the effects of the disruption. More importantly, by avoiding complete replanning of the system, and by attempting to isolate disruptions, it attempts to retain as much of the distributed plan as possible.

In order to find a response which is least disruptive to other agents, the affected agent must have some information about how its actions will affect those other agents. Because an individual agent does not have global knowledge about the system, some form of negotiation is needed as a means of gathering information about other agents. Negotiation is a well-studied concept in the field of *distributed artificial intelligence* (DAI). The disrupted agent searches for the least disruptive response by negotiating with other agents which could possibly be disrupted by its actions.

In this paper, we explore the issues of disruption and coordinated recovery in the domain of distributed job shop rescheduling. While finding a good schedule is often a very hard problem, handling the disruption of an already existing schedule also presents an important problem. We consider "polite rescheduling," the application of polite replanning to the problem of recovering from such a disruption in a

Manuscript received September 19, 1994; revised September 14, 1995. This work was supported in part by the NSF under Grant IRI-9209031 and Grant DDM-9313222.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor MI 48109 USA.

Publisher Item Identifier S 1042-296X(96)02541-4.

distributed group of manufacturing cells, and describe PRIAM (*polite rescheduler for intelligent automated manufacturing*), a rescheduling architecture for decentralized manufacturing.

This paper is organized as follows. Section II provides some background on the DAI field and AI research in scheduling, and explains how polite replanning is related to this research. Section III presents a formal model and an outline of our polite replanning ideas. Section IV describes the application of our ideas to the job shop rescheduling domain, and describes the PRIAM architecture. Section V presents simulation results of our work. Section VI presents a summary of this work.

II. Distributed Problem Solving and Polite Replanning

A. DAI and Intelligent Scheduling

Polite rescheduling attempts to find solution methods for schedule revision from the field of DAI, which has become an increasingly important field of AI during the last 15 years [4], [9]. *Distributed problem solving* (DPS), a branch of DAI, studies how several agents cooperate to solve a common problem. The two most important issues in DPS are problem decomposition and task organization. These issues are addressed in the important early work in DAI. The *contract net* protocol of Davis and Smith [5] introduced the concept of negotiation protocols for task decomposition, while Durfee and others have explored the problem of organization among agents with interacting goals in a dynamic environment [7].

In the past dozen years, starting with the work of Fox [8] and Smith [17], there has been much AI research in the field of factory scheduling, much of which involves reasoning about schedule constraints and analyzing resource capacity. Recently, there has been some work on distributed approaches to scheduling, including Parunak's distributed run-time elaboration of a high-level coarse schedule [15], Sycara's work on distributed job-shop resource allocation [18], and Sen's work on distributed meeting scheduling [16].

Our main concern, schedule revision in a decentralized manufacturing environment, is not directly addressed in these works. It is loosely related to the problem of backtracking in DPS, as discussed in [18] and [13], and the problem of avoiding "ripple effect" propagation of plan changes is briefly discussed in Kambhampati's work on distributed "hybrid" planning [11]. There has been important work on schedule revision in a single-agent environment, including work by Minton [12] and Zweben [21] in the domain of space applications.

B. The Polite Replanning Approach

The problems which we are investigating are those in which one agent in a system of loosely-coupled agents needs to recover from a local disruption. By "loosely-coupled," we mean that the agents are not necessarily cooperating closely on any particular task, but they may affect one another, and in particular, the actions of one may hinder another from achieving its goals. A cellular manufacturing system is a good example of a loosely-coupled system, as cells affect one another through resource sharing and part delivery, but work largely in isolation on their own tasks. By a "local disruption," we mean an unexpected event which is recognized only by one agent, and the immediate effects of which concern only that agent.

In the framework provided by the contract net and similar models, the problem decomposition and task organization are as follows. The task of finding a "least disruptive" solution is centralized at the disrupted agent, because this agent is the only one with knowledge of the disruption and its immediate effects. However, this disrupted agent may lack the knowledge required to find a good solution, because such a solution depends upon the states of other agents. Thus, the responsibility for negotiation lies with this agent. As in the contract net protocol, it "contracts" other potentially affected agents to determine how disruptive a proposed solution may be. If a proposed solution is accepted, then the disrupted agent leaves as problems for other agents how to respond to the effects of this proposed solution.

Most DPS approaches involve the construction of plans or problem-solutions through general cooperation and exchange of plan and goal information. Our polite replanning approach is distinguished by addressing the situation in which agents already have plans, and in which one agent must revise its plan in the context of other agents' plans. Obviously, if disruptions never occur, this style of cooperation is unnecessary. Likewise, if disruptions occur so frequently that agents are constantly revising their plans, and are revising plans simultaneously, then some form of reactive planning [1] or global supervision is better suited to the problem than cooperation about constant plan revisions. Our approach, however, may be appropriate for problems in between these two poles, in which disruptions may occur and thus must be dealt with, but in which these disruptions are infrequent enough that having an initial plan is still useful. Likewise, while plan revision is an unimportant problem if plan generation is very easy, our approach for avoiding plan disruption is appropriate when plan generation is not easy, as in job shop scheduling. Thus, for both these reasons, we believe that our approach is appropriate for the problem of rescheduling in a decentralized manufacturing system.

III. Formal Model

In order to discuss concepts dealing with disruption propagation and recovery actions in a multiagent environment, we propose the following model. Let C be the set of n cells. For each cell i, there is a set $S_i = \{s_{i1}, \ldots, s_{im}\}$ of states. In the job shop scheduling domain, for example, this set would be the set of all schedules. For each cell i, there is a set $D_i \subset S_i$, which is the set of *disrupted* states. A state in S_i which is not in D_i is a *safe* state. A disruption is an event which puts a cell into a disrupted state. A disrupted state in job shop scheduling would be an infeasible schedule. Let set $A_i = \{a_{i1}, \ldots, a_{ip}\}$ be the set of actions which can be taken at cell i. These sets of states and actions need not be explicitly enumerated; rather, they represent search spaces through which proper actions are found, as described below.

For each pair of cells i and k, there is a transition function $\mathcal{F}_{ik}: A_i \times S_k \rightarrow S_k$ which describes how an action taken by cell i affects cell k. Thus, if cell k is in state s and cell i takes action a, then cell k will be put into state $\mathcal{F}_{ik}(a, s)$. If cell i takes an action which puts cell j into a disrupted state, we say cell j has been *disrupted* by cell i. If cell i is in state s, and then takes action a, it will itself be put into state $\mathcal{F}_{ii}(a, s)$.

For cell i and state $s \in D_i$, let $\mathrm{RA}_i(s) = \{a: a \in A_i, \mathcal{F}_{ii}(a, s) \in S_i - D_i\}$ be the set of recovery actions for cell i while it is in disrupted state s. For cell i and state $s \in D_i$, let $\mathrm{GA}_i(s) = \{a: a \in \mathrm{RA}_i(s), \mathcal{F}_{ij}(a, s') = s' \text{ for all } s' \in S_j, j \in C, j \neq i\}$ be the set of *guaranteed-local* recovery actions. A guaranteed-local recovery action will not affect any other cell, regardless of what state it is in. For cell i, let $T_i = \{s: s \in D_i, \mathrm{GA}_i(s) \neq \emptyset\}$ be the set of *semisafe* states for cell i. A semisafe state is a disrupted state from which a cell can recover with a guaranteed-local recovery action. For cell i and action $a \in A_i$, let $M_i(a) = \{j: j \in C, j \neq i, \exists s \in S_j, \mathcal{F}_{ij}(a, s) \in D_j\}$ be the set of remote cells which could possibly be disrupted by cell i taking action a. Clearly, if $a \in \mathrm{GA}_i(s)$ for some cell i and action a, then $M_i(a) = \emptyset$.

Disruptions can be classified by how much they result in propagation of disruptions. Consider a system state x in which cell i is in disrupted state $x_i = s_d \in D_i$, and in which every other cell is in a safe or semisafe state. We call the disruption which caused cell i to be disrupted a disruption of *type 0* if $\mathrm{GA}_i(s_d) \neq \emptyset$. By taking an action $a \in \mathrm{GA}_i$, cell i can recovery from a type 0 disruption without disrupting other cells. Likewise, we call the disruption a disruption of *type* l, where $l > 0$, if there is an action $a \in \mathrm{RA}_i(s_d)$ such that $M_i(a) \neq \emptyset$, and a cell $k \in M_i(a)$, such that $\mathcal{F}_{ij}(a, x_j) \in T_j$ for all $j \in M_i(a)$, $j \neq k$, and such that, if cell k is disrupted by action a (that is, if state $\mathcal{F}_{ik}(a, x_k) \in D_k$), it is a disruption of type $l - 1$. Cell i can recover from a disruption of type n without the disruption being propagated more that l levels. If a disruption is not of any of these types, then this model cannot describe how the system can recover from this disruption.

A. Outline of Approach

In our "polite replanning" approach, we assume that, in searching for the lowest cost response to an outside disruption, it is best to try to limit the propagation of the disruptions. Even though the best solution might entail the disruption of every cell in the system, we limit the search space by trying to find a solution which involves the least disruption propagation.

When a cell experiences a disruption, it tries to determine whether this disruption is of type 0. If it determines this, then it takes a recovery action which will not result in a disruption of another cell. If not, then it tries to determine through negotiation with other cells whether the disruption is of type 1. If it determines this, it takes the action which results in a propagation of the disruption of at most one level. Here we do not go beyond disruptions of type 1; in our cellular manufacturing domain, there is not a large number of cells, so that at greater levels of propagation, the whole system is affected. This approach can be extended to disruptions of type l in systems of greater numbers of cells. In such a case, a small group of cells may cooperate more fully to prevent the disruption from propagating beyond that group.

Consider a disruption which puts cell i into disrupted state s_d. We do not assume that the controller at cell i already knows its full set of possible actions, and their effects on other cells. Instead, the controller at cell i uses some heuristic G to try to find a guaranteed-local action $a \in \mathrm{GA}(s_d)$. If it can find such an action, it will take that action. If not, some communication is necessary for the selection of a good recovery action. Thus, the controller at cell i uses some heuristic H to select a recovery action $a' \in \mathrm{RA}(s_d)$ which seems likely, given local information, not to be very disruptive to other cells. Cell i then sends a proposal message to all members of $M_i(a')$, proposing action a'.

When a cell j receives a proposal message proposing action a', it first determines whether action a' will cause a disruption at cell j. If not, then it returns an `ok-0` message. Otherwise, it tries to determine whether the disruption caused by a' will be one of type 0, which can be handled locally. If so, it returns an `ok-1` message. Otherwise, it will return a `not-ok` message, perhaps along with some information J which can be used by the disrupting cell's heuristic H to propose a better solution.

When the controller at the disrupted cell receives replies to its proposal, if all replies are `ok-0` messages, then it takes the proposed action. If all replies are either `ok-0` or `ok-1` messages, then the controller can take the proposed action. If there is a `not-ok` reply, then the controller knows that action a' will not isolate the disruption to the cells in $M_i(a')$, so, with whatever information has been gathered, it uses heuristic H again to propose a new recovery action, unless it determines that further negotiation will not be useful.

This approach is of course only a simple outline of an algorithm for handling this problem. The real issues are what kinds of heuristics G and H are, what kinds of information J is to be exchanged, and what to do when no proposal is acceptable to the other cells. At least some of these answers are domain dependent, and cannot be more fully described in this very general model.

IV. Polite Rescheduling

A. Background

In this section, we consider the use of polite replanning in the domain of scheduling in a cellular manufacturing system. The scheduling domain is an appropriate one in which to investigate this problem, because there are easily definable interactions among cells, in the form of precedence constraints among jobs. Scheduling is the assignment of jobs to machines at specified times, and may be done statically (before execution) or dynamically (during execution). In dynamic scheduling, all scheduling decisions are made at run-time, by dispatch rules [3], or by least-commitment opportunistic planning [14]. While fast and widely used in practice, dynamic scheduling suffers from being myopic and unpredictable (i.e., scheduling actions are unknown before run-time). Static sched-

uling, constructing a schedule before execution, allows use of time consuming optimization methods which can allocate resources much more efficiently. Static scheduling also allows prediction of task completion times and machine utilization.

One often overlooked but important aspect of scheduling is the actual execution of an statically-constructed schedule (a *preschedule*). During execution, unexpected events, such as machine breakdown or new job arrival, may disrupt the preschedule. Revising the schedule on the shop floor may be difficult, as time is constrained, and shop floor computing resources may be limited. Thus, even if optimization methods were used to construct the preschedule, they may be unavailable for its revision. One approach to handling unexpected events is dynamic scheduling, in which no preschedule is constructed. Another approach is to construct a new schedule when events render the old one infeasible. One very fast way of doing this is to "push back" the existing schedule until it becomes feasible. This method is widely used in practice, but very often produces an inefficient schedule.

These approaches, however, do not make good use of the preschedule. We choose instead to follow the matchup scheduling approach of Bean *et al.* [2]. In this approach, when unexpected events disrupt the preschedule, the scheduler attempts to schedule production so that the system can return to ("match up with") the original preschedule. Thus, the good preschedule need not be discarded when disruptions occur, and commitments based upon that schedule need not necessarily be broken.

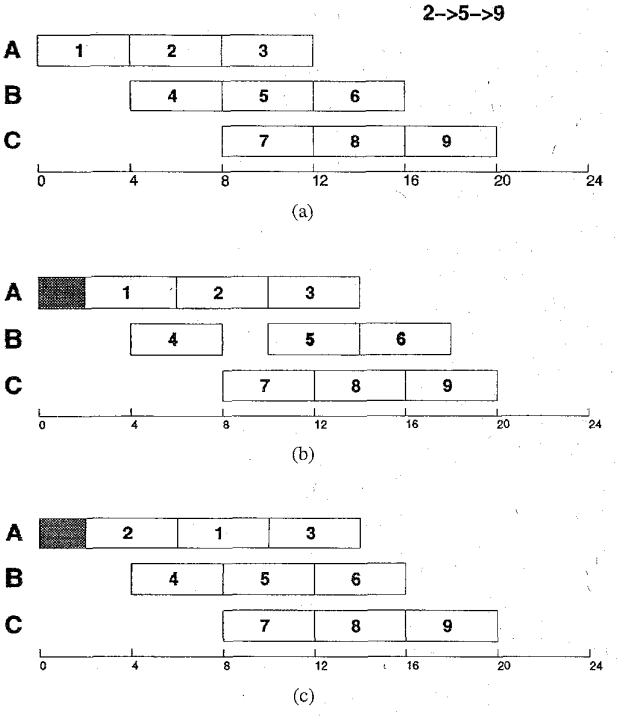


Fig. 1. A simple example. (a) Preschedule. (b) Pushed-back schedule. (c) Rescheduled schedule.

B. Polite Rescheduling

Our approach, as previously discussed, is to have local cell schedule controllers reschedule in response to schedule disruptions in such a way as to limit the disruption, either to the cell itself, or to a small subset of cells. In order to evaluate various rescheduling approaches, we consider the following class of job shop problems. Each job is to be processed on any machine of one specific cell. Jobs may have successors at other cells; a successor job may start processing only after its predecessor has been completed. We assume that a preschedule has already been constructed for this set of jobs, and that this preschedule tries to minimize the sum of tardiness over all the jobs. Tardiness is a common measure, but minimizing tardiness for even simple problems is NP-hard.

As intelligent schedulers reason about schedule constraints, an intelligent rescheduler should reason about inter-cell schedule constraints. We assume that each cell has knowledge from the preschedule about these constraints, including which of its jobs have successor jobs, and the times those successor jobs are scheduled to begin processing at other cells. We call the times the *precedence deadlines* of the predecessor jobs; precedence deadlines are not to be confused with due times. Likewise, each cell has information about which of its jobs have predecessor jobs. However, cells do not have any other information about the schedules at other cells.

In this type of problem, cells interact solely through precedence constraints among jobs. Consider the very simple example in Fig. 1. Here there are three cells with one machine per cell. Here job 2 has job 5 as a successor, which in turn has job 9 as a successor. The preschedule is shown in Fig. 1(a). In Fig. 1(b), the machine at cell A is unable to process any job from time 0 to time 2. Cell A's schedule has been pushed back, disrupting the schedule at cell B because of the late processing of job 2.

The algorithm we propose is based upon the outline described in Section III. When a disruption is identified at a cell, that cell will try to reschedule itself without disrupting schedules at other cells; such rescheduling would be a guaranteed-local recovery action. It will thus try to find a new schedule in which jobs with successors complete processing before their successors are scheduled to begin processing (in the preschedule). If such a nondisrupting schedule can be found, then the cell will attempt to implement a good nondisrupting schedule.

If such a schedule cannot be found, then the cell will try to find a schedule that is likely to be least disruptive to other cells. It then will propose that schedule to the cells which may be affected by it. Each of these other cells will either accept this schedule, if it determines that it can reschedule in response to any disruptions caused by the proposed schedule without disrupting other cells, or reject this schedule, if it cannot determine this. If all of these cells accept the proposed schedule, then the originally disrupted cell will implement it, and the cells disrupted will find and implement new nondisrupting schedules which deal with the disruptions caused by the proposed schedule.

In the simple example described before in Fig. 1, while the pushed-back schedule in Fig. 1(b) resulted in a schedule

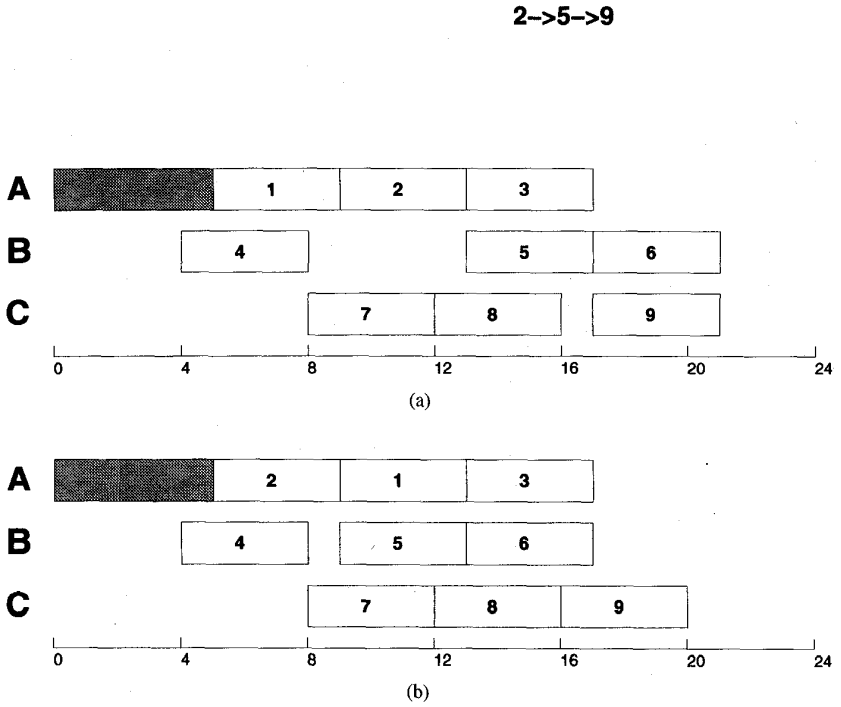


Fig. 2. A simple example. (a) Push-back schedule. (b) Rescheduled schedule.

disruption at cell B, the schedule in Fig. 1(c) reschedules cell A without disrupting cell B. In our algorithm, cell A would try to find such a schedule before beginning any negotiations with any other cells. Had the machine of cell A been down from time 0 to time 5 instead, as in Fig. 2, then cell A first would try to find a nondisruptive schedule, and would fail because none exists. It then would try to find a schedule least likely to be disruptive to cell B. It would then propose this schedule. Were it to propose the pushed-back schedule as in Fig. 2(a), cell B would not accept the proposal, as it would be unable to avoid disrupting the schedule at cell C. The schedule in Fig. 2(b), if proposed by cell A, would be accepted by cell B, as it can find a nondisruptive schedule to address the late completion of job 2.

C. Implementation

Our architecture for investigating polite rescheduling is called PRIAM (Polite Rescheduler for Intelligent Automated Manufacturing). PRIAM consists of a *rescheduler* module and a *negotiator* module, as illustrated in Fig. 3. The rescheduler produces new schedules according to priorities determined by the negotiator. The negotiator module determines what kinds of schedules to propose to other cells, and determines the priorities through the use of information resident at the node or gathered through communication. This division is natural, because negotiating is a higher level task which is not concerned specifically with making schedules, while rescheduling is a lower level task in which negotiation does not play a direct role.

The priorities given to the rescheduler determine what kind of schedule will be produced. For example, if a disruption

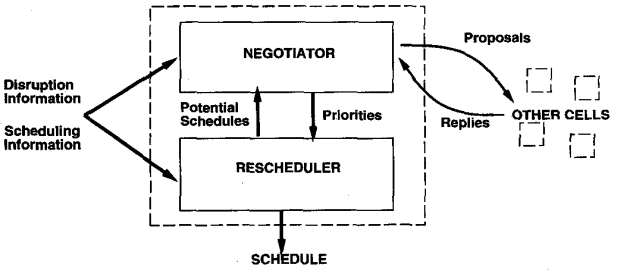


Fig. 3. The PRIAM architecture.

has just been identified, then the initial action will be to try to find a nondisrupting schedule. The priority for such a schedule is for all jobs with successors to complete processing before their precedence deadlines. Except for one machine scheduling, even this simple problem of scheduling to meet deadlines is NP-hard, so the rescheduler usually cannot search for optimal solutions. We use a heuristic priority scheduling algorithm (Fig. 4) with three priority levels: high priority for predecessor jobs, low priority for predecessor jobs, and lowest priority for nonpredecessor jobs. The algorithm first schedules high priority jobs, and then inserts low priority jobs into the partial schedule without pushing high priority jobs past their precedence deadlines. Lowest priority jobs are likewise inserted into the schedule. In general, the rescheduler must be fast, because the time allowed for rescheduling on-line after a disruption is likely to be much shorter than the time allowed for construction of the preschedule. Likewise, while the negotiator makes decisions on a higher level, it must also use heuristics.

Given n high priority predecessor jobs, m low priority predecessor jobs, and s non-predecessor jobs, and a preferred dispatch rule D:

1. Schedule the high-priority predecessor jobs by earliest precedence deadline, and label these jobs $1, 2, \ldots, n$;
2. For each scheduled high-priority predecessor job i in order,
   2.1. Define the slack time s_j for job j, $i \leq j \leq n$ as the idle time in the current schedule between the completion time of job $i-1$ and the precedence deadline for job j;
   2.2. If there is an unscheduled low-priority predecessor job which has processing time less than $\min_{i \leq j \leq n}(s_j)$, insert into the schedule before job i the job which has the smallest precedence deadline of such jobs, and go to 2.1.
3. Schedule the remaining unscheduled low-priority predecessor jobs by earliest precedence deadline.
4. Relabel the scheduled predecessor jobs in order of starting time $1, 2, \ldots n+m$;
5. For each scheduled predecessor job in order,
   5.1. Define the slack time s_j for job j, $i \leq j \leq n+m$ as the idle time in the current schedule between the completion time of job $j-1$ and the precedence deadline for job j;
   5.2. If there is an unscheduled job which has processing time less than $\min_{i \leq j \leq n}(s_j)$, insert into the schedule before job i one such job chosen by D, and go to 5.1.
6. Schedule the remaining unscheduled jobs by D.

Fig. 4. Priority scheduling algorithm for polite scheduling.

V. Evaluation

A. Simulation Model

We evaluate priority scheduling algorithms in PRIAM through simulation of disruptions in a generic manufacturing system. In these simulations, a preschedule is constructed for a generic manufacturing system, which consists of four groups of three cells each: a set of machining cells, two sets of subassembly cells, and a final assembly cell. Each cell has two identical machines, and a job may be processed only at one specified cell. Jobs may have precedence constraints: a job at a machining cell may have a successor at a subassembly 1 cell, a job at a subassembly 1 cell may have a successor at a subassembly 2 cell, and a job at a subassembly 2 cell may have a successor at a final assembly cell. But there may be jobs at any cell which do not have successors or predecessors.

The preschedule is generated from a randomly generated set of 192 jobs (16 per cell). One parameter in the generation of the job set is p, the probability that any given job is the predecessor of some other job (excepting final assembly jobs). By varying p, job sets with different levels of precedence constraints are generated. In each of the job sets, a job may have at most one successor, but may have several predecessors. We assume, for simplicity, that setup times are not sequence-dependent, and can be ignored. In each simulation, one of the machines at a machining cell is disabled for a given interval, and the system is rescheduled using each of the rescheduling methods described above.

We are chiefly concerned with how disruptive the rescheduling process is to the manufacturing system. Our primary measure of disruptiveness is the number of cells which are affected by the disruption. Other measures that we consider are the total number of times cells need to reschedule, the number of jobs whose scheduled completion times are changed, and the number of jobs which are rescheduled on a machine different from that on which it was originally scheduled. Our secondary measure is the makespan, the completion time of the last job to finish. Makespan is the measure used in constructing the original preschedule, and it is a measure of the quality of the resulting schedule.

We consider three different negotiation strategies in these simulations. The negotiation strategies that can be used depends upon what kind of information can be obtained from other cells. For the polite negotiation algorithm for these

negotiations, we assume that, when a disrupted cell proposes a new schedule to a remote cell, that remote cell will reply `ok-1` if it can reschedule without disrupting other cells; otherwise, it will include in its reply the identities of the cells it will disrupt if the proposed schedule is implemented. From this information, the disrupted cell will have an estimate of how many cells will be disrupted by a proposed schedule. In the first polite negotiation algorithm (POL-NEG1), the disrupted cell will propose a small number of possible schedules generated from different priorities, and will decide to implement the first proposal which elicits only `ok-1` replies from other cells. Otherwise it will implement the proposal causing fewest disruptions. In the second polite negotiation algorithm (POL-NEG2), the disrupted cell will also propose a small number of possible schedules and will decide to implement the proposal causing fewest disruptions. The third negotiation algorithm (POL-NEG3) is the same as POL-NEG2, except that the number of proposals is smaller. Only the originally disrupted cell uses these polite negotiation algorithms; if a cell is disrupted only by the late completion of a predecessor job, it will not use negotiation.

In these simulations, we compare the results from our polite negotiation algorithms with the results from a polite algorithm (POL) using the previously described priority scheduling algorithm but without negotiation. In this algorithm, the disrupted cell generates a small number of possible schedules and tries to limit the number of disrupted cells, but does so without negotiation with other cells. We also compare these results with the results from two similarly fast algorithms which do not consider how the rescheduling of one cell may affect another: the pushback algorithm (PB), in which schedules at disrupted cells are simply pushed back, and the largest-remaining-processing-time-first dispatch rule (LPT), which is used to achieve a low makespan, but does not consider the problem of disrupting other cells. These algorithms do not include optimization techniques. Such techniques usually consume much time and computation, and our assumption is that schedule revision at a cell will not take place on powerful computing platforms dedicated to execution of long intensive tasks, as the cell controller is responsible for other local management tasks.

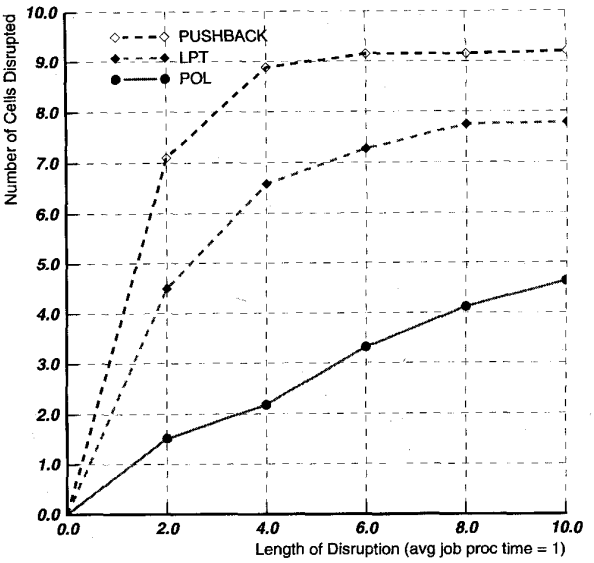


Fig. 5. Number of cells disrupted.

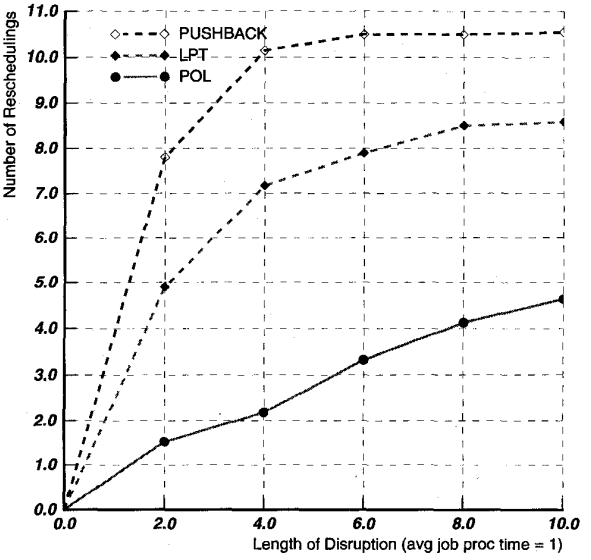


Fig. 6. Number of reschedulings.

B. Results

Figs. 5 through 16 show results for three simulations for each of twenty job sets with the constraint parameter $p = 0.6$. First we consider only the PB, LPT, and POL algorithms. Fig. 5 shows the number of cells eventually disrupted from the propagation of one machine disruption, versus the length of the original disruption. These results show that the POL algorithm isolates disruptions much more than the two other nonnegotiation rescheduling methods. Fig. 6 shows the number of times cells eventually have to reschedule. Again, the POL algorithm is much better for preventing other cells from having to reschedule. Fig. 7 shows the makespan after all rescheduling is finished. The POL algorithm is slightly better than the LPT algorithm at keeping the makespan from being affected by the disruption.

Figs. 8 through 14 show simulation results for the polite negotiation algorithms. Figs. 8 and 9 show that for short disruptions, negotiation does not seem to provide an advantage; for longer disruptions, both polite negotiation algorithms show a significant advantage over the POL algorithm without negotiation. There seems to be little difference in how well the POL-NEG1 and POL-NEG2 algorithms prevent the spread of disruptions, while Fig. 10 shows that the POL-NEG1 algorithm requires fewer message exchanges. Fig. 11 shows how many levels of propagation are caused by the original disruption. Figs. 12 through 14 compare the POL-NEG2 and POL-NEG3 algorithms, and show that while fewer proposals reduce the number of messages required, it also reduces the advantage of using negotiation. Figs. 15 and 16 show the

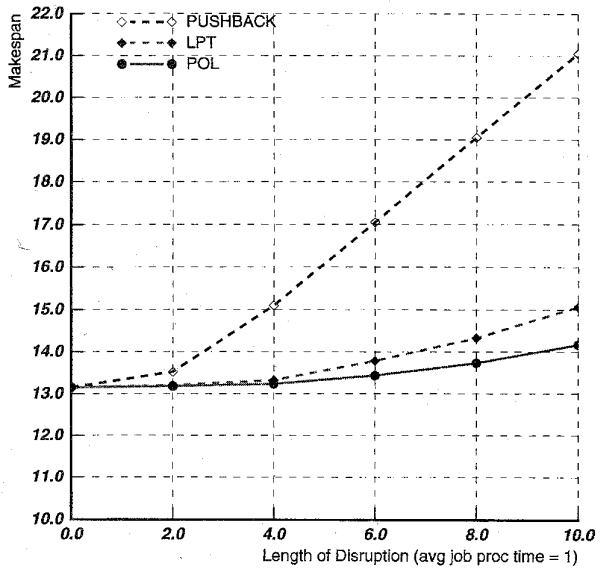


Fig. 7. Makespan.

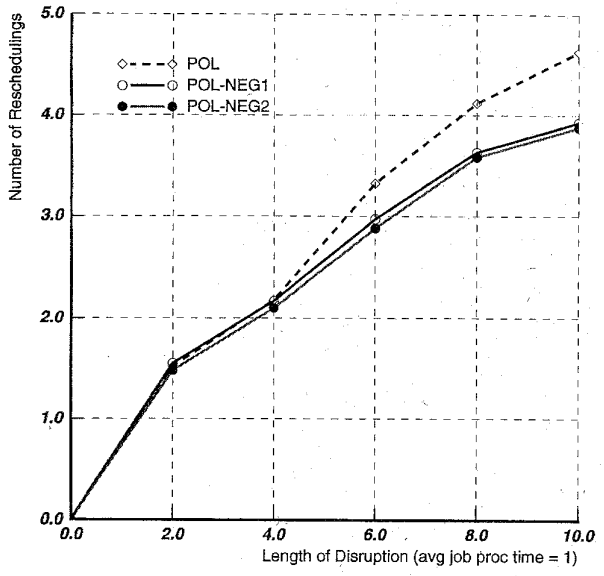


Fig. 9. Number of reschedulings.

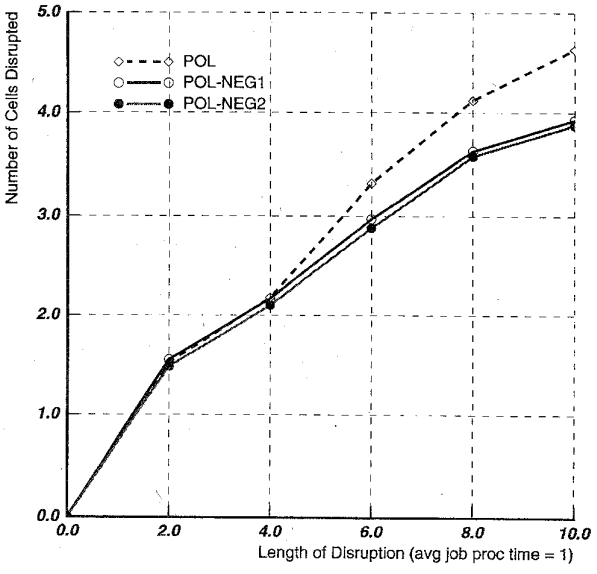


Fig. 8. Number of cells disrupted.

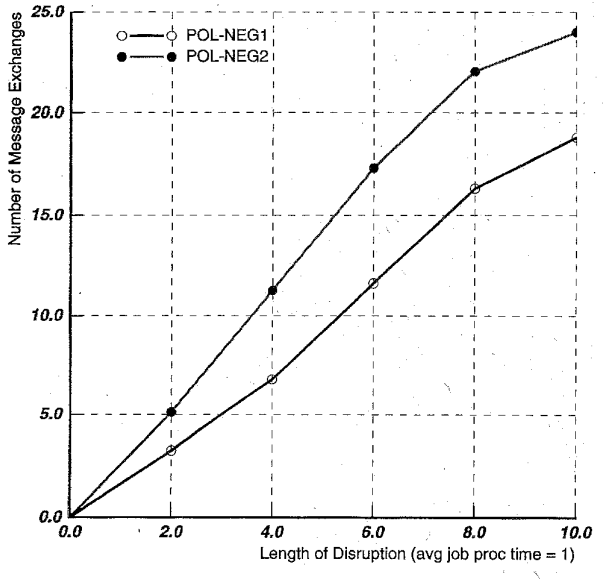


Fig. 10. Number of message exchanges.

performance of the various algorithms using the measures of the number of jobs which are rescheduled, and the number of jobs that are moved to a different machine during rescheduling. These show that, when the disruption length is low, the polite algorithms reschedule and move many fewer jobs than the LPT algorithm. When the disruption length becomes large, it is much harder to avoid rescheduling jobs.

C. Discussion

While any use of negotiation in distributed manufacturing systems is necessarily very domain-dependent, our generic simulations allow us to make some observations about the usefulness of the polite approach and polite negotiation. These simulations show that polite rescheduling provides an advantage for a distributed system, allowing a cell to respond to a schedule disruption while reducing the spread of this disruption to other cells. However, they also indicate under what conditions polite rescheduling is likely to be useful. When the constraints of the rescheduling problem are light, as when the initial machine disruption is short, the use of negotiation does not provide much advantage, because the disruption is not likely to be propagated even if the possibility of propagation is not considered. Likewise, other simulation results not presented here indicate that when the scheduling problem itself is less constrained (because there are fewer precedence relations among jobs), the polite approach with and without negotiation provide less significant advantages,

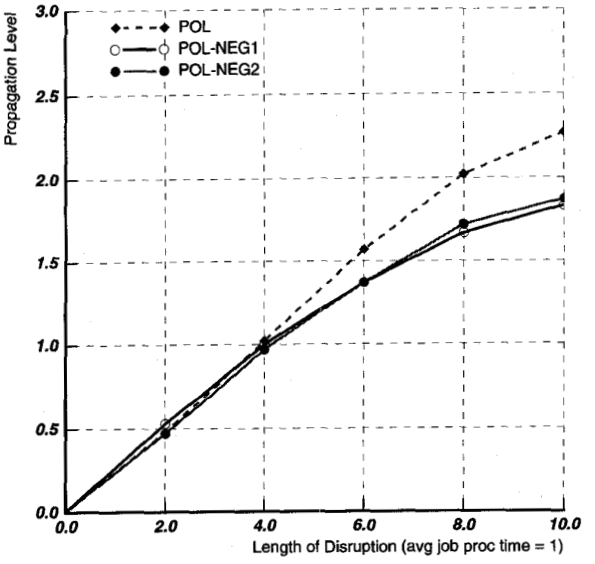


Fig. 11. Propagation level.

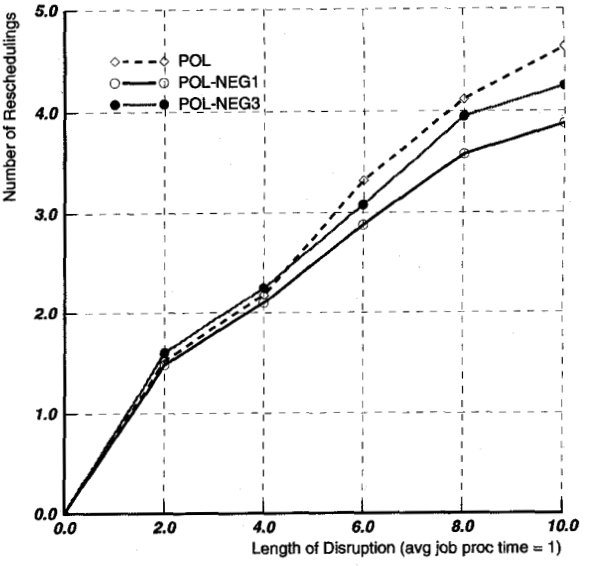


Fig. 13. Number of reschedulings.

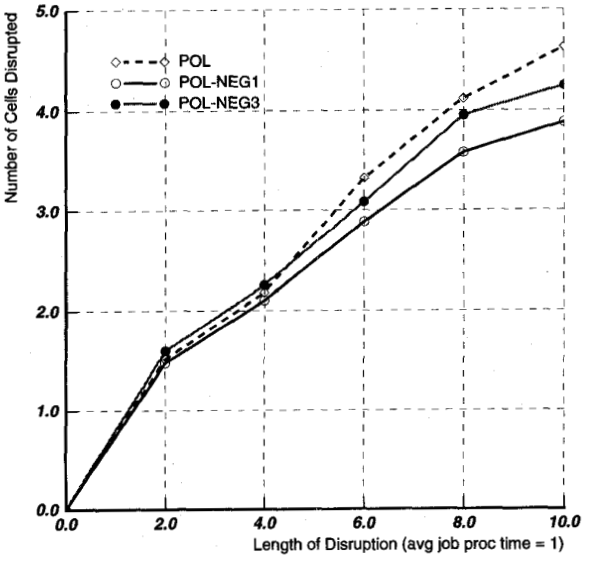


Fig. 12. Number of cells disrupted.

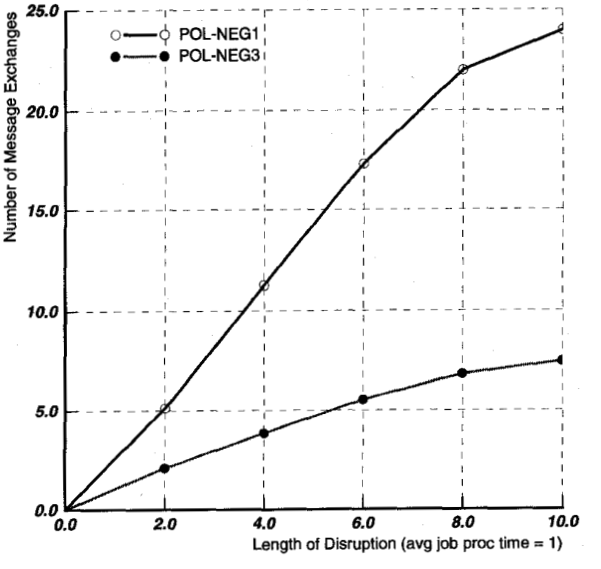


Fig. 14. Number of message exchanges.

once again because the disruption is much less likely to be propagated.

When the rescheduling problem is more constrained, as when the initial disruption is longer, the polite strategies which use negotiation provided a much more significant advantage over those strategies not using negotiation. Under these conditions, a cell which does not have global information is not likely to have enough information about how its actions will affect other cells, and thus communication allows the cell to make better-informed decisions. However, our other simulation results suggest that when the scheduling problem is much more constrained (because there are more precedence relations among jobs), the polite approaches provide less of an advantage over other approaches. Under these conditions, the many constraints of the problem almost ensure that any disruption will be propagated throughout the system, and attempts to respond to the disruption using negotiation will merely confirm that the propagation is almost unavoidable. Thus, there is a "window" of usefulness for our polite approach; it is most useful when the problem is constrained enough that a cell cannot have sufficient information without communication, but not so constrained that propagation of disruptions cannot be avoided. The size of this window is obviously very domain-dependent.

Another important issue for evaluation of the polite approach is cost. There are at least two dimensions for cost in this type of problem. The first involves the quality of the resulting schedule based upon some scheduling measure disregarding

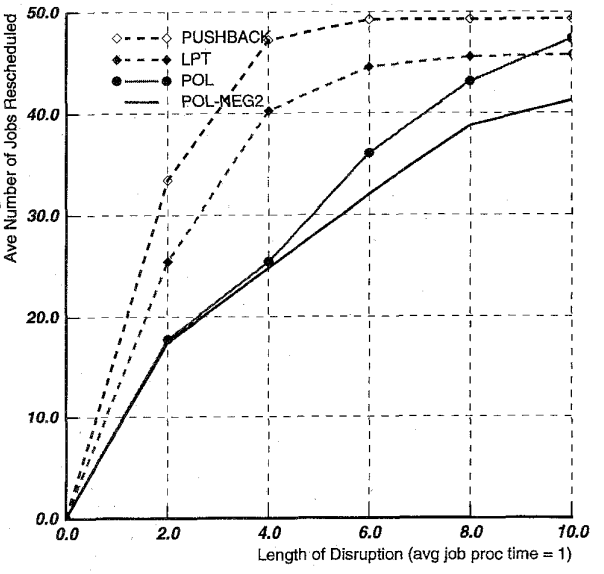


Fig. 15. Number of jobs rescheduled.

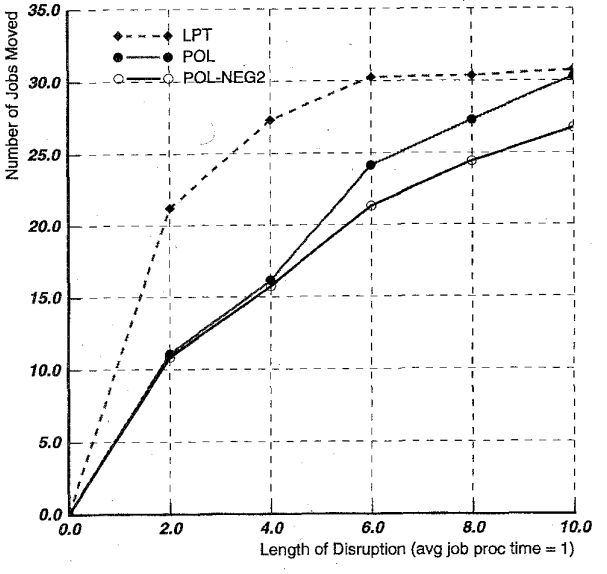


Fig. 16. Number of jobs moved.

disruption. In these simulations, we considered the makespan measure, which is a single "global" measure rather than an aggregate one. Using this measure, our polite methods performed well, because preventing disruption propagation also prevents delays in the completion times of jobs with predecessors. However, in previous simulation results which we have reported in [19], we considered the average tardiness measure, which is an aggregate measure. When this measure was considered, our polite methods produced schedules of less quality (higher tardiness) than other approaches which allowed greater disruption propagation, because the polite methods trade off increased tardiness at the disrupted cell against the spread of disruptions. Thus, there can be a tradeoff between the measure for *scheduling* (e.g., tardiness) and the measure for *rescheduling* (e.g., number of cells disrupted). The other dimension of cost is the computation and communication overhead for the polite approaches. This overhead is an important consideration for any use of negotiation, especially if communication and processing resources are limited, as might be true for a cell controller computer. Our simulations show that the communication costs (in number of message exchanges) grow as the size of the disruption grows (because more negotiation is needed to find a good rescheduling solution), and that additional gains in preventing disruption propagation come at the cost of additional communication (because more proposals are required to find a better solution).

In these simulations, for simplicity, we have used the same job characteristic parameters for the job sets for each cell. Greater heterogeneity among cells is more realistic, as different cells may process different classes of tasks. Our ongoing work investigates how such heterogeneity can determine the utility of different negotiation strategies.

VI. Summary

We have presented polite rescheduling, a new approach to handling schedule disruptions in a distributed manufacturing system. Our approach takes into consideration the possibility that responding to a disruption in one part of the system may cause disruptions in other parts of the system. It thus attempts to respond to disruptions local to one manufacturing cell so that other cells are disrupted as little as possible. Our simulation results with PRIAM show the advantages of using a scheduling algorithm that emphasizes precedence constraints over other scheduling considerations, and demonstrate the advantages of negotiation. We believe that there are other fruitful applications of DAI concepts to the area of manufacturing systems, and we are exploring in other domains the problem of handling disruptions in decentralized environments.

References

[1] P. E. Agre and D. Chapman, "Pengi: An implementation of a theory of activity," in *Proc. National Conf. Artificial Intelligence*, 1987, pp. 268–272.

[2] J. C. Bean *et al.*, "Matchup scheduling with multiple resources, release dates and disruptions," *Operations Res.*, vol. 39, no. 3, pp. 470–483, May–June 1991.

[3] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatch rules for manufacturing job shop operations," *Int. J. Prod. Res.*, vol. 20, no. 1, pp. 27–45, 1982.

[4] A. H. Bond and L. Gasser, "An analysis of problems and research in DAI," in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser, Eds. San Mateo, CA: Morgan Kaufmann, 1988, pp. 3–35.

[5] R. Davis and R. G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intell.*, vol. 20, pp. 63–109, 1983.

[6] N. A. Duffie *et al.*, "Fault-tolerant heterarchical control of heterogeneous manufacturing system entities," *J. Manufact. Syst.*, vol. 7, no. 4, pp. 315–328, 1988.

[7] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Trans. Comput.*, vol. C-36, no. 11, pp. 1275–1291, Nov. 1987.

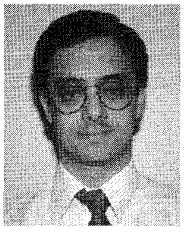
[8] M. S. Fox and S. F. Smith, "ISIS—A knowledge-based system for factory scheduling," *Expert Syst.*, vol. 1, no. 1, pp. 25–49, 1984.

[9] L. Gasser and M. N. Huhns, Eds., *Distributed Artificial Intelligence*, vol. 2. San Mateo, CA: Morgan Kaufmann, 1989.

[10] A. C. Jones and C. R. McClean, "A proposed hierarchical control model for automated manufacturing systems," *J. Manufact. Syst.*, vol. 5, no. 1, pp. 15–25, 1986.

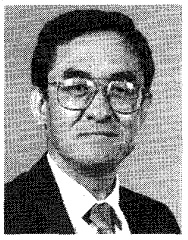
[11] S. Kambhampati *et al.*, "Integrating general purpose planners and specialized reasoners: Case study of a hybrid planning architecture,"

IEEE Trans. Syst., Man, Cybern., vol. 23, no. 6, pp. 1503–1518, Nov. 1993.
[12] S. Minton and A. B. Philips, "Applying a heuristic repair method to the hst scheduling problem," in *Proc. DARPA Workshop Planning, Scheduling and Control*, 1990, pp. 215–219.
[13] Y. Nishibe *et al.*, "Effects of heuristics in distributed constraint satisfaction problems," in *Proc. 11th Int. Workshop DAI*, 1992, pp. 285–302.
[14] P. S. Ow and S. F. Smith, "Viewing scheduling as an opportunistic problem-solving process," *Ann. Operation Res.*, vol. 12, pp. 85–108, 1988.
[15] H. V. D. Parunak, "Distributed artificial intelligence systems," in *Artificial Intelligence Implications for Computer Integrated Manufacturing*, A. Kusiak, Ed. Bedford: IFS, 1988, pp. 225–251.
[16] S. Sen and E. H. Durfee, "Unsupervised surrogate agents and search bias change in flexible distributed scheduling," in *Proc. 1st Int. Conf. Multi-Agent Systems*, 1995, pp. 336–342.
[17] S. Smith, "OPIS: A methodology and architecture for reactive scheduling," in *Intelligent Scheduling*, M. Zweben and M. S. Fox, Eds. San Mateo, CA: Morgan Kaufmann, 1994, pp. 29–66.
[18] K. Sycara *et al.*, "An investigation into distributed constraint-directed factory scheduling," in *Proc. IEEE AI Applications*, 1990, pp. 94–100.
[19] T. K. Tsukada and K. G. Shin, "Polite rescheduling: Responding to schedule disruptions in a distributed manufacturing system," in *Proc. 1994 IEEE Int. Conf. Robotics and Automation*, 1994, pp. 1986–1991.
[20] D. J. Williams and P. Rogers, Eds., *Manufacturing Cells: Control, Programming and Integration*. Oxford: Butterworth-Heinemann, 1991.
[21] M. Zweben *et al.*, "Scheduling and rescheduling with iterative repair," in *Intelligent Scheduling*, M. Zweben and M. S. Fox, Eds. San Mateo, CA: Morgan Kaufmann, 1994, pp. 241–255.



Thomas K. Tsukada (S'91–M'93) received the B.A. degree in economics from Haverford College, Haverford, PA, in 1987, the B.S. degree in electrical and computer engineering from the State University of New York at Buffalo in 1989, and the M.S. degree in computer science from the University of Michigan, Ann Arbor, in 1991.

Currently, he is a Ph.D. candidate and a Research Assistant in the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science at the University of Michigan. His research interests include computer integrated manufacturing, scheduling, and distributed artificial intelligence.



Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratories, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA, IBM T. J. Watson Research Center, and Software Engineering Institute at Carnegie Mellon University, Pittsburgh, PA. He also chaired the Computer Science and Engineering Division, EECS Department at the University of Michigan for three years beginning January 1991. He is currently Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. He has authored/coauthored over 350 technical papers (more than 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He is currently writing (jointly with C. M. Krishna) a textbook *Real-Time Systems*, which is scheduled to be published by McGraw-Hill in 1996. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, and middleware services for distributed real-time fault-tolerant applications. He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes.

Dr. Shin received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning in 1987. In 1989, he received the Research Excellence Award from the University of Michigan. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems, a Program Co-Chair for the 1992 International Conference on Parallel Processing, and served numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991–1993, was a Distinguished Visitor of the Computer Society of the IEEE, an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED COMPUTING, and an Area Editor of *International Journal of Time-Critical Computing Systems*.