Real time distributed shop floor scheduling using an agent-based service-oriented architecture

Wang, C.; Ghenniwa, H.; Shen, W.

Publisher's version / Version de l'éditeur:

International Journal of Production Research, 46, 9, pp. 2433-2452, 2008-05-01

Real time distributed shop floor scheduling using an agent-based service-oriented architecture

NRCC-50323

Wang, C.; Ghenniwa, H.; Shen, W.

May 2008

A version of this document is published in / Une version de ce document se trouve dans:
International Journal of Production Research, v. 46, no. 9, 2008, .pp 2433-2452

Real Time Distributed Shop Floor Scheduling Using an Agent-Based Service-Oriented Architecture

Chun Wang[1], Hamada Ghenniwa[1], Weiming Shen[1,2]

[1] *Department of Electrical and Computer Engineering*
The University of Western Ontario, London, Ontario, Canada
[2]*Integrated Manufacturing Technologies Institute*
National Research Council Canada, London, Ontario, Canada
cwang28@engga.uwo.ca; hghenniwa@eng.uwo.ca; weiming.shen@nrc.gc.ca

Abstract *This paper proposes a distributed manufacturing scheduling framework at the shop floor level. The shop floor is modeled as a collection of multiple workcells, each of which is modeled as a flexible manufacturing system. The framework consists of a distributed shop floor control structure, dynamic distributed scheduling algorithms, multi-agent system modeling of workcell, and service oriented integration of the shop floor. At the workcell level, the designated scheduler allocates jobs to resources and deals with any dynamic events locally, if possible. Otherwise, it collaborates with the other peer schedulers of workcells. Workcells are modeled as multi-agent systems. Local dynamic scheduling is achieved by the cooperation of the scheduler agent, the real time control agent and resource agents. Distributed scheduling is conducted through Web services facilitated by the service oriented shop floor integration. The proposed distributed control structure, dynamic distributed scheduling mechanisms and the system integration have been implemented using an agent-based service-oriented approach and validated through a case study.*

Keyword: Shop floor scheduling, distributed control structure, dynamic distributed scheduling algorithms, agent based service oriented integration.

1. Introduction

Globalization of markets has driven manufacturing enterprises to shed the security of mass production and shift to a new paradigm, mass customization. An essential goal of this transformation is to respond to market changes in a timely and cost effective manner. As an integral component of manufacturing management, scheduling needs to be effectively integrated with other components of manufacturing systems such as supply chain management, ERP, and shop floor control. Dynamic changes can derive from either outside parties in the market, such as the supply side (representing suppliers), demand side (representing customers) or within the enterprise, such as real-time events from the shop floor. In a real world shop floor environment, it is rarely the case to execute exactly as planned. Operation durations tend to vary, machines break down, raw materials fail to arrive on time, new customer orders appear, others get

cancelled, etc. Such disruptions incur higher costs due to missed customer delivery dates, higher work-in-process inventory, and lower resource utilization. To deal with these issues, practical scheduling systems need to be able to effectively reorganize the shop floor production plan and repair or redo the production schedule accordingly. Scheduling systems with the capability of revising or re-optimizing a schedule, in response to unexpected events, becomes key for companies to sustain their productivity.

This research is concerned with developing real time distributed scheduling systems at the shop floor level. The shop floor is modeled as a collection of workcells. The workcells, in turn, are modeled as Flexible Manufacturing Systems. The scheduling is performed cooperatively and collectively by the group of schedulers, each delegated to a specific workcell. Dynamic scheduling in this environment requires real time scheduling algorithms and their effective integration with the distributed shop floor control structure. Dynamic scheduling has been extensively studied in the literature (Kocjan, 2002; Shanker and Tzen, 1985). There have been research efforts focusing on distributed scheduling as well (Baker, 1998; Neiman et al., 1994; Sycara et al., 1991). In this paper, we study dynamic and distributed scheduling algorithms in the multiple workcell shop floor setting. In addition to the individual algorithms, we investigate how to integrate them in a way that the overall shop floor scheduling agility and solution quality are enhanced.

2. Scheduling Problems in Multiple Workcell Shop Floor

As illustrated in Figure 1, the shop floor considered here consists of a collection of workcells. Each of them is modeled as a flexible manufacturing system. Within a workcell, jobs need to be scheduled on resources. The scheduling problem at this level is a dynamic FMS scheduling problem, which is handled by a designated scheduler for the workcell. At the shop floor level, due to workcell capability limitation or unexpected events, some workcells may have jobs that need to be assigned to other workcells. The scheduling problem at this level is a dynamic distributed scheduling problem, which needs to be solved collectively by a group of schedulers through cooperation.

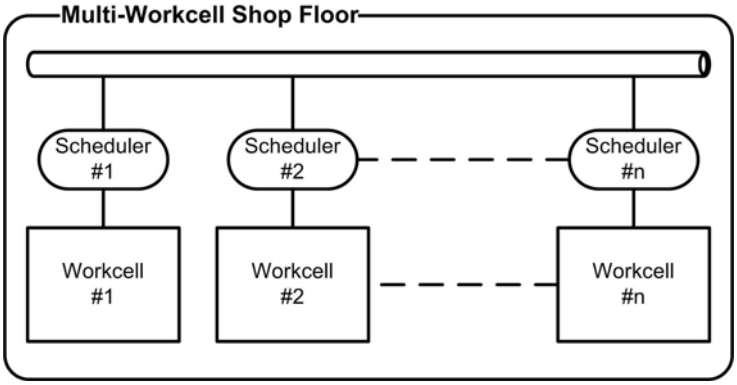


Figure 1 Scheduling in Multiple Workcell Shop Floor

In this paper, we only consider the class of reactive dynamic scheduling algorithms [Sadeh, 1991]. Therefore it is not necessary to model the randomness caused by dynamic events in the following problem models.

2.1. Workcell Scheduling Problem

In the workcell scheduling problem setting, workcells are implemented by FMS systems. Among many FMS scheduling models, we focus our attention on a class of problems in which machines have Partially Overlapping capabilities [Kamel and Ghenniwa, 1995]. As illustrated in Figure 2, a workcell consists of various types of resources, such as computer numerical controlled machines, automated guided vehicles (AGV) and a workpiece storage system. These resources are controlled by resource controllers and the whole workcell is controlled by a real time controller. An operator can program the processing of the workcell through the interaction with the real time controller. In this paper, we are interested in the impacts of partially overlapped characteristics of the workcell scheduling problem. We have simplified the workcell scheduling model by assuming that the transportation times of jobs between machines are equal and have been modeled in machine processing times. Therefore, there is no need to explicitly model the AGV. At the same time, storage and port are treated as independent resources, like machines. This allows us to differentiate the resources only by their capability sets[1], not by the relationships among them. For some resources, e.g. machines - their capability sets may be partially overlapped.

[1] The capability set of a resource is the set of functionalities provided by that resource. For example, a machine's capability set contains all operations that the machine can process.

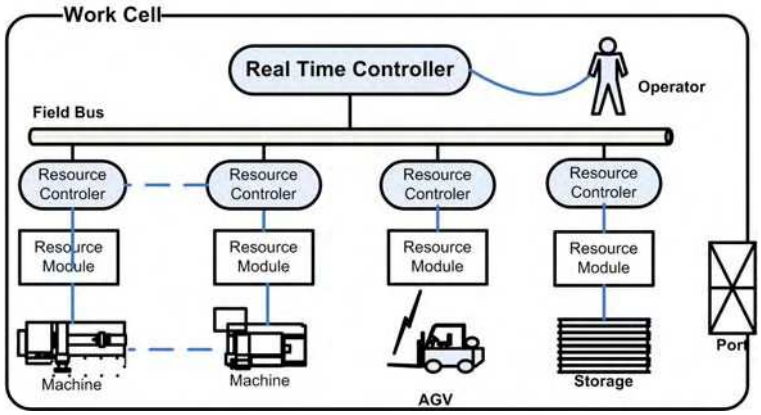


Figure 2 Workcell resources and the control structure

Formally, an instance of the class of scheduling problems in partially overlapping systems consists of a set of *n* jobs, denoted by $J = \{J_1, J_2, ..., J_n\}$, to be processed by a set of m resources, denoted by $M = \{M_1, ..., M_m\}$. Each job J_j ($j = 1, ..., n$) requires the processing of a sequence operations $o_{j,k}$ $(k = 1, ..., n_j)$, where n_j is the number of operations belong to J_j. An operation $o_{j,k}$ corresponds to an uninterrupted physical process which has to be performed on a resource. Each resource M_i ($i = 1, ..., m$) is defined by a set of operations, which represent its capability. If $o_{j,k} \in M_i$, $o_{j,k}$ can be processed by resource M_i. For any two resources $M_i, M_l \in M$, $M_i \cap M_l$ may not be empty. That is, resources have overlapping capabilities. If a resource M_i $(1 \le i \le m)$ is capable of processing an operation $o_{j,k}$ $(1 \le k \le n_j; 1 \le j \le n)$, a processing time $p_{i,j,k} \in R^+$ is given. $p_{i,j,k}$ may not be equal to $p_{l,j,k}$, for $i \ne l$, $i,l \in \{1, ..., m\}$, which means the same operation may have different processing times on different resources. In the workcell scheduling problem, we don't model the resource eligibility constraints. Instead, we assign processing time $p_{i,j,k} = +\infty$, if M_i can not process $o_{j,k}$. In addition, with each job J_j $(j = 1, ..., n)$ we associate two values: release time of a job J_j – r_j, due date for the completion of a job J_j - d_j. There are precedence constraints among operations of each job. The objective is to minimize makespan of the solution schedule. Using the following variables,

$S_{j,k}$, the starting time of the operation k of job j,

$$X_{i,j,k} = \begin{cases} 1 & \text{if machine } i \text{ is chosen to perform operation } k \text{ of job } j \\ 0 & \text{otherwise.} \end{cases}$$

$$Y_{j,k,\hat{j},\hat{k}} = \begin{cases} 1 \text{ if operation } k \text{ of job } j \text{ is performed before operation} \\ \quad \hat{k} \text{ of job } \hat{j} \text{ on the same machine;} \\ 0 \text{ if operation } k \text{ of job } j \text{ is performed after operation} \\ \quad \hat{k} \text{ of job } \hat{j} \text{ on the same machine; } j \neq \hat{j}. \end{cases}$$

the partially overlapping scheduling problem can be formulated as a mixed integer programming as follows.

$$min\left\{\max_{J_j \in J}\left\{S_{j,n_j} + \sum_{i=1}^{m} p_{i,j,n_j} X_{i,j,n_j}\right\}\right\} \tag{1}$$

Subject to

$$S_{j,1} \geq r_j \ , \qquad \forall J_j \in J \tag{2}$$

$$S_{j,k-1} + \sum_{i=1}^{I} p_{i,j,k-1} - S_{j,k} \leq 0 \ , \forall j, 1 < k \leq n_j \tag{3}$$

$$S_{j,k} + \sum_{i=1}^{I} p_{i,j,k} X_{i,j,k} - S_{\hat{j},\hat{k}} + HX_{i,j,k} + HX_{i,\hat{j},\hat{k}} + HY_{j,k,\hat{j},\hat{k}} \leq 3H, \quad \forall j, \hat{j}, j \neq \hat{j}, 1 \leq k \leq n_j, 1 \leq \hat{k} \leq n_{\hat{j}} \tag{4}$$

$$Y_{j,k,\hat{j},\hat{k}} + Y_{\hat{j},\hat{k},j,k} = 1, \quad \forall j, \hat{j}, j \neq \hat{j}, 1 \leq k \leq n_j, 1 \leq \hat{k} \leq n_{\hat{j}} \tag{5}$$

$$\sum_{i=1}^{I} X_{i,j,k} = 1 \ , \ \forall j, 1 \leq k \leq n_j \tag{6}$$

$$X_{i,j,k} \in \{0,1\}, \qquad \forall i, j, 1 \leq k \leq n_j \tag{7}$$

$$Y_{j,k,\hat{j},\hat{k}} \in \{0,1\} \ , \qquad \forall j, \hat{j}, j \neq \hat{j}, 1 \leq k \leq n_j, 1 \leq \hat{k} \leq n_{\hat{j}} \tag{8}$$

$$S_{j,k} \geq 0 \ , \qquad \forall j, 1 \leq k \leq n_j \tag{9}$$

The objective function (1) is to minimize the makespan of the solution schedule. The set of constraints (2) ensure that a job does not start before its release time. The set of constraints (3) ensure that an operation does not start before the previous operation of the same job has completed. The set of constraints (4) and (5) ensure that at most one job can be processed by a resource at a time. In (4) H is a large finite positive number. Constraints (6) say one operation can and only can be processed by one resource. Constraints (7), (8), and (9) are non-negative and integer constraints.

2.2. Dynamic Scheduling Problem

Manufacturing is a process often fraught with contingencies. It is rarely the case that tasks are executed exactly as planned. Small disruptions such as minor deviations in operation durations

often do not warrant major modifications to the schedule. However, as the impact of small disruptions accumulate or as more severe disruptions occur, such as long machine breakdowns, it is sometimes desirable to re-optimize the schedule from a more global perspective [Sadeh, 1991]. In many cases, this re-optimization means re-scheduling all operations that have not been processed by the time of disruption. We distinguish two types of dynamic scheduling situations at the workcell level, namely minor disruptions and severe disruptions. In case of a minor disruption, a schedule repair procedure which minimizes the perturbation to the original schedule is appropriate. On the other hand, if the disruption is severe (caused either by the accumulation of small disruptions or a major resource malfunction), a re-optimization from a more global perspective is usually desirable. An obvious issue is how to decide the severity of a dynamic disruption. The threshold of distinguishing minor and severe disruption should be set by the workcell operator as it involves the trade-off between the overall solution quality and the perturbation to the original schedule. Frequent schedule re-optimization can result in instability and lack of continuity in detailed shop floor plans, resulting in increased costs attributable to what has been termed "shop floor nervousness"[McKay et al.1998].

2.3. Distributed Scheduling Problem

At the shop floor level, the task of scheduling is to coordinate the local schedules of workcells in a way that the good solution quality of the shop floor schedule is achieved. In this context, individual workcell scheduling problems are tied together by two elements, shop floor level objective and inter-workcell constraints. Because each workcell tries to minimize/maximize its own objective function at shop floor level, the scheduling problem can be modeled as a multi-objective optimization problem. The overall solution quality can be measured by Pareto efficiency or some forms of aggregation of the individual objectives of workcells. In the workcell scheduling problem formulated in the previous section, the objective of each workcell scheduler is to minimize the makespan. Accordingly, we define the objective of the shop floor scheduling problem as the weighted sum of makespans over all workcells.

The inter-workcell constraints in the decentralized shop floor scheduling problems are derived from the machine capacity dependency among workcells. Solving the constraints is to achieve coordination among schedulers of workcells. The process of solving the inter-workcell constraints is to find a value assignment to some shared variables that satisfies all the local

constraints of the workcells involved. Specifically, when solving the local scheduling problem, each scheduler has some variables and tries to determine their values. However, there exist inter-workcell constraints because of the capacity dependency and the value assignment must satisfy these constraints. Formally, there exist l workcells $1,2,...,l$. X_h ($h=1,...,l$) is the set that contains all variables the scheduler h needs to assign values to in order to determine a schedule for the workcell h. Because of the inter-workcell constraints, some schedulers need to share a subset of their variables. However, such a case can be formalized as these schedulers have different variables, and there exist constraints that these variables must have the same value. We say that the constraints of a distributed shop floor scheduling problem are satisfied, iff

1. For any scheduler h and $\forall x \in X_h$, the value of x is assigned to d, any constraints in L_h is satisfied under the assignment $x = d$, where L_h is the set of local constraints of workcell h.
2. If $\bar{x}$ is a shared variable between workcell h and workcell q, $\bar{x} = d_h$ in workcell h, $\bar{x} = d_q$ in workcell l, then $d_h = d_q$.

Let $S^I_{h,q}$ denote the set of schedules which satisfy inter-workcell constraints between workcell h and q; let S^L_h denote the set of schedules which satisfy local constraints of workcell h. The distributed shop floor scheduling problem can be formulated as follows.

$$\min \sum_{h=0}^{l} w_h M_h(S)$$

$$\text{s.t.} \quad S \in S^L_h,\ h = 1,...,l,$$

$$S \in S^I_{h,q},\ \ h = 1,...,l,\ \ q = 1,...,l,\ h \neq q.$$

where w_h is the weight of the workcell h, $M_h(S)$ is the latest completion time of all jobs belonging to the workcell h, the allocations of jobs of all workcells form an overall shop floor schedule S.

3. Agent-based Scheduling at Workcell Level

This section presents an agent-based scheduling system at the workcell level. The system consists of a multiagent system architecture, coordination mechanisms among agents and agents' local decision-making schemes.

3.1. Multiagent System Architecture

Figure 3 shows a multiagent system architecture proposed for the agent-based scheduling system at the workcell level. The system consists of resource agents, a real time controller agent, a directory facilitator and a scheduler agent. All agents are connected to the network independently. The Ethernet is taken as the infrastructure for communication and cooperation among agents within a workcell. In principle, each agent can reach any of the other agents in the system by sending messages. However, communication that goes outside the workcell is mediated by the scheduler agent. The functionalities of the agents are described as follows:

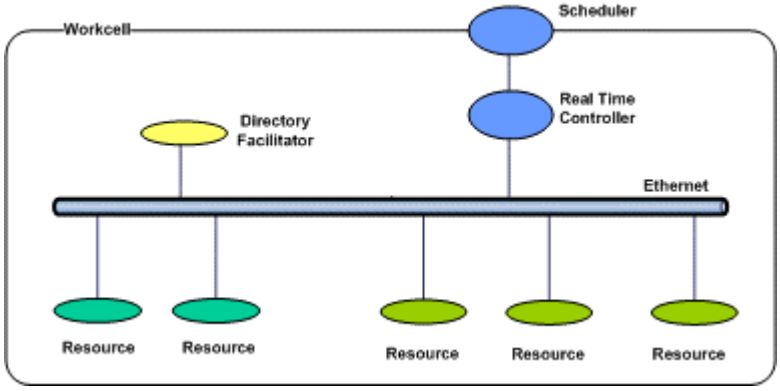


Figure 3 Multiagent System Architecture for a Workcell

1. Resource agents represent manufacturing resources in the workcell. Each agent corresponds to one resource. Resource agents receive job assignments from the Real Time Controller agent and report the working status of their resources to the Real Time Controller agent and the status information, including routine processing data and unexpected disruptions.

2. The Directory Facilitator (DF) has the registration service functionalities for other agents in the agent system, keeps up-to-date agent registration, informs all registered agents with updated registry and provides lookup and matchmaking services to the system.

3. The Real Time Controller is the agent that represents the overall control of a workcell. It accepts production schedules from the workcell scheduler and distributes them to resources in the workcell. At the same time it monitors the processing status of resources, analyzes and aggregates the raw resource processing data. If unexpected changes in the workcell affect the

execution of the schedule, it will report to the scheduler with high level scheduling related processing information.

4. The Scheduler performs the main scheduling functionality in the system. At the workcell level it works with the Real Time Controller implementing the dynamic scheduling within the workcell. At the shop floor level, it cooperatively works with other peer schedulers in achieving distributed shop floor scheduling.

3.2. Coordination Mechanisms among Agents

The workcell level dynamic scheduling requires coordination between the Scheduler, the Real Time Controller and the Resources agents. The coordination between Scheduler and Real Time Controller implements the monitoring and control functionalities required by dynamic scheduling. The Scheduler passes the generated schedules to the Real Time Controller to be executed in the workcell. Workcell resource statuses can be reported to the Scheduler through monitoring. The protocol adopted for the coordination mechanism between the Scheduler and the Real Time Controller is FIPA Query Protocol (http://www.fipa.org). The protocol has been implemented in different ways to fulfill the different functional requirements to achieve coordination.

Figure 4 depicts the schedule deployment protocol between the Scheduler and the Real Time Controller. Once a new schedule has been calculated, the Scheduler deploys it to the workcell by sending a Deploy message. The Real Time Controller receives the updated schedule and passes it to the Resource agents to be deployed. Depending on different deploying results the Real Time Controller may reply to the Scheduler with "*inform-done*" (deployment finished), "*failure*" (deployment failed) or "*not-understood*" if part of the schedule is not understandable.

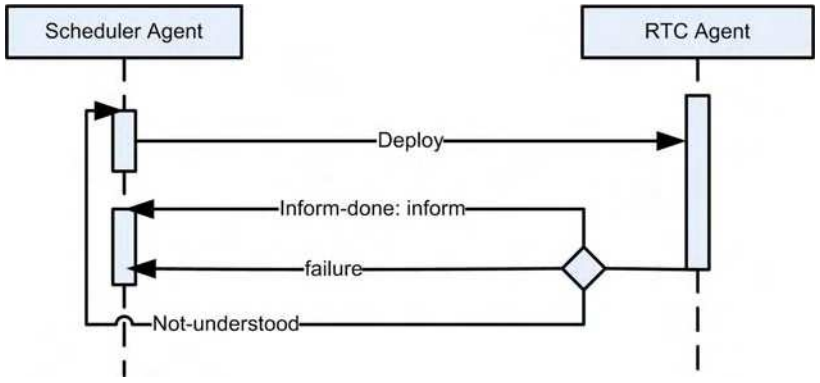


Figure 4 Schedule deployment protocol

Figure 5 shows the disturbance reporting protocol between the Scheduler and the Real Time Controller. Once a disturbance happens in a workcell, the Real Time Controller reports it to the Scheduler by sending a Request message. The Scheduler receives the disturbance report and replies with an "*inform-done*" message. If the reported message is not understandable, it will ask the Real Time Controller to re-send the request by sending a "*not-understood*" message.

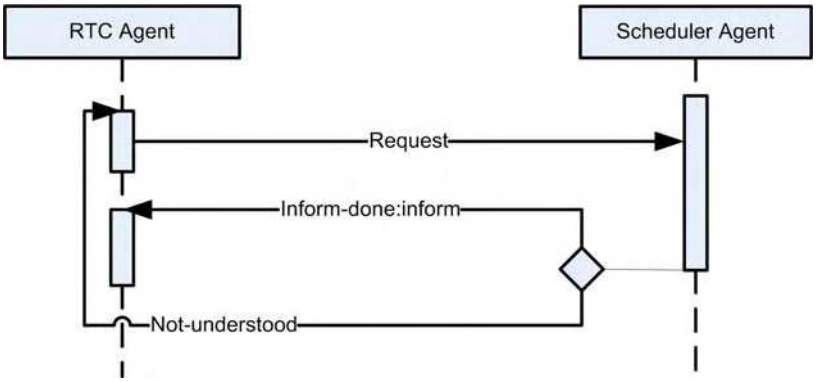


Figure 5 Disturbance reporting protocol

Coordination between the Real Time Controller and the Resource agents utilizes the same FIPA Query Protocol as those used between the Scheduler and the Real Time Controller. The Real Time Controller uses the schedule deployment protocol to distribute a schedule to Resource agents in the workcell and the Resource agents report any disturbances regarding the execution of the deployed schedule, using the disturbance reporting protocol. All agents register their capabilities to the DF before the actual scheduling process starts. If a new Resource agent is integrated into the workcell after the starting time of scheduling, it registers its capabilities to the DF as well. The Real Time Controller is notified by the DF for the updated workcell configuration information and reports any newly integrated resources to the Scheduler using the disturbance reporting protocol. Similarly, if a resource is withdrawn from the workcell after the start time of scheduling, it must report to the Real Time Controller as a disturbance and delete its registration in the DF.

3.3. Agent's Local Decision Making Algorithm

At the workcell level, the schedules are computed by a heuristic scheduling algorithm encapsulated in the Scheduler. The Real Time Controller and the Resource agents provide related information needed by the Scheduler. However, they are not directly involved in the actual

scheduling decision making. Therefore, in terms of decision making schemes, we focus on the Scheduler and, for the Real Time Controller agent and the Resource agents, we give a brief overview.

The Scheduler agent's local decision-making scheme is mainly implemented by a static scheduling algorithm (for the Workcell Scheduling Problem) and a dynamic scheduling algorithm (for the Dynamic Scheduling Problem). Details of these two algorithms are described as follows.

3.3.1. The Static Scheduling Algorithm

The workcell scheduling problem modeled in Section 2.1 is a class of FMS scheduling problems. The majority of the approaches developed for these problems are heuristic oriented (e.g., Nof et al., 1979; Stecke and Solberg, 1981; Chang et al., 1985) and artificial intelligence based (e.g., Fox and Smith, 1984; Smith and Ow, 1985; Sadeh, 91). Some Genetic Algorithm-based approaches (e.g., Fang and Xi, 1997; Qi et al., 2000) can be considered as meta-heuristic methods. Most of these methods use simulation to generate or evaluate schedules. A comprehensive survey on simulation approaches in FMS scheduling can be found in (Chan et al., 2002). Basnet et al. (1994) reviewed the literature concerning the operational aspects of FMSs. Zweben and Fox (1994) provided a comprehensive reference for artificial intelligence-based scheduling approaches. The problem we are focusing on is a class of FMS scheduling problems with a partially overlapping system structure, which has been proven to be NP-hard (Wang et al., 2006). Exact methods, which find optimal schedules, are not practical because of the prohibitive computation demanded. While many approaches proposed in the literature focus on other aspects of the FMS scheduling problems, we propose a dispatching rule-based heuristic algorithm, leveraging the partially overlapping characteristics of the problem.

The heuristic algorithm combines a set of dispatching rules. The basic ideas are: (1) effectively utilize the flexibility provided by the partially overlapping characteristic of the system to balance the work loads, at the same time, (2) assign operations to resources which can finish them faster. Based on the above heuristics, we propose two dispatching rules: Flexible Operation Last (FOL) and Earliest Finishing Time First (EFT). FOL is a composite of two elementary dispatching rules: Longest Processing Time first (LPT) [Panwalkar and Iskander, 1977] and Least Flexible Job first (LFJ). As a composite dispatching rule, FOL is modeled as a ranking

expression that combines LPT and LFJ. This combination can be implemented as the following function:

$$f_{j,k} = \frac{\exp\left(\frac{n_{j,k}}{Q}\right)}{l_{j,k}}$$

where $f_{j,k}$ is the ranking index of FOL, defined as the Flexibility of $o_{j,k}$, $n_{j,k}$ is the number of resources in the workcell that can perform $o_{j,k}$; $l_{j,k}$ is the average processing time of the operation, which is calculated based on historical data. Q is the scaling parameter that can be determined empirically. If Q is very large, the FOL rule reduces to the LPT rule. If Q is very small, the rule reduces to the LFJ rule. FOL selects operations to be scheduled according to their Flexibilities. Operations with higher Flexibilities (short average processing time and more eligible resources) are placed towards the end of the schedule, where they can be used to balance loads more effectively. Once the operation to be scheduled has been selected by FOL, EFT find a resource for the operation based on its completion time. The resource with earliest completion time is chosen.

The algorithm consists of two steps. Before scheduling an operation to a resource, the FOL rule is used to select an operation from Eligible Operation Set[1] (EOS). Then the EFT rule is used to designate the selected operation to a resource based on the current partial schedule. The algorithm implements the EFT rule by considering two factors, the workload of a resource in the current partial schedule which has been established by previous operations and the processing speed of this resource for the selected operation. The resource which can finish the operation first is chosen. Briefly, the algorithm can be described as following.

1. Set EOS= ϕ;
2. If (J_j has unscheduled operations, $\forall J_j \in J$) then
   2.1. Move eligible operations from J_j to EOS;
   2.2. Select an operation to be scheduled from EOS based on FOL rule;
   2.3. Allocate selected operation to a capable machine based on EFT rule.
   2.4. Remove the allocated operation from EOS;
3. Go to 2.

[1] Eligible Operation Set contains operations for which the job release time and operation preceding constraints are satisfied.

4. If EOS is not empty
   4.1. Go to 2.2
5. Terminate

3.3.2. The Dynamic Scheduling Algorithm

In section 2.2 dynamic scheduling problems were classified based on the severity of the disruption happening in the workcells. In the cases of minor disruption, schedule repair procedures are appropriate to provide a fast response and small permutation. In the cases of severe disruption, the re-optimization the algorithms must be considered. While the static workcell scheduling algorithm can be applied directly to the severe disruptions as a re-optimization algorithm, we present a scheduling repair procedure as follows.

In repairing a schedule, the schedule repair algorithm first identifies a number of operations affected by the disruption and must be un-scheduled, then, allocates them to the available resources using the workcell scheduling algorithm. Once a dynamic disruption happens in a work cell, some operations are affected by the event directly. At the same time, others may be affected indirectly by the conflict propagation caused by various constraints. For example, if resource m breaks down at time t_m, an operation $o_{j,k}$ scheduled on m has not started the processing or has not been finished, $c_{j,k} > t_m$, can no longer be processed on m (where $c_{j,k}$ is the completion time of $o_{j,k}$ and assume m cannot be recovered before the end time of the schedule's execution). We call $o_{j,k}$ a directly affected operation. All directly affected operations form a set, denoted by O_{DA}, $o_{j,k} \in O_{DA}$. Operations belong to O_{DA} have to be rescheduled on other capable resources. In addition to operations in O_{DA}, an operation $o_{\hat{j},\hat{k}}$ scheduled on other resources which have precedence constraints with an operation in O_{DA}, say $o_{j,k}$, and has been scheduled after $o_{j,k}$, $s_{\hat{j},\hat{k}} > c_{j,k}$, may need to be rescheduled as well because $c_{j,k}$ may change too much after its reschedule such that $s_{\hat{j},\hat{k}} > c_{j,k}$ is no longer true. We call $o_{\hat{j},\hat{k}}$ an indirectly affected operation. All indirectly affected operations form a set, denoted by O_{IA}, $o_{\hat{j},\hat{k}} \in O_{IA}$. Clearly, not all operations in O_{IA} need to be rescheduled in order to generate a valid schedule repair. To minimize the perturbation to the original schedule, we propose a two step scheduling repair procedure: (1) a schedule repair first un-schedule operations in O_{DA}; (2) if these operations are not sufficient to

enable a new solution to be generated, the unscheduled operations are expanded incrementally to operations in O_{IA} until a solution is found. Based on the modeling and analysis mentioned above, we propose a dynamic scheduling repair algorithm described as following. We assume that resource m breaks down at time t_m and cannot be recovered within the time period to be scheduled in a workcell.

1. Set Eligible Operation Set $EOS = \phi$;
2. Evaluate operations scheduled on m, find all $o_{j,k} \in O_{DA}$, such that $c_{j,k} > t_m$;
3. Find all indirectly affected operations $o_{\hat{j},\hat{k}} \in O_{IA}$ based on the operations in O_{DA};
4. Unschedule operations in O_{DA};
5. If (J_j has unscheduled operations, $\forall J_j \in J$) then
   5.1. Move eligible operations from J_j to EOS;
   5.2. Select an operation $o_{j,k}$ to be scheduled from EOS based on FOL rule;
   5.3. Allocate $o_{j,k}$ to a capable resource based on EFT rule;
   5.4. If no allocation is feasible (violate constraints),
      5.4.1. Find the nearest successor of $o_{j,k}$ which is not in EOS based on precedence constraints;
      5.4.2. Unschedule the successor found;
      5.4.3. Go to 5;
   5.5. Remove the allocated operation from EOS;
   5.6. Go to 5;
6. If EOS is not empty, go to 5.2;
7. Terminate

The Scheduler agent's scheduling states keep changing in dynamic scheduling situations. These changes can be modeled as a Finite State Machine as shown in Figure 6. The Scheduler agent has six scheduling states. Among them, Monitoring, Scheduling, and Deploying are of importance. After Initialization, the schedule has been calculated and deployed to the workcell. The Scheduler agent will be in the state of Monitoring. Dynamic events, which represent the changes from a workcell, can trigger the transition from the Monitoring state to the Scheduling state at which repairing or rescheduling algorithms respond to the occurrence of these events and

work out a new schedule. If dynamic events happened in the process of Scheduling, the rescheduling procedure may be restarted to accommodate the newly occurring events. Once a new schedule is ready, the Scheduler agent changes to the Deploying state where the new schedule is deployed to the workcell. If deployment failed due to unexpected changes in the workcell, it reverts back to scheduling state, and rescheduling will be restarted again.

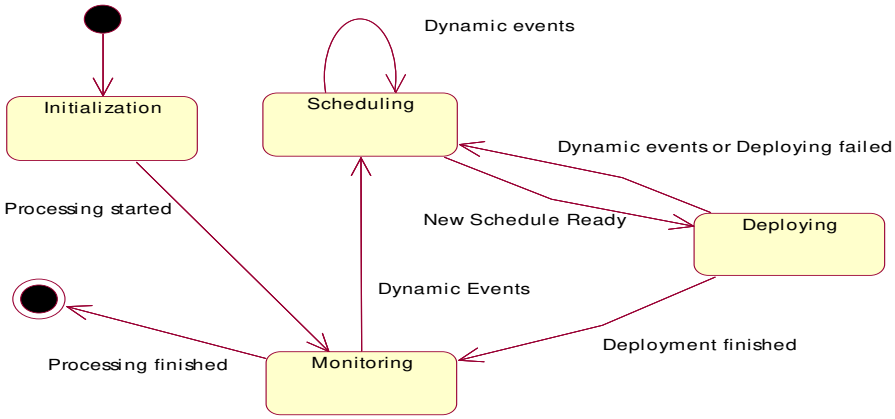


Figure 6 Finite State Machine model of the Scheduler agent

3.3.3. Local Decision Making of Real Time Controller and Resource Agents

The Real Time Controller's local decision-making contains a set of information processing rules to aggregate information reported from the Resource agents to a higher level abstraction which is intelligible by the Scheduler, and to breakdown a schedule computed by the Scheduler into sub-schedules each for a Resource agent. In addition, it also contains control logics used to implement the coordination protocols. The Resource agents' local decision-making mainly contains resource monitoring procedures and control logics for disturbance reporting, schedule deployment, and capability registration.

4. Service Oriented System Integration at Shop Floor Level

The control of multiple workcells on a shop floor has been integrated using an agent-based Web service integration framework (AWS) (Shen et al. 2006). Agent-orientation is an appropriate design paradigm to enable automatic and dynamic collaborations. It is a natural system design and implementation choice in capturing the distributed and dynamic natures of the distributed real time shop floor scheduling. In addition, the software agent paradigm has technical advantages in software modularization, legacy systems integration, distributed problem solving,

and semantics-based interaction with complex and distributed transactions. Established technologies in these areas provide the necessary foundation for the design and implementation of distributed real time scheduling systems. On the other hand, service-orientation is suitable in designing system integration at the shop floor level. The Web Services paradigm is fast evolving and has been supported by several industrial leaders. This led to the development of various supporting technologies for Web Services that enable deploying, publishing, discovering, invoking and composing services in a standard and consistent way. This enables an open, flexible, standardized integration of manufacturing control at the shop floor, enterprise, and supply chain levels.

The merging of service-oriented and agent-based approaches has been a hot topic of research in recent years. Petrie et al. (2003) discussed the shortcomings of Web services standards and how logical AI techniques like declarative commands, agents, and AI planning techniques can be used to address some of these shortcomings. They proposed an FX-Agent approach to address Web services discovery and composition of Web services. Matskin et al. (2005) identified Web services composition as an important issue for efficient selection and integration of inter-organizational and heterogeneous services on the Web and they believed that software agents can help make Web services "pro-active". In their system, provider's Web services are wrapped into individual Providers' Agents on an agent-based marketplace providing services for Customers' Agents. Maamar et al. (2005) presented an agent-based and context-oriented approach that supports the composition of Web services. During the service composition process, software agents engage in conversations with their peers to agree on the Web services that participate in this process. Liu et al. (2004) proposed a conceptual model of agent-mediated Web services for intelligent service matchmaking. In fact, most of research efforts in the literature such as the above mentioned approaches can be roughly categorized as "agentification" of Web services into an agent community. We proposed a different approach for agent and Web services integration (Li et al., 2003). In our AWS framework, an agent core is built into each Web service, so that a Web service is itself an agent. No matter the "agentification" of Web services as agents in a multi-agent system (Maamar et al., 2005) or encapsulation of agents as Web services over the Internet (Li et al., 2003), both approaches share the common goal that, by taking the advantages of Web services and agents, the resultant integrated solution will produce a sophisticated paradigm for Internet computing.

4.1. System Overview

Figure 7 illustrates an agent-based Web service integration for the distributed real time shop floor scheduling system. The proposed system integration is composed of a UDDI and several scheduling service sub-systems, each represents a workcell. As shown in Figure 7, the scheduler of each workcell has two identities, scheduling service and scheduling agent. When working with Real Time Controllers, it is exposed as an agent communicating with ACL. On the other hand, it is exposed as a Web service when working with its peer schedulers on the shop floor. The UDDI is a static repository that provides schedulers' information with standard terms that contains workcell's capabilities and constraints. At the shop floor level, communication among schedulers is based on Web Services standards; at the workcell level, communication among agents is base on ACL messages.

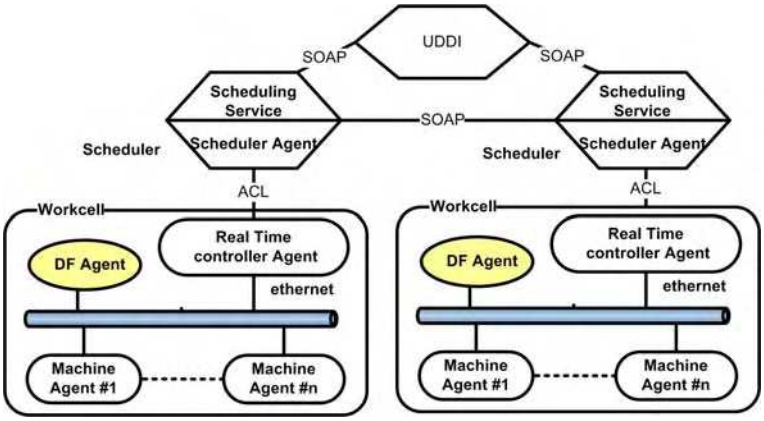


Figure7 Agent-based Web service integration

4.2. Distributed Scheduling

The scheduling problem at the shop floor level contains multiple distributed workcells. Thus, it is actually a distributed scheduling problem. In a workcell, some dynamic events, such as a resource malfunction, may happen. If this disruption cannot be contained inside the workcell, in other words, some disrupted jobs can no longer be scheduled in the same workcell because of the lack of processing capabilities caused by the resource breakdown, the scheduler of this workcell needs to assign the un-scheduled operations to other workcells.

We propose a Contract Net-based distributed scheduling algorithm for the shop floor level scheduling problem in the context of the service oriented integration of multiple workcells. Briefly, the algorithm can be described as follows:

1. Dynamic events at a workcell (call it initiator in terms of the contract net protocol) causes some operations to no longer be able to be processed in the workcell. The initiator finds eligible workcells (call them responders) which can process the un-scheduled operations through UDDI. Note that, a responder does not have to be able to process all un-scheduled operations of the initiator. The definition of eligible workcell requires that the workcell can process at least one of the un-scheduled operations.

2. The initiator sends out a call for proposal (CFP) to all responders. The CFP contains un-scheduled operations and their associated constraints, such as precedence and release dates.

3. The responders try to accommodate the un-scheduled operations from the initiator into their own local schedules based on their scheduling objectives respectively. Once the scheduling on the responders is finished, the responders send proposals back to the initiator. Each of the proposals contains a solution schedule for the un-scheduled operations of the initiator. Note that, the schedule solution from a responder may not contain all un-scheduled operations. Therefore, for some responders, they can just provide schedules for some of the unscheduled operations due to their capability constraints or the availability of the resource processing time. If it is not feasible for the responder to schedule any of the assigned operations or the responder is not interested in the assignment, it sends back a Refuse message.

4. Upon receiving the proposals from the responders, the initiator selects one or a combination of them based on its scheduling objectives to form a final schedule for the unscheduled operations and inform the responders which are included in the final schedule by sending Award messages.

5. If the proposal is accepted by the initiator, the responder deploys the contingent schedule and sends the initiator a message indicating the result of the deployment. If no dynamic events happened during the negotiation process, the responder should be able to successfully deploy the schedule and send the initiator an "*inform-done*" message. If some dynamic events happened during the negotiation process made the contingent schedule impossible to be deployed, a "*failure*" or "*inform-result*" message will be sent to the responder indicating how the contingent schedule is impacted.

4.3. Scheduler Agent Design

In a distributed shop floor scheduling system, workcells are modelled as multi-agent systems. However, at the shop floor level, these multi-agent systems are integrated through Web services. The coexistence of these two different environments poses challenges in systems integration. Because all interactions between the two environments are facilitated by the scheduler, it is not really necessary to implement a general Web services agent gateway between agent and Web service environments. Our approach is to encapsulate the gateway functionality into the scheduler agent, such that it can communicate with both environments concurrently. Based on the CIR-Agent architecture (Ghenniwa and Kamel, 2000), we have designed problem solver, interaction and communication components in the scheduler agent. However, in order to communicate to different environments, both the interaction and communication components into the scheduler agent are split into two parts. As shown in Figure 8, the workcell scheduling interaction and ACL communication are used by the local controller of the problem solver to interact with the real time controller agent in the workcell; the scheduling service interaction and SOAP communication are used by the remote controller of the problem solver to interact with scheduling services provided by other workcells. The problem solver component consists of the local controller, remote controller and scheduling algorithms designed for scheduling at different levels of the shop floor.

Other agents in the system, such as resource agents and real time controller agents are also designed based on the CIR-Agent architecture. Because they only exist in the agent environment, they are not equipped with SOAP communication and Service interaction components.

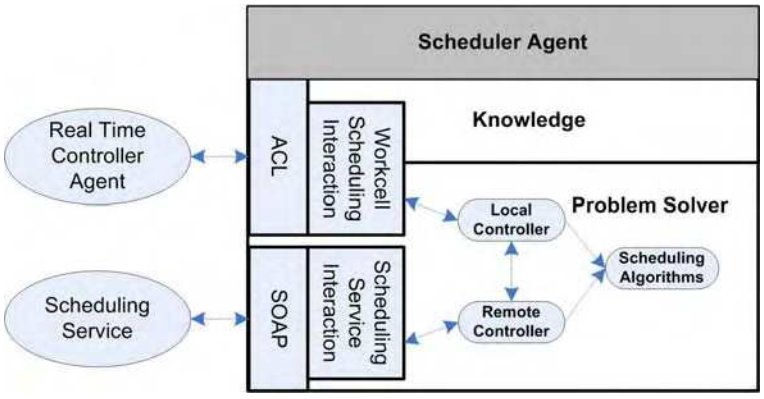


Figure 8 Design of scheduler agent

4.4. System Deployment

The real time distributed shop floor scheduling system has been implemented in Java on the JADE agent development platform (http://jade.tilab.com) and Java Web Services tools (http://java.sun.com/webservices). Figure 9 illustrates a two-workcell deployment of the system. The scheduling system for a workcell contains a scheduler agent, a real time controller agent, several resource agents and a scheduling service. All agents sit on a distributed JADE platform across several hosts. A Java Web Service environment is installed on the same host that the scheduler agent sits on, which allows the scheduling service of a workcell to be connected with the scheduling services of other workcells. Together, the scheduling services and the scheduler agents fulfill the functionality of the real time distributed shop floor scheduling.

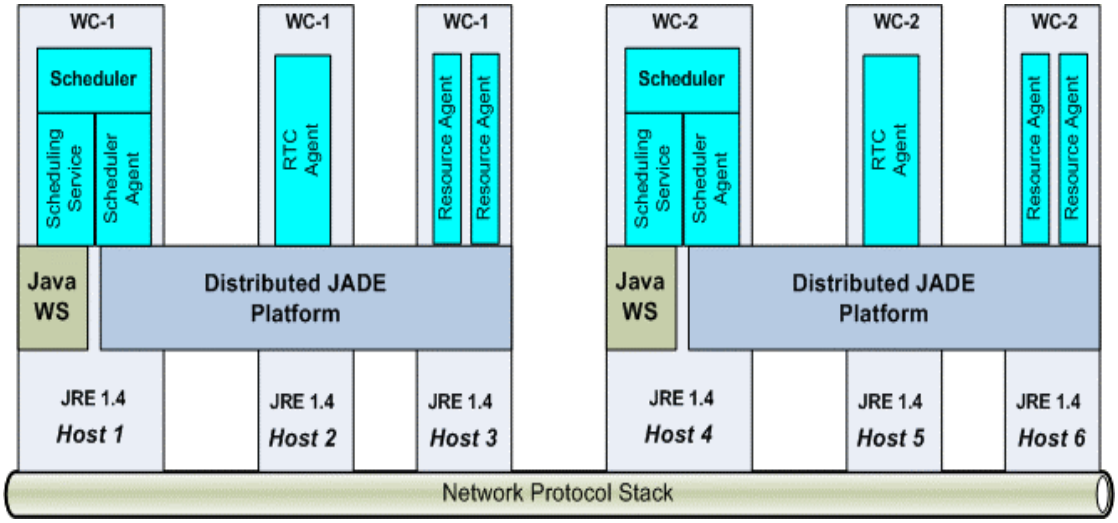


Figure 9 Deployment of the real time distributed scheduling system

5. A Case Study

This section presents a case study which demonstrates how the proposed components, at different levels of the multiple workcell shopfloor, collaboratively provide robust scheduling under the proposed Agent-based Web Service integration framework. For the sake of clearly demonstrating the interactions among workcells, we consider a shop floor with two workcells. Since the contract net (a one to many negotiation protocol) is used for assigning jobs, the proposed system can also accommodate the dynamic scheduling on shop floors with larger number of workcells. As illustrated in Figure 10, the shop floor has two workcells (Workcell A

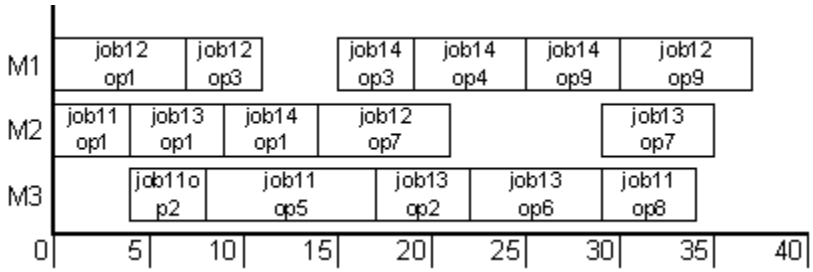
and Workcell B). Each of them has a set of jobs to be scheduled. The experimental scenario goes as follows.

Workcell Scheduling The schedulers of Workcells A and B perform scheduling using the static workcell scheduling algorithm described in Section 3.3.1. At this stage we assume that all jobs can be scheduled in local workcells. The generated schedules are passed to the Real Time Controllers of Workcells A and B respectively. Figure 10(a) shows the assigned schedule for Workcell A and Figure 10(b) shows the assigned schedule for Workcell B in the form of a Gantt chart. Workcell A and Workcell B have the same workcell configuration (only include three machines). The two job sets that need to be allocated have the same configuration as well (however, different job names are used). In the chart, the horizontal bars indicate the length of time allocated to each operation. The x-axis of the chart is subdivided into equal units of time (say hours in our case). The y-axis, on the other hand, lists all the resources in the workcell.

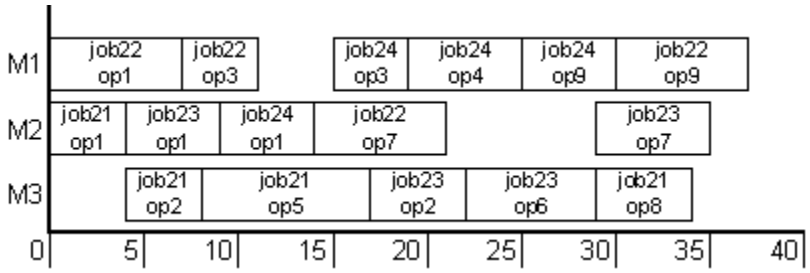
Dynamic Scheduling For dynamic scheduling, we demonstrate how a machine down event is accommodated by the dynamic scheduling algorithm proposed in Section 3.3.2. Say Machine 3 of Workcell A breaks down at hour 25. This disruption is passed to the Scheduler through the Real Time Controller. The schedule repair algorithm first identifies operations (job13-op6, job13-op7, and job11-op8 in this case) that are affected by the disruption and need to be re-scheduled, then, allocates them to available machines using the workcell scheduling algorithm. The repaired schedule is passed to the Real Time Controller and executed in Workcell A. As illustrated in Figure 10(c), job13-op6 is rescheduled on machine 2. To accommodate job13-op6, job13-op7 is shuffled two hours towards the end of the schedule. However, job11-op8 can no longer be processed by Workcell A because Machine 3 is the only one eligible in Workcell A. It needs to be assigned to other workcells on the shop floor by the distributed scheduling algorithm

Distributed scheduling To assign job11-op8 to other workcells, the scheduler of Workcell A first tries to find all eligible workcells on the shop floor that can process the operation through the lookup service provided by the UDDI (Workcell B turns out to be the only eligible one). The scheduler A sends out a service request which contains a call for proposal to Scheduler B including the operation name (job11-op8) and the operation release time (at hour 18 because its precedent operation job11-op5 ends at hour 18). Upon receiving the request from Scheduler A, Scheduler B calculates a solution for job11-op8 and sends back a bid indicating when the operation will be processed. Scheduler A awards this operation to scheduler B. Scheduler B

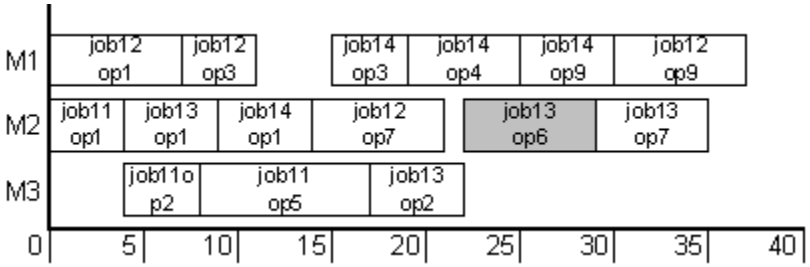
passes the modified schedule (including the assignment of job11-op8) to the Real Time Controller of Workcell B for execution. As shown in Figure 10(d), the operation is added to the end of Machine3's schedule in Workcell B.



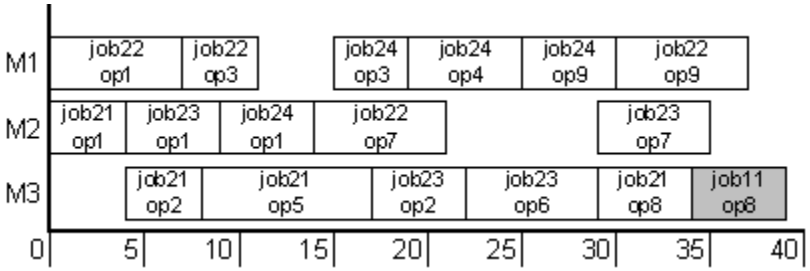
(a) Original schedule of workcell A.



(b) Original schedule of workcell B.



(c) Repaired schedule of workcell A.



(d) Refined schedule of workcell B accommodating the op8 of job 11 from work cell A.

Figure 10 Gantt charts of schedules

To demonstrate the integration of scheduling algorithms more clearly and intuitively, we have intentionally used a simple scenario in this case study. The performance of the algorithms has been tested using more complicated problem sets. Interested readers may refer to (Wang et al., 2005, 2006).

6. Conclusion

Generally speaking, any manufacturing enterprise is distributed. Distribution can be geographical, logical, temporal, or spatial. In the manufacturing domain, it is not uncommon for production to be distributed geographically, sometimes on a continental scale (the automobile industry is a prime example). An enterprise can logically be distributed, reflecting its organizational structure. Organizational structuring can be a necessity in order to decompose the enterprise's problems into manageable chunks and to better exploit available expertise. Scheduling is an essential functionality required by manufacturing control and management at various levels of manufacturing. We have proposed a real time distributed scheduling framework for multi-workcell shop floors. Since distributed environments exist at other levels of manufacturing management, in many cases, it is justified to apply the proposed distributed control structure and even some algorithms (e.g. the distributed scheduling algorithm) to inter-enterprise, enterprise and plant environments as well. For example, at the enterprise level, if a set of customer orders need the cooperation of several divisions of an enterprise, in a dynamic market environment, the scheduling problem involved is a real time distributed one. Currently, most of the enterprise planning and scheduling as in ERP/MRP systems are conducted in a centralized way. One of the criticisms of these systems is the fact that they are complex and inflexible. As a result, there has been interest in the development of decentralized strategies for enterprise systems. We see this as a potential application domain of real time distributed scheduling systems.

In many real world environments, scheduling exhibits a decentralized nature and is conducted through negotiation processes. This observation triggers one of our important future research directions, which is the application of economic based resource allocation mechanisms, such as various auctions, to real time distributed manufacturing scheduling. In many business-to-business transactions, production scheduling parameters (e.g. due dates) are set through a negotiation process between the customer and the service or product provider. In some cases, a firm may consider the possibility of "outsourcing" some time-sensitive orders through a negotiation mechanism if the system is highly congested where completing all the orders in-house would lead to very high tardiness penalties. As many manufacturing

management applications require scheduling functionality in decentralized environments, we see that economic-based scheduling mechanisms are good candidates in such environments.

Acknowledgement

The research work presented in this paper was partially supported by the Material and Manufacturing Ontario and Timelog International Inc. through a collaborative project.

Reference

Baker, A.D., Merchant, M.E., Automatic Factories: How Will They Be Controlled. *IEEE Potentials*, 1993, **12**(4), 15-20.

Baker, A.D., A Survey of Factory Control Algorithms which Can be Implemented in a Multi-Agent Heterarchy: Dispatching, Scheduling, and Pull. *Journal of Manufacturing Systems,* 1998, **17**(4), 297-320.

Basnet, C., Mize, J., Scheduling and Control of Flexible Manufacturing Systems: A Critical Review. *International Journal of Computer Integrated Manufacturing*, 1994, **7**(6), 340-355.

Chan, F.T.S., Chan, H.K., Lau H.C.W., The State of the Art in Simulation Study on FMS Scheduling: A Comprehensive Survey. *International Journal on Advanced Manufacturing Technologies*. 2002, *19*, 830-849.

Chang, Y.L., Sullivan, R.S., Bagchi, U., Wilson, J.R., Experimental investigation of real-time scheduling in flexible manufacturing systems. *Annals of Operations Research*, 1985, **3**, 355–377,

Duffie, N.A., Piper, R.S., Humphrey, B.J., Hierarchical and Non-Hierarchical Manufacturing Cell Control with Dynamic Part-Oriented Scheduling. *Proceedings of the 14th NAMRC*, North American Manufacturing Research Conference, Minneapolis, MN, 1986, pp. 504-507.

Fang, J., Xi, Y., A rolling horizon job ship rescheduling strategy in the dynamic environment. *International Journal of Advanced Manufacturing Technology*, 1997, **13**, 227–232.

Fox, M.S., and Smith, S.F., ISIS: a knowledge-based system for factory scheduling. Expert Systems, 1(1):25-49.

Ghenniwa, H, Kamel, M, (2000) Interaction Devices for Coordinating Cooperative Distributed System. *Automation and Soft Computing*, 1984, **6**(2), 173-184.

Kamel, M., Ghenniwa, H., Partially-Overlapped Systems: The Scheduling Problem. *Design and Implementation of Intelligent Manufacturing Systems*, Parsaei, H. and Jamshidi, M. (Eds.), Prentice-Hall, 1995, pp. 241-274.

Kocjan, W., Dynamic scheduling: State of the art report. Technical Report, T2002:28, SICS, 2002.

Li, Y., Ghenniwa, H.H., Shen, W., Integrated description for Agent-based Web Services in eMarketplaces. *Proceedings of the Business Agents and the Semantic Web Workshop*, Halifax, Nova Scotia, Canada. 2003, pp. 11-17.

Lin, G.Y.-J., Solberg, J.J., Flexible Routing Control and Scheduling. *Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems,* K. E. Stecke and R. Suri, Eds., Elsevier, Amsterdam, 1989, pp. 155-160.

Liu, F., Yao, L., Zhang, W., Liu, H., Zhang, H., A conceptual model of agent mediated Web service. *Proceedings of IEEE International Conference on Services Computing (SCC 2004)*, Shanghai, China, 2004, pp. 638-42.

Maamar, Z., Mostéfaoui, S.K., Yahyaoui, H., Toward an agent-based and context-oriented approach for Web services composition. *IEEE Transactions on Knowledge and Data Engineering*, 2005, **17**(5), 686-97.

Matskin, M., Küngas, P., Rao, J., Sampson, J., Peterson, S.A., Enabling Web services composition with software agents. *Proceedings of the Ninth IASTED International Conference on Internet and Multimedia Systems, and Applications* (IMSA 2005 #477-122), Honolulu, Hawaii, USA, 2005.

McKay, K.N., Safayeni, F.R., Buzacott, J.A., Job shop scheduling theory: What is relevant? *Interfaces*, 1998, 18(4): 84-90.

Neiman, D., Hildum, D., Lesser, V., Sandholm, T., Exploiting meta-level information in a distributed scheduling system. *Proceeding of Twelfth National Conference on Artificial Intelligence* (AAAI-94), 1994.

Nof, S., Barash, M., Solberg, J., Operational control of item flow in versatile manufacturing systems. *International Journal of Production Research*, 1979, **17**(5), 479–489.

Panwalkar, S., Iskander, W., A survey of scheduling rules. *Operations Research*, 1977, **25**(1), 45-61.

Petrie, C., Bussler, C., Service agents and virtual enterprises: A survey. *IEEE Internet Computing*, 2003, **7**(4), 68-78.

Qi, J.G., Burns, G.R., Harrison, D.K., The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *International Journal of Advanced Manufacturing Technology*, 2000, **16**, 609–615.

Ramaswamy, S.E., Joshi, S., Distributed Control of Automated Manufacturing Systems. *Proceedings of 27th CIRP International Seminar on Manufacturing Systems,* Ann Arbor, MI, 1995.

Sadeh, N., Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

Shanker, K., Tzen, Y.J., A loading and dispatching problem in a random flexible manufacturing system. *International Journal of Production Research*, 1985, **23**, 579-595.

Shaw, M.J., Whinston, A.B., Distributed Planning in Cellular Flexible Manufacturing Systems. Management Information Research Center Technical Report, Purdue University, West Lafayette, IN, 1983.

Shen, W., Norrie, D.H., Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems*, 1999, **1**(2), 129-156.

Shen, W., Distributed Manufacturing Scheduling Using Intelligent Agents. *IEEE Intelligent Systems*, 2002, **17**(1), 88-94.

Shen, W., Li, Y., Hao, Q., Wang, S., Ghenniwa, H., A Service Oriented Integration Framework for Collaborative Intelligent Manufacturing. *Robotics and Computer-Integrated Manufacturing*, 2007, **23**(3), 315-325.

Smith, R.G., The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 1980, **C-29**(12), 1104-1113.

Stecke, K.E., Solberg, J., Loading and control policies for flexible manufacturing systems. *International Journal of Production Research*, 1981, **19**(5), 481–490.

Sycara, K., Roth, S., Sadeh, N., Fox, M., Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991, **21** (6), 1446-1461.

Wang, C., Ghenniwa, H., Shen, W., Heuristic Scheduling Algorithm for Flexible Manufacturing Systems with Partially Overlapping Machine Capabilities. *Proceedings of IEEE ICMA 2005*, Niagara Falls, Canada, 2005, pp. 1139-1144.

Wang, C., Ghenniwa, H., Shen, W., Scheduling Multi-operation Jobs in Partially Overlapping Systems. *International Journal of Computer Integrated Manufacturing*, 2006, **19**(5), 453-462.

Zweben, M., Fox, M.S., (Eds.), *Intelligent Scheduling*. Morgan Kaufman Publishers, San Francisco, CA, 1994.