15th Triennial World Congress, Barcelona, Spain

CONTEXT-DEPENDENT AGENTS FOR REAL-TIME SCHEDULING IN MANUFACTURING SYSTEMS

Toshiyuki Miyamoto * **Bruce Krogh** **,[1]
Sadatoshi Kumagai *

* *Dept. of Electrical Eng., Osaka Univ., Suita, Japan*
** *Dept. of Electrical and Comuputer Eng., Carnegie Mellon Univ., Pittsburgh, U.S.A.*

Abstract: Autonomous distributed manufacturing systems(ADMS) consist of multiple intelligent components with each component acting according to its own judgments. The ADMS objective is to realize more agile and adaptive manufacturing systems. This paper presents the introduction of context-dependent agents (CDAs) in ADMS that switch strategies depending on system conditions to achieve better performance than can be realized by agents that use the same strategies under all system conditions. For the real-time job scheduling problem, the paper presents a basic CDA architecture and the results of an extensive empirical evaluation its performance relative to other rule-based schemes based on several common indices for real-time dispatch.

Keywords: Intelligent Manufacturing Systems, Agents, Real-time

1. INTRODUCTION

Recent progress in information technology has opened a door to Next Generation Manufacturing Systems (NGMS, 2000). NGMS are envisioned to be: flexible and adaptable, information- and knowledge-based, and modular to support distribution and autonomy and reconfiguration. In the NGMS program of the Intelligent Manufacturing Systems project (IMS, 2001), architectures for autonomous distributed manufacturing systems (ADMS, 2001), which are manufacturing systems comprising multiple intelligent components (agents) with each component acting according to its own judgments (Lin, 1992) have been investigated. This is in contrast to traditional centralized control architectures, which suffer from a lack of, or a difficulty to implement, agility and flexibility. The ADMS structure makes it easy to design or modify the controller, and makes it easy for different generations of technology to coexist.

This paper addresses a common problem that arises in the design of control agents for the ADMS (and other such systems). Since agent decisions are made independently, it is possible that agents can end up working at cross purposes, leading to a degradation in overall system performance. This occurs when the system conditions are different from the conditions that were assumed when the agent strategies were designed. Agent strategies that are very good in some situations can turn out to be very bad for other situations. This problem is addressed by the concept of *context dependent agents* (CDAs) (Krogh et al., 2001). In a CDA, local decision policies are designed for different assumptions about the overall system context. The appropriate policies are then invoked

[1] The research of B. H. Krogh was supported in part by the Complex Systems/Networks Initiative of EPRI, Palo Alto, CA, and the U.S. Department of Defense.

on-line based on real-time monitoring of system performance.

This paper explores the application of the CDA concept to ADMS, focusing on the problem of real-time job-shop scheduling (Rovithakis *et al*, 2001; Aydin and Oztemel, 2000; Cicirello and Smith, 2001). The scheduling problem for job shops is well known as a NP complete problem (Smith, 1992). Moreover, in applications, there are many performance criteria that apply depending on the situation. Even if a good schedule is computed off-line, it is seldom possible to carry it out without making real-time modifications to accommodate new jobs and unanticipated changes in the system. We are interested in performing the scheduling task using distributed agents that decide their next actions based on a dispatch rules. CDA switching changes the criteria for the agent decisions based on the system status. This paper presents the basic system and CDA architecture, along with a specific switching policy for the job shop scheduling problem. The performance of the CDA approach is compared with the performance of fixed controllers using dispatch rules based on several common indices used rank the urgency of performing given tasks.

2. AGENT-BASED JOB-SHOP CONTROL

Job-shop manufacturing systems consist of three types of physical components: manufacturing cells, AGVs, and storage (e.g., a warehouse). For the purposes of scheduling, the AGV system can be represented as transport delays for moving parts between cells. Figure 1 shows the ADMS agent configuration for a job shop with three cells. Each cell has a cell manager and cell controller. These agents are connected with a communication network. A software agent, called a blackboard, has information about the whole system, and other agents can get information from the blackboard agent as needed. The solid lines indicate communication links. The dashed and single-dotted lines show which agents can query blackboard for the system status and can send state-change information. The dashed lines show that the cell manager can watch the status of its own cell, and can send control messages to it.

In this paper, the job shop capabilities are specified by a table describing the functions that can be performed by each cell and its capacity (see Table 1). A second table specifies the jobs to be done (see Table 2). An order request is given with the number of batches, due date, a sequence of functions and associated process amounts (total effort to perform each function). Order requests are not necessarily known initially, that is, new jobs can arrive while the system is operating. The

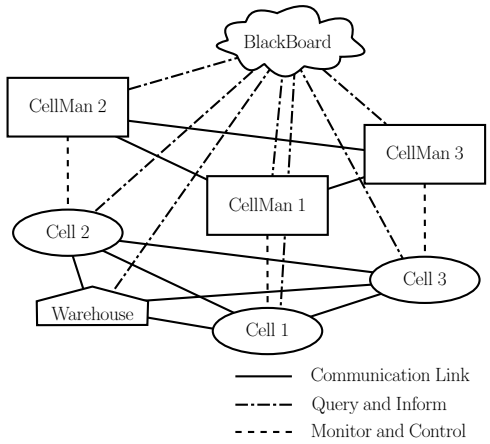

Fig. 1. Agents for job shop problem.

Table 1. Cell Specification

cell	A	B	C	D	E	capacity
Cell_1	1	0	2	0	1	10
Cell_2	1	1	0	1	0	10
Cell_3	0	0	0	2	0	10
Cell_4	2	0	0	0	0	10
Warehouse	-	-	-	-	-	20

Table 2. Order Specification

order	N	D	process sequence			
Odr_1	3	520	D/38	A/36	A/34	B/22
Odr_2	6	470	A/28	A/28	A/20	C/30
Odr_3	4	490	E/36	A/34	C/24	A/24

N : the number of batches
D : due date

functions for a batch must be performed in the order given by the process sequence. In the column "process sequence" of Table 2, A, B, C, D, and E are functions, and integer numbers are process amounts. The process time for a function is calculated by dividing the process amount by the ability of a cell. For example, if Cell_4 processes the second step of Odr_1, its process time is 18. Constants specify the time required to transport jobs between cells and the setup time require to start a function at each cell.

It is assumed that the system is pull style. That is, the cell sends a request to another cell or warehouse when it does not have the next job. The cell agents must make the following decisions:

- What job to do next;
- From which cell or warehouse to a request a job; and
- Whether to give a job in its buffer to an agent requesting it.

The remainder of the paper focuses primarily on the first question.

3. CONTEXT-DEPENDENT AGENTS

Figure 2 shows the proposed CDA architecture. Cell_k is a target to control and x_k is its state

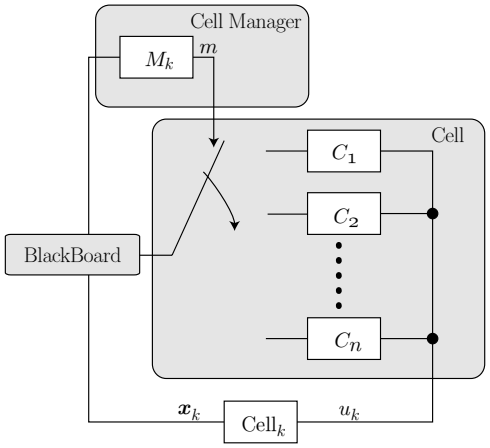

Fig. 2. A cell agent and a cell manager agent implementing a CDA.

vector. The cell agent has a set of controllers $C_1, C_2, \cdots, C_n$. Each controller implements a strategy designed for a certain performance objective. The cell manager can switch the controller according to the control signal m from the cell manager agent. The output u_k of the cell agent is control signal to Cell_k. The cell manager agent collects information about the whole system and decides a suitable control strategy m for the cell agent.

The control strategies are based on dispatch indices described in the following section. These dispatch indices are used to create dispatch rules, where a dispatch rule is a quantitative ranking of the urgency of doing each of the next possible jobs. The cell controller selects the job with the highs ranking (most urgent). Section 5 describes how the dispatch rules are constructed. A set of dispatch rules and a switching strategy for a cell manager is then described in section 6.

4. DISPATCH INDICES

Dispatching is typically performed by quantifying the urgency of performing each of the possible next tasks. For job shops, several dispatching indices have been defined for this purpose, each reflecting a particular performance objective (Jaymohan and Rajendran, 2000; Gargeya and Deane, 1999). The dispatch indices used in this paper is shown in Table 3, where the definitions of the indices use the following notation and terminology (Jaymohan and Rajendran, 2000):

- τ time at which the decision is made.
- D_i due date of order i.
- O_i number of operational steps of order i.
- $\mathcal{F}$ the set of functions.
- $\mathcal{O}$ the set of orders.
- J_{ij} a job at j-th operational step of order i.
- F_{ij} required function for j-th operation of order i.
- V_{ij} required process amount to complete j-th operation of order i.
- $W_{ij}(\tau)$ number of jobs waiting for j-th operation of order i at the instant τ in the whole system.
- $P_{ij}(\tau)$ number of jobs being processed in the j-th operation of order i at the instant τ in the whole system.
- $N_{ij}(\tau)$ number of remaining jobs for j-th operation of order i at the instant τ in the whole system. $N_{ij}(\tau) = \sum_{l=1}^{j} W_{il}(\tau) + \sum_{l=1}^{j-1} P_{il}(\tau)$.
- $Z_{ij}(\tau)$ priority index of j-th operation of order i for the current control strategy at the instant τ (to be computed by the agents).
- $\mathcal{C}$ the set of cells.
- $\mathcal{C}_\text{O}$ the set of other cells.
- $\mathcal{S}(\tau)$ the set of idle cells at the instant τ. The idle cell is defined as follows:
  - at the instant τ, there is no job for it, and
  - in the future, it may process some jobs.
- B_{kf} ability value with cell k for function f.
- $\max B_f$ the maximum value of ability among cells for function f, given as $\max B_f = \max_{k\in\mathcal{C}} B_{kf}$.
- $\text{r}_f(\tau)$ remaining process amount for function f, given as $\text{r}_f(\tau) = \sum_{i\in\mathcal{O}} \sum_{j=1}^{O_i} \alpha_{ijf}(\tau) V_{ij}$, where

$$\alpha_{ijf}(\tau) = \begin{cases} N_{ij}(\tau) & \text{if} F_{ij} = f \\ 0 & \text{otherwise} \end{cases}$$

- $\text{rem}_f(\tau)$ remaining process amount divided by the sum of ability value for function f, given as $\text{rem}_f = \frac{\text{r}_f}{\text{sum}B_f}$.
- T_{ijk} time to process j-th operation of order i by cell k. If $B_{kF_{ij}} > 0$, then $T_{ijk} = \frac{V_{ij}}{B_{kF_{ij}}}$, else T_{ijk} is not defined.

SPT (Shortest Process Time) rule is commonly used to minimize mean flow time and percentage of tardy jobs. In the table, $T_\text{max} = \max_{i\in\mathcal{O}, 1\le j\le O_i, k\in\mathcal{C}} T_{ijk}$, and
$T_\text{min} = \min_{i\in\mathcal{O}, 1\le j\le O_i, k\in\mathcal{C}} T_{ijk}$.

EDD (Earliest Due Date) rule is one of the oldest indices commonly used for easy implementation. With this rule, a cell tries to process a job with earliest due date. In the table, $D_\text{max} = \max_{i\in\mathcal{O}} D_i$, and $D_\text{min} = \min_{i\in\mathcal{O}} D_i$.

BAF (Best Ability First) rule tries to use better function than other cells. In the table, m is the cell itself. In the rest of this paper, each constant is given as follows: $b_1 = 1.05$, $b_2 = 0.9$.

MRF (Most average Remain Function) rule uses the remaining process amount for each function which is dynamic information, and tries to process the critical function at the instance.

Table 3. Dispatch Indices

Rule	Index
SPT	$Z_{ij}^{\mathrm{SPT}} = \begin{cases} \dfrac{T_{\max} - T_{ijk}}{T_{\max} - T_{\min}} & \text{if } B_{kF_{ij}} > 0 \\ 0 & \text{else,} \end{cases}$
EDD	$Z_{ij}^{\mathrm{EDD}} = \begin{cases} \dfrac{D_{\max} - D_i}{D_{\max} - \tau} & \text{if } D_{\max} > \tau \\ \dfrac{\tau - D_i}{\tau - D_{\min}} & \text{otherwise,} \end{cases}$
BAF*	$Z_{ij}^{\mathrm{BAF}} = \dfrac{B_{mF_{ij}}}{\max B_{F_{ij}}}$ When $Z_{ij}^{\mathrm{BAF}} = 1$ for more than one function, the index would be replaced by $Z_{ij}^{\mathrm{BAF}} = \begin{cases} b_1 & \text{if } \forall k \in \mathcal{C}_{\mathrm{O}}, B_{kF_{ij}} < B_{mF_{ij}}, \\ b_2 & \text{if } \exists k \in \mathcal{C}_{\mathrm{O}}, \forall f \in \mathcal{F}, \\ & f \neq F_{ij}, B_{kf} = 0, \\ Z_{ij}^{\mathrm{BAF}} & \text{othersize.} \end{cases}$
MRF*	$Z_{ij}^{\mathrm{MRF}}(\tau) = \dfrac{\mathrm{rem}_{F_{ij}}(\tau)}{\max_{x \in \mathcal{F}} \mathrm{rem}_x(\tau)}$
PJI*	$Z_{ij}^{\mathrm{PJI}}(\tau) = \max_{x=j+1}^{O_i} s_{ix}(\tau)$
LOC	$Z_{ij}^{\mathrm{LOC}}(\tau) = \begin{cases} l_1 & \text{if } J_{ij} \text{ is in buffer of itself,} \\ l_2 & \text{else if } J_{ij} \text{ is in buffer of others, and all of them cannot process } J_{ij}, \\ l_3 & \text{else if } J_{ij} \text{ is in the warehouse,} \\ l_4 & \text{else if } J_{ij} \text{ is in buffer of others, and some of them can process } J_{ij}, \\ l_5 & \text{else.} \end{cases}$
SRF*	$Z_{ij}^{\mathrm{SRF}}(\tau) = \dfrac{S_{ij}(\tau)}{\max_{y \in \mathcal{O}} \max_{z=1}^{O_y} S_{yz}(\tau)}$
LPF*	$Z_{ij}^{\mathrm{LPF}}(\tau) = \begin{cases} n_1 & \text{if } \dfrac{x^*_{mF_{ij}}}{\max_{f \in \mathcal{F}} x^*_{mf}} > th \\ n_2 & \text{otherwise,} \end{cases}$

* New indexes introduced in this paper

PJI (Produce Job for Idle cell) rule looks ahead, and tries to produce a job for idle cells. In the table, $s_{ix}(\tau) = \dfrac{\sum_{k \in S(\tau)} B_{kF_{ix}}}{\mathrm{sum}B_{F_{ix}}}$.

LOC (Loc dependent) rule gives a priority depending on the current location of the job. In the following part of this paper, each constant is given as follows: $l_1 = l_2 = l_3 = 0.1$, $l_4 = 0.05$, and $l_5 = 0.0$.

SRF (Sum of average Remain Function) rule tries to process a job whose remaining process amount is more. In the table,

$$S_{ij}(\tau) = \begin{cases} \displaystyle\sum_{x=j}^{O_i} \mathrm{rem}_{F_{ix}}(\tau) & \text{if } N_{ij}(\tau) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

LPF (Linear Programming Filtering) rule uses the following linear program to minimize the total process time. Let x_{kf} denote a length of time for which cell k processes with function f, and T be a total process time to complete orders. Then the minimization problem can be written

Table 4. Priority Index on Choice Cells or the Warehouse

Index	Condition
0.1	A cell has the job, and it cannot process the job.
0.2	A cell has the job. Though it can process the job, it has never rejected a request for the job.
0.3	The warehouse has the job.
0.4	A cell has the job. It can process the job, and it has rejected a request for the job in the latest δ clocks.
0.5	It does not have the job.

δ is set to 10 in this paper.

$$\begin{aligned} &\text{minimize } T \\ &\text{subject to } \sum_{k \in \mathcal{C}} B_{kf} x_{kf} \geq r_f(\tau), \forall f \in \mathcal{F} \\ &\qquad \sum_{f \in \mathcal{F}} x_{kf} \leq T, \forall k \in \mathcal{C}, \end{aligned}$$

and x^*_{kf} in the table is an argument of the optimal solution. In the table, th is a threshold. In the following, each constants are given as follows: $n_1 = 1.0$, $n_2 = 0.1$, and $th = 0.1$.

5. CELL CONTROLLERS

The control strategy is to prioritize possible next tasks using measures constructed from the dispatch indices. These measures are expressed by the following format:

$$index[, index][; index[, index]]$$

where [xxx] means any number of iterations of xxx, and ',' is a separator for elements and ';' is a separator for sets. The order of the sets of indices indicate their priority in the rule, with the first set having the highest priority. For example, $index1, index2; index3$ means $index1$ and $index2$ are in the set of the first priority and $index3$ are in the second priority set. The index Z_{ij} is calculated by the weighted sum of an index for each rule except a case that some of the rules are filtering rules like LPF as follows:

$$Z_{ij}^{index1} + Z_{ij}^{index2} + \gamma \times Z_{ij}^{index3}$$

where γ is a discounting factor and is set to 0.1 in this paper. If $index1$ is a filtering index, then Z_{ij} of this strategy is calculated as follows:

$$Z_{ij}^{index1} \times (Z_{ij}^{index2} + \gamma \times Z_{ij}^{index2})$$

If one of the indices is used more than one times in different sets, the second or later entry is ignored. The cell agent calculates Z_{ij} for each existing job, and it puts jobs into its schedule in descending order.

Cell tries to process the first job in the schedule. If it does not have the job, it tries to get the job from another cell or the warehouse. In this paper, the following procedure to decide which cell or warehouse to send a request is used:

1 Calculate a priority index for each other cells and the warehouse using Table 4.
2 A target which has the smallest index is selected. If multiple cells have the same index, a cell which has the most jobs in its buffer is selected.

If the request is rejected, it goes back to reschedule, and the reject information is stored. The agent schedules at the next clock. If the request is accepted, the cell gets the job, and it is put in its buffer.

Let ϵ be a estimated time to transport a job from another place. In this paper, it is fixed to 2. If the cell can process the first job in the schedule, it will make the next scheduling at ϵ clock before of expected finish clock of current job.

When a request message from another cell, Cell decides whether it will give a job to the cell or not by the following rule:

- If the requested job is in the schedule, it is rejected. But when this cell has more batches than in the schedule, they could be distributed.
- If more than one cell requests for the same job at the same time, a priority of the cell is given lexicographically.

6. CELL MANAGER: CONTEXT-DEPENDENT SWITCHING

Developing the set of controllers and switching rules is not easy, because even if the strategy seems to work well for some cases, there may be other cases for which the strategy works badly, and vice versa. The following empirical procedure for developing CDA switching rules is used:

1 Select the dispatch indices to be used.
2 Define a collection of dispatch rules (controllers) using the indices.
3 Design rules for switching among the controllers.
4 Generate sets of sample orders randomly.
5 Simulate the CDA with the sample sets.
6 Analyze the result, and find a sample for which the strategy does not work well.
7 Change the switching rule (if necessary), and return to the simulation step.

The CellManager described in this section has been designed to minimize the total process time. This agent decides the next control strategy for its cell agent. The control strategy is the set of dispatching indices with two priorities. Hereafter, the first priority set is called major rules, and the second priority set is called minor rules.

The possible control strategies available to this CellManager are expressed as follows:

$$\{\mathrm{LOC}, \mathrm{SRF} | \{\mathrm{BAF}|\mathrm{MRF}, \mathrm{SRF}\}, \{-|\mathrm{LPF}\}\};$$
$$\mathrm{LOC}, \{\mathrm{SRF}|\mathrm{PJI}\}$$

where $\{A|B\}$ means a choice of A or B, and $-$ means no index is selected. Thus the CellManager has to make four choices; the result of these four choices is a particular control strategy. After extensive simulation studies, the following rules for making these four choices has been developed.

Rule1 If the cell has only one function, CellManager chooses "LOC,SRF". Otherwise, it selects $\{\mathrm{BAF}|\mathrm{MRF}, \mathrm{SRF}\}, \{-|\mathrm{LPF}\}$ depending on the status.

Rule2 To minimize the total process time, each cell should select a function for which the cell has more ability. In this sense, BAF should be used. But if most of remaining jobs require limited functions, then the cell which has the function should use it. In this case, "MRF,SRF" should be used. A guard condition for switching from "BAF" to "MRF,SRF" is as follows:

$$\exists f \in \mathcal{F}, B_{kf} > 0, \mathrm{rem}_f > ave + \Delta, \Delta > \frac{ave}{2},$$

where ave is the average of rem_f, and Δ is the standard deviation of rem_f. Its negation is a guard condition for reverse switching.

Rule3 The CellManager activates LPF, when it has to restrict a use of some functions. The condition to activate LPF is as follows:

$$\exists f \in \mathcal{F}, B_{mf} > 0, \frac{x_{mf}}{\max_{j \in \mathcal{F}} x_{mj}} < th,$$

where m is the cell itself, and th is given in section 4. Its negation is a guard to reverse transition.

Rule4 CellManager selects "PJI" when the following conditions are true:

1 $\mathcal{S} \neq \emptyset$
2 $\exists f, \exists k \in \mathcal{S}, B_{kf} > 0, \frac{rem_f}{\max_{j \in \mathcal{F}} rem_j} > 0.5$

Its negation is a guard to reverse transition.

7. EXPERIMENTAL RESULTS

Simulation software is written in Java using the Multi Agent Net (MAN) library(Miyamoto and Kumagai, 1999).

Randomly generated a set of 100 examples has been used for simulations on 17 strategies. Sixteen cases were done with fixed control strategy, and one case was done with switching by the CDA. Table 5 shows experimental results, and each column is the strategy, the average of the total process time, and the average of the different

Table 5. Experimental Results

strategy	$\overline{T}$	$\overline{T-T^*}$
BAF,MRF,LPF;LOC,SRF	575.3	11.5
BAF,MRF;LOC,SRF	588.1	24.3
BAF,LPF;LOC,SRF	596.2	32.7
BAF;LOC,SRF	585.2	21.4
EDD;LOC,SRF	684.0	120.2
SPT;LOC,SRF	680.1	116.3
CDA	572.3	8.5

from the best total process time. Here, we show only results of selected strategies because of the limitation of papers. Strategy CDA refers to the results generated by implementing the strategy switching rules described in the previous section.

Indices EDD and SPT are well-known and commonly used. Since they are not designed to minimize the total process time, results are the worst.

The result of "BAF;LOC,SRF" was better and "BAF,MRF;LOC,SRF" or "BAF,LPF;LOC,SRF". This means making a complex control rule makes the result worse in some cases.

Control strategies "BAF,MRF,LPF;LOC,SRF" and "CDA" use almost the same set of rules. This means the performance could be improved with the CDA switching.

8. CONCLUSIONS

This paper describes an agent-based real-time scheduling system for job shops. The scheduling problem is an optimization problem with multiple criteria, and which criterion is important at that point may vary depending on the state of the system. The scheduler must react to changes in system conditions. And even for one objective, control strategy should be changed depending on the system configuration. Without CDA switching, realizing a controller satisfies everything would be difficult. This paper showed a basic agent architecture to realize the switching among strategies.

A switching strategy to minimize the total process time has been designed, and a simulation software has been developed. An evaluation about strategies has been done based on experimental data. The experimental results show that the performance could be improved with the CDA.

Future works include designing a switching rules for multiple criteria, developing a systematic method to define a CDA structure and switching rules, and developing an evaluation method of strategies.

REFERENCES

Autonomous Distributed Manufacturing Systems, *http://www.ims.mstc.or.jp/project /project_end/95002/95002_3-1_contents.html*.

Aydin, M. E. and E. Oztemel (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, **33**, 169-78.

Cicirello, V. A. and S. F. Smith (2001). Wasp Nets for Self-Configurable Factories. *Proc. of the 5-th International Conf. on Autonomous Agents*, 473-480.

Gargeya, V. B. and R. H. Deane (1999). Scheduling in the dynamic job shop under auxiliary resource constraints: a simulation study. *Intl. Journal of Production Research*, **37**, 2817-2834.

Intelligent Manufacturing Systems, *http://www.ims.org*.

Jayamohan, M. S. and C. Rajendran (2000). New dispatching rules for shop scheduling: a step forward. *Intl. Journal of Production Research*, **38**, 563-86.

Krogh, B. H. ed. (2001). Context-Dependent Network Agents. *EPRI/DoD Complex Interactive Networks/Systems Initiative: Second Annual Report*. Technical Report no. 1006094, EPRI, Palo Alto, CA, 2001.

Lin, G. Y. J. and J. J. Solberg (1992). Integrated shop floor control using autonomous agents. *IEE Transactions*, **24**, 55-71.

Miyamoto, T. and S. Kumagai (1999). A Multi Agent Net Model and the Realization of Software Environment. *Proceedings of Workshop of Petri Nets to intelligent system development* with 20th International Conference on Application and Theory of Petri Nets, 83-92.

Project Report of NGMS (2000), IMS95002. *http://www.ims.org*.

Rovithakis, G. A., S. E. Perrakis, and M. A. Christodoulou (2001). Application of a neural-network scheduler on a real manufacturing system. *IEEE Trans. on Control Systems Technology*, **9**, 261-270.

Smith, S. F. (1992) Knowledge-based production management: approaches, results and prospects. *Production Planning and Control*, **3**, 350-380.