
Journal of Intelligent Manufacturing 12, 269–279, 2001

A virtual factory based approach to on-line simulation and scheduling for an FMS and a case study

MING-HUNG LIN and LI-CHEN FU

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

Received September 1999 and accepted April 2000

The recent years have witnessed a revolution in computing capabilities. At the same time, modern manufacturing systems are becoming increasingly complex and capital-intensive. Here, we propose a virtual factory wherein an efficient prototyping testbed will be provided. It will be possible to develop detailed models to support system design and operation, to test different system configurations. Besides, It will be possible to manipulate all system features which impact performance, and to see the results of these manipulations, all without disrupting the actual system. This paper is one among very few that tries to represent the virtual factory in an analytic form so that many existing mathematical analyses can be applied. New pseudo resources can be added to form a new virtual environment, and control policy designed by engineers will be evaluated before being issued. Since a state-transition sequence of the virtual factory from the initial state to the final state can be seen as a schedule of the modeled system. An effective schedule of the processing can be obtained by using an A^* based search algorithm, namely, Limited-Expansion A algorithm.

Keywords: Virtual factory, mathematical analyses, limited-expansion A algorithm, flexible manufacturing system

1. Introduction

The recent years have witnessed a revolution in computing capabilities. Hardware and software are cheaper, faster and increasingly powerful. Nowadays, the market place is continuously changing and unpredictable, and hence an efficient prototyping environment is crucial. Modern manufacturing systems are becoming increasingly complex and capital-intensive. Costly system design requires a great deal of modeling and analysis to ensure acceptable system performance after implementation. Modeling and analysis methodologies have progressed tremendously through an enormous amount of researches. The concept of virtual factory or virtual manufacturing is brought forth-taking advantage of powerful computing capability of computers to achieve the goal of manufacturing in computer. In the virtual factory, it will be possible to develop detailed models to support system design and operation, to test different system configurations. It will be possible to manipulate all system features which impact performance, and to see the results of these manipulations, all without disrupting the actual system.

Nowadays, advent of advanced information technologies such as computer networking and 3-dimensional graphics drives more and more industrial companies to reform from traditional human-oriented semi-automation to information-oriented full-automation (Iwata *et al*., 1997; Muller *et al*., 1996; Eccleston, 1996). Lee (1997) developed a prototype of a virtual factory using an expert system shell and a database management system on internet, by which various manufacturing strategies are examined. A virtual factory refers to virtual manufacturing activities carried out not in one central plant but rather in

multiple locations by suppliers and partner firms (Orsak and Etter, 1994; Stefano *et al*., 1997; Bailey *et al*., 1997). To establish a virtual factory, one must have a clear understanding of the manufacturing capabilities of all parties in the production network (Vairaktarakis *et al*., 1996).

Macedo, Hosseini and Syam, (1996) proposed an artificial intelligence based tool that helps to select the partners of a virtual factory. Bodner and Reveliotis, (1997) developed modeling tools that support rapid development of detailed simulation models to assess system performance. Henning and Reveliotis, (1995) presented the simulation functions as an interactive computer-based virtual factory, providing a business insurance policy that allows companies to model their proposed facilities, layout, procedures, and equipment in the computer. Jain (1995) described the role of modeling and analysis during the manufacturing system development cycle and proposed a virtual factory framework for their systematic and efficient use. Finally, Saraswat (1994, 1993) proposed an approach to build a highly flexible computer-controlled manufacturing facility.

To summarize from the previous works on the topic of virtual factory, we can see that some aimed at developing the simulation models to assess system performance. Besides, we know that some emphasized on virtual teaming through multi-enterprise partnerships using the multimedia technologies in the production network and some works on virtual prototyping. However, they are conceptual than realistic for those results are either too coarse or somewhat fragmentary. In this paper, we propose a virtual factory wherein a brand-new efficient prototyping testbed will be provided. New pseudo resources can be added to form a new virtual environment, control policy designed by engineers can be evaluated before being issued, and the performance analysis can be carried out. This paper is one among very few that tries to represent the virtual factory in an analytic form so that many existing mathematical analyses can be applied. Since the states of the virtual factory are deterministic during the evolution of the pertaining state-transition sequence from the initial state, we can directly use the states of the virtual factory to genuinely describe the states of the system. All the reachable states can represent the state space of the modeled system. Then, a state-space search method can be applied to obtain a promising evolution path of states and the state-transition sequences from the initial state to the final state can be viewed as a schedule of the modeled system.

The organization of the paper is as follows. In the next section, we will try to represent the virtual factory in an analytic form in section. In Section 3, an operation model is proposed, which will provide an operating procedure for the virtual factory. Section 4 presents the limited-expansion *A* algorithm for seeking an effective schedule based on the proposed virtual factory. In Section 5, the proposed virtual factory is implemented on an assembly cell, which is compared with a testbed proposed earlier (Huang, Fu and Hsu, 1997). A brief conclusion is given in Section 6.

2. Problem statement, analysis and modeling

The prototyping testbed that we call virtual factory is a complete integrated developing environment. Its mission is to resolve the following problems.

(1) How to define an efficient prototyping skeleton for integrating and automating the manufacturing system software and hardware.

(2) How to define an operating procedure to respond to the requirement from the environment or manufacturing engineers subsequently.

(3) How to provide a schedule for the proposed modeling system.

2.1. *Pseudo domain and physical domain*

Let $X_D(t_k)$ and $X_C(t_k)$ be the discrete and continuous state variable at time t_k, respectively. We denote the time of occurrence of event k as t_k. We assume $X_D(t_k)$ remains unchanged and $X_C(t_k)$ varies continuously with speed vector $v(t)$ at time t in the interval $[t_{k-1}, t_k)$. t_k^- is denoted as the time right before but arbitrarily close to t_k, i.e.,

$$X_D(t_k) = G[X_D(t_{k-1})]$$

and

$$X_C(t_k) = X_C(t_{k-1}) + \int_{t_{k-1}}^{t_k} v(u)du$$

G is a nonlinear function. We denote the state variable at time instant t as $X(t)$. It is possible to deduce from

the state variable at time t_{k-1} and the history of the speed vector v from t_{k-1} to t the reachable state variable at time t, namely, $X(t)$, as follows:

$$X(t) = G[X_D(t_{k-1})] + \int_{t_{k-1}}^{t} v(u)du \qquad (1)$$

The virtual environment consists of a pseudo domain D_p and a physical domain D_s. A pseudo domain D_p is a triplet $\langle PX, PQ, PT\rangle$, where PX, PQ, PT are pseudo resource-state variables, pseudo system-state variables and pseudo time-state variables, respectively, $t^P \in PT$, $PX(t^P) \in PX(t)$, $PQ(t^P) \in PQ(t)$. Pseudo resource-state variables may characterize the changes in the pseudo resources or pseudo data, e.g., the motion trajectory of a pseudo robot that is running on computer, or the pseudo-manufacturing data generated from computer software. The resources here are composed of hardware and software processes. On the other hand, the pseudo system-state variable represents the state of the system and it is updated via computer simulation. The physical domain D_s is also a triplet $\langle SX, SQ, ST\rangle$, where SX, SQ, ST are physical resource-state variables, physical system-state variables and physical time-state variables, respectively, $t^S \in ST$, $SX(t^S) \in SX(t)$, $SQ(t^S) \in SQ(t)$.

The difference between the physical domain and the pseudo domain lies in that the former reflects the reality but the latter only reveals what to be expected if some pseudo resources are going to reality. In general, physical resource-state variables include physical hardware and software, e.g., hardware is like robot, buffer, loader and software is like vision processing package. Physical system-state variable represents the state of the overall manufacturing system. Because pseudo time-state variables PT is the time horizon used by a computer.

Based on the previous definition, we denote a set of virtual resource-state variables as VX, which is given by $VX = PX \cup SX$. Let VQ be a set of virtual system-state variables such that $VQ = PQ \cup SQ$. Considering a transformation function T, where $T : PX \rightarrow SX$, let $x(t^P) = T[y(t^S)], x \in PX$, $y \in SX$. If there exists an inverse transformation function T^{-1} such that $y(t^S) = T^{-1}[x(t^P)]$, then there is an isomorphic relation between x and y, e.g., actual robot PUMA and pseudo robot PUMA.

2.2. *Virtual automaton*

We define a virtual factory as VF, $VF = [VA, LB, SP]$, where VA, LB, SP are denoted as virtual automaton, librarian broker, and specifications, respectively. A virtual automaton VA is a 9-tuple set $VA = \langle Q, I, \rho, q_0, W, O, \psi, \xi, \tau\rangle$, where

(a) Q is a finite set of states, $Q \subseteq VQ$.
(b) I is a finite set of inputs, $I \subseteq VX$.
(c) ρ is a set of the next-state functions. Let σ be a set of the next-state functions such that

$$\sigma : PQ \times SQ \times PX \times SX \rightarrow PQ \times SQ$$

where

$$\rho = \rho_p \cup \rho_s \cup \rho_m$$

$$\rho_p : PQ \times \emptyset \times PX \times \emptyset \rightarrow PQ \times \emptyset$$

$$\rho_s : \emptyset \times SQ \times \emptyset \times SX \rightarrow \emptyset \times SQ$$

and

$$\rho_m \subset \sigma - (\rho_p \cup \rho_s)$$

(d) $q_0 \in Q$ is an initial state, and $W \subseteq Q$ is a set of final states.
(e) O is a finite set of outputs, $O \subseteq PX$; and ψ is output function $\psi : Q \times I \rightarrow O$.
(f) ξ is an output function such that $\xi : PX \rightarrow SX$.
(g) τ is an input function such that $\tau : SX \rightarrow PX$.

Definition 2.1
A virtual factory VF is called a monitoring machine, if we let $Q = Q - PQ$, $I = I - PX$, *and* $\rho = \rho_s$.

Definition 2.2
A virtual factory VF is called a simulation machine, if we let $Q = Q - SQ$, $I = T[SX]$, *and* $\rho = \rho_p$.

Definition 2.3
A virtual factory VF is called an advanced mixing machine, if we let $PT \neq ST$.

Definition 2.4
A virtual factory VF is called an advanced testing machine, if we let $\forall sx \in SX$, *sx is at idle state.*

Let P_M be a performance function such that $P_M : CD \rightarrow R$, where CD is a set of control polices, R is a real number and M is a performance criteria, e.g.,

throughput, completion time. We let the cost function of variable VX be denoted as Cv, $Cv : VX \rightarrow R$, where R is real number. Considering two policies ζ and χ, where ζ is designed with incorporating the virtual factory whereas χ is generated based on the traditional testbed. Both policies are designed to solve the same problem κ. Let VX^{χ} be the set of variables that is used to develop χ, and VX^{ζ} be the set of variables that is used to develop ζ.

Definition 2.5
The gain of virtual factory as to a traditional testbed B for problem κ can be defined as the following equation,

$$G_{vf}(\kappa, B) = \frac{\sum_M P_M(\zeta) \sum_i Cv(vx_i^{\chi})}{\sum_M P_M(\chi) \sum_i Cv(vx_i^{\zeta})} \qquad (2)$$

subject to

$$\forall vx_i \in VX, 0 \leq Cv(vx_i)$$

There is an isomorphic relation $z, z(SX_i) = PX_i$ such that

$$Cv(PX_i) \leq Cv(SX_i)$$

Librarian broker LB is defined as an information manager that explores and extracts the information from various resources, e.g., statistic data generator, equations of queuing network, simulation generator, and historic or temporal data. The specifications SP represent the services which virtual factory can provide.

3. System operation

The block diagram of operation model of virtual factory is given in Fig. 1. The device interface function DI represents data communication and transformation between the physical devices and the VF. Note that the physical devices consist of both hardware and software. There are two time-clock generators, TCG_a and TCG_b. The TCG_a provides the time clock to refresh the elements in the pseudo domain D_p. The TCG_a provides the time clock to refresh the elements in the physical domain D_s.

A manufacturing system MS can be described as a 6-tupleset $MS = \langle MQ, I, \rho, \Gamma, O, \psi \rangle$, where

(a) MQ is a finite set of states, and I is a finite set of inputs.

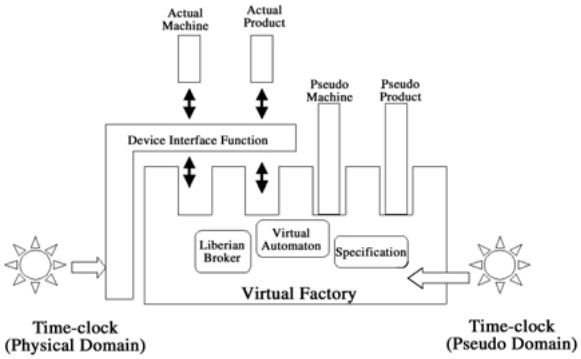

Fig. 1. The block diagram of the operation model for a virtual factory.

(b) Γ is a finite set of control policies or scheduling rules.

(c) ρ is a set of the next-state functions, $\rho : MQ \times I \times \Gamma \rightarrow MQ \times \Gamma$.

(d) O is a finite set of outputs, and ψ is output function $\psi : MQ \times I \times \Gamma \rightarrow O$.

Each control policy being evaluated can be represented as $R = (T_0, p_1, A_1, \ldots, p_n, A_n, T_n)$, where p_i is the ith procedure of R. T_{i-1} is an approximate starting time required by p_i. T_i is an approximate completion time, and $A_i \subset VX$ is the set of resources to be required for p_i. On the other hand, the initial set of resource variables H, where $H =$. The local variable i is initialized such that $i = 0$. The model operation can be summarized in the following steps:

(1) Run the device interface function DI such that $SX = DI[I], SQ = DI[MQ]$.

(2) Check the state of each physical resource-state variable $SX(t^S)$, where $T_{i-1} \leq t^S \leq T_i$. If $SX(t^S) \in A_i$ and $SX(t^S)$ is at an idle state, then $H = H \cup \{SX(t^S)\}$.

(3) Tune the time clock TCG_a, check the state of each pseudo resource-state variable $PX(t^P)$, where $T_{i-1} \leq DI[t^P] \leq T_i$. If $PX(t^P) \in A_i$, then $H = H \cup \{PX(t^P)\}$.

(4) For each $px \in PX \cap H$ and $sx \in SX \cap H$, if there is an isomorphic relation between px and sx, then calculate and maximize the gain of virtual factory $G_{vf}(\kappa, B)$ for $H - \{px\}$ and $H - \{px\}$ individually.

(5) Increase i, and do step (2)(3)(4) until the $i = n$.

Notice that step (1) is just to map each physical device (e.g., a robot) to its counterpart in computer, and always keep the changes of Q and SQ

synchronous. Although the proposed virtual factory is tied closely to the existent manufacturing system, it is our objective not to interfere the running of the manufacturing system. In Step (2), based on this consideration, *SX* can be used only when it is idle.

4. Search-based scheduling method

Once the virtual factory model of an FMS is constructed using the modeling approach described in the previous section, the evolution of the system can be described by the changes of state of the virtual factory. Since the virtual factory model can represent routing flexibility, shared resources, and various lot sizes, as well as concurrency and precedence constraints, all the possible system behavior described here can be completely tracked down within the set of reachable states of the virtual factory.

The general FMS scheduling problem has been shown to be NP-complete. Therefore, we resort to the heuristic search algorithm to solve this problem. The A^* algorithm is known as a heuristic best-first search procedure and guarantees to reach the optimal solution if an appropriate cost function is incorporated. However, this would also mean high memory requirement to store all the nodes generated by the best-first search, and exponential time with respect to the problem size. To avoid these drawbacks, an algorithm, called limited-expansion A algorithm, modified from A^* algorithm, is used. The relation between this algorithm and A^* algorithm is analogous to that between the beam search and the breadth-first search.

The idea of limited-expansion A algorithm we proposed earlier (Sun *et al*., 1994) is similar to that of staged search (Nilsson, 1971), i.e., when the number of nodes in OPEN list exceeds a given number, pruning procedure takes place and only a specified number of the ''best'' nodes (of minimum estimated cost) are kept for further processing. In our approach, the limited-expansion A algorithm assumes that the OPEN list only has a given maximum capacity b. In other words, at most b best nodes are kept on OPEN list at any step.

By modifying the general A^* algorithm, we can construct the following limited-expansion A algorithm for non-delay scheduling.

Step 1: Place initial state M_0 on the list OPEN.
Step 2: If OPEN is empty, terminate with failure.
Step 3: Choose a state M from the list OPEN with minimal cost $f(M)$ and move it from the list OPEN to the list CLOSE.
Step 4: If M is the final state, construct the searched path from the initial state to the final state and terminate.
Step 5: Generate the successor states for each enabled transition, and set pointers from the successors to M.
Step 6: For each successor state M', compute its cost $f(M')$ and do the following:

(1) If state M' is not already on list OPEN or list CLOSE, then put M' on list OPEN.
(2) Else if state M' is already on OPEN and a shorter path is found, then direct its pointer along the current path.
(3) Else if state M' is already on list CLOSE and a shorter path is found, then direct its pointer along the current path and move M' from list CLOSE to list OPEN.

Step 7: If there are more than b states on OPEN, truncate the state M_k from OPEN with maximum cost $f(M_k)$. Go to Step 7.
Step 8: Go to Step 2.

Obviously, by using this approach it is not guaranteed that the algorithm will find the optimal schedule, even if an admissible heuristic cost estimate is used. One could view limited-expansion A as a relaxed version of A^*. If the capacity of the OPEN list in limited-expansion A algorithm is b, we can see that limited-expansion A becomes A^* when $b \to \infty$.

For a problem with n operations in total, no more than bn nodes will be expanded in the worst case. The worst-case complexity of this algorithm in terms of the expanded nodes will then be $O(bn)$, if node cost evaluation is considered as the main source of complexity. Therefore, we can say that limited-expansion A algorithm trade the schedule optimality for reduced memory requirement and lower algorithm's complexity.

5. A case study

5.1. *Robotic flexible assembly system*

In a robotic flexible assembly cell, there are different types of equipment, which cooperate with one another to assemble the parts sent into the system.

Fig. 2. Two-robot assembly cell. 1: Part loader, 2: Parts, 3: Conveyor belt, 4: Optical sensors, 5: CRS robot arm, 6: Overhead CCD camera, 7: rotatory buffer, 8: Grippers of Adept, 9: Adept robot arm.

Our experimental environment is a two-robot assembly cell shown in Fig. 2 that is dedicated to assembling various types of mechanical parts serially sent in through a conveyor belt. The cell is composed of several pieces of hardware. The equipment structure is shown in Fig. 3. There are two prod-

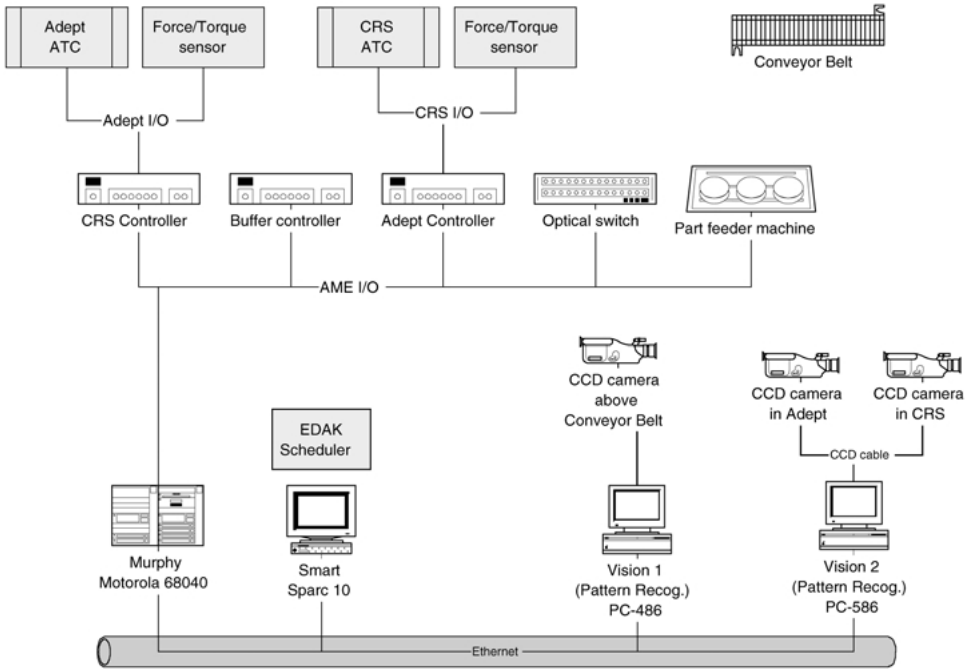

Fig. 3. The equipment structure in an assembly cell.

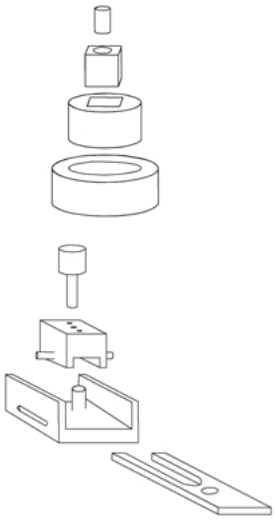

Fig. 4. Two products produced in an assembly cell.

ucts produced in our assembly cell as shown in Fig. 4.

Each product has four parts. The first product is assembled with only vertical insertion operations. The second product includes operations that are more complex. To assemble the second part with the base part for the second product, the robot needs to do vertical insertion and then a rotation to fasten the part with the base part. Sixteen parts can be placed randomly in the pallet of the loader. The loader will load the part onto the conveyor belt one at a time upon request. In the following, the implementation of virtual factory at the flexible assembly cell is described. The modeling is explained first and the interaction between the components are discussed later.

5.2. *Virtual factory modeling*

We model our virtual factory as multiple processes and libraries that work together.

- *Physical Domain:* The robotic assembly cell has two robots, named as Adept and CRS, respectively. The robot Adept has four assemblies sites and the CRS has two. The other types of equipment are explained as in the previous sub-section.
- *Pseudo Domain*:

  (1) The robotic assembly cell has four pseudo robots, respectively named as Pseudo Adept, Pseudo CRS, Pseudo PUMA, and Pseudo A-arm. The robot Pseudo PUMA has three assembly sites whereas the one Pseudo A-arm has two. Since Pseudo Adept is an emulator of Adept, it also means that there is an isomorphic relation between them. Similarly, Pseudo CRS is an emulator of CRS. Here, software modules represent the four pseudo robots.

  (2) In addition to the two products produced in our assembly system, there are two new pseudo products which will be produced virtually. These two new ones are similar to the two old ones, but the color of them are different. Each product also has four parts as mentioned earlier.
- *Virtual Automaton:*

  (1) Six robots are included in the set of virtual resource-state variables *VX*, namely, Pseudo Adept, Pseudo CRS, Pseudo PUMA, Pseudo A-arm, Adept, and CRS.

  (2) The variable *VX* also includes the parts of two products produced in our real assembly cell and the parts of two new pseudo products which will be produced virtually.

  (3) The set of next-state functions ρ in the virtual automaton are partitioned into two parts, one behaves just like the real assembly cell, whereas the other is the set of relations between the pseudo resource-states and the pseudo system-states.
- *Librarian Broker:* We have three databases, namely, mathematics database, manufacturing database, and prediction database. They are in charge of storing the data needed for optimal feedback process, system operation process and prediction diagnosis process, respectively. All information can be extracted through the librarian broker.
- *Specifications:* There are three processes, which are optimal feedback process, multi-objective negotiation process, and prediction diagnosis process. Each process can be performed independently.
- *Device Interface Function:* In this experiment, we use a commercial software package, Wonderware InTouch, which can view and interact with the execution of an entire operation through graphical representations of the produc-

Table 1. Product requirements of experimental scheduling

Product	O_{i1}	O_{i2}	O_{i3}
J_1	M1	M3	M1/M2
J_2	M2	M3	M1/M2

tion processes. We use it to perform communication and transformation between the real assembly cell and the virtual factory.

5.3. *Scheduling results*

We have shown, in the previous section, that an RFAC can be modeled using virtual factory and an effective schedule can be generated using heuristic search method. Now, in order to show how the scheduling approach can really solve the scheduling problem of an FMS, we chose an RFAC as a target system for implementation.

In our experiment of scheduling, we assume that there are two products J_1, J_2 produced in our assembly cell and Table 1 shows the operation requirements of these products. We use the symbols M_1, M_2, and M_3 to represent the Adept and CRS and Pseudo Puma, respectively. The table gives the product requirements of alternative operations and the necessary resources for processing. It also gives the technological precedence constraints among the processes.

The processing time of each operation for a certain product is listed in Table 2. In the table, $OP_{j,i,k}$ means that it is an operation i of product j performed on machine k. Note that processing time includes tool set-up time of a machine for simplicity.

In the heuristic search algorithm, the performance criterion we take is the total completion time and the makespan. This example is solved using the limited-expansion An algorithm with the following heuristic function (Lee, 1992):

$$h(n) = -w^* dep(n)$$

where w is a weighting factor and $dep(n)$ is the total number of accomplished operations for all products at node n.

Table 2. Operation time of products

Operation	*Operation time*
$OP_{1,1,1}$	82
$OP_{1,1,2}$	102
$OP_{1,2,3}$	27
$OP_{1,3,1}$	45
$OP_{1,3,2}$	32
$OP_{2,1,3}$	50
$OP_{2,2,1}$	53
$OP_{2,2,2}$	25

Table 3. Scheduling results (cost only)

Product sizes		*Total completion time*
Product 1	*Product 2*	
1	1	370
2	2	536
5	5	1403
10	10	2435

Several different product sizes of this example are tested and the results, makespan only, are shown in Table 3. Note that, the capacity of the list OPEN is limited to 16 and the weighting factor of the heuristic function $h(n)$ is set to be 3 in our limited-expansion A algorithm.

5.4. *On-line experiment results*

Various experiments are performed under different cases for the proposed model.

Case 1. Monitoring Machine. If all of the resources are limited to real physical resources, it means that we only use the Adept and CRS. The main functionality of virtual factory is monitoring to see whether the given task is executed properly in the system. In the experiment, part loading machine loads part into the real assembly cell randomly, and the robot will either assemble it or store it on buffer. We try to balance the number of parts for different types, but the order of the part is completely randomly. While each robot's assembly sites are full, any finished product will be removed.

Case 2. Simulation Machine. The experiment is performed on the pseudo robots, i.e., to use the pseudo robots: Pseudo Adept and Pseudo CRS. We replace the physical equipment with its emulator. In the experiment, part-loading machine loads part into the virtual factory according to some distribution processes given from the librarian broker.

Table 4. Cost function for each variable

VX	*Pseudo PUMA*	*Pseudo A-ram*	*Pseudo Adept*
Cv	1.7	2.3	1.8
VX	Adept	CRS	Pseudo CRS
Cv	40.8	49.6	1.1
VX	Puma	A-arm	
Cv	70.7	60.3	

Table 5. Experiment results for case 3 and case 4

	Case 3	*Case 4*
Cost of VX (greedy)	221.4	133.9
Cost of VX (A^*)	94.4	6.9
$P_{\text{Throughput(greedy)}}$	0.38	0.39
$P_{Throughput(A^*)}$	0.31	0.16
Gain	1.91	7.96

Case 3. *Advanced Mixing Machine*. It is a complicated experiment. In the experiment, Pseudo PUMA, Pseudo A-arm, Adept and CRS are running together. They form a four-robotic virtual assembly cell. While the part-loading machine loads part into the real assembly cell randomly, the pseudo parts of the two new pseudo products have also been loaded into the virtual factory according to some distribution processes.

Case 4. *Advanced Testing machine*. Similar to the simulation machine, here the difference is to add to pseudo robots (Pseudo PUMA and Pseudo A-arm), and the parts of two new pseudo products into the experiment.

The graphical representation of the virtual factory for our experiment is shown in Fig. 5. The user interface is implemented via a commercial software package Wonderware InTouch.

Considering the assembly problem κ, the parameters that we will be examined are the gain of virtual factory $G_{vf}(k)$ given in Equation 2. We assume the cost function *Cv* for each variable *VX* can be represented in Table 4, where the unit is \$/hour. We denote the number of finished products per second as our throughput. Let us first consider the assembly scheduling designed within a traditional testbed EMFAK (event-driven model of flexible automation kernel (Huang, 1997) which is clearly a greedy method. There the robots can assemble the parts or subassemblies whenever they satisfy the geometric constraints. The schedule designed in use of virtual factory is an A^* algorithm. Because the functions of simulation and monitoring are the same as those for EMFAK, we have no gain in it. The results of gain we examined for cases 3 and 4 explained in the previous sub-section are shown in Table 5. In Table 5, the greeday method is better than A^* programming when the parts are placed randomly in the pallet of the loader. However, we will know that the value of gain

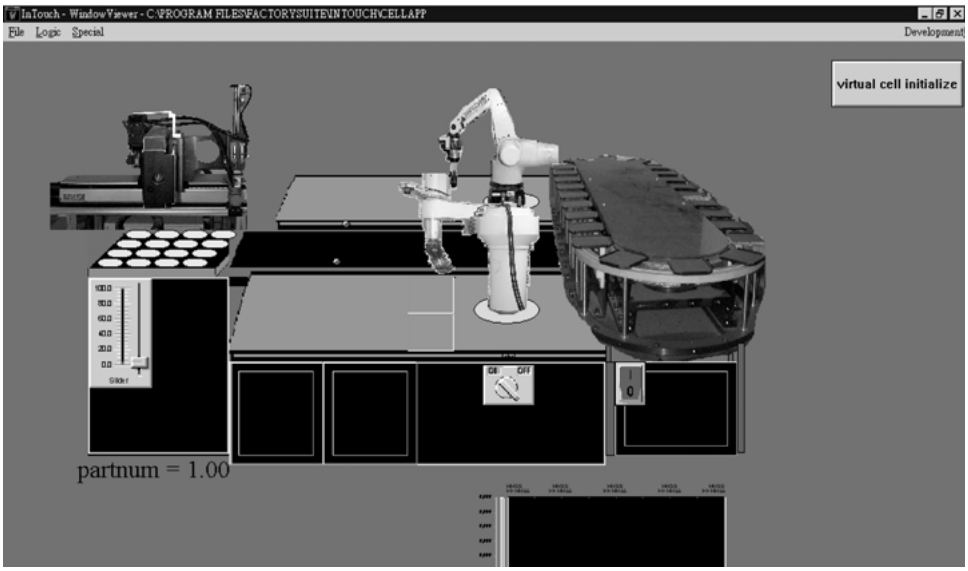

Fig. 5. Graphical representation of an implemented virtual factory.

becomes larger when more pseudo robots instead of real ones are used. We can also use the virtual factory to develop powerful schedules not limited to capacity of resources.

6. Concluding remarks

This paper is one among very few that tries to represent the virtual factory in an analytic form so that much existing mathematical analysis can be applied. The proposed operation model of the virtual factory is capable of coordinating with hybrid methods and targets to build an open system such that it can be self-configured dynamically to respond to a changing market place. New pseudo resources can be added to form a new virtual environment, control policy designed by engineers will be evaluated before issued. In order to obtain an effective schedule while avoiding the NP-complete computing complexity, we applied the so-called limited-expansion A algorithm, which is based on the A^* heuristic search algorithm. This method can give solutions close to the optimal but also can easily be implemented on computers for the memory requirement that is bounded and adjustable.

Although this paper contains both theoretical development and practical examples, there remains various kinds possible improvement. It is needed to have a better modeling formalism that the automaton based virtual factory can be applied. Queueing networks (Hock, Kleinrock:1; Kleinrock:2) and petri nets (Desrochers and AI-Jaar) are well known formalisms for the description and analysis of a variety of discrete event systems. Automaton based virtual factory can be modeled via petri nets or queueing networks. Queueing networks offer a concise graphical description of the service station, queue plus queueing discipline and stochastic routing. Unfortunately queueing networks lack the ability to model more complicated structure like blocking/locking, fork/join, signal/trigger, simultaneous resource and synchronization. Generalized stochastic petri nets (GSPN) are formalism for both qualitative and quantitative analysis of systems. Nevertheless, it is a hard work to represent a scheduling rule via the pure GSPN. Therefore, it is desirable to find a combination of GSPN and queueing networks which benefits from advantages of both formalisms. Thus, our future works are to find a combination of GSPN and queueing networks to model the proposed automaton based virtual factory.

References

Bailey, D., Dessouky, M., Verma, S., Adiga, S., Bekey, G. and Kazlauskas, E. (1997) Virtual factory for manufacturing education, in the Proceedings of the 1997 6th Annual Industrial Engineering Research Conference, pp. 879–884.

Bodner, D. and Reveliotis, S. (1997) Virtual factories: an object-oriented simulation-based framework for real-time FMS control, in the Proceedings of the 1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation, pp. 208–213.

Desrochers, A. A. and AI-Jaar, R. Y. (1995) *Application of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*, IEEE Press.

Di Stefano, F., Lo Bello, F. and Mirabella, L. (1997) Virtual lab: A Java application for distance learning, in the Proceedings of the 1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation, pp. 93–98.

Eccleston, M., (1996) Virtual prototyping. *Manufacturing Engineer*, **75**(3), 129–132.

Henning, K. and Reveliotis, S. (1995) Simulation: The virtual factory, Proceedings of the 1995 38th APICA International Conference and Exhibition Orlando, pp. 12–14.

Hock, N. C. (1990) *Queueing Modeling Fundamentals*, John Wiley & Sons Press.

Huang, H.-S., Fu, L. C. and Jen Hsu, J. Y. (1997) Rapid setup of system control in a flexible automated production systems, in the Proceedings IEEE International Conference on Robotics and Automation, pp. 1517–1522.

Iwata, K., Onosato, M., Teramoto, K. and Osaki, S. (1997) Virtual manufacturing systems as advanced information infrastructure for integration manufacturing resources and Activities. *Annals of the CIRP*, **46**, 335–338.

Jain, S. (1995) Virtual factory framework: A key enabler for agile manufacturing, in the Proceedings of the 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, pp. 247–258.

Kleinrock, L. (1975) *Queueing Systems Volume 1: Theory*, John Wiley & Sons Press.

Kleinrock, L. (1976) *Queueing Systems Volume 2*: *Computer Applications*, John Wiley & Sons Press.

Lee, D. Y. and DiCesare, F. (1992) Experimental study of a heuristic function for FMS scheduling. *JAPAN/USA Symposium on Flexible Automation*, **2**, 1171–1177.

Lee, K. I. (1997) Virtual manufacturing systems—a test-bed of engineering activities. *Annals of the CIRP*, **46**, 347–350.

Macedo, J., Hosseini, J. and Syam, S. (1996) Intelligent reference models for designing virtual factories, in the Proceedings of the 1996 IEEE International Engineering Management Conference Vancouver, pp. 346–350.

McGehee, J., Hebley, J. and Mahaffey, J. (1994) The MMST computer-integrated manufacturing system framework. *IEEE Transaction on Semiconductor Manufacturing*, **7**, 107–116.

Ming-Hung, L. and Fu, L.-C. (1998) Systematic creation and application for virtual factory with object oriented concept, in the Proceedings IEEE International Conference on Robotics and Automation, pp. 304–309.

Muller, P., Svan, E., Terlouw, P. and de Vreede, G.-J. (1996) Multimedia and co-operative work in new product development: A participatory approach to virtual prototyping, in the Proceedings of the 1996 IEEE Conference on Emerging Technologies and Factory Automation, pp. 777–782.

Nilsson, N. J. (1971) *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York.

Orsak, G. C. and Etter, D. M. (1994) Connecting the engineer to the 21st century through virtual teaming. *IEEE Transaction on Semiconductor Manufacturing*, **7**(2), 107–116.

Orfali, R., Harkey, D. and Edwards, J. (1987) *The Essential Distributed Objects Survival Guide*. John Wiley & Sons Press.

Saraswat, K. (1993) Programmable factor for IC manufacturing for the 21st century, in the Proceedings of the IEEE/SEMI International Semiconductor Manufacturing Science Symposium, San Francisco, pp. 2–6.

Saraswat, K. (1994) Adaptable IC manufacturing systems for the 21st century, in the Proceedings of the 1993 Symposium D on Integrated Processing for Micro and Optoelectronics of the 1993 E-MRS Spring Meeting Conference Strasbourg, pp. 131–137.

Sun, T.-H., Chen, C.-W. and Fu, L.-C. (1994) Petri-net based modeling and scheduling for a flexible manufacturing system. *IEEE Transactions on Industrial Electronics*, **41**(6), 593–601.

Vairaktarakis, G., Hosseini, J. and Syam, S. (1996) Forming partnerships in a virtual factory, in the Proceedings of the 1996 27th Annual Meeting of the Decision Sciences Institute, pp. 1394–1396.

VM1 (1994) Virtual manufacturing user workshop final report. *Virtual Manufacturing User Workshop*, Dayton, Ohio, July, 12–13.

VM2 (1994) Virtual manufacturing technical workshop final report. *Virtual Manufacturing Technical Workshop*, Dayton, Ohio, October, 25–26.