BEBR

FACULTY WORKING PAPER NO. 1159

A Distributed Knowledge-Based Approach To Flexible Automation

Michael J. Shaw
Andrew B. Whinston

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign

BEBR

FACULTY WORKING PAPER NO. 1159

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

July, 1985

A Distributed Knowledge-Based Approach to Flexible Automation

Michael J. Shaw, Assistant Professor
Department of Business Administration

Andrew B. Whinston
Purdue University

ABSTRACT

This paper applies automatic planning and distributed problem solving methods to the on-line planning and control of cellular flexible manufacturing systems, consisting of asynchronous manufacturing cells. A knowledge-based approach is used to determine the course of action, resource sharing, and processor assignments. Within each cell there is an embedded nonlinear planning system that executes dynamic scheduling and supervises manufacturing operations. Because of the decentralized control, real-time task assignments are carried out by a negotiation process among cell hosts. The negotiation process is modeled by augmented Petri nets--the combination of production rules and Petri nets--and is executed by a distributed, rule-based algorithm.

1. Introduction

Flexible automation--automation that can handle a large and constantly changing variety of produced items--has played an increasingly important role in the efforts to improve the productivity of the manufacturing industry [15] [23]. The recent progress in computer technologies has accelerated the realization of flexible automation. The use of computers in manufacturing, such as the numerically controlled (NC) machines, adds programmability and thus versatility into manufacturing systems. More important, computers also provide on-line execution of manufacturing planning and decision making. These two capabilities, computerized control and on-line planning, are integrated into a well-orchestrated, automated manufacturing system--referred to as the Computer Integrated Manufacturing Systems (CIMS)--that can produce wide-ranging items efficiently.

In implementing CIMS, the cellular architecture has emerged as the most effective and economical organization for the system [5], [22], [29]. Such a system, referred to as the cellular flexible manufacturing system (CFMS), consists of a collection of "manufacturing cells." As shown in Figures 1 and 2, each cell is controlled by a host computer, which supervises the activities in the cell. Any cell can communicate with other cells through a communications network (e.g., a local area network). The computer architecture is highly distributed, e.g., the host computers used in the National Bureau of Standard facility range from a VAX 11/780 to 8086- or 68000-based single board computers and multiple processor systems [22].

Insert Figures 1 and 2 Here

Intelligent manufacturing planning and control in a CIMS requires knowledge bases that contain information about current tasks, manufacturing procedures, and the production environment [22]. Furthermore, the evolution of the CIMS into distributed architectures has complicated the information processing requirements. Issues that must be considered include the effective planning and problem solving in the distributed environment; the structuring of knowledge bases to facilitate knowledge sharing between system components; and the coordination and communication among manufacturing cells. To achieve these, this paper develops a distributed knowledge-based approach for manufacturing planning and control. In it, an embedded nonlinear planning system in each cell generates production procedures, supervises resource sharing, and guides manufacturing executions. This planning system also keeps track of the manufacturing environment, directing adjustments on the production procedures when necessary.

Moreover, since a job may consist of a set of tasks to be assigned to several manufacturing cells, the system needs to supply a coordinating mechanism that, through the communications network, permits the matching of tasks to cells. To achieve such coordination by decentralized control, three issues need to be addressed:

(1) the design of an interface language that enables effective communication among cell hosts;

(2) the model of the problem-solving process which, through the communications network, dynamically distributes tasks among the cells; and

(3) the programming and execution of this problem-solving process at each cell in a decentralized manner.

Following the distributed problem solving framework [8], [31], this paper uses the "contract net" approach to meet these requirements. A negotiation protocol, a high-level protocol, is used to ensure orderly information transformation and events sequencing between asynchronous, cooperating cells. The underlying idea is to structure the interactions among cells by tasks negotiation.

In order to carry out the process systematically, it is important to have a good representation of the negotiation protocol and to capture the dynamic, concurrent nature of the protocol. Also, this representation should be integrated into executable programs to coordinate task execution.

The augmented Petri net, an integration of production rules and Petri nets, is used to model the negotiation protocol. The automation of this augmented Petri nets leads to a distributed algorithm for task allocation. Since the procedure is designed to be implemented by a variant of a rule-based system--i.e., the control production system, the knowledge-based system in a sense contains two kinds of program knowledge: the knowledge for task planning and the knowledge for intercell cooperation.

The remainder of this paper is organized as follows. Section II characterizes the distributed problem-solving system in the CFMS environment and discusses the planning approach for flexible automation. Section III shows the use of negotiation protocol for coordinating the tasks; this protocol is then represented by the augmented Petri net model. Section IV illustrates the implementation of the negotiation protocol. The final section summarizes the paper.

II. Knowledge-Based Planning

The planning system, when organized as a knowledge-based system, treats knowledge on three levels: data, knowledge base, and control. By contrast, conventional programs treat knowledge on only two levels: data and program. At the data level a knowledge-based system stores declarative knowledge about the goals, the current situation of the world, and the semifinished plan. At the knowledge-base level is stored the domain-specific, procedural knowledge. This knowledge models the actions of the world, and it is often represented by production rules or operators. Finally, at the control level the knowledge about the strategy of plan construction is stored; this is the control knowledge indicating how to select operators, and when to apply them.

Because planning involves exploration of alternative sequences of actions, a symbolic model of the real world, the "world model," represents the environment as the plans evolve. For any given planning problem, the initial condition and the stated goal condition are both treated as instances in the world model. The generation function of a planning system, then, is to construct a course of action that transforms one state of the world model, which contains an initial condition, to a state which satisfies the goal condition. Thus, as shown in Figure 3, a planning system for the CIMS has three basic components:

Insert Figure 3 Here

---

(1) the world model, which contains a symbolic description of the real world. This world model is represented by the collection of first-

order predicates in a database. Any instance of the database is called a state of the world model. Examples of such database elements in a world model are shown in Table 1.

Insert Table 1 Here

(2) The action model, which describes the transformational effects of actions that map states to other states. Such transformations are usually modeled by operators similar to the STRIPS operators defined in [10]. In such an action model, each operator can be specified

< Action - name >	:	< list-of-arguments >
< Precondition >	:	< list-of-precondition-literals >
< Add list >	:	< list-of-add-list-literals >
< Delete list >	:	< list-of-delete-list-literals >
< Resource >	:	< resource-name >
< Duration >	:	< length-of-duration >

In addition to the standard STRIPS formalism--which specifies an action by the add list, delete list, and preconditions--we have also included two more descriptions for each action--the "resource" used during the action, and the "duration" of the action. There are two advantages to this addition: the increased representational power of the action model and the resulting acceleration of conflict detection and conflict resolution. An example is shown in Figure 4.

Insert Figure 4 Here

(3) The inference engine, which directs the plan generation process. It selects a sequence of operators to achieve the goal state from a given initial state.

The linear plans can be generated by any STRIPS-like plan generation system [9], [10]. Such a planning system can use a backward-chaining method in searching for the best actions [26]--i.e., it works backward from the goal state and find a sequence of actions that could produce this goal state from the initial state. The process of plan generation, then, can be viewed as finding the solution path in a search tree. The root of the tree is the goal state, and instances of operators define the branches. The solution path starts with the root (the goal state) and leads to the leaves (the initial state), thereby defining the plan. An example of the search trees generated by the planning system are depicted in Figure 5.

Insert Figure 5 Here

Within this planning framework, the manufacturing process corresponding to each task is modeled by state-changing transformations, represented by operators. If the manufacturing processes for different tasks are independent, then, in principle, they can be executed in parallel. In reality, however, different tasks usually are competing for machines, tools, and other resources; therefore, the planning system must take into account the interactions among the processes. In the planning literature, this interaction problem between tasks have been treated by imposing constraints between planning steps to avoid any potential conflicts [28], [34], [35]. Two kinds of schemes have been used for this purpose: the "critic" mechanism [28], and the "reasoning about resource" scheme [35]. Because the plans generated by these methods are partially ordered, these methods are called "nonlinear planning."

For the flexible manufacturing environment, Shaw [29] used a non-linear plan-generation scheme, implemented in Franz LISP, to derive the desired production plans within each cell. The scheme requires four steps:

Step 1. Generate a linearly-sequenced plan for each task.

Step 2. Identify problematic interactions between the planning steps.

Step 3. Use precedence constraints to avoid conflicts.

Step 4. Identify alternative planning steps to improve the performance.

An example of the partially ordered plan resulting from the scheme is shown in Figure 6, where each of the nodes represents a planning operator.

Insert Figure 6 Here

This approach displays some desirable characteristics for real-time planning and control in the CIMS environment. First, it is goal-directed, i.e., users only need to specify the goals and the planning system would, accordingly, derive the necessary steps on-line. Second, it is dynamically adjustable. New goals can be accomodated while the current production plan is still being executed; also, plans can be modified when unexpected events occur (e.g., tool or machine breakdowns). A "plan revision scheme" is initiated when bottlenecks are detected; the scheme, in turn, seeks to use alternative resources to improve the throughput [29]. In addition, travel paths taken by guided carts or the arm movements of neighboring robots should be analyzed so that any potential conflicts or interferences are avoided [2], [20].

Formally, the problem to be solved in the planning system can be defined as a quadruple

$$PR = \langle S, O, IS, G \rangle$$

where S is the set of states in the database, O is the set of operators, defined as functions $S \rightarrow S$. Both IS and G belong to S; IS denotes the initial state, and G denotes the goal state. The inference engine selects the sequence of operators in the search space based on predefined control strategies.

To perform planning by a distributed system with a group of "agents," the problem PR is decomposed into subproblems. The final plan consists of a collection of plans for these subproblems, coordinated to be applicable to all initial states IS and to achieve the goal state G. The coordination between the planning agents may be accomplished through messages passed between agents. The key issue, then, is how to automate this coordination to result in orderly interactions. The strategy we shall use throughout this paper consists of four phases:

(1) Goal decomposition. A job is decomposed into tasks to be executed by different cells.

(2) Tasks distribution. Tasks are distributed among the cells so that each task is performed by the most appropriate cell available.

(3) Tasks execution. Cells perform the assigned tasks as required.

(4) Tasks synthesis. Finished tasks of the same job are assembled.

This strategy is referred to as the task-sharing strategy. Another widely used strategy is result-sharing--i.e., each agent will independently initiate its problem solving activities and subsequently aggregates its results [8], [19]. In the foregoing task-sharing scheme, the task distribution problem needs special attention because of the loosely coupled, decentralized control structure. The following sections discuss the use of a negotiation process to achieve it.

III. The Contract Net Framework

A. The Negotiation Procedure

In a CFMS, each manufacturing cell specializes in a predetermined part family so as to reduce the set-up cost. Several cells may be assigned to a single job if its components belong to different part families. To utilize the system resources efficiently, each task of a job must be assigned to the most appropriate cell--this process is referred to as task sharing. Task sharing also occurs when a cell is overloaded and needs to distribute some of its tasks.

A widely used metaphor for distributed problem solving is the "society of experts metaphor"--i.e., the system under study is viewed as a group of experts that solve problems cooperatively [4], [8], [17], [24]. Applying this metaphor to the CFMS environment, we will treat each cell's host computer as an agent planning and managing its local tasks. The whole CFMS, then, is a group of such agents with different areas of specialties, and task sharing can be carried out by proper cooperation among the cells. A cell should "negotiate" with other cells to determine how to share tasks. Specifically, announcement-bid-award cycles are used to distribute tasks to appropriate cells.

The anouncement-bid-award cycle is initiated when a manufacturing cell has a task it is not capable of handling. The cell announces the task to other cells to seek assistance. The announcement messages contain three types of task-dependent information:

a) The eligibility specification: listing the qualification for a cell to submit a bid.

b) The task abstraction: providing a brief description of the task to allow interested cells to evaluate the task by comparing it with other announced tasks.

3) The bid specification: specifying the expected format of the bid to be submitted.

A cell in the network keeps an "active-task announcement list" for every machine in the cell and ranks each announced task in the list according to its type (Table A.1). When a machine becomes idle, the cell selects a task at the top of the list and submits a bid to the cell which announced the task (the manager cell). A manager cell may receive several such bids in response to a single task announcement. The award decision is made based on all the bids received, and the manager cell selects the most preferable cell based on a ranking function, calculated by the distance between the two cells, the estimated processing time, and the loading factor of the bidder. The successful bidders are informed of the award through award messages from the manager cells and the task will be transferred accordingly.

B. The Negotiation Protocol

To execute the negotiation process by way of communications activities, a set of communication rules must be established to regulate the interactions between the cells so that they proceed in an orderly fashion. Furthermore, the protocol also has to carry out the task sharing process--the process of distributing tasks among cells. A formal model for the CFMS protocol should contain three components:

1. A formalism for the message content. This task-dependent language describes the task information in messages.
2. A formalism for the message format. This formalism, independent of the task domain, is used to format all the messages, so that the message content can be properly interpreted. For instance, the formalism for a task announcement message is:

```
<task-announcement>: = TASK-ANNOUNCEMENT
                       {name}
                       {task-abstraction}
                       {eligibility-specification}
                       {bid-specification}
```

   An instance of such a message is shown in Figure 7.

Insert Figure 7 Here
----

3. A formalism for the negotiation process. This formalism is used to describe the proper sequencing of actions to carry out negotiation procedure among cells. Since there may be several manager cells at the same time, this formalism needs to describe asynchronous, parallel processes.

The first two formalisms can use the context free grammar with BNF expressions. This method is fairly standard in modeling protocols and will not be discussed here. Details of it can be found in [31], [33]. Subsequent discussions focus on the formalism for task negotiation.

The network on which the negotiation protocol is implemented can be modelled as a three-layer structure. This structure is illustrated in Figure 8. The negotiation protocol is a high-level, problem-oriented protocol governing the coordination among cell hosts. The host-to-host protocol, or the transport protocol, is used to provide reliable communication between processes in different host computers; it is often implemented by programs called transport stations in the host's operating system. The lowest level of the protocol, the transmission protocol, is responsible for the correct transmission packeting and routing of data between cells; the transmission layer actually incorporates the functions of the physical layer, the data-link layer, and the network layer in the more common ISO multilayer protocol model defined in [32]. Based on this layered protocol structure, task sharing among cells can be carried out by the negotiation protocol at the problem-solving layer, with proper communications support from the transport and the transmission layers. Section IV will illustrate the utilization of a graph model, the augmented Petri net, to specify and, subsequently, to program the negotiation protocol.

Insert Figure 8 Here

IV. Modeling and Programming the Negotiation Protocol

A. A Model for the Negotiation Process

A good model for the negotiation process must describe two aspects of task negotiation:

1. a procedural representation of the communication and coordination mechanisms between the cells; and
2. a declarative representation of the local problem solving process within a cell.

We use the augmented Petri net model [36] to achieve these requirements. The augmented Petri net is an integration of two representation models: production rules are used to model the individual events (the declarative representation), and the Petri net is used to model the interactions and temporal relationships between these events (the procedural representation). The augmented Petri net model has been proven effective in modeling asynchronous, concurrent processes where the combination of state variables grows exponentially. In it, each transition corresponds to a production rule and the Petri net structure of the model can be viewed as the interactions between the productions. To understand the mechanism of an augmented Petri net, let us first review some features of the Petri net as a modeling tool.

Originally designed to model process concurrency and precedence relations, the Petri net model has been used to model, specify, and verify communication protocols [6], [27]. The definition of the Petri net follows:

protocol at the problem-solving layer, with proper communications support from the transport and the transmission layers. Section IV will illustrate the utilization of a graph model, the augmented Petri net, to specify and, subsequently, to program the negotiation protocol.

Insert Figure 8 Here

IV. Modeling and Programming the Negotiation Protocol

A. A Model for the Negotiation Process

A good model for the negotiation process must describe two aspects of task negotiation:

1. a procedural representation of the communication and coordination mechanisms between the cells; and
2. a declarative representation of the local problem solving process within a cell.

We use the augmented Petri net model [35] to achieve these requirements. The augmented Petri net is an integration of two representation models: production rules are used to model the individual events (the declarative representation), and the Petri net is used to model the interactions and temporal relationships between these events (the procedural representation). The augmented Petri net model has been proven effective in modeling asynchronous, concurrent processes where the combination of state variables grows exponentially. In it, each transition corresponds to a production rule and the Petri net structure of the model can be viewed as the interactions between the productions. To understand the mechanism of an augmented Petri net, let us first review some features of the Petri net as a modeling tool.

<u>Definition 1</u> (Petri Net)

A Petri net, W, is a quadruple, $W = \langle P,T,I,O \rangle$, where P is the set of places, T is the set of transitions; $I:T \rightarrow P^*$ defines the input function, and $O:T \rightarrow P^*$ defines the output function.

A place is marked if it has one or more tokens; a transition is enabled if each of its input places are marked. The firing of an enabled transition removes one token from each of its input places and adds one token to each of its output places. A token distribution among the available places in a Petri net is called a marking of the net.

Corresponding to each Petri net and an initial marking, Petri net language is defined as follows:

<u>Definition 2</u> (Petri Net Language)

If there exists a Petri net, $W = \langle P,T,I,O \rangle$, a labelling function l for the transition $l:T \rightarrow Z$, and an initial marking λ, then all the possible sequences of transition firings constitute the Petri net language:

$$L(l) = \{ l(\beta) \in Z^* \mid \beta \in T^* \text{ and } \delta(\lambda,\beta) \}$$

where δ is the next-state function. For a sequence of transitions t_{j1}, t_{j2}, ..., t_{jk}, $\delta(\lambda, t_{j1}t_{j2}t_{j3}\dots t_{jk})$ represents that the firing of the transition sequence, t_{j1}, t_{j2}, up to t_{jk}, is legal.

We now proceed to define formally an augmented Petri net:

<u>Definition 3</u> (Augmented Petri Nets)

An augmented Petri net is composed of seven elements:

APN = $\langle P,T,I,O,\lambda,AP,D\rangle$

where $\langle P,T,I,O\rangle$ is a Petri net as defined in Definition 1; λ is the initial marking of this net. The set of transitions, T, also defines the set of productions, with each transition corresponding to one production rule. D is the set of database elements in the production system and AP is the set of active productions whose conditions are satisfied by D.

A transition t in T is "firable" iff

(1) $t \in AP$; and

(2) I(t) is marked; I(t) represents the set of input places of the transition t.

In the augmented Petri net model, since there is a production corresponding to every transition, we can label the transition and the associated production rule with the same labelling function. The Petri net language in the augmented Petri net can thus be seen as either the set of all possible sequences of transitions or, alternatively, as the set of all allowable sequences of production rule invocations. If each transition corresponds to an activity, the Petri net language generates the correct sequence of activities.

Task negotiations for several tasks are usually executed concurrently. The manager cell may be ranking the incoming bids while the potential contractors at the same time are collecting task-announcements and deciding on whether to submit bids. Consequently, the transfer of messages (e.g., task-announcements, bids) from one cell to another requires synchronized activities among the cells involved. Augmented Petri nets can help ensure the correct implementation of these synchronized activities.

To use the augmented Petri net model, the negotiation process can be represented by two subsets: one (Figure 9) models the necessary actions of the manager cell who announces a task to other cells, processing the incoming bids and awards the task to the selected cell; the other sub-net (Figure 10) models the corresponding actions of the cells who receive the task-announcement (the contractor cells). This sub-net deals with the decision on submitting bids.

Insert Figures 9 and 10 Here

Each activity in the process is represented by a production rule, and the interactions among these activities are represented by the Petri net. Each transition in the Petri net (denoted by a bar in the figures) corresponds to one production rule. When a transition is enabled (i.e., all input places are marked), the corresponding rules will determine the firing condition.

Table 2 lists the set of production rules that correspond to the transitions in the two augmented Petri nets in Figure 9 and Figure 10. Rules T1 to T9 correspond to the task-announcement process of the manager cell; rules T10 to T16 correspond to the bid-submitting process of the contractor cell. At each step in the process, the augmented Petri nets guide the negotiation process of all cells so that the activities for task negotiation are correctly carried out.

Insert Table 2 Here

B. Implementing the Protocol

Consistent with the standard hierarchical structure for communications protocol [1], the negotiation protocol represented by the production system listed in Table 2 is specified at a higher level. Necessary support to carry out this high-level protocol is provided by protocol programs of lower layers, such as the transport and the transmission layers in Figure 6. Programming the production system in Table 2 requires three types of literals:

1. Simple predicates; these are state descriptions stored in the database (Table 3);
2. Functions; they return a binary value after their invocations. Functions are used mostly for conditions checking (Table 4);
3. Procedures; these are used to execute corresponding actions when rules are triggered. Some of these actions may include communication operations (Table 5);

Insert Tables 3, 4, 5 Here

Table 6 shows a language description, called PNL, for the Petri nets in Figures 7 and 8. The purpose of expressing Petri nets with a language like PNL is to have a machine-processable representation of the Petri net; the PNL is translated, in turn, into a procedure language recognizable to a host compiler [25]. To generate unambiguous Petri net language, the description and processing of PNL require that unique names be assigned to the places and transitions.

Insert Table 6 Here

C. Controlled Production Systems

The Petri net language discussed in Definition 2 can serve as the "control language" to regulate the invocation of production rules in the production system during its inference process. Such a production system whose control structure is represented explicitly is called a controlled production system.

Using control language in a rule-based production system gives two advantages: first, the execution becomes more efficient because the control language reduces the applicable set of production rules by eliminating irrelevant production rules from consideration; second, since the control structure is explicitly represented by this control language, it can be modified without having to change the contents of the production system. This separation between control and programs is an important feature of the knowlege-based programming [13], [18]. Now let us define a controlled production system and its relation with the augmented Petri net model.

Formally, a production system can be defined as follows:

Definition 4 (Production System)

A production system is a triple

$$PS = \langle R,D,h\rangle$$

where R is the set of production names, D is the set of database elements, and h is the interpretation of the production R, represented as:

$$h:\ R \rightarrow (q,r)$$

q is the set of conditions and r the set of actions corresponding to R. The state of the production system is defined by the contents of the database elements. When the conditions of a production P_1, denoted $q(P_1)$, is satisfied by the current database, then P_1 is said to be "invocable." If P_1 is invoked, then the database is transformed to a new state, denoted by $r(P_1)$.

Definition 5 (Controlled Production System)

A controlled production system is defined as a quadruple

$$M = \langle R,D,h,C\rangle.$$

The subset $\langle R,D,h\rangle$ is a production system defined in Definition 4. A state in the CPS is defined by a pair $S = \langle u,X_1\rangle$ with $u \varepsilon C$ and $X_1 \varepsilon D$. A production rule p is said to be "applicable" if $up \varepsilon C$. A state $\langle up,X_2\rangle$ is said to be derivable from the state $\langle u,X_1\rangle$, denoted

$$\langle u,X_1\rangle \rightarrow \langle up,X_2\rangle$$

If p is applicable at $\langle u,X_1\rangle$ ($up \varepsilon C$) and the database elements in X_1 satisfy the preconditions of p ($q(X_1)$ = TRUE), then the actions of p change X_1 to X_2 ($r(X_1) = X_2$).

The control language in effect guides the allowable sequences of production invocations, i.e., a production is applicable only if it is accepted by the control language. At each stage of the execution, the control language acts to "focus" the control on a subset of the productions, the applicable productions, and prohibits the other productions from being invoked.

Based on Definitions 1 through 5, we can now propose a theorem which equates the augmented Petri net model to a production system controlled by the Petri net language.

<u>Theorem 1</u>:

For any augmented Petri net

$$APN = \langle P,T,I,O,\lambda,AP,D\rangle$$

there exists a controlled production system,

$$M = \langle R,D,h,C\rangle$$

such that APN and M generate the same sequence of production rules.

<u>Proof</u>

For an augmented Petri net APN, the first five elements $\langle P,T,I,O,\lambda\rangle$ can define a Petri net language L(1). (Definition 2)

Since the active set of productions, AP, in APN is generated by matching preconditions of the productions in T against the database elements in D, AP is derivable from the production system $\langle R,D,h\rangle$ in M. (Definition 4)

Now, if we let the Petri net language L(1) in APN be the control language C in M, then:

(1) ($\rightarrow$) If a transition t is firable in APN, t must satisfy (a) $t \in AP$ and (b) I(t) is marked. These are equivalent to the conditions (a) the production t is invocable in the production system $\langle R,D,h\rangle$ and (b) t is accepted by the language C. Therefore, t is applicable in M. (Definition 5)

(2) ($\leftarrow$) If a production p is applicable in M, p must satisfy (a) p is invocable in <R,D,h> and (b) p is accepted by the control language C; these conditions are equivalent to the conditions (a) $p \in AP$ and (b) $p \in L(1)$; therefore the transition corresponding to p is firable in $\langle P,T,I,O,\lambda \rangle$. Thus, p is also firable in APN. (Definition 3)

Q.E.D.

This isomorphism between (1) the augmented Petri net model and (2) a production system model with a separate control language enables us to deal with the task-sharing problem by using the production system listed in Table 1, and the Petri nets (shown in Figure 9 and Figure 10) can serve as the control structure. Such an algorithm is similar to the inference procedure used in production systems, as the one used in OPS5 [11]. The only difference is that in the beginning of each cycle the algorithm picks the "applicable" rules by the control language.

```
Procedure - Task-sharing {executed by a manager cell}
Input:  a task T, consisting of a set of decomposable subtasks ($t_i$)
Begin
Repeat {Based on Controlled Production System}
(Step 1) <Control> --
    Determine the set of productions accepted by the control language
    $\rightarrow$ p';
(Step 2) <Selection> --
    Select the productions among p' which are invocable $\rightarrow$ CF {the
    conflict set};
```

```
(Step 3) <Conflict resolution> —

    Select a production from the CF set according to the conflict
    resolution strategy;

(Step 4) <Execution> --

    Activate the "then" part of the selected production;

Until the goal condition appears in the database;

end {task-sharing}.
```

The procedure uses the data-directed search scheme [3], [7]. In Step 1, p' is the set of production rules whose corresponding transitions are firable by the current marking in the Petri net. Step 2 to Step 4 is the "recognition-action cycle:" a rule is invocable whenever there are database elements that satisfy the conditions of the rule. If more than one rule is invocable, then these rules are collected into the conflict set CF. A conflict resolution strategy is used in Step 3 to select one rule from the set CF. Although many conflict resolution strategies have been considered [3] [21], Step 3 can be straightforward: select the first production rule that is applicable. Step 4, in turn, makes changes in the database according to the "then" component of the selected rule. The cycle continues until either of two conditions occurs:

(1) the goal state is derived and the inference procedure is successful; or

(2) the goal state has not been achieved, but the conflict set in Step 2 is empty.

In the second case, the inference procedure has failed; an exception handling routine will be called upon to take over.

D. Performance

The performance of the task-sharing algorithm is affected by two factors: the performance of (1) the inference procedure in the controlled production system and (2) the negotiation process among the cells. The former is determined by the organization and search strategies of the production system; the latter is determined by the parameters of the negotiation protocol. This section will discuss both aspects.

For a conventional rule-based production executing data-directed inference procedure, the cycle time of the recognize-act cycle is [21]:

$$\begin{aligned} T.cycle &= T.condition\text{-}match + T.action \\ &= (P \times M \times T.match) + ((A/P) \times T.act), \end{aligned}$$

where P = size of the production system;

M = size of the database (number of predicate literals);

T.match = Average time to find a match between preconditions and the database elements;

A = number of action elements of all rules;

T.act = average time needed to execute the action parts of a rule; and

A/P = average number of actions of a rule.

Since the task-sharing algorithm utilizes a control language to screen the applicable production rules first, the cycle time of the production system is modified as

$$T.cycle' = T.control + (P' \times M \times T.match) + ((A/P) \times T.act)$$

where T.control is the time needed to locate the applicable productions by checking the control language. In our case, it is the time taken for the Petri net language to find the firable transitions based on the current markings.

P' is the size of the active production rules decided by the control language; in general, $P' < P$.

The response time of the negotiation process is the sum of four components:

$$T = T_1 + T_2 + T_3 + T_4,$$

where T_1 is the time for making task announcements, T_2 is the time for evaluating announced tasks by the bidders, T_3 is the time for submitting bids, and T_4 is the time for bid evaluation by the manager cell.

Among the four components, T_1 and T_3 are communications delays for transmitting messages, thus they are affected by the load of the communications network. T_2 is the maximum time allowed for the bidder to evaluate tasks before submitting the bid. It should be adaptable to such factors as the deadline of tasks and frequency of job arrivals. If there is a deadline T_D for the job, then the manager cell should impose a time constraint on T_2:

$$T_2 \leq T_D - T_1 - T_3 - T_4 - T_5,$$

where T_5 is the estimated travel time for the workpiece to get to the bidder cell from the manager cell.

E. Summary of the Approach

The production system, the inference engine, and the database for executing task-sharing are integrated into a knowledge-based system called the contract processor, to be implemented in every cell host. In terms of the distributed problem-solving strategy presented in Section II, the primary function of such a contract processor is to distribute tasks among the cells. According to this scheme, the complete distributed planning algorithm is shown in Figure 11.

The major thrust of using augmented Petri nets and the corresponding controlled production system is that it provides a suitably powerful modeling language for executing the task-sharing process. Because of the problem-solving ability of the negotiation protocol, the approach also has flexibility in achieving task sharing in different situations. More research needs to be done in exploring the optimization behavior of the negotiation process. Nevertheless, the knowledge-based approach has proved very effective in its performing the planning and control functions. In particular: (1) It enforces the separation of the knowledge description and the control structure. Knowledge is described by production rules and the control structure is represented by the augmented Petri nets. (2) It applies a uniform approach to both the multi-task planning within a cell and the task-sharing planning among the cells. As such, each cell's knowledge-based planning system contains two types of knowledge: first, knowledge for planning and scheduling; and second, knowledge for task sharing and communication. (3) It uses a decentralized scheme to perform intelligent manufacturing planning and control. The approach, therefore, enjoys the benefits associated with distributed processing systems such as: graceful degradation, modularity, extensibilty, improved performance, and reliability [12].

Moreover, the utilization of the knowledge-based approach for handling information processing also permits future incorporation of expert rules and heuristics. In this context the generalized manufacturing information system for a CIMS becomes a distributed expert system, at each node the knowledge base contains manufacturing operators, task-sharing rules, and judgmental rules. This expert system's task is to utilize the system resources effectively in the dynamically changing environment.

V. Concluding Remarks

This paper has shown an approach for intelligent manufacturing information processing based on the distributed knowledge-based framework. The nonlinear planning system schedules tasks within each cell, and the negotiation protocol coordinates task sharing among cells. A model based on the augmented Petri nets has been developed to specify the negotiation protocol and to capture the dynamic and concurrent nature of the protocol.

The implementation of the protocol by a knowledge-based system enforces the separation of logic and control components in the computer program that executes the protocol; this gives flexibilities in the software design aspect of the implementation. The use of knowledge-based program also enables us to utilize unified representations for both the planning activities and the task negotiation process.

References

[1] G. Bochman and C. Sunshine, "Formal methods in communication protocol design," IEEE Transactions on Communications, pp. 624-631, 1980.

[2] D. Bourne and P. Fussell, "Designing programming languages for manufacturing cells," The Robotics Institute, CMU-R1-Tr-82-5, Carnegie-Mellon Universsity, 1982.

[3] B. Buchanan and R. Duda, "Principles of rule-based expert systems," Computer Science Department, HPP-82-14, Stanford University, 1982.

[4] B. Chandrasekaran, "Natural and social system metaphors for distributed problem solving," IEEE Trans. on Systems, Man and Cybernetics, Vol. 11, No. 1, pp. 1-5, January 1981.

[5] Cutkosky, et. al., "Precision machining cells within a manufacturing system," The Robotics Institute, Carnegie-Mellon University, 1983.

[6] A. Danthine, "Protocol representation with finite state models," IEEE Transactions on Communications, pp. 632-642, 1980.

[7] R. Davis, et. al., "Production rules as a representation for a knowledge-based consultation program," Artificial Intelligence, Vol. 8, pp. 15-45, 1977.

[8] R. Davis and R. Smith, "Negotiation as a metaphor for distributed problem solving," Artificial Intelligence, Vol. 20, pp. 63-109, 1983.

[9] R. E. Fikes, P. E. Hart and N. J. Nilson, "Learning and executing generalized robot plans," Artificial Intelligence, 3(4), pp. 251-288.

[10] R. E. Fikes and N. J. Nilson, "STRIPS: A new approach to the application of theorem proving to problem solving," Artificial Intelligence, 2(3/4), 1971, pp. 189-208.

[11] C. Forgy, OPS5 User's Manual. Department of Computer Science, CMU-CS-81-135, Carnegie-Mellon University, 1981.

[12] P. H. Enslow, "What is a Distributed Data Processing System," Computer, Jan. 1978, pp. 13-21.

[13] M. Georgeff, "Procedural control in production systems," Artificial Intelligence, Vol. 18, pp. 175-201, 1982.

[14] C. Hewitt, "Viewing control structures as patterns of passing messages," Artificial Intelligence, Vol. 8, pp. 323-364, 1977.

[15] G. K. Hutchingson, "Flexibility is key to economic feasibility of automating small batch manufacturing," Industrial Engineering, pp. 77-86, June 1984.

[16] R. Kieburtz, "A hierarchical multicomputer for problem-solving by decomposition," Proceedings of the IEEE Distributed Computing Systems, pp. 631-71, 1979.

[17] W. Kornfeld and C. Hewitt, "The scientific community metaphor," IEEE Systems, Man and Cybernetics, 1981.

[18] R. Kowalski, "Algorithm = logic + control," Communications of the ACM, Vol. 22, pp. 424-436, August 1979.

[19] V. Lesser and D. Corkill, "The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving networks," The A.I. Magazine, Vol. 4, No. 3, pp. 15-33, 1983.

[20] T. Lozana-Perez, "Robot programming," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A.I. Memo No. 698, 1982.

[21] D. McDermott and C. Forgy, "Production system conflict resolution strategies," Pattern Directed Inference Systems, D. Waterman and F. Hayes-Roth, Eds., Academic Press, 1978.

[22] C. McLean, M. Mitchell and E. Barkmeyer, "A computer architecture for small-batch manufacturing," IEEE Spectrum, pp. 59-64, May 1983.

[23] M. E. Merchang, "Production: a dynamic challenge," IEEE Spectrum, pp. 36-39, May 1983.

[24] M. Minsky, The society theory of thinking," Artificial Intelligence - An MIT Perspective, P. Winton, Ed., MIT Press, 1979.

[25] R. Nelson, L. Haibt and P. Sheridan, "Casting Petri nets into programs," IEEE Trans. on Software Engineering, Vol. SE-9, No. 5, pp. 590-602, September 1983.

[26] R. Nelson, Principles of Artificial Intelligence. Palo Alto: Tioga, 1980.

[27] J. Peterson, Petri Net and the Modeling of Systems. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[28] E. D. Sacerdoti, A Structure for Plans and Behavior. New York: North-Holland, 1977.

[29] M. J. Shaw, "The Design of a Distributed Knowledge-Based System for Intelligent Manufacturing Information Systems," Ph.D. Thesis, Purdue University, 1984.

[30] M. J. Shaw and A. B. Whinston, "Automatic Planning and Flexible Scheduling: A Knowledge-Based Approach," Proceedings IEEE International Conference on Automation and Robotics, St. Louis, 1985.

[31] R. G. Smith, "The contract net protocol: high level communication and control in a distributed problem solver," IEEE Transactions on Computer, Vol. 29, pp. 1104-1113, 1980.

[32] A. Tanenbaum, Computer Networks. New Jersey: Prentice-Hall, 1981.

[33] A. Teng and M. Liu, "A formal approach to the design and implementation of network communication protocol," Proceedings of COMPSAC, pp. 114-128, 1978.

[34] S. Vere, "Planning in time: windows and duration for activities and goals," Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, pp. 246-266, May 1983.

[35] D. E. Wilkins, "Domain independent planning: representation and plan generation," Artificial Intelligence, 22, pp. 269-301, 1984.

[36] M. Zisman, "Use of production systems for modelling asynchronous concurrent processes," in Pattern Directed Inference Systems, D. Waterman and F. Hayes-Roth, Eds., Academic Press, 1978.

APPENDIX

Insert Table A.1 Here

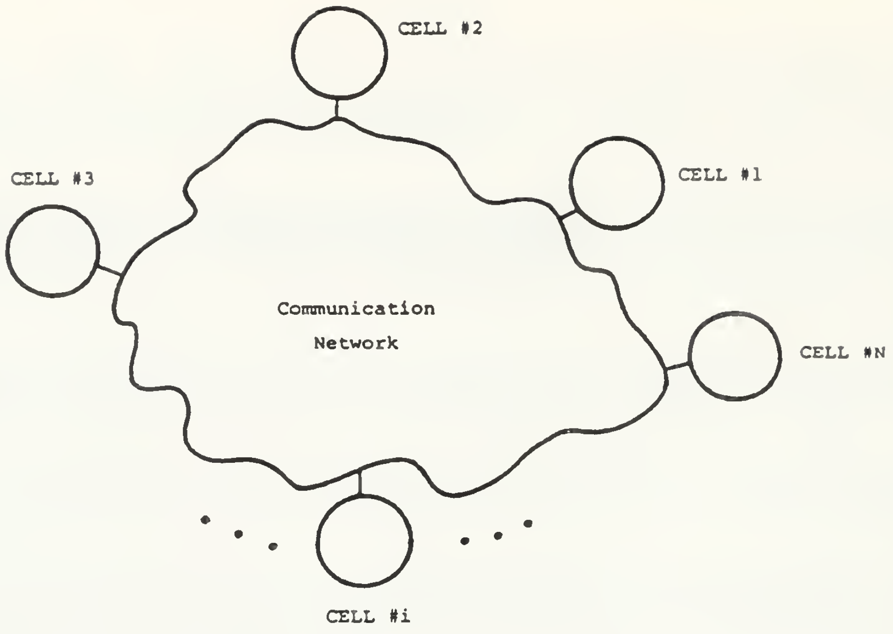

Figure 1. A Network of Manufacturing Cells

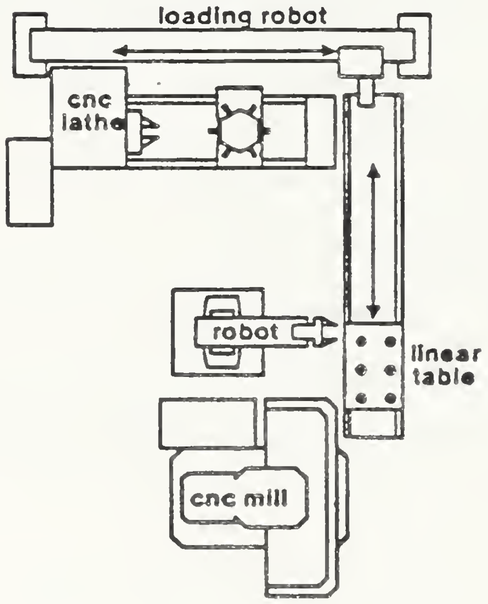

Figure 2 The Organization of a Manufacturing Cell
(Adapted from Cutkosky 1983)

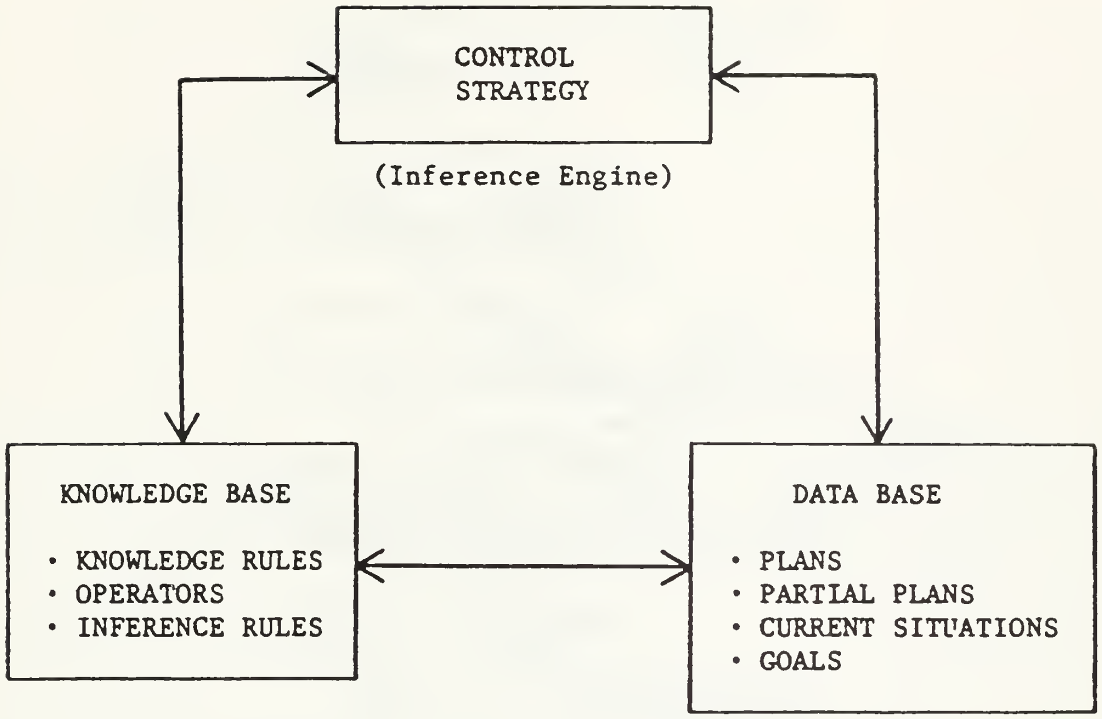

Figure 3. Basic Structure of a Planning System

TRANSFER(M, M', PT, t) : Transfer part PT from machine M to machine M' at time t.

Precondition : FINISH-OP(M, OP, PT, t)

DIFFERENT(M, M')

PT-NEXTOP(OP, OP', PT)

MACH-OP(M', OP')

IDLE(M', t)

Add-list : MACH-PT(M', OP', PT, t)

IDLE(M, t)

Delete-list : FINISH-OP(M, OP, PT, t)

IDLE(M', t)

Resource : M'

Duration : 2

NEXTOP(M, OP, OP', PT, t) : Perform operation OP' on part PT following operation OP on the same machine M.

Preconditon : FINISH-OP(M, OP, PT, t)

PT-NEXTOP(OP, OP', PT)

MACH-OP(M, OP')

Add-list : MACH-PT(M, OP', PT, t)

Delete-list : FINISH-OP(M, OP, PT, t)

Resource : M

Duration : 0

Figure 4. Examples Of Operators In The Action Model

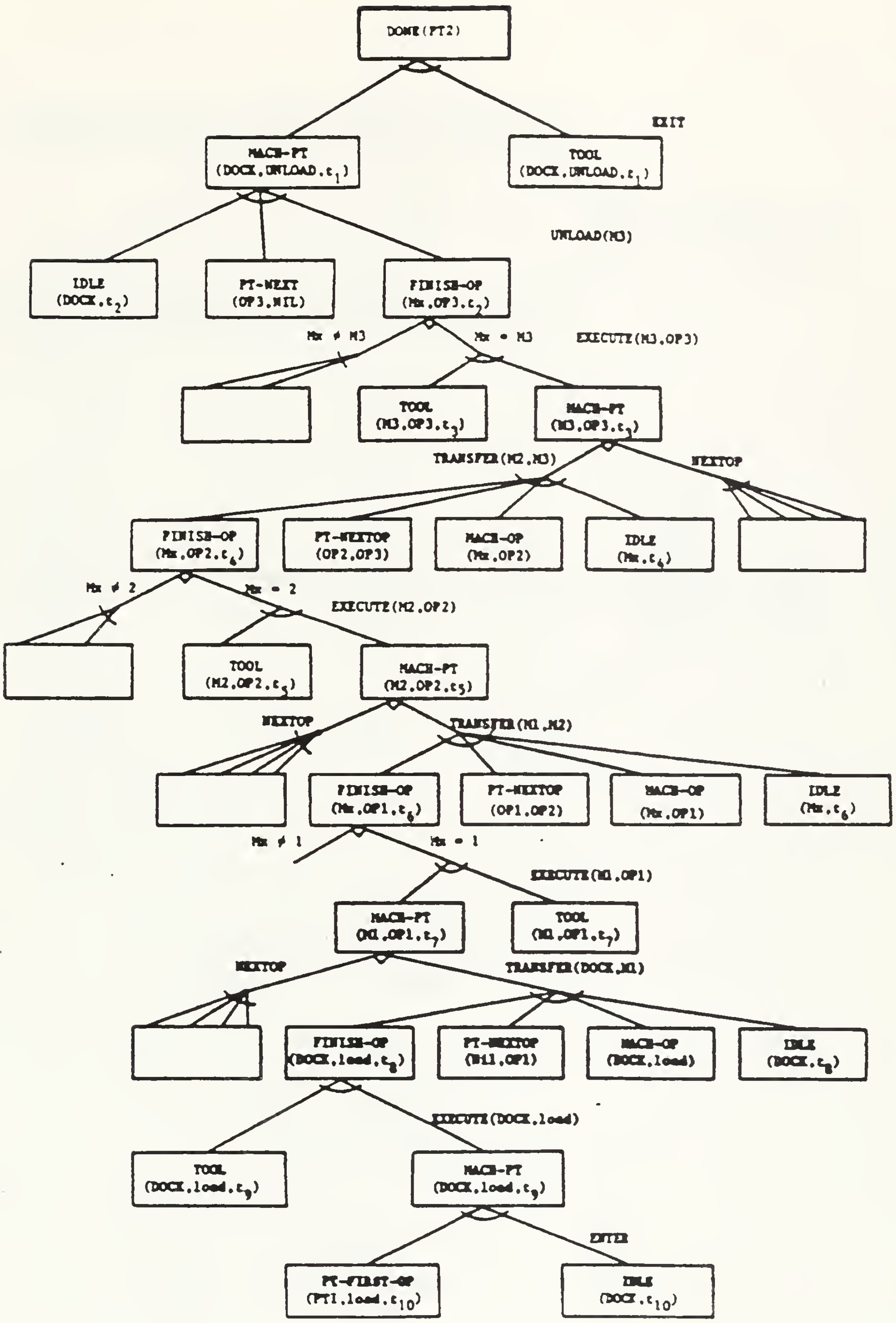

Figure 5. The Search Tree Generated for the Planning of One Job

$$\begin{tikzcd}
 & & & p4 \arrow[r] & p5 \arrow[r] \arrow[ddr] & p6 \arrow[r] & p7 \arrow[r] & p8 \arrow[r] & p9 \arrow[r] & p10 & \\
p1 \arrow[r] & p2 \arrow[r] & p3 \arrow[ur] \arrow[dr] & & & & & & & & \\
 & & & q1 \arrow[r] & q2 \arrow[r] & q3 \arrow[r] & q4 \arrow[r] & q5 \arrow[r] & q6 \arrow[r] & q7 \arrow[r] \arrow[uulll] & q8
\end{tikzcd}$$

Figure 6. A Partially-ordered Network Generated by the Nonlinear Planning System

Eligibility Specification

MUST-HAVE: CONVEX-CUTTER

Task Abstraction

TASK-TYPE: Milling

BATCH-SIZE: 45

CELL-CODE: 0015

Bid Specification

COMPLETION TIME

Expiration Time

1200/3/10/1984

Figure 7. An Example of the Task-Announcement Message

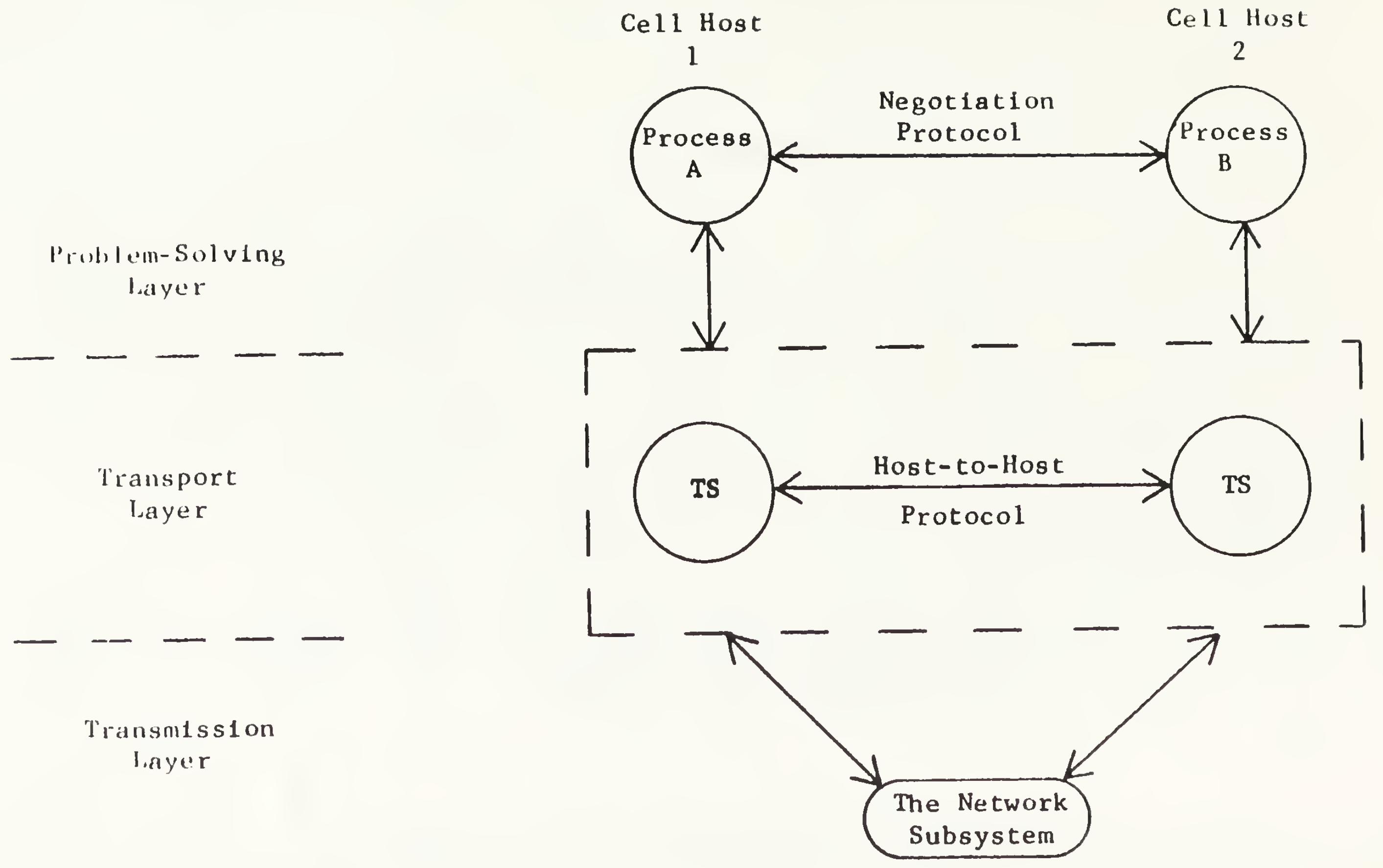

Figure 8. The Protocol Hierarchy

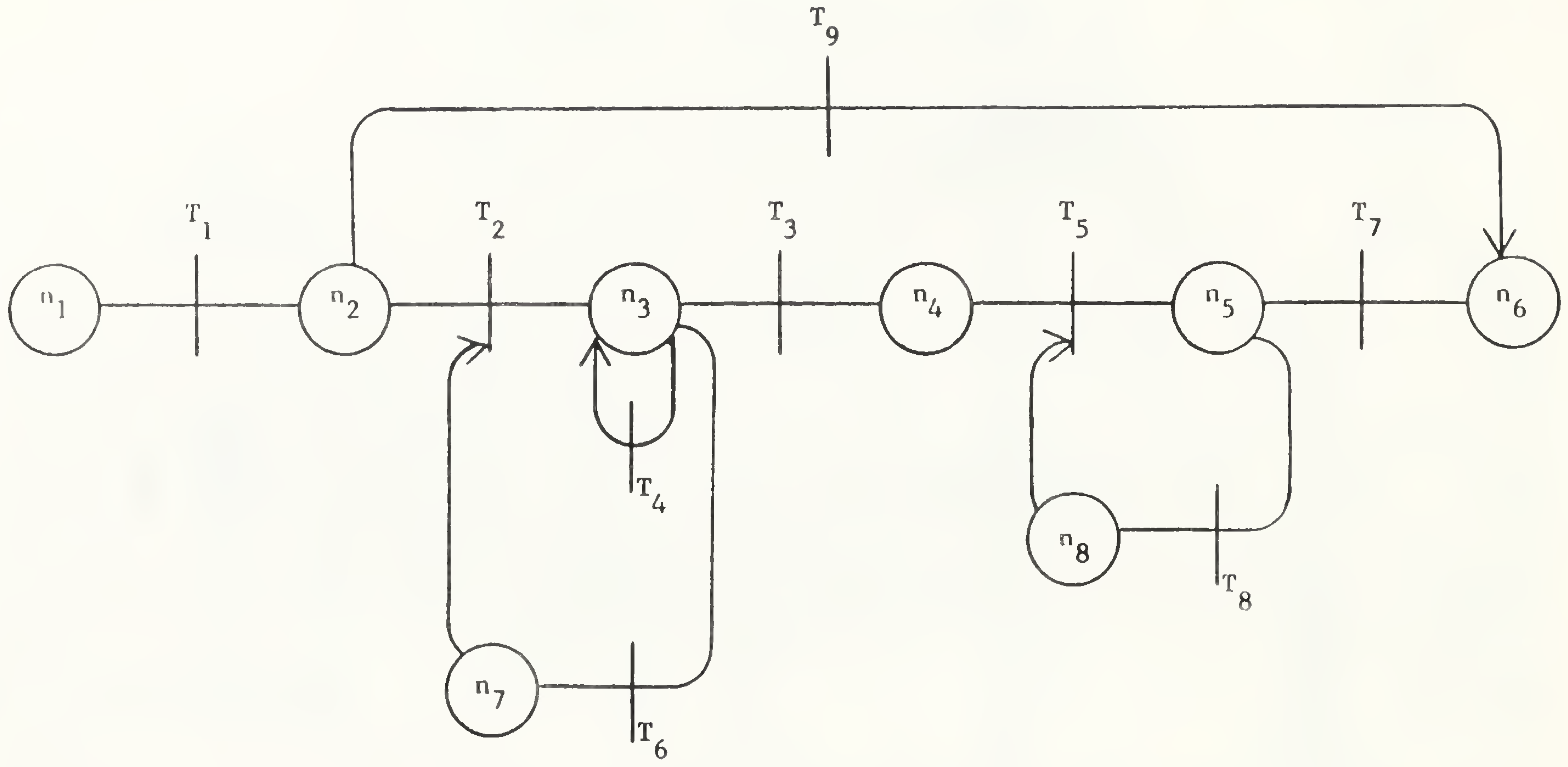

Figure 9. The Task-Announcement Process

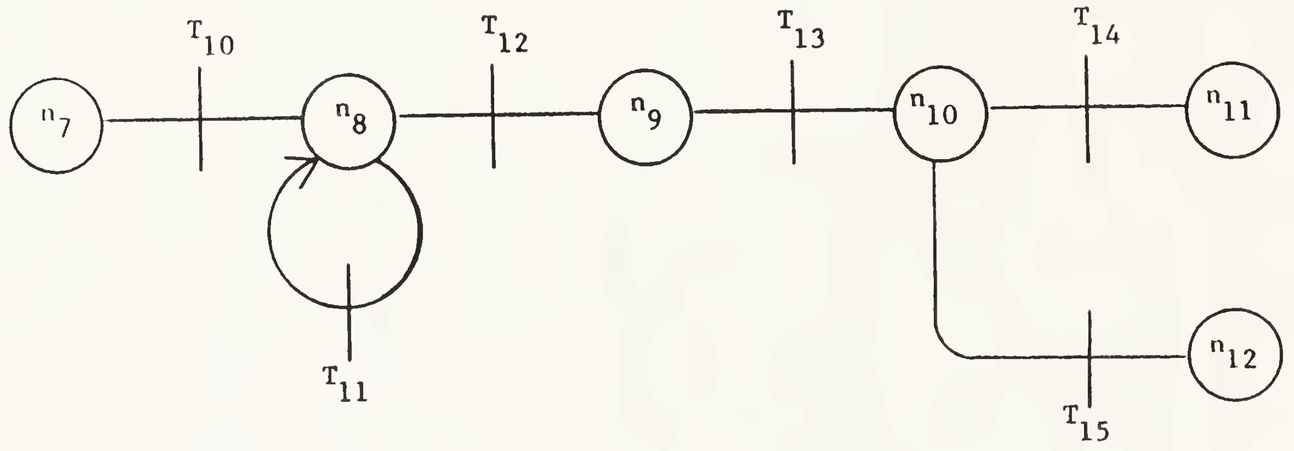

Figure 10. The Bidding Process

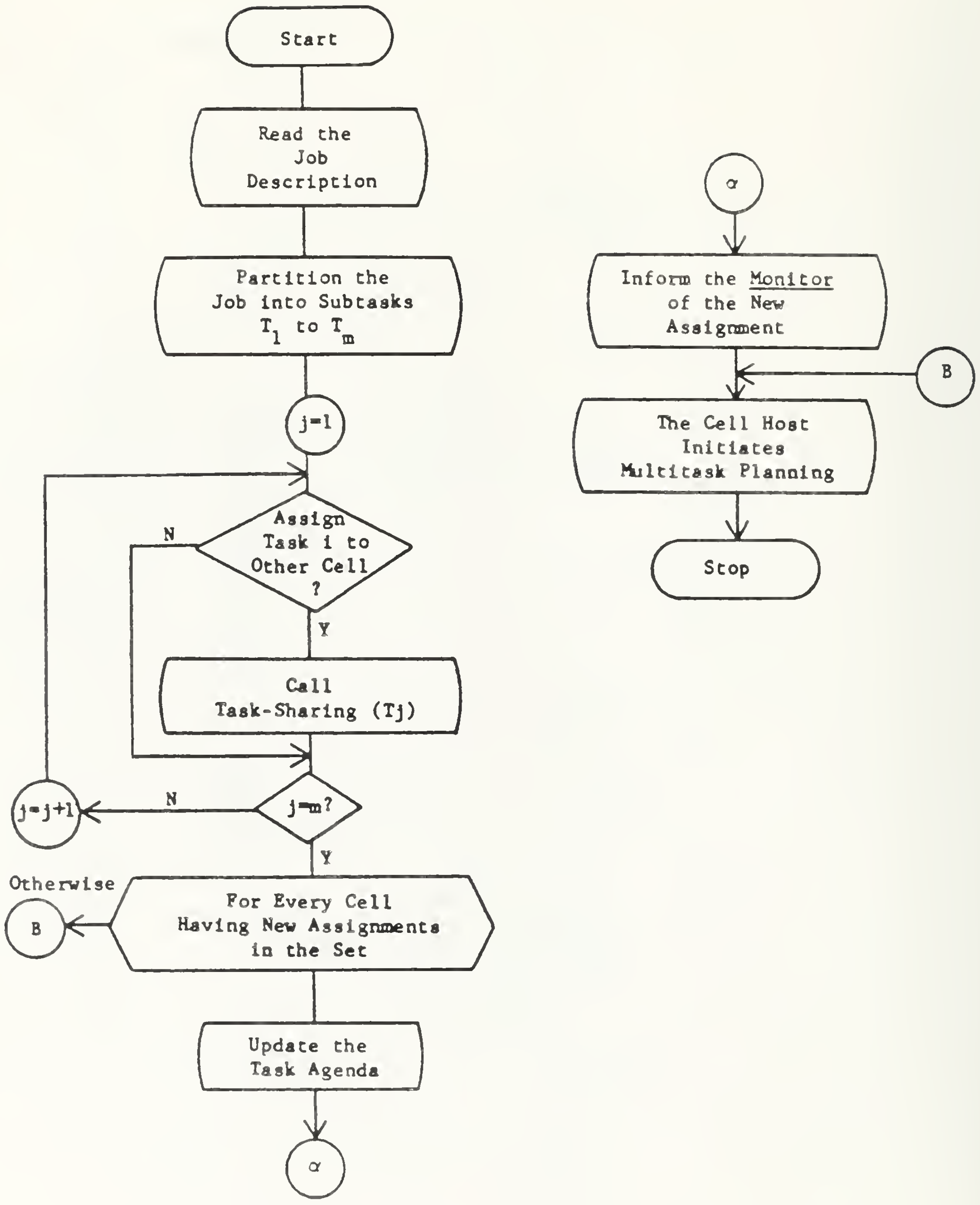
Start
Read the Job Description
Partition the Job into Subtasks T_1 to T_m
j=1
Assign Task i to Other Cell ?
N
Y
Call Task-Sharing (Tj)
j=m?
N
j=j+1
Y
Otherwise
B
For Every Cell Having New Assignments in the Set
Update the Task Agenda
α
α
Inform the Monitor of the New Assignment
B
The Cell Host Initiates Multitask Planning
Stop

Figure 11. The Flow Chart of the Task-Sharing Algorithm

Table 1 Examples of Predicates Used in the World Model

IDLE(M,t): Machine M is idle at time t

MACH-PT(M,OP,PT,t): Machine M begins operation OP on part PT at time t

FINISH-OP(M,OP,PT,t): Machine M completes operation OP on part PT at time t

DIFFERENT(M,M'): Machine M is a machine different from machine M'

MACH-OP(M,OP): Machine M is capable of performing operation OP

PT-FIRST-OP(OP,PT): Operation OP is the first operation on part PT

PT-NEXTOP(OP,OP',PT): Operation OP' should be performed on part PT immediately after operation OP

DONE(PT,t): All operations on part PT are completed at time t

TOOL(M,OP,t): The tool for operation OP is available to the machine M at time t

Table 2 Specifying the Production System in Program Form

```
T1      if (NEW-TASK task)
        then (TASK-INITIALIZATION task)

T2      if (TASK-EVALUATE task)
        then (TASK-ANNOUNCEMENT task)

T3      if (BID-RETURN bid)
           AND (LEQ time-now deadline)
        then (BID-PROCESSING bid)

T4      if (LEQ time-now deadline)
        then

T5      if (GT time-now deadline)
           AND (NE bid-list blank)
        then (BID-AWARD bid-list)

T6      if (GT time-now deadline)
           AND (EQ bid-list blank)
        then (REANNOUNCE task)

T7      if (REPLY-TO-AWARD accept)
        then (LIST-ASSIGNMENT task)

T8      if (REPLY-TO-AWARD reject)
        then (RE-AWARD task)

T9      if (NOT(TASK-EVALUATE task))
        then (LIST-AGENDA task)

T10     if (TASK-ANNOUNCED task)
           AND (BID-EVALUATE task)
        then (TASK-RANKING task)

T11     if (EQ(PROCESSOR-FOR-TASK task)busy)
        then (LIST-ACTIVE-TASK-ANNOUNCEMENT task)

T12     if (EQ(PROCESSOR-FOR-TASK task)idle)
        then (BID-REPLY(BID-SELECT a-t-a-l))

T13     if (LEQ time-now deadline)
        then (BIDDING task)

T14     if (BID-REPLY accept)
           AND (CELL-CONDITION normal)
        then (LIST-AGENDA task)
           AND (REPLY-TO-AWARD accept)

T15     if (BID-REPLY accept)
           AND (CELL-CONDITION not-normal)
        then (REPLY-TO-AWARD reject)

T16     if (BID-REPLY reject)
        then (RE-BIDDING(BID-SELECT a-t-a-l))
```

1. (NEW-TASK task)
2. (BID-RETURN bid)
3. (LEQ time-now dead-line)
4. (REPLY-TO-AWARD accept)
5. (REPLY-TO-AWARD reject)
6. (NE bid-list blank)
7. (TASK-ANNOUNCED task)
8. (BID-REPLY accept)
9. (CELL-CONDITION normal)

Table 3. Predicate Literals Used in the Negotiation Protocol

1. (TASK-EVALUATE task):

   To evaluate the new task, the current loading condition of the cell, and the requirement of the task; the function returns binary values:

   TRUE: the host decides not to accept the task, will announce the task.

   FALSE: the host will execute the task.

2. (Bid-EVALUATE task):

   Similar to TASK-EVALUATE except now it is to decide whether to bid or not. Also returns binary values:

   TRUE: the cell decides to participate in the bidding.

   FALSE: otherwise.

3. (PROCESSOR-FOR-TASK task)

   Returns two answers:

   IDLE: a candidate processor within the cell can execute the task now.

   BUSY: all the candidate processors for the task are currently busy.

Table 4. Functions Used in the Negotiation Protocol

1. TASK-INITIALIZATION
2. TASK-ANNOUNCEMENT*
3. BID-PROCESSING
4. BID-AWARD*
5. REANNOUNCE*
6. REPLY-TO-AWARD*
7. REBIDDING*
8. LIST-ASSIGNMENT
9. REAWARD*
10. LIST-AGENDA
11. TASK-RANKING
12. LIST-ACTIVE-TASK-ASSIGNMENT
13. BID-REPLY*
14. BIDDING*

* communication operations are involved

Table 5. Procedures Used in the Negotiation Protocol

Table 6. The PNL Description for Petri Nets of the Negotiation Process

T_1:	INIT(n_1)	T_8:	TRANS(n_8)
n_1:	PLACE(T_1)	n_8:	PLACE(T_5)
T_1:	TRANS(n_2)	T_9:	TRANS(n_6)
n_2:	PLACE(T_2, T_9)	T_{10}:	TRANS(n_8)
T_2:	TRANS(n_3)	n_8:	PLACE(T_{11}, T_{12})
n_3:	PLACE(T_3, T_4, T_6)	T_{11}:	TRANS(n_8)
T_3:	TRANS(n_4)	T_{12}:	TRANS(n_9)
T_4:	TRANS(n_3)	n_9:	PLACE(T_{13})
n_4:	PLACE(T_5)	T_{13}:	TRANS(n_{10})
T_5:	TRANS(n_5)	n_{10}:	PLACE(T_{14}, T_{15})
n_5:	PLACE(T_7, T_8)	T_{14}:	TRANS(n_{11})
T_6:	TRANS(n_7)	T_{15}:	TRANS(n_{12})
n_6:	PLACE(T_E)		
n_7:	PLACE(T_2, T_{10})		
T_7:	TRANS(n_6)		

1. Active-Task-Announcement-List (a-t-a-l):

   For each processor in the cell, this list keeps records on the tasks that have been announced but not expired, and within the capability of the processor. The cell-host will choose a task from this list once the corresponding processor gets idle.

2. Assignment List:

   Each manager cell keeps an assignment-list on all the tasks awarded and the corresponding contractor cell.

3. Bid-List:

   Each manager cell keeps a bid-list on all the bids received after a task is announced.

4. Task-Agenda:

   Each cell has a task-agenda which contains all the tasks assigned to each processor.

Table A.1 The Lists Used in the Negotiation Protocol