INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Shaw, Jeng-Ping

THE DESIGN OF A DISTRIBUTED KNOWLEDGE-BASED SYSTEM FOR THE INTELLIGENT MANUFACTURING INFORMATION SYSTEM

Purdue University PH.D. 1984

THE DESIGN OF A DISTRIBUTED KNOWLEDGE-BASED SYSTEM FOR THE INTELLIGENT MANUFACTURING INFORMATION SYSTEM

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jeng-Ping Shaw

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 1984

PURDUE UNIVERSITY

Graduate School

This is to certify that the thesis prepared

By Jeng-Ping Shaw

Entitled The Design of a Distributed Knowledge-Based System for the Intelligent Manufacturing Information System

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of:

Doctor of Philosophy

Signed by the final examining committee:

Andrew Whinston, chair

Robert Bonczek

C. H. Holsapple

Robert P. Minch

Approved by the head of school or department:

August 2 1984 Donald C. King

☐ is
This thesis ☒ is not to be regarded as confidential

Andrew Whinston
Major professor

Grad. School
Form No. 9
Revised 11-83

ACKNOWLEDGEMENTS

I wish to express sincere thanks to my advisor, Professor Andrew B. Whinston, for his guidance and support throughout the course of my doctoral studies. His efforts and insights have contributed immeasurably to this research and to my knowledge in information systems.

I am grateful to Professors Robert Bonczek, Clyde Holsapple, and Robert Minch, members of my dissertation committee, for their invaluable assistance to my dissertation research, and to Professor Timothy Lowe, for his constant help and advice.

I would like to acknowledge the support made possible by grants from Purdue Research Foundation, CIDMAC, and IBM. Also, thanks are due to Ms. Kathy Smith who typed the final manuscript. Her conscientious efforts in preparing the difficult figures were much needed and appreciated.

Finally, I wish to thank my wife Crystal. Without her support and understanding, this dissertation would never have been completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Appendix Figure

ABSTRACT

Shaw, Jeng-Ping. Ph.D., Purdue University, August 1984. The Design of a Distributed Knowledge-Based System for the Intelligent Manufacturing Information System. Major Professor: Andrew B. Whinston.

This thesis incorporates planning and distributed problem solving into the design of manufacturing systems. Such an information system is characterized by the hierarchical organization and distributed control. The environment consists of a group of manufacturing cells; each cell uses a knowledge-based planning system to manage the jobs within the cell while interacting with other cells through the communication network.

First, a framework for using the knowledge-based method to perform the planning and control of manufacturing jobs is developed. The dynamic environment is represented by a world model, the operations are represented by state-changing transformations, and manufacturing steps are derived by the plan generation. The coordination of concurrent activities and the management of shared resources are emphasized by including the duration and the resource in the planning formalism.

Plans are constructed in three steps. First, a linearly-sequenced plan is generated for each job independently by a search procedure. In the second step, a plan generator is used to establish necessary precedence relationships between operations by performing look ahead and avoiding any conflicts. Conflict detection can be

achieved either by a "critic" mechanism or a "reasoning about resource" mechanism.

In step three, a plan-revision scheme is used to improve the plan so that the final plan has the shortest duration. The planning system can function as a scheduler in a manufacturing cell, characterized by being goal-directed, event-driven, and able to perform both static and dynamic types of on-line scheduling.

Because the control is decentralized, the whole information system can be viewed as a society of experts and each cell is a problem solving agent with predefined expertise. A negotiation protocol is used to regulate communication and task allocation among cells. This thesis attempts to use the market structure to organize the information system: tasks are viewed as commodities and cells as the bidders with varying preferences.

The augmented Petri net model is used to represent the negotiation protocol and is implemented in a controlled rule-based system. The execution of the negotiation protocol is accordingly accomplished by an inference procedure in the rule-based system. Thus, this thesis in effect has adopted a unified approach to the planning process and the allocation process.

CHAPTER 1
INTRODUCTION

1.1 Overview

In this thesis, issues concerning the design of an information system for the computer integrated manufacturing environment are addressed. Such an information system is characterized by multi-level organization, distributed control, dynamic environment, concurrent activities, resource sharing, and real-time, on-line planning and control. The concepts and results from the field of artificial intelligence in many ways can be effectively used to meet these requirements. The focus of this thesis is centered around the application of two areas: knowledge-based planning and distributed problem solving.

One goal of this thesis is to develop a framework for using the knowledge-based approach to perform planning and control of the manufacturing processes on an on-line basis. Based on the approach, the dynamically changing environment is represented by a world model, the manufacturing processes are represented by state-changing transformations, and manufacturing plans are derived by an inference engine. In the manufacturing domain, the coordination of concurrent activities and the management of shared resources are especially important in the planning. This planning system can then be used to schedule operations, decide the routing of manufacturing jobs, assign machines and other resources, and monitor the execution of manufacturing plans.

Another goal of this thesis is to explore the methodology that can incorporate distributed control in the information system. The type of computer integrated manufacturing system dealt with in this thesis is composed of a group of modules - called the manufacturing cells - which are autonomous units connected by a communications network. A distributed problem solving method is developed to coordinate the activities in different cells and to enable the group of cells to share manufacturing jobs in a cooperative fashion. The coordination and interaction among the cells are modeled by a negotiation process, which can be activated and controlled by a rule-based system. In this context, the proposed information system is referred to as a distributed knowledge-based system.

Finally, this thesis also attempts to explore the structure and the processing methodology for general, decentralized information systems. Because of the developments in computer networking and processor fabrication technologies, the structure of information systems is evolving toward decentralization: a group of microcomputer-based processors are connected by a communications network, performing problem solving jobs collectively. However, a new communications approach is needed that enables the processors to cooperate effectively during their problem solving. This approach not only regulates how the processors communicate with each other, but also "what" they communicate with. Every processor should think and reason about the interactions with others in order to accomplish, with global efficiency, the tasks they share. This can be accomplished by adding still another protocol layer - called the problem-solving layer - to the traditional

network protocol, so that the interactions between processors proceed in a problem solving manner. Besides the computer integrated manufacturing domain, the office information system can also fit into this structure: a group of office work-stations connected by a local area network, performing electronic business procedures in a cooperative fashion.

The organization of this kind of decentralized information systems can be based on the "society of experts" metaphor. Each module in the system - whether it is a manufacturing cell or an office work-station - is viewed as an expert specializing in performing certain types of tasks. For any given job entered into the system, the relevant experts form a team that can accomplish the job efficiently. The structure of the information system is organized as a "contract net" in this thesis, with tasks viewed as commodities and the processor modules as the bidders with varying preferences. Just like the way commodities are allocated to economic agents through the market, the tasks can be allocated through the contract net to the modules efficiently.

1.2 The Environment

The on-line availability of computers in manufacturing systems helps realize the concept of flexible automation - automation that can handle a large and constantly changing variety of produced parts. Moreover, the use of computers also brings about a second capability: real-time optimization of manufacturing processes and problem solving. These two capabilities are integrated into a fully automated manufacturing environment - referred to as the computer-integrated manufacturing system (CIMS).

An architecture for a CIMS has been proposed by the National Bureau of Standard (McLean et. al. [1983]). Consisting of manufacturing cells as the component modules, systems with the proposed architecture can be built incrementally with cells and their interfaces as the standard units. The organization is composed of five levels: facility, shop, cell, work-station, and equipment. The control system at each level of the architecture takes commands from only one higher-level system, but it may direct the control system at the next lower level. This thesis uses a similar, three-level organization as the target environment, leaving out the managerial issues at the facility level and combining work-stations and equipments to be the machine level. This three-level organization for the CIMS is as follows.

1) Level one: the shop level. A shop is composed of a group of manufacturing cells connected by a local area network. This level is responsible for the real-time management of jobs and resources on the shop floor by performing two primary functions: task allocation and resource assignment. For a given job, the shop system first executes process planning - selecting the sequence of manufacturing operations that can complete the job according to the specifications; the shop system then selects appropriate cells for the operations and delivers tools and other resources to the cells if necessary. Furthermore, the shop manages the automation programs by an on-line storage facility and supplies the automation programs to the cell host when a part is sent to a cell. Because the shop-level system does not have global control of the cells, it is a decentralized system.

2) Level two: the cell level. A flexible manufacturing cell consists of a group of numerical machines with integral material-handling equipments under the control of a host computer. The host computer is responsible for performing planning and control within a cell. It schedules tasks to machines, coordinates the machines in performing the multiple jobs, and manages the resources in the cell. Communications interfaces are provided at this level so that the cell host can communicate with other cells. In a CIMS, a flexible manufacturing cell is viewed as an autonomous unit.

3) Level three: the machine level. The control system at this level includes the controllers for machines, parts handling equipments, and robots. In order to make a manufacturing cell fully autonomous, it is important for the controllers at this level to be able to send and receive complex messages from the cell host; moreover, the controller must be linked directly to the host and allow the cell host to do the following: (1) Command the execution of automation programs on the machine controllers. (2) Down-load automation programs to the machine controllers and receive up-loaded programs from the controllers. (3) Access the state of the machine controllers in order to maintain a detailed account of the status of the cell.

Designing an information system in such an environment requires that activities of autonomous components in the CIMS be well coordinated and that the planning and control of the manufacturing processes be executed in an efficient and responsive manner. The primary issues

can be broken down into the following:

-- scheduling of operations

-- coordination of processes for different jobs

-- sharing of resources

-- monitoring of plan executions.

These issues are to be addressed by various kinds of problem solving performed at different levels of the system, as will be described in the following section.

1.3 Problem Solving in CIMS

The information system for the CIMS environment should possess problem solving capabilities so that manufacturing processes can efficiently transform the raw material into end products. The necessary problem solving capabilities include:

Process selection

The purpose of process selection is to produce a process plan for machining a part, given its drawing. The process plan should specify the machining operations to be executed, the precedence ordering, the machine tools to be used, and the surface requirements. In order to produce the desired part with reasonable efficiency, such planning involves taking into account both technological and economic considerations. The former is concerned with the properties of manufacturing operations, the latter the cost factors. For example, a reaming operation is always preceded by a drilling operation and not vice versa, this is constrained by the technological consideration. Economic considerations are exemplified by the rule that, since to execute consecutive operations on the same machine can reduce the set-up cost,

operations that can use the same machine should be clustered together in the process plan.

Because of the rules like those above, in the real-world, the process selection problem is solved mostly by constraint-satisfaction and by heuristics rather than by optimization. Nau and Chang [1981] propose to use expert systems to handle the process selection problem; the expert system uses a frame-based organization where various judgemental rules or decision heuristics are stored as expertise. In this thesis, the plan, which is specified by a sequence of operations, resulting from the process-selection procedure is assumed given.

Scheduling and routing

The scheduling problem can be defined as the problem of selecting a sequence of operations to be performed by various machines in a system. The execution of such a schedule results in the completion of jobs, and assigning machine times and resources to each operation at the right moments. Scheduling in the CIMS is particularly difficult because of the flexibility of the system and the versatilities of machines; more than one machine can perform a given operation and several jobs are competing for shared resources. The scheduling problem in such an environment is characterized by the concurrent activities and the resource-sharing among the jobs.

This thesis uses the knowledge-based planning approach to handle the scheduling of multiple jobs. Specifically, nonlinear planning is used to coordinate the execution of different jobs; the total duration of the resulting schedule is minimized by maximizing the concurrency among the jobs. During the plan-generation process, machine

capabilities, operation precedence orderings, and resource availabilities have been taken into account to decide the final schedule.

For the real-world problem, where the problem size is large and many factors affecting the solutions must be considered, the scheduling problem is often computationally intractable. Some heuristics must be used to reduce the search space to a more tractable size. To this end, Fox [1983] uses a constraint reasoning heuristic to direct the search for schedules in a large-scale job shop environment. The key part of the search for a schedule is the application of constraints in reducing the search space. After each application of an operator, the generated states are rated by applying the relevant constraints, and only the best "n" states are kept for the next iteration of operator application.

By organizing the manufacturing system as a collection of manufacturing cells, as is the case of the environment under study in this thesis, the complexity of the scheduling problem can also be reduced. The underlying strategy is similar to the problem reduction principle. A job is decomposed into tasks to be performed by several cells and the scheduling problem for each cell becomes much more manageable than the scheduling of the original jobs for the whole system. Based on this approach, the scheduling of jobs is accomplished by two steps:

1. task allocation at the shop level; and
2. operation scheduling at the cell level.

Decentralized task allocation

A CIMS consists of a group of manufacturing cells as processor modules, each of which possesses different areas of expertise in terms

of manufacturing operations. Because the cells are connected by a packet-switched local area network and the control is distributed, a task entered into the system should be assigned to the best processor module available in a decentralized fashion and the decision of allocation has to be made based on local information, without any central dispatcher.

The task allocation problem can be handled by an algorithm analogous to the contract negotiation process, as proposed in Chapter 4. This procedure consists of announcement-bid-award sequences to distribute a task to appropriate cells; it is characterized as a mutual selection process: a manager cell with tasks to be distributed attempts to select the most suitable cell to handle the tasks, while a cell with idle machines is also selecting among the announced tasks and submitting bids on those tasks it prefers. A contract of task allocation is established when a bidding cell is selected by the manager cell.

Resource management

The resource management problem is the problem of assigning the limited number of each type of resources in the CIMS environment, including machines, tools, fixtures, pallets, and conveyors. This thesis deals with the resource management problem in terms of feasibility rather than optimality; the resources are subject to the constraint that only one job can have access to a resource at a time. The proposed planning system treats the resource management problem in the manufacturing domain as the resource management in the multiprocessing operating system environment. Synchronizing mechanisms are therefore

used in the real-time planning program to postulate the mutually exclusive condition when several jobs are competing for the resource. In a similar context, Nof et. al. [1980] propose the use of a "manufacturing operating system" to direct concurrent processes and manage shared resources in the manufacturing system. They use a variant of the Petri net model to represent the operational logic so as to regulate the flows of manufacturing processes.

System configuration

One of the advantages to have decentralized control and modular structure is the dynamic configuration/reconfiguration capability. In the CIMS environment, the individual manufacturing cells serve as the basic modules. To complete the tasks required by a job, several cells are "connected" to share the tasks; these cells then play the role of the work-stations in a transfer line until the job is completed. The set of connections of cells for individual jobs in the system is called a configuration of the cells. Reconfiguration is needed when new jobs are entered into the system to be assigned. By using the time-sharing concept developed in Operating Systems, when a cell is assigned with several jobs, it is a "virtual cell" for each job. The on-line planning system in each cell-host computer is responsible for coordinating the multiple jobs in the cell.

Chapter 4 will adopt the market as the information system structure among the cells. Since there is no direct control between manufacturing cells, the communication among cells is accomplished by means of task-announcement and bid-submission messages. A negotiation process is used to ensure that tasks be distributed from a cell as the

manager to a cell as the contractor. The relationships between cells, which determine the configuration, are dynamically established and a cell can be the manager of one task while being the contractor of another.

1.4 Planning and Problem Solving

Planning is a type of problem solving and its primary objective is to develop an appropriate course of actions, among all the possible actions, that transform the system (referred to as the "world") from the current condition to a desired goal condition. The course of actions generated by this process is called a plan. Such a planning system is exemplified by the robot planning system, where robot plans are generated by selecting and synthesizing robot actions to achieve some stated tasks in a given environment.

The planning system can be organized in the form of a knowledge-based system; that is, the knowledge is organized on three levels: data, knowledge base, and control - as opposed to conventional programs where the knowledge is organized on just two levels: data and program. In the knowledge-based planning system, the declarative knowledge about the goals, the current situation of the world, and the semi-finished plan constructed are stored in a database at the data level. On the other hand, in the knowledge-base level, is the domain-specific, procedural knowledge. This knowledge is used to model the behavior of the world, and is often given in the form of rules or operators. Finally, in the control level is the knowledge about the strategy of plan construction; it is related to the decisions of how to select operators and when to apply them. This separation of control from the

program is one of the major characteristics of knowledge-based systems.

In a conventional computer program, knowledge pertinent to the program and methods for utilizing this knowledge are all intermixed, so that it is difficult to change the program. Using the knowledge-based approach, the program itself is only a reasoning and control mechanism; the system can be changed or remodeled by simply adding or subtracting rules in the knowledge base.

Because planning involves exploration of alternative sequences of actions, a symbolic model of the real world, referred to as a world model, is used to serve as an abstraction of 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 general function of a planning system, then, is to construct a course of actions that transform one world model containing an initial condition to a world model which matches the goal condition. Thus, a planning system must have three basic components:

1. The world model, containing a symbolic description of the real world. This world model is represented by the collection of first-order predicates in a database. An instance of the database is called a state-description in the world model.
2. The action model, describing the transformational effects of actions that map states to other states. Such transformations are usually modeled by operators, as the STRIPS operators defined in Nilson [1980].
3. The inference engine, which is the control unit of the planning system that directs the plan generation process. A sequence of

operators are selected to achieve a described goal state from a given initial state.

1.5 Distributed Problem Solving

Distributed problem solving is concerned with problem solving situations in which a group of problem solvers cooperate in performing tasks to achieve a common set of objectives or goals. Such systems are usually characterized by distributed knowledge and limited interactions among the problem solving agents; both the processing power of the agents and the communication capacities among the agents are considered limited resources in relation to the size of problems to be solved.

Due to the limitations of interactions among problem solving agents, each agent's view of the global activities in the distributed system is both localized and limited. Every agent in the distributed system must be able to direct its own activities in concert with the activities of the other agents purely based on local information; the aggregation of these local problem solving should be globally consistent and coherent. There are many possible assignments of subproblem tasks to problem solvers. Since these problem solvers possess different kinds of expertise, they have differing degrees of appropriateness with respect to a given task. The matchability between the expertise of a problem solver and the requirement of a task is reflected by the processing cost of the problem solver: the better the problem solver can perform the task, the lower the processing cost.

The primary issue of problem solving in a distributed environment is then how to allocate subproblem tasks among the agents in a decentralized manner, while minimizing the cost of processing as well as

communication. How to overcome resource limitations - limitations on storage capacities, processing capabilities, and communications activities - is of central importance to the design of distributed problem solving. These characteristics and issues of distributed problem solving systems are reminiscent of the characteristics and issues that research in economics has been dealt with for quite some time. The discipline of economics is concerned with the efficient use of scarce resources. The traditional domain of economics contains systems that are distributed at various levels: the level of the individual actor (economic man of business forms), the level of markets, the level of an entire economy, and, finally, the level of the world economy. Within this framework, economics theories have addressed issues related to distributed systems since Adam Smith's work on division of labor to the more recent research developments on organizations (March and Simon [1958]) and on the theories of teams (Marschack [1972]).

If processes and tasks in distributed problem solving systems are viewed as commodities, then the task assignment problem can be related to the resource allocation problems in economic systems. Problem solvers possess differing degrees of appropriateness for a given task just as economic actors have different levels of utilities for a given commodity. In economic systems, it is well known that the market mechanism solves the economic problem of equating supply and demand by successive approximations to the equilibrating prices. The ideal market mechanism, according to Simon [1981], is

> "a dazzling piece of machinery that combines the optimizing choices of a host of substantively rational economic actors

into a collective decision that is Pareto optimal for the society."

Thus the market mechanism in a sense is also a computational method for solving the problem of optimal resource allocation. In economic systems, if such allocation problems are solved by a centralized procedure rather than the market, the enormous number of equations makes such a procedure impossible because of the processing and storage requirements. The market mechanism, on the other hand, is characterized by reduced and localized computational requirements. At each iteration in the operation of the market mechanism, each economic actor adjusts its tentative allocation, making use of information only about the current tentative prices and its own utility function. The adjustment of tentative prices, at the same time, is established by competitive bidding. It is the minimization of information requirements for each participant in the economy that constitutes the virtue of using the market mechanism.

Since distributed problem solving systems exhibit characteristics of economic systems, as described above, it is natural that the market mechanism has been used to allocate resources, the subproblem tasks, among problem solving agents. Smith [1978] and Malone [1983] are two examples of this application. When an agent has a subproblem task to be assigned, this task is treated as a commodity to be traded in the market place: a description of the task is announced and those agents who are interested will respond with bids (a bid contains information about estimated completion time); the task is then assigned to the best bidder. By treating each manufacturing cell in a CIMS as an agent

with its own area of expertise, we shall apply the same market mechanism to perform task allocation among the cells. This will be discussed in Chapter 4.

1.6 An Outline of the Thesis

In Chapter 2, various issues related to distributed problem solving will be discussed. We will especially analyze distributed problem solving from the standpoint of economics and argue that generalized distributed problem solving methods have long existed in economic systems and human organizations. After reviewing several representative distributed problem solving systems, we shall describe strategies that can be used to achieve effective distributed problem solving, under the constraints of restricted communications and limited computational resources.

In Chapter 3, mechanisms will be described for using a knowledge-based planning system to manage the computer-integrated manufacturing system which is characterized as a distributed environment. Two kinds of information in the action formalism are emphasized: resource and duration. The planning system uses a "reasoning about resources" mechanism to maintain the correctness of machine usages when several manufacturing jobs are competing for the machine. A plan-generator, called PLAN-AHEAD, determines the precedence ordering between conflicting actions by means of the duration information; the final plan - a partially ordered network - has maximized parallelism. A plan-revision mechanism is used to reassign a job to an alternative machine if the job is delayed in a machine queue. We will show that the planning system can play the role of an on-line scheduler. The

scheduler is goal directed, event-driven, and able to cope with the dynamic environments. Operating system techniques will be used to provide appropriate synchronization and communication in coordinating concurrent manufacturing activities. Planning steps are grouped into critical sections according to the resources they need and thus maintain the mutual exclusion of shared resources.

Chapter 4 will be concerned with the decentralized task allocation problem at the shop level, referred to as the "cellular flexible manufacturing system." A contract net protocol will be developed to regulate orderly interactions between asynchronous, cooperating cells. The underlying idea is to structure the interaction between cells as the process of negotiation. Augmented Petri nets, an integration of production rules and Petri nets, are used to model the contract net protocol. The automation of this augmented Petri net provides the basis to implement task allocation algorithm in a decentralized fashion.

Finally, Chapter 5 will conclude this thesis and propose possible directions for future research.

CHAPTER 2
DISTRIBUTED PROBLEM SOLVING:
A FRAMEWORK FOR INTELLIGENT INFORMATION PROCESSING

2.1 Characteristics of Distributed Problem Solving Systems

2.1.1 Introduction

The purpose of this chapter is to explore the characteristics and structures of distributed problem solving systems. Problem solving may be described as finding a series of state changing actions that will achieve a desired goal state given the initial state. A problem solving system consists of either a single problem solving agent or a set of agents that perform problem solving tasks to achieve the common goal state or set of goal states. A system is considered distributed if it consists of a set of agents that in general collaborate and generally differ with respect to expertise and information. The system's coordination of the process may be handled from a central location or be decentralized among the set of problem solvers. The two main givens of such a system are the expertise of the individual agents and the location and type of control in the problem solving process. Expertise is the set of abilities and knowledge an agent possesses. The knowledge an agent possesses may change over time and be transferred between agents during the problem solving process while abilities are generally given characteristics of the agents. If agents have identical expertise, then the system is homogeneous and

subproblem tasks are distributed throughout the system to balance the work load.

Since a distributed problem solving system is comprised of a set of problem solving agents, their actions must be coordinated to achieve global solutions. Further, they must be connected by a communication network. This will limit to varying degrees their interactions and the coordination of the global problem solution. At any given point in time an agent has unique information (localized) and a limited view of the global problem solution and solution process. Both the processing power of the individual problem solving agents and the communication capacities are limited resources within the system. Further, the timing of communication and subproblem task solutions is important in the coordination process. Information such as data to be used in the subproblem task execution, subproblem tasks, and messages (defined as data on the state of the system or the tasks during the solution process) may pass between agents in the problem solving process. This flow of information and an agent's other activities must be efficiently directed and timed in concert with the activities of the other agents to achieve a globally consistent and coherent solution to the problem. This is the control of the system.

Problem solving in such a system is a dynamic process that usually solves each problem as it enters the system in the following four phases:

1. The decomposition of the problem into subproblem tasks.
2. The allocation of the subproblem tasks among the problem solvers.
3. The solution of the subproblem tasks by the problem solvers.

4. The integration of the solutions to obtain a global solution. This phase may be viewed as another of the subproblem tasks to be executed by the individual problem solvers and included in the third phase.

The allocation phase has received by far the most attention in the literature. The system's goal in this phase is to efficiently allocate the subproblem tasks among the problem solving agents. Since they are distributed, no single agent has direct knowledge about the information held by others. If the problem solvers differ with respect to expertise (heterogeneous system), different assignments of subproblem tasks will incur different solution costs. In fact, often there will be subproblem tasks that certain problem solvers cannot solve. The overall solution time is also important. Balancing the task load among the problem solvers involves considering the tradeoff between allowing a task to wait in line for execution or letting a less capable problem solver execute it. The matching of problem solver expertise and load to task requirements so as to minimize the overall solution cost (including waiting time cost) is the requirement for an efficient allocation.

The particular process that a system uses to allocate subproblem tasks may be more or less efficient in a specific environment than another process. Further, a process that is very efficient in one environment may be very inefficient in another. From a single problem solver's point of view (with local information), the subproblem allocation phase is an information gathering process. Its efficiency depends upon the flow of information within the system. First, the messages

which pass information between problem solvers may vary in size. In most environments, smaller messages are less costly than larger ones. Secondly, since the transmission of a message takes time and uses system resources, a large number of messages is more costly than a small number. And thirdly, the distance the messages travel may be important. For example, a message sent to several problem solvers is usually more costly than one sent to a single system member.

The remainder of this chapter is organized as follows. First, various issues related to the distributed problem solving system are addressed. Economic aspects will be discussed in Section 2.1.2 where it is argued that generalized distributed problem solving methods have long existed in economic systems and human organizations. The advantages of using a distributed problem solving system over a centralized system is presented in Section 2.1.3. Then, in Section 2.4, the communication network in distributed problem solving systems is characterized; it is an important feature but is also a restricted resource in the system. Finally, the issues of allocating various capabilities to agents are addressed. These include the distribution of knowledge, abilities, and control among the agents.

In Section 2.2, four distributed problem solving systems are described. Each of these four systems has unique features and approaches in solving problems. These systems are: the HERESAY II system, the Ether system, the Contract Net system, and the DPS networks.

Section 2.3 describes various strategies that have been used to achieve effective distributed problem solving. These strategies

include coordination schemes (Section 2.3.2), cooperation strategies (Section 2.3.3), organization structuring (Section 2.3.4), and the "satisficing" strategy (Section 2.3.5). In the end, various representative distributed problem solving systems are compared in terms of their characteristics and strategies.

2.1.2 Economic Aspects of Distributed Problem Solving

It is difficult to identify when the systematic study of distributed problem solving systems began. Certainly the early work of Adam Smith in economics pointed out many of the advantages of distributed problem solving. Smith viewed the main economic problem as that of the production of material wealth and argued that the division of labor brought great advantages. "This great increase of the quantity of work which, in consequence of the division of labour, the same number of people are capable of performing is owing to three different circumstances; first to the increase of dexterity in every particular workman; secondly, to the saving of time which is commonly lost in passing from one species of work to another; and lastly, to the invention of a great number of machines which facilitates and abridge labour, and enable one man to do the work of many" (Smith [1976]). The division of labor means that rather than each worker solving the problem of the production of a single good alone, several workers cooperate in the solution. According to Smith, they will differ with respect to expertise as specialization leads to differences in both knowledge and abilities. Also, the second advantage of the division of labor showed that Smith recognized that production (problem solving) time is also an important factor. On the question of

coordination, Smith believed that buyers and sellers, if left alone, would find it in their own self interest to work, produce, and exchange goods in a way that would promote the efficient production of material wealth. This distribution of control would occur as a "consequence of a certain propensity in human nature... to truck, barter, and exchange one thing for another."

Modern economics views the basic economic problem slightly differently than Smith did, but still views it, essentially, as problem solving. In the modern view, the basic economic problem is to allocate given scarce resources to the production of various goods so as to maximize the consumer's utility (satisfaction). At the level of a single firm, the production of a single good (problem) is performed by a single person (non-distributed problem solving system) or by a group of workers with division of labor (distributed problem solving). If a single firm produces more than a single good, then this joint production is a case where there may be advantages to solving more than one problem at a time. That is, producing one good makes producing the second good more cost effective.

A market is another problem solving system in economics. In this example, the problem of the production of a number of goods is handled in a distributed problem solving system comprised of individual firms. The market serves as a central location that helps to coordinate the agents' interactions in a very decentralized manner. The messages passed between agents (firms) are all in the form of prices. An entire economy is another level in economics that acts as a problem solving system. Here again, firms are the problem

solving agents and in a market economy, the price mechanism coordinates the problem solving process. Economic externalities are advantages to solving more than one problem at a time if they are external benefits and are disadvantages if they are external costs.

Perhaps the simpliest example of a problem solving system in economics is an international trade model. Adam Smith argued for the opening up of freer trade among nations to further take advantage of distributed systems, but David Ricardo was the first to use economic modeling in his analysis. Ricardo rigorously proved the law of comparative advantage that stated that two countries would both gain if each specialized in the production of those goods that were relatively low-cost items.

A resource allocation in an economic environment is essentially a subproblem task allocation. The goal of a resource allocation process is to distribute the resources of an economy to a group of possibly heteroegenous economic agents in an efficient manner. An economic agent possesses unique information and is acquiring further information in a basically decentralized manner to exchange resources in the search for efficient allocations. Some resource allocation processes use a central coordinator (referred to as an auctioneer in the economic literature) while others do not.

A tatonnement and a non-tatonnement resource allocation process imply very different problem solving coordination strategies. One type of tatonnement resource allocation process employs an auctioneer (referred to as a Walrasian auctioneer) who coordinates the search for trading partners by establishing resource prices that will lead to an

efficient allocation. The auctioneer offers possible prices and agents respond with quantities (demands for allocated item). Once an efficient set of prices is established, all markets simultaneously clear (reallocation occurs). The strategy of the auctioneer is to continue to search until an efficient set of prices is known and only then will exchange occur (through some central market). Without actually having a central coordinator (auctioneer), enough information would have to be exchanged between agents so that all agents can decide upon trading partners prior to the exchange taking place. A similar class of tatonnement processes employs a coordinator who adjusts quantities rather than prices (a Marshallian auctioneer). The only difference in this process is that agents respond to the coordinator's quantity assignments with price responses.

Under a non-tatonnement allocation process, much less searching occurs. When a coordinator (Walrasian auctioneer) announces a set of prices in his search for the set that will lead to efficiency, agents respond as before with quantity changes (revealing their demand) but reallocation occurs each time period at the current auctioneer prices. Hence, exchange occurs each period and the search for the efficient allocation involves much less searching.

2.1.3 The Reasons for Using Distributed Problem Solving Systems

The importance of the role played by distributed systems in the development of information processing is attributable to several factors. Some information processing systems are inherently distributed, where expertise or control may be easily decomposed into a number of relatively independent modules either physically (spatially

distributed system), functionally, or both. A number of problem solving systems are spatially distributed. In these, problem solving agents are placed at different physical locations but require assistance from each other. The communication protocol is thus an important focus of their designs. Examples include the sensory networks (Smith [1978], Corkill [1982]) and the intelligent consultant for ARPANET (e.g., Rosenschein [1982], where intelligent agents located on various computers in the ARPANET are used to help construct and execute plans in the operating system domain).

The problem solving systems that are functionally distributed are often decomposed into divisions, with each division corresponding to a set of specific functions. The system that is purely functionally but not spatially distributed would have no communication problems since there is no physical separation of the problem solving agents. An example of this might be a human organization where the problem solving system is located in a single office that performs several different separate functions. A single worker with several clearly defined and separated functions may also be viewed as such a problem solving system.

A problem solving system that is both spatially and functionally distributed is exemplified by an office information system where a collection of work stations share tasks. Each work station is assigned a specific set of functions in terms of capabilities and responsibilities. Another example is the cellular system where each manufacturing cell specializes in producing "part families" that require similar manufacturing operations. To accomplish a job in such a system often

requires the joint effort of several cells associated with different families.

Other advantages to employing a distributed problem solving system include better reliability, performance speed-up, graceful degradation, system extensibility, and modularity (Enslow [1977]). Reliability refers to the ability of the problem solving system to continue to function when a portion of the system network fails. Performance speed-up occurs because problem solvers may execute sub-problem tasks concurrently rather than serially. Graceful degradation refers to the ability of the system to not only continue to function when a portion of the system network fails but to continue to perform relatively efficiently. System extensibility allows problem solvers to be added without major system redesign. And modularity refers to the flexibility of the network configuration. That is, each agent has the same control structure so that the responsibilities of an agent may be reassigned to other agents in case of failure. Modularity also implies conceptual clarity and simplicity of design and further results in system extensibility. Another motivation of using the distributed structure comes from the fabrication technologies of computational processors that employ very large scale integrated circuits. They have made it less expensive to make a number of smaller processors for an information system rather than a single large processor. And lastly, frequently, a problem solving system may contain too much knowledge for a single problem solving agent to function efficiently. Capacity limitations demand that the information be distributed among several distributed problem solvers.

2.1.4 Communication Networks

When a problem solving system is distributed spatially, the interactions between problem solvers must go through a communication network and are therefore constrained by the network's physical limitations. The limited processing capability of each problem solving agent adds to the communication needs of a distributed system. In order to disseminate messages to one another to solve problems cooperatively, the agents may have to compete for time slots on the communication channel. Since most distributed problem solving systems are physically distributed, the coordination problem with the communication network limitations has been an important topic in the literature.

Simon [1981] has termed the limitations on the processing capability of each problem solving agent as bounded rationality. This limitation applies both to the amount of information which can be effectively handled by a problem solver to arrive at a decision and to the processing power which each agent possesses. Also, the coordination activities of the system consume processing resources and further tightens the limitations on the processing power of the problem solvers.

The communication networks of distributed problem solving systems may be categorized as either loosely-coupled or tightly-coupled. The processing speed for computational processors is generally faster than the speed of communication networks linking the processors. Therefore, in a computer environment in particular, the tradeoff between the time a system spends communicating and the time it spends processing is

important. If more processing power is spent in executing subproblem tasks than in communicating with other processors, then the system is loosely-coupled. Otherwise, the communication network is tightly-coupled. While most of the distributed problem solving systems are loosely-coupled, the Heresay II system (Erman et. al. [1980]) described later in this chapter is an example of a tightly-coupled network.

The structure of the communication network within the organization is a main determinant of the degree to which interactions between agents are limited. The structure of the organization determines to a large degree the roles and the responsibilities of the problem solvers. Malone [1983] employs the market as a metaphor for distributed problem solving systems and coordinates the problem solving activities through a variant of the price system of the market. The Heresay II system (Erman et. al. [1980]) uses a hierarchical structure similar to that of a corporation. Other metaphors of organizational structures used in problem solving systems are scientific communities (Kornfeld and Hewitt [1981]) and committees (Chandrasekaran [1981]). Corkill [1982] goes further and proposes a self-designing organization. That is, the structure of the communication network is altered with each different problem entering the system. The communication requirements associated with coordinating such problem solving activities may be a source of ideas for new or extended networking technologies, in which each site in the network is treated as an intelligent agent instead of a dumb terminal.

Another approach to the problem of limited interactions between agents is to focus on the task decomposition phase of problem solving.

For example, Davis and Smith [1983] focus on the modularity and the size of the subproblem tasks resulting from the problem decomposition. The basic idea of their scheme is that if subproblem tasks are made modular, they have fewer interdependencies and, therefore, fewer processors coordinating the problem solving.

2.1.5 Distributed Expertise and Control

The expertise of an agent includes the knowledge and abilities possessed by the agent. Knowledge is the information - both long-term and short-term - acquired by the agent; ability is the inherent capability of the agent, which constrains the kind of knowledge the expert can possess. For example, a computer processor may contain regression subroutines and time-series data files; then the processor as a problem solver possesses the "knowledge" to do regression analysis on that given data. However, this problem solver has a different kind of "ability" than that of an interactive graphic terminal. Another example of distributed knowledge can be found in the multiprocessor system Cm*, where the routines for the network operating system are located at various processors. A system with distributed ability is exemplified by the sensor network discussed in Davis and Smith [1983], where the nodes consist of either sensors (for detecting image data) or processors (for processing image data).

The expertise possessed by a problem solving system may be distributed among the agents. The distribution may result from the nature of the system, as with, for example, a sensor network where sensory signals are processed at different locations in the network. Distribution of knowledge may result from the limited capacity of individual

agents to store enough knowledge for a single problem. In this case, each agent is assigned a specific portion from the whole knowledge domain to be its area of expertise. The distributed problem solving system can then be viewed as a set of cooperating experts where the subproblem tasks are assigned to match the expertise of the agents.

The way the expertise within a system is distributed raises the distinction between a homogeneous and a heterogeneous system. A heterogeneous problem solving system is likely to be modularized where problem solving agents possess different kinds of expertise. The expertise of an agent may overlap in varying degrees with that of other agents. When there is very little overlap, there are potential bottlenecks in terms of certain types of subproblem tasks that may require certain types of scarce expertise. The problem of allocating knowledge to a given set of agents is similar to the file placement problem in computer networks (Chen [1980] and Wah [1984]). The main problem in the task allocation phase of problem solving is to match the subproblem tasks to the given distribution of expertise efficiently.

In the case of homogeneous network systems, each agent has a full range of problem solving capabilities. Matching tasks to problem solvers is no longer a concern in the allocation phase of problem solving. The main focus in these systems is on load balancing to improve performance (Efe [1982] and Tantawi and Towsley [1984]).

In view of the fact that the individual agent in a distributed problem solving system is capable of only a limited amount of problem solving, the overall intelligence, or capability, of the system may be viewed as a result of the interactions among the set of agents. This

effect is referred to as the emergent intelligence of the system.

The control of a system is distributed when processing and communication are not focused at a particular problem solver, but rather every agent is capable of accepting and assigning tasks. The distribution of control helps avoid bottlenecks that could degrade performance. However, when the control is distributed, the problem solving activities need to be coordinated among the agents to ensure global coherence in solving the problems. Various coordination methods to exercise distributed control are discussed in Section 2.3.

2.2 A Review of Existing Systems

In this section, four representative distributed problem solving systems are discussed. Each of these systems employs unique concepts distinctly different from other distributed problem solving systems. The Heresay II system is one of the early distributed problem solving systems that is distinguished by its hierarchical organization and the use of the blackboard to coordinate its problem solving activities. The Ether system emphasizes exploring the parallelism and concurrencies among the problem solving activities. The contract net approach treats a distributed system as an economic system and uses a decentralized coordination scheme similar to the market mechanism. And finally, the work on DPS networks by Corkill employs the most generalized approach to distributed problem solving to date. By using meta-level control to coordinate agents, the system structure changes with each new problem and is treated as one of the subproblem tasks when a problem is entered into the system.

2.2.1 The Heresay II System

The Heresay II system (HSII) is an implementation of a class of distributed problem solving organizations. The primary characteristics of these organizations include: (1) multiple, diverse, independent, and asynchronously executing knowledge sources (KS's); (2) the KS's are invoked in a data-directed, cooperating fashion; and (3) coordination among KS's are achieved via a shared database called the blackboard. The original tasks that HSII dealt with are speech understanding which needs to search in a large space of possible interpretations for the utterance that best fits the input data, i.e., the speech waveform signals.

The knowledge in the task domain is represented in separate KS's. Each KS can be viewed as an expert in its particular fields, communicating with other experts via the blackboard. The basic data unit of the blackboard is the hypothesis. A hypothesis represents a partial solution to the overall problem expressed at one of the levels of the blackboard. By reading or writing information on the blackboard, a KS can either place a hypotehsis on the blackboard or test the blackboard hypothesis produced by other KS's. The knowledge in the different KS's can be used to interpret the utterance at different levels of representation. The levels of representation form a hierarchy and each level is built upon a lower level. The job of a KS is to solve problems at its level of expertise by postulating subproblems at a lower level or by solving the subproblems from a higher level. The problem solving activities are data-directed, invoked by the current state of the database.

The problem solving approach is as follows. The basic execution cycle begins with the execution of a knowledge source that makes changes to the blackboard monitor and determines what additional knowledge sources should be executed in response to the changes. These knowledge sources are placed on the scheduling queue that is ordered by the scheduler based on the progress of problem solving in the system. When the currently executing knowledge source has completed, the highest rated knowledge source on the queue is executed, and the cycle repeats.

2.2.2 The Ether System

The Ether system (Kornfeld [1979]) carries out problem solving activities with an emphasis on allowing parallel processing of the sub-problem tasks. Computation in Ether is carried out by computation elements known as sprites. Using the pattern-directed control method, a sprite consists of two parts, a pattern and a body. If the pattern successfully matches an assertion that has been broadcast in the system, the body of the sprite is executed and new sprites and assertions will be created.

Following the Actor model created by Hewitt [1977], the Ether system realizes all communication and control among sprites by disseminating messages. Two kinds of communication elements may be broadcast: assertions and goals. Assertions are intermediate results of computations that need to be broadcast to inform related agents of the new facts. The goal of each part of the system must be communicated to other parts embodying expertise that may help achieve the goal.

There is an allocation mechanism to distribute the available processing resources to the active sprites. An agent of the system that provides processing resources is called a sponsor. All sprites capable of triggering do so through the support of a sponsor. A sponsor may reallocate its resource on the basis of relative merits of sprites.

Ether carries out parallel pattern directed invocation procedures where all the applicable sprites (sprites whose patterns are matched) which can get support from a sponsor will work on the goals concurrently. The coordination between different parts of the system is achieved by messages passing without any requirements of shared memory or critical regions. Messages are broadcast through the system, the interested agents have the option of intercepting and subsequently, adopting the information.

Besides the speed-up due to parallel processing, Kornfeld [1982] argues that because of the information shared between running sprites, the problem solving activities may be further facilitated. This effect is referred to as "combinatorial implosion."

In analyzing the problem solving behavior of Ether, Kornfeld and Hewitt [1981] use the metaphor of scientific communities. A system of sprites can be compared to a community of scientists working concurrently, keeping track of the new developments of the community. They broadcast their goals and results in the form of proposals and publications, while looking for sponsors of resources to help achieve the goal.

2.2.3 Contract Nets

The work on contract nets (Smith [1978] and Davis and Smith [1983]) defines a framework for distributed problem solving based on the market metaphor. A contract net protocol is developed to specify communication and control in a network of problem solvers. Task distribution is viewed as an interactive process; negotiations are carried on between an agent with a task to be executed and a group of agents able to execute the task. A negotiation protocol is used to help determine the content of the information transmitted, rather than simply provide a means of sending bits from one agent to another as the traditional network protocol does.

With the approach, the collection of problem solving agents is referred to as a contract net and the execution of a task is determined by a contract between two agents. Each agent in the net takes on one of two roles related to the execution of an individual task: manager or contractor. A manager is responsible for monitoring the execution of a task and processing the results of its execution. A contractor is responsible for the actual execution of the task. Individual agents are not designated a priori as managers or contractors; these are only roles, and any agent can take on either role dynamically during the course of problem solving.

A contract is established by a process of local mutual selections based on a two-way transfer of information. Available contractors evaluate task announcements made by several managers and submit bids on those for which they are suited. The managers evaluate the bids and award contracts to the agents they determine to be the

most appropriate. The negotiation process may then recur. A contractor may further partition a task and award contracts to other agents.

In summary, the negotiation process has four important characteristics: 1) it is a local process that does not involve centralized control, 2) there is a two-way exchange of information, 3) each party to the negotiation evaluates the information from its own perspective, and 4) final agreements are achieved by mutual selection.

Based on the Contract Net approach, Malon [1983] uses a prototype system called Enterprise that schedules tasks to processors in a decentralized fashion. The environment is typically a local area network of high-performance personal computers connected with an Ethernet. By using the contract negotiation process to match tasks with processors, better overall system performance is reported. This new methodology has long-lasting effects on the design philosophy of office information systems. By modeling the environment as a contract net, personal workstations are most of the time dedicated to their owners. But when the owners are not using them, these personal workstations become general purpose servers, available to other users on the network. Thus programs may be written to take advantage of the maximum amount of processing power and parallelism available on a network at any time.

2.2.4 The DPS Network

Corkill [1982] uses a network of problem solving nodes, each with a complete Heresay II architecture to investigate the cooperative problem solving behavior in such a distributed network. The specific problem domain is a distributed vehicle monitoring system that has a number of processing nodes, with associated acoustic sensors,

geographically distributed over the area to be monitored. Each node can communicate with one another over a packet radio communication network.

The problem solving in the network requires a structure and organization in which the nodes cooperatively converge to acceptable answers in the face of incorrect, inaccurate, and inconsistent intermediate results, as is the characteristic of acoustic signals. A functionally accurate, cooperative approach is developed using an iterative, co-routine type of node interaction in which a node's tentative partial results are iteratively revised and extended through interaction with other nodes. It is claimed that by using this approach much less communication is required to exchange these high-level partial results than the communication of the massive raw data. In addition, the nodes can operate asynchronously, resulting in increased parallelism and better performance. Further, because of the functionally accurate nature, better reliability is also expected.

Besides the basic blackboard structure used in Heresay II, Corkill has integrated goal-directed control structure into each node. This has been accomplished through the addition of a goal blackboard. Goals are created on the blackboard as a result of a change in the data blackboard, or they are sent by other nodes. The planner responds to the insertion of goals on the goal blackboard by developing plans for their achievement. Internode communication is added to the node architecture by the inclusion of the communication knowledge source. These knowledge sources allow the exchange of hypotehses and goals among nodes. In a sense, these communication knowledge sources are

functionally similar to the negotiation protocol processor, implemented in the form of knowledge bases, as proposed in Shaw and Whinston [1983]. The communication activities are carried out through limited broadcasting. Each communication knowledge source contains a list of nodes that may be interested in a particular message and blocks other nodes from receiving the message.

An important feature incorporated in the distributed problem solving network is the use of meta-level control to guide the cooperation among nodes. The meta-level control is represented as a network organizational structure that specifies in a general way the information and control relationships among the nodes. The organizational role assigned to each node, its responsibilities, and the interaction patterns are all represented by this meta-level control. The local control component of each node elaborates these relationships into precise activities to be performed by the node. Thus, the coordination of activities among nodes is achieved by the combination of two concurrent activities: organizational structuring activities and local control activities.

Meta-level control via organizational structuring is introduced into the node structure by the specifications in the interest areas of the architecture. These interest areas exercise their influences on the problem solving activities by modifying the priority rating of goals, specifying the list of nodes they are to send messages to or receive messages from, and rating the importance of other nodes.

The various authority relationships among the nodes in the network are specified by a relative weighing of activities generated locally

versus activities proposed by other nodes. Together, they represent the control knowledge about the responsibility of the node in terms of problem solving activities and their relationships between nodes.

Although currently the organizational structures are specified directly into interest areas of the nodes, Corkill proposes to use an organization blackboard for determining plausible structures for the system and evaluating potential candidates for network reorganization.

2.3 Strategies of Distributed Problem Solving

2.3.1 The Global-Coherence Issues

Due to the limited interactions among problem solving agents, as characterized in Section 2.1, each agent's view of the global activities in the distributed system is localized and restricted. It is impractical to keep every agent constantly informed of the development of other agents' activities or the changes in their databases. The total requirements of communication activities are too enormous. In addition, the use of a centralized controller is also precluded for the sake of reliability; by using decentralized control, the network's performance degrades gracefully when a portion of the network fails. Therefore, some kinds of coordination are required to enable each agent to direct its own activities in concert with the activities of the other agents based on local decision and information; the aggregation of these local activities should satisfy global coherence.

Thus, the primary issue in distributed problem solving is that the solutions produced by individual agents not only are locally good, achieving the assigned tasks, but that the aggregation of these local

sclutions should result in overall solutions that are globally acceptable. This global coherence must be achieved by local problem solving in a decentralized manner.

The difficulty in obtaining coordinated behavior when each agent has only limited, local information of the system can be resolved by effective - but limited - interaction between agents. Crucial information is exchanged between agents in the most efficient forms; but the amount of information exchanged should be enough so that each agent can produce solutions that are globally coherent. The required global coherence of distributed problem solving can be achieved by:

1) the cooperation among agents; or
2) interactions that can either resolve conflicts between agents or help satisfy interdependent constraints among agents.

In the following section, various strategies that have been used to address these issues are presented.

2.3.2 Coordination Schemes

In order to effectively organize the information flows among agents and direct local activities to achieve global coherence, various schemes of coordination have been used. In most distributed problem solving systems, the coordination is done by disseminating messages among the agents. There are basically three kinds of messages for these purposes: tasks, goals, and data. In terms of the amount of semantic information, task messages contain more information than goal messages, and goal messages in turn contain more information than data messages.

Four coordination schemes will be discussed: externally-directed, self-directed, negotiation, and meta-level controlled. A system is using externally-directed coordination if agents do not initiate any activities unless they receive messages from other agents informing them what to do. Under the internally-directed scheme, on the other hand, the agents generate their own decisions on their local activities. Both the negotiation scheme and the meta-level controlled scheme can be viewed as the combination of the first two approaches. However, the meta-level approach is more generalized in the sense that it can contain a negotiation procedure. In general, the more semantic information the messages contain, the more "external" is the coordination scheme (one has to be clear in giving orders). Thus, under the self-directed scheme, the messages mostly contain only data; whereas under the externally-directed scheme, the messages contain tasks that need to be done.

(1) Externally-Directed

When the coordination is externally-directed, an agent is required to perform some actions in response to the receipt of messages. In a sense, the scheme is similar to that of pattern-directed invocations; patterns are contained in messages and sent by other agents. Since an agent's activities are directed and invoked by other agents through disseminating messages, the agent has less flexibilities in its processing strategies; this is a more structured scheme for network coordination.

The Actor formalism developed by Hewitt [1977] and the Ether system by Kornfeld [1979] are two examples of externally-directed

schemes. The knowledge of an agent is stored as a collection of "activities" and each activity is directed by some patterns. These patterns will be matched against contents of the received messages and, if matched successfully, the corresponding activities will be carried out. In the actor model, each agent is modeled as an "actor", and the corresponding activities are called the scripts.

(2) Self-Directed

When the coordination is self-directed, each agent determines the portion of the overall tasks it should perform and the information it should exchange with other agents, as opposed to externally-directed systems where an agent is completely directed by other agents. The subproblems in each agent, under the self-directed scheme, are solved in a bottom-up manner. The task decomposition and node assignment information is not known to the agents; subproblems in an agent are generated from raw data collected by that agent. Lesser and Erman [1980] and Corkill [1982] discuss ways of syntehsizing subproblems from different agents to construct a consistent overall solution using the self-directed approach. The major advantage of using self-directed coordination is the flexibility the agents have in determining local activities. They do not have to wait for messages from other agents to initiate activities. This also contributes to graceful degradation in case of system failures. An agent can immediately undertake activities to circumvent troubled portions of the system without interacting with other agents at all.

(3) Negotiation

The negotiation process is both self-directed and externally-directed. It has the ingredient of externally-directed coordination in that, when a task message is announced, those potential contractors (those agents who can do the task) must react to the announcement and initiate a task evaluation procedure. These activities are invoked externally by messages sent from other agents (the managers).

On the other hand, part of the negotiation process is self-directed. For example, the manager's decision to delegate a task is a self-directed action, without any control exercised by other agents; so is the contractor's decision to accept a task.

This integration of self-directed and externally-directed approaches is referred to by Galbraith [1977] as a "mutual-adjustment" process for coordination. This approach possesses both the flexibility of the self-directed approach and the efficiency of the externally-directed coordination.

(4) Meta-level Coordination

The meta-level coordination scheme, as used in Corkill [1982], can be split into two concurrent activities:

a) construction and maintenance of an organizational structure for the agents; and

b) continuous local elaboration of this structure into precise activities using the local control of each agent.

The organizational structure is used to provide each agent with a high-level view of problem solving in the system. It specifies the responsibility of each agent and the relationships among the set of

agents (these specifications are recorded on an organization blackboard). The local problem solving activities are planned in response to the goals and subgoals. These goals are either created locally or as a result of externally-directed requests from other agents (communicated goals). Various types of network coordination can be achieved by adjusting the responsibility (the "role") of agents or by adjusting the authority relationship (the power structure) among agents. The specification of responsibilities of an agent is done by determining and modifying the priority ratings of goals and tasks in the agent; the authority relationships among the agents is specified by a relative weighting given to the importance of local problem solving versus activities proposed by other agents. Thus, meta-level coordination is a more general type of integration of externally-directed and self-directed schemes than the contract net formalism.

2.3.3 Cooperation Strategies

The fact that each agent has only local, limited knowledge encourages the cooperation among problem solving agents since none of them can solve the whole problem. Davis and Smith [1983] develop a contract net protocol to achieve the cooperation among agents, using "tasks" as the basis of forming cooperation. Agents who share tasks for the same problem work independently and synthesize their results when completed. They argue that two different approaches can be used to achieve cooperation among agents: task sharing and result sharing.

Using the task-sharing scheme, agents divide the set of tasks among themselves and each agent can independently solve subproblem tasks using local knowledge. The major concern in this mode of

cooperation is then to decide the task allocation - determining who will do what - in a decentralized fashion. The contract net formalism is an example of this type of cooperation.

In the result-sharing scheme, agents assist each other by exchanging their partial results. Based on the updated information from other agents, an agent would adjust its activities in the hope that better solutions can be achieved. The Ether system in Kornfeld and Hewitt [1981] uses this mode of cooperation. The set of agents is likened to a scientific community, where scientists share their intermediate results through publications - in the hope that this can help facilitate problem solving.

The cooperation scheme advocated by Lesser and Corkill [1981] represents yet another form of cooperation. The cooperation between agents is based on "data". In their problem, the input data to each agent is, by nature, usually incorrect. Therefore, each agent has to perform problem solving with incomplete data while simultaneously exchanging the intermediate results of its processing with other agents. It is hoped that by exchanging their intermediate results they can construct cooperatively a complete solution, eliminating erroneous results. Thus, the objective of cooperation in this case is for consistency between agents, accomplished by iterations of exchanging partial results.

Another form of cooperation occurs when the agents' tasks are not independent and the agents need to cooperate in order to avoid possible conflicts. For example, if one agent is to achieve ON(A,B) while another agent is to perform PICKUP(B) in the block world, there is a

conflict between these two agents. One way for them to cooperate is by suspending the task for ON(A,B) until block B is freed by the second agent. This type of cooperation, which uses the suspension of actions to avoid any conflicts among the agents, can be implemented by the synchronizing mechanisms used in operating systems for real-time resource management.

2.3.4 Organization Structuring

To resolve the inherent problems of bounded rationality and limited interactions for each problem solving agent, distributed systems often use structures that are tailored to the particular task domains. The objectives of such system structuring are to reduce the processing requirements and uncertainties faced by individual agents.

Simon [1981] uses markets and hierarchies as two prime examples of structures that can be used to distribute information processing throughout complex systems. In systems with these structures, the processing requirement for each agent is little, but together the system can achieve tasks such as decentralized resource allocations that have enormous overall complexities.

Based on these viewpoints, distributed problem solving systems may take advantages of the structures among the agents to effectively distribute the problems. An organizational structure specifies the responsibilities, the control pattern, and the interaction pattern among the agents. It influences how a larger task should be decomposed and then properly distributed, or how individual agents should behave in specific situations. For example, functional hierarchies and product-hierarchies are two possible structures for organizations;

problems to be solved would have to be decomposed and then allocated differently using the two structures. Since different organizational structures imply different distribution of control as well as different information paths among agents, the problem that can be decomposed and solved efficiently under one organization structure may not reach the same level of performance in systems of other kinds of organizational structures.

Several research studies embody aspects of market metaphors into the design of their organizational structures (for example, Smith [1978], Malone [1983], and Shaw and Whinston [1983]). The primary advantage of using market structure to organize agents is the small amount of information transferred. This is described by Hayek [1945]:

> "... the most significant fact about this (market) system is the economy of knowledge with which it operates, or how little the individual participants need to know in order to be able to take the right actions. In abbreviated form, by a kind of symbol, only the most essential information is passed on, and passed on to those concerned. ... the price system is used for registering changes in order to adjust their activities to changes of which they may never know more than is reflected in the price movement ...".

By adopting the market as the basic structure in distributed problem solving systems, it is hoped that the same kind of information efficiency would take place in task allocation. The system with a market structure eliminates all forms of direct control between agents. Communication among agents is accomplished by means of task announcement and the bid submission messages. The market mechanism will ensure that tasks be allocated through negotiation between the manager (supplier) and the contractor (buyers). By using the contract net

formalism to emulate the market, contract managers adopt an evaluation function to rank their bidders; this evaluation function, for instance, may be a function of the completion time needed by a bidder.

Another characteristic of the market metaphor for DPS is that the relationships between agents are dynamically established. In a sense, they are many-to-many relationships. An agent can be the manager of one task and the contractor of another. Moreover, when a contract of a task is established, a hierarchical relationship is initiated: the manager acts as the superior to the contractors. The construction of such relationships is referred to as "dynamic reconfiguration" since the new relations between agents are added when new tasks are allocated; a task is always assigned to the best agent available. Thus, the market structure for distributed problem solving systems introduces both efficiency and flexibility to the process of task allocation.

Hierarchies represent the other structure suited for distributed problem solving systems, especially when there are many agents in the system and the amount of information exchanged is prohibitively large. The basic idea of using hierarchical structure is to divide the system properly into units, so that most of the required information flows occur within the unit. The few information exchanges then are handled by the set of unit managers.

Identifying the reduced information flows in hierarchies, Simon [1981] observes:

> "In fact the main advantage to be gained from hierarchic authority is identical with that gained from using prices as communicators: matters of fact can be determined at the particular loci in an organization that are best equipped by

> skill and information to determine them, and they can then be communicated to 'collecting points' where all the facts relevant to a specific issue can be put together and a decision reached. Only a small part of the source knowledge and information and expertise need be present at the collecting points, and these points can themselves be numerous and dispersed through the organization."

An extension of the hierarchical structure, referred to as the modular-network structure, has been used for various distributed problem solving applications. The systems of this structure consist of a group of modules, with each module being a hierarchy controlled by a manager. Examples of this structure can be found in the computer network system Cm*, in the cellular system in Shaw and Whinston [1983] and in the distributed problem solving network (Corkill [1982]). The underlying strategy is similar to the notion of "division of labor" in organization theory - the objective of both approaches is to reduce the task complexity. Two different mechanisms are integrated and used to coordinate activities in this kind of organization: a coordinating mechanism that operates through the set of managers and a coordinating mechanism operating within each unit.

Shaw and Whinston [1983] incorporate the market mechanism into such organizational structures. Specifically, the contract negotiation process is used to allocate subproblems among the set of managers. Once assigned, the manager of each unit in turn coordinates the agents in its unit to complete the subproblem tasks.

2.3.5 Satisficing versus Optimizing

As has been discussed, in order to better utilize the resources, an agent must spend more time computing and problem solving than communicating and the system should be loosely-coupled. On the other

hand, the necessity to keep global coherence among the agents demands that their problem solving activities be well-coordinated to accommodate any interdependency or conflict; communication activities are required to support this coordination. What is desired is a balance between coordination and problem solving so that global coherence is in effect and the combined cost of both activities is acceptable. Thus, sometimes the emphasis is shifted from optimizing the performance of problem solving to achieving an acceptable performance level, in order to avoid the costly searches for optimalities. This trade-off is referred to as satisficing versus optimizing.

This satisficing principle is adopted by existing distributed problem solving systems. In the contract net formalism, for instance, the negotiation process can only achieve suboptimal task assignments. There are two reasons that prohibit global solutions. First, the negotiation process results in myopic assignments. Since the future tasks are unpredictable, it may happen that all agents would get better assignments (better matched to their expertise) if they would wait longer so that more tasks are included in determining the assignment. Second, it is a greedy method: each agent always selects the highest ranked task; however, it may happen that the group as a whole would be better off (according to some kinds of welfare functions) if some agents did not choose their highest ranked tasks. The primary advantage of using the negotiation process is the efficiency in determining assignments, at the expense of optimalities.

2.3.6 Summary

Various strategies used in the distributed problem solving systems discussed in Section 2.2 are summarized in Table 2.1.

Table 2.1 A Comparative Study of Four Distributed Problem Solving Systems

	Heresay II	Ether	Contract Net	DPS Net
Global Coherence	Yes	Yes	Yes	No/Yes
Coordinating Methods	Self-Directed	Externally-Directed	Negotiation	Meta-Level Controlled
Organizational Structuring	Hierarchical	Scientific Communities	Markets	Organizational Self-design
Task Allocation	Schedular in the Blackboard	Sponsors	Contract Net Protocol	Organizational Blackboard
Communications Policies	Tightly-Coupled Shared Blackboard	Tightly-Coupled Broadcasting	Loosely-Coupled; Broadcasting or Point-to-point	Loosely-Coupled; Limited Broadcasting

CHAPTER 3
PLANNING IN A DISTRIBUTED ENVIRONMENT

3.1 Introduction

This chapter describes the methods for planning in a distributed environment where a group of agents cooperate to generate plans automatically. This approach is applied to the computer integrated manufacturing environment, where robots and other agents need to organize their activities and complete operations for different jobs in the system. The planning system is used to develop a course of actions for the agents and to transform the system from the initial condition to the desired goal conditions. The course of actions generated, including the actions required for coordination, form the resulting "plan". The planning system is then used to monitor the execution of the plan and to modify the plan dynamically if required.

A general planning system that generates plans automatically involves: representing the world, representing actions and their effects on the world, searching for sequences of actions to achieve certain goals, reasoning about interactions of actions that are taking place concurrently, eliminating harmful interactions between concurrent actions, and monitoring the execution of the resulting plan.

The addition of two elements - "resources" and "durations" - to the standard STRIPS formalism is emphasized. Besides providing a better description of the actions relevant for planning, both

descriptions can also be used to develop effective heuristics in the plan construction. Resources can be used to identify the conflicting actions in parallel subplans; the duration information is used in deciding the precedence relationship between the conflicting actions.

If the planning is to be accomplished by several agents, referred to as multi-agent planning, two approaches are plausible. The first is the centralized approach, where a single planning agent generates plans to be carried out by other agents and then hands out the pieces of the plan to the relevant individuals. The second approach is a decentralized one, where each agent concurrently constructs portions of the plan; together, the collection of plans generated by individual agents achieve the desired goal conditions.

This chapter will also address the decentralized, multi-agent planning problem, commonly referred to as distributed planning. The primary issue to be addressed is the coordination and synchronization of the activities of individual planning agents to form an integrated plan collectively. Since the plan is generated by the group of agents in a decentralized fashion, the necessary interactions should be carried out by communication activities among the agents. With this approach, the communication activities are treated the same as the other activities in the domain. In other words, cooperating with other agents is part of the expertise of each planning agent.

There is an issue that is important in distributed planning but is ignored in this chapter: the decomposition of the original problem and the assignment of subproblems to the group of agents, referred to as "task allocation". This aspect of distributed planning is treated

in Chapter 4, where a decentralized scheme similar to the market mechanism is used to achieve task allocation. In this chapter, we simplify the problem and assume that the original goal to be achieved is a conjunction of several subgoals, with each subgoal assigned to an agent.

This assumption is valid for computer-integrated manufacturing systems. For example, the goal posted may be the completion of m parts. Carried by mobile robots, each part requires different sequences of operations. A natural way of task allocation is to decompose the goal into m subgoals; each subgoal corresponds to a part with a sequence of operations. Thus, the stated planning problem is: how to let each robot generate a plan for the part it carries, while the multiple plans efficiently utilize the available machines without conflicting in the activities of other robots.

Coordinating plan generations among multiple agents is similar to managing concurrent processes in a multiprocessor operating system, where numerous processes contend for access to the processors and other system resources. The job of the operating system is to provide synchronization and mutual exclusion among processes, so that none of the processes will interfere with one another.

By adopting the synchronization and communication techniques from the operating system field, the multi-agent planning is modeled as concurrent processes. The interactions between single-agent plans are handled by synchronization mechanisms using the message-passing approach. The portion of the plan that uses a resource is treated as the critical region - where mutual exclusion is ensured. This method

of treating and analyzing a resource explicitly is adapted from the method and viewpoint used in the SIPE system (Wilkins [1982]). In a similar context, Nof et. al. [1980] propose a "manufacturing operating system" to manage concurrent manufacturing activities; a variant of the Petri net model is used to regulate the operational logic and thus direct the flow of manufacturing processes among the processors.

The organization of the remainder of the chapter is as follows. In the next section, the general knowledge-based planning method is introduced and various issues concerning knowledge representation methods and control strategies are addressed. In the following section, the nonlinear planning approaches are discussed, focusing on the handling of subproblem interactions. Existing nonlinear planning systems, e.g., NOAH (Sacerdoti [1977]), NONLIN (Tate [1977]), DCOMP (Nilson [1980]), SIPE (Wilkins [1982]) and DEVISER (Vere [1983]), are compared. To illustrate the conflict-avoiding method used to maximize concurrency between subplans, an example that applies nonlinear planning method to the scheduling problem in a flexible manufacturing cell is discussed. In the last section, distributed planning is treated as an extension of the nonlinear planning approach, with synchronization primitives inserted in single-agent plans for proper interaction. A two-robot machine loading problem in a flexible manufacturing system is used to illustrate the distributed planning approach.

3.2 The Knowledge-Based Approach to Planning

The primary objective of a planning system is to develop an appropriate course of actions, among all the possible actions, that

transform the system (referred to as the "world") from the current condition to a desired goal condition. The course of actions generated by this process is called a plan. Such a planning system is exemplified by the robot planning system, where robot plans are generated by selecting and synthesizing robot actions to achieve some stated tasks in a given environment.

The planning system is organized in the form of a knowledge-based system; that is, the knowledge is organized on three levels: data, knowledge base, and control - as opposed to conventional programs where the knowledge is organized on just two levels: data and program. In the knowledge-based planning system, the declarative knowledge about the goals, the current situation of the world, and the semi-finished plan constructed are stored in a database at the data level. On the other hand, in the knowledge-base level is the domain-specific, procedural knowledge. This knowledge is used to model the behavior of the world, and is often given in the form of rules or operators. Finally, in the control level is the knowledge about the strategy of plan construction; it is related to the decisions of how to select operators and when to apply them. This separation of control from the program is one of the major characteristics of knowledge-based systems.

In a conventional computer program, knowledge pertinent to the program and methods for utilizing this knowledge are all intermixed, so that it is difficult to change the program. Using the knowledge-base approach, the program itself is only a reasoning and control mechanism; the system can be changed or remodeled by simply adding or

subtracting rules in the knowledge base.

Because planning involves exploration of alternative sequences of actions, a symbolic model of the real world, referred to as a world model, is used to serve as an abstraction of 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 general function of a planning system, then, is to construct a course of actions that transform one world model containing an initial condition to a world model which matches the goal condition. Thus, a planning system must have three basic components:

1) A world model. Represented as declarative knowledge at the data level, this world model contains relevant descriptions about the environment. This knowledge is usually about the properties of domain objects (e.g., whether a machine is idle) or about the relationships among the objects (e.g., whether a part is at a certain machine). The most prevalent knowledge representation for the world model is the use of first-order predicate calculus to describe properties, functions or relations of objects (Nilson [1980]). In the planning system, corresponding to every possible situation of the environment, the world model is represented by a conjunction of the predicate formulas that hold true in that particular situation. This conjunction of instances of predicate formulas defines a "state" which describes the corresponding situation in the real-world environment.

Some of the properties of objects and relationships among objects may not change over time. For example, the capabilities of a machine or the connectivity between two machines do not change with the

application of operators; these properties are called the invariant properties of domain objects. This separation of state-changing knowledge and invariant knowledge is analogous to the reasoning process of human experts; the knowledge that changes with actions is stored in a working memory, whereas the knowledge that remains unchanged is stored in the long-term memory. The same kind of idea is applied in representing the world model.

The invariant properties of domain objects can be structuredly represented by semantic networks of frame-based systems to take advantage of their greater efficiency. In these representations, all the relevant information is collected together and properties can be inherited; accessing and manipulating the information can thus be facilitated. Further, constraints can be specified to regulate the possible values of variable instantiations, reducing the feasible domain for the solution-searching procedures.

2) An action model. This action model, represented at the domain-specific knowledge-base level, formalizes the description of applicabilities of each action and its impact on the world model when applied. An action is represented by a transformation from one state of the world model to another state; each action is represented as an operator. Using the pattern-directed representation, each operator consists of two components: preconditions and postconditions. The preconditions are represented by a predicate calculus expression, if the expression - the pattern - is matched by the state descriptions in the world model, the operator becomes applicable. The postconditions define the literals that are added or deleted by the application of

the operator, called an add list and a delete list, respectively.

When using the action model, the "frame problem" must be considered. The frame problem is the problem of specifying which conditions in a state description should change and which should not, whereas almost all conditions which hold true in a given state continue to hold true after an action has been performed. Based on the assumption of the STRIPS system, the action model assumes that all conditions that are not indicated to be changed by the operator remain the same; that is, all the changes in the world are accounted for by the postconditions of the applied operator. In our view, this assumption is valid for the manufacturing environment where the effects of the manufacturing operations can be clearly defined. A counter example of this assumption is the ill-designed block world, where stacking a block onto a stack of blocks, for instance, might topple the whole stack of blocks. Such a world is difficult for planning because of the unpredictability of actions' effects.

While the operators are used to describe the actions relevant to the problem domain, there could be other rules which provide good judgement of actions to take when specific situations arise. These are judgemental or empirical knowledge in the knowledge base, usually acquired by learning from human experts. This type of knowledge is excluded from robot planning systems, where actions are well-defined and situations are tightly controlled. In the general computer integrated manufacturing environment, however, the introduction of these judgemental rules may reduce the complexities of the problems, and result in satisficing solutions.

3) An inference engine, as the control unit. The major control decision, exercised by the inference engine, is concerned with the selection of action sequences which are based on the current state of the world model and the decision of what action best leads to the final state. Since the generation of an action sequence for a plan typically involves extensive searches among alternative actions, the inference engine produces a search tree along the plan-generation process. The goal description is at the root node, with instances of operators defining branches, and the intermediate state defining the nodes. The plan generation is then equivalent to a graph search procedure; standard search strategies such as best-first or back-tracking can be used to guide the search. Moreover, heuristic rules are sometimes used to facilitate the search of operators. An example of using heuristic rules is the "means-ends analysis". When an operator is applied with this algorithm, the difference between the new state and the goal state is determined and the operator that can best reduce the difference is chosen. This "searching cycle" continues until the goal is satisfied in the new state. If the chosen operator is not applicable, its preconditions are established as a new intermediate subgoal. The algorithm is then applied in a recursive fashion to achieve the subgoal by the same searching cycles.

To illustrate the strategies involved in means-ends analysis, the search algorithm for generating a linearly sequenced plan is shown in the following procedures:

```
Procedure OPERATOR-SEARCH(G)
Input:
     S:  the initial state
     G:  the goal state
Output:
     Plan:  the resulting linearly-sequenced plan
Begin
     1)  Plan := Nil
     2)  Until S matches G Do Begin
     3)     g := G - S /* unsatisfied goal */
     4)     op := operators whose add list contains a literal that
                matches g
                /* applicable set */
     5)     p* := nondeterministically select an operator from op
     6)     Plan := concatenate (Plan,p*)
                /* Plan contains all operators used so far */
     7)     q := precondition formula of appropriate instance of p*
                /* subgoal */
     8)     Call OPERATOR-SEARCH(q)
                /* node expansion */
     9)     S := result of applying p* to S
            End
End
```

This procedure is similar to the plan generation procedure used in STRIPS (Nilson [1980]). Using the "means-ends analysis" planning procedure, those components of G unmatched by S are treated as the

difference between the goal state and the current state, denoted by g; operators whose add list contain any literal in g are considered relevant to reducing the difference and therefore are applicable. Many operators may be applicable, but only one can be selected nondeterministically.

For large planning systems, the number of alternatives to search at each choice point can grow very fast with the size of the problem. There are two basic methods for overcoming the combinatorial explosion associated with the search in extremely complex planning problems. These two methods are:

(1) to search the space more efficiently, or

(2) to transform the search space into smaller manageable chunks that can be searched efficiently.

The planning strategies based on the first method include heuristic search and constraint satisfaction; the strategies based on the second method include hierarchical planning, problem reduction, divide and conquer, and least commitment.

As a summary of this section, Figure 3.1 illustrates the basic structure of a knowledge-based planning system.

3.3 Planning for Multiple Jobs: The Nonlinear Planning System

According to the knowledge-based approach discussed above, plans are generated by selecting a sequence of actions to achieve the desired goal state. The resulting plans are linearly sequenced, with strict precedence ordering between the selected actions. This is the simplest planning system - the linear planning system.

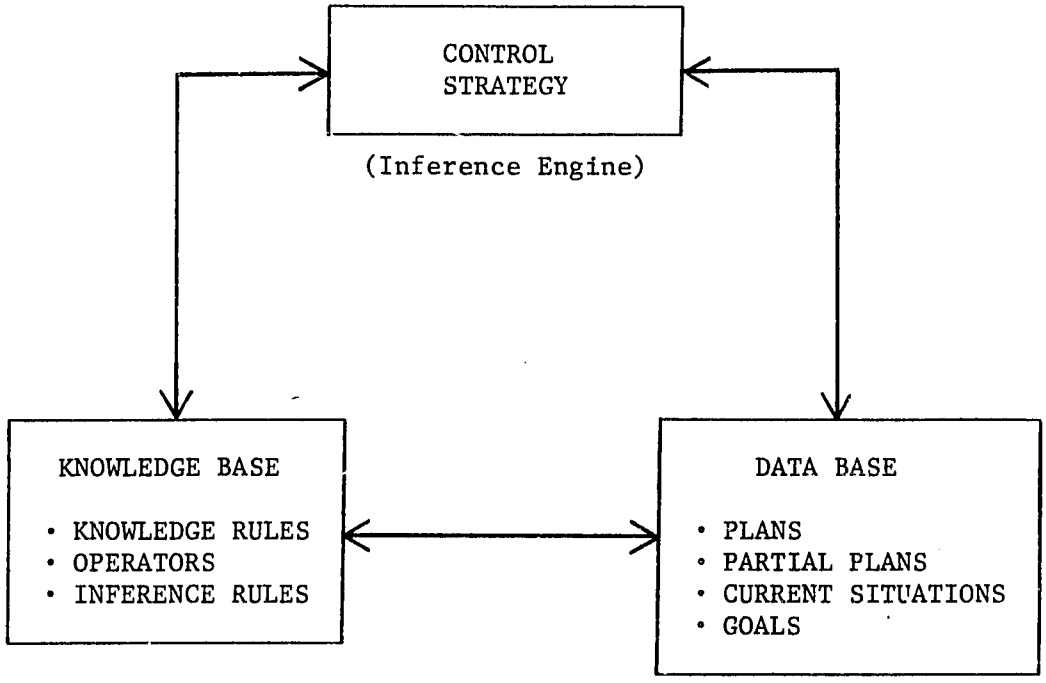


Figure 3.1 Basic Structure of a Planning System

Many planning systems have had the ability to break a problem into subproblems, that is, to take advantage of the "divide and conquer" strategy in generating plans. A common practice is breaking the original goal into a set of subgoals, developing linearly-sequenced plans for the individual subgoals, and synthesizing the plans for subgoals into the final plan. In contrast to the linear planning, however, the actions in this final plan may not have precedence relationships. The plan, no longer linearly sequenced, is referred to as a "nonlinear plan". This section is an attempt to review important nonlinear planning techniques that have been developed and to provide an overview of the planning approaches we will use in later sections in dealing with the manufacturing environment.

3.3.1 Previous Planning Systems

In nonlinear planning, if the subproblems are independent with each other, then, in principle, the plan-generation processes for subgoals can be parallel. Two segments of a plan are parallel if the partial ordering of the plan does not specify any particular precedence relationship; the total duration of such a plan is often reduced. Parallelism is considered beneficial for the planning process because of the corresponding efficiency, which is the highest when subproblems are completely solved in parallel. In most of the cases, however, the actions of one plan interacts with the actions of other plans, causing "conflicts" between these linearly-sequenced plans and disrupting the correctness of the plan. The nonlinear planning system needs to take these interactions into account and take measures to avoid conflicts between parallel actions.

The concept of nonlinear planning was first implemented in the NOAH system developed by Sacerdoti [1977]. NOAH performs planning with interfering conjunctive goals by considering the plan steps as parallel for as long as possible. Where interactions between plan steps are detected, a critics mechanism is used to sequence the steps so that conflicts are avoided. If an action for one goal deleted an expression that was a precondition of a conjunctive goal, then the action with the endangered precondition would be performed first by imposing a precedence constraint between these two actions. The resulting final plan is a partial ordering of operators, referred to as the procedure network. Nilson [1980] uses a simplified version of NOAH, called the DCOMP method, to illustrate nonlinear planning. In DCOMP, planning operators are at a single level as opposed to the hierarchical structure used to define operators in NOAH.

Tate [1977] extends the techniques used in NOAH and develops a planning system called NONLIN. While capable of solving some problems that are beyond NOAH's capabilities, NONLIN also attempts to use the nonlinear planning method to aid the project management technique in operations research. The partially ordered network of actions generated by the NONLIN system can be used to automate the process of specifying consistent jobs and identifying the precedence relationships between jobs in the construction of the project network. Techniques such as critical path analysis can then be applied to establish schedules and allocate resources. NONLIN differs with NOAH in two aspects:

1) NONLIN is capable of backtracking by keeping the backtracking points in the searching for actions.
2) By ignoring the effects of actions which were not relevant to the application of latter operators, NONLIN can search more efficiently.

For every operator that appears in the plan, a "goal structure" is used in NONLIN to keep a list of operators whose application could make the precondition hold. This allows NONLIN to recognize relevant interactions and to be sensitive to important effects of operators in resolving conflicts between actions.

The SIPE system developed by Wilkins [1982] is a domain-independent planning system that generates hierarchical, partially ordered plans. Unlike other nonlinear planning systems, SIPE is designed to also allow interaction with users throughout the planning and plan execution processes. While most planning systems use a single representation for their world model, SIPE incorporates the frame-based system, for its efficiency, to represent the invariant properties of domain objects. Predicate calculus is used, for its representational power, to represent state-changing properties. Thus, the number of literals in the state descriptions is reduced and pattern-matching is made more efficient.

The second contribution of SIPE is its ability to reason about resources. The representation of operators includes the specification of the object employed by the corresponding operator as a resource. When an operator is applied, the planning system automatically checks for the availability conditions of the required resource and avoids any possible conflict. The declaration of a resource increases the

representational power of an operator and facilitates the detection of conflicts between operators.

The third important aspect of SIPE is its use of constraints to restrict the possible values of domain variable instantiation, enabling it to be capable of constructing partial descriptions of unspecified objects. Incorporating the constraint satisfaction technique in the searching of the solution enables SIPE to carry out more efficient planning.

DEVISER (Vere [1983]) is a nonlinear planning system that accounts for time and durations explicitly. Parallel plans are synthesized to achieve goals with imposed time constraints; actions and events may have computable, deterministic durations. The final plans generated by DEVISER are partially ordered networks of activities, with a duration and a start time "window" presented with each activity.

A window specifies the upper and lower bound on the time when an activity may occur. Windows for activities are computed dynamically during plan generation and are derived from goal windows by the consideration of the durations of interfering activities and the times of occurrence of prespecified events. Constraints on the window boundaries must be satisfied when two nodes are sequential or consecutive; if a time constraint is violated, the planner must backtrack to search for activities consistent with the time constraint.

An alternative approach can be found in Bullers et. al. [1980], where they apply the problem reduction approach to perform planning and control in the manufacturing domain. Predicate logic and theorem proving techniques are used in deriving manufacturing steps in the dynamic environment.

3.3.2 An Overview of the Proposed Method

The planning method we will develop for the computer integrated manufacturing environment has characteristics resembling those of the systems previously discussed. We shall briefly present its features in this section and then illustrate the method in more detail by solving an example problem in the next section.

Based on the nonlinear planning methods developed in NOAH and DCOMP, our method begins by constructing a linear plan for each subgoal. However, instead of testing the preconditions and the postconditions of the operators to decide their interactions, we will use resources as the basis for detecting harmful interactions. Since most of the conflicts between parallel plans in the manufacturing environment are related to resources, our method is more explicit and easier to use.

Although the idea of explicitly declaring resources was originated by Wilkins' SIPE [1982], our method differs with his in one important aspect. SIPE's embedded planner checks on the resources availability when operators are applied, and the operator will not be used at all if the resource is not available. Our method is a greedy algorithm in the sense that a linear-sequenced plan is generated for each subgoal, regardless of the resource availability, and then use precedence constraints to maintain the correctness in resource accesses.

As with DIVISER (Vere [1983]), our method uses the duration as an important description in the planning, especially in deciding precedence constraints. However, by emphasizing resources used by activities, the resulting plan network is more expressive for

managerial analysis, e.g., sensitivity analysis like "what if a certain resource is not available."

To minimize the total time taken by the resulting plan, we use a plan generator to decide the best ordering among the activities. The plan generator runs the activities in the way they are arranged by the linear planner, with each operator scheduled into an event-list. An operator is blocked in a queue if the resource it requests is occupied, thus, a precedence constraint must be established in the plan between the operator currently using the resource and the operator which is blocked; the former is, then, the predecessor, the latter is the successor. The partial ordering thus constructed will minimize total duration of the plan.

3.4 An Application: The Sequencing and Scheduling Problem in a Manufacturing Cell

3.4.1 The Problem

We shall now illustrate the application of the planning method presented above to a sequencing and scheduling problem in a flexible manufacturing cell.

A manufacturing cell is a flexible manufacturing system (FMS) and is usually a modular unit in a large-scaled computer integrated manufacturing system. A typical manufacturing cell has several computer-controlled machines and robots, with an automatic handling system (e.g., the linear table in Figure 3.2) transporting parts between machines. Such integrated systems are characterized by their flexibilities in making parts and their capabilities of performing a wide range of operations.

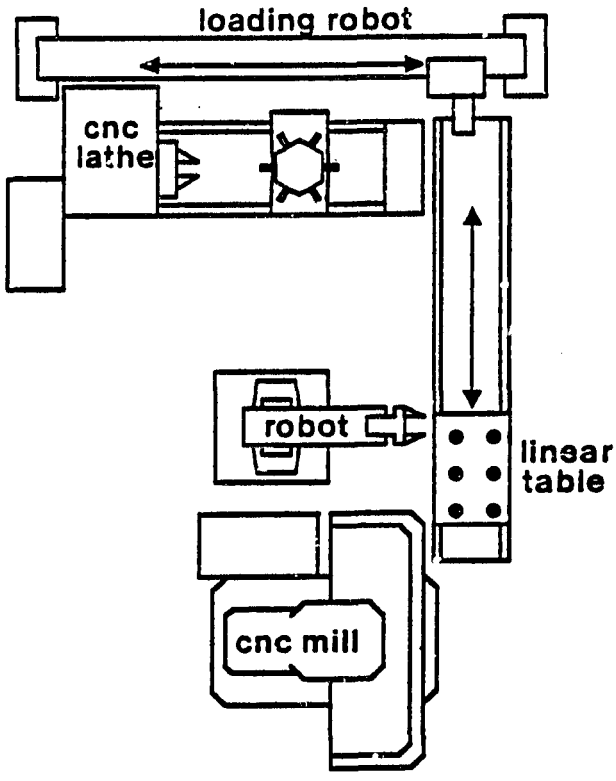


Figure 3.2 The Organization of a Manufacturing Cell

The problem to be solved is the following. A certain number, say N, parts are assigned to the manufacturing cell; each part requires a given set of linearly-sequenced operations to be performed by the three machines. Since the machines have varying efficiencies for different operations, each part is routed among the machines in the hope that every operation it needs is performed by the most appropriate machine available.

The objective of the problem is to schedule the N parts concurrently by developing a schedule for each part traveling among the machines; the makespan - the duration taken for completing all the required operations - should be minimized, while avoiding any conflicts arising from assigning parts to a busy machine.

The problem of scheduling these parts is treated as the problem of generating plans for all the parts. An N-part-M machine scheduling problem can be decomposed into N subproblems, with each subproblem defined as the routing of one part. The nonlinear planning method discussed in Section 3.3 can thus be utilized to generate a plan for the N subproblems; the primary "interactions" between these subproblems are their sharing of the M machines. The objectives of the scheduling problem - to minimize makespan and to avoid conflicting assignments - can be translated to the criteria of the plan generation: to maximize concurrency and to avoid harmful interactions among the subplans. Before discussing the planning system, the parameters of the problem are introduced.

For this particular example, the flexible manufacturing cell, as shown in Figure 3.2, consists of three computer-controlled machines,

as denoted by MACH 1, MACH 2 and MACH 3 in Table 3.1.

Table 3.1 The Machines

CNC lathe	MACH 1
Welding robot	MACH 2
CNC mill	MACH 3

Suppose there are two parts entering the system that need to be scheduled; these parts require different sequences of machining operations, as listed in Table 3.2 and Table 3.3 below:

Table 3.2 The Operations

OP 1	surfacing
OP 2	welding
OP 3	finishing

Table 3.3 Operation Requirements of Each Part

Part 1 (PT1)	OP1, OP2, OP3
Part 2 (PT2)	OP1, OP3

Each machine is capable of performing a different set of operations, while some operations may be performed by several machines. The operations each machine is capable of performing are shown in Table 3.4. The machines perform the set of operations with varying speeds, as reflected in the average time taken for the operations on each of the machines shown in Table 3.5. Finally, the average time

Table 3.4 The Capabilities of Machines

MACH 1	OP1, OP3
MACH 2	OP2
MACH 3	OP1, OP3

Table 3.5 The Average Operation Times (unit)

MACH \ OP	1	2	3
1	3	-	7
2	-	6	-
3	8	-	4

to transport the parts from one machine to another through the linear table and the time for the robot to load or unload the parts are shown in Table 3.6 as follows.

Table 3.6 Average Transfer and Load/Unload Time

Average Transfer Time (Linear Table)	2
Average Load/Unload Time (Robot)	1

3.4.2 The Knowledge-based Planning System

Using the knowledge-base approach, the expert planning system is organized on three levels: data, knowledge, and control. A world model at the data level is used as a description of the environment of the manufacturing cell. A set of pattern-invoked operators in the knowledge base is used to model the actions to be planned. An inference engine applies the control knowledge to search for the best course of actions in constructing the plan while reasoning about resources, interactions, and precedence orderings.

In the remainder of this subsection, we shall address the necessary knowledge at the three levels of the expert planning system to solve the stated problem.

3.4.2.1 The Declarative Knowledge in the Database

There are three kinds of knowledge stored in the database: the world model, task descriptions, and the plans. The world model used to describe the environment of the manufacturing cell is shown in Table 3.7, where first-order predicate literals are used for knowledge representation. This set of predicate literals have different

Table 3.7 The Set of Predicates

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

SAME(M,M') : Machine M is machin e M'

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

characteristics in terms of the properties or relations they are describing. There are three types of literals:

1) Invariant literals. This set of literals would not be affected by the application of operators. Examples of these literals are: MACH-OP, PT-FIRST-OP, PT-NEXTOP, etc.
2) Functional literals. They are used to describe functional relationships. Examples are SAME, DIFFERENT, LE, etc.
3) State-descriptions literals. These literals change with the applications of operators and are the state-changing elements of the world model. Examples of this type of literals include IDLE, MACH-PT, FINISH-OP, and DONE.

There is another kind of knowledge stored at the data level: the representation of the generated plan by a partially ordered network of activities. The plan representation is used to monitor the execution of the planned activities; if there are deviations between the conditions specified by the plan and the conditions in the real world, the planning system should take measures to modify the rest of the plan.

The plan representation can also be used to accommodate dynamically changing environments. For instance, in the manufacturing example, new parts entering the system may request to be scheduled while the existing parts in the system are not completely done. By monitoring the progress of the execution, the planning system is capable of detecting the current actions that remain to be executed. One approach is to leave the remaining plan intact, thus, generating a plan for the new part that does not conflict with the existing plan.

3.4.2.2 The Domain-specific, Knowledge-base Level

A set of operators, stored in the knowledge base, are used to represent actions the system may perform. Each operator contains information about the object that participates in the actions, what the actions are attempting to achieve, the effects of the actions when they are performed, and the conditions necessary before the actions can be performed. The action formalism represented by such an operator is specified as follows:

<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>

Besides the standard STRIPS formalism, which specifies an action by its add list, delete list, and preconditions, we have 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. One is the increased representational power of the action model; the other is the facilitation of conflict detection and conflict resolution.

One of the major functions of the planning system is the effective coordination of resource usage. Resources are to be employed during a particular action and then released; reasoning about resources is essential to ensure that only one action is using the resources at any given time. For the manufacturing environment, the major forms of

resources are machines, robots, and human operators.

If the resource used by an operator is not represented explicitly, then resource availability would have to be recognized solely by the preconditions and postconditions of the operators. Stating the use of the resource explicitly, on the other hand, is a way of saying that the resource has to be available before the operator can be applied. It is easier for the planning system to reason about resource availabilities by the explicit resource declaration than by checking with the various pre- and postconditions of operators. This point will be further discussed in Section 3.4.2.3.

The duration information is important because the constraints and the goals of planning problems in the manufacturing domain are often related to time; for example, the due date of a job, the machine time available, etc. The set of operators is shown in Table 3.8.

Furthermore, planning a manufacturing process usually includes the objective that the total duration of the process be minimized. This can only be achieved if the duration of each action is represented explicitly. As will be explained in Section 3.4.2.3, the durations are used to decide the ordering between the conflicting actions and to help achieve a minimized-duration plan.

3.4.2.3 The Control System

At the control level, an embedded inference engine is used to develop and organize the necessary actions to accomplish the goals. Since, in this example, there is a natural decomposition of the goal into M subgoals, with each subgoal able to generate a linear plan for the corresponding part. The generation of plans can be accomplished

Table 3.8 The Set of Operators

ENTER(PT , t) : Part PT enters the system at time t

Precondition : PT-FIRST-OP(PT,LOAD)

IDLE(DOCK,t)

Add-list : MACH-PT (DOCK, LOAD, PT, t)

Delete-list : IDLE(DOCK, t)

Resource : DOCK

Duration : 0

EXECUTE(M, OP, PT, t) : Execute operation OP on part PT on machine M at time t.

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

TOOL(M, OP, t)

Add-list : FINISH-OP(M, OP, PT, $t+\delta t$)

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

Resource : M

Duration : δt

Table 3.8, continued.

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

Table 3.8, continued.

UNLOAD(M, DOCK, PT, t) : Unload part PT from machine M onto the unloading dock DOCK at time t.

Precondition : IDLE(DOCK, t)

PT-NEXTOP(OP, NIL, PT)

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

Add-list : MACH-PT(DOCK, "unload", PT, t)

Delete-list : IDLE(DOCK, t)

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

Resource : M

Duration : 3

EXIT(PT, t, δt) : Part PT which is unloaded at the unloading dock DOCK at time t leaves the system at time t+δt.

Precondition : MACH-PT(DOCK, "unload", PT, t)

TOOL(DOCK, "unload", t)

Add-list : DONE(PT, t+δt)

IDLE(DOCK, t+δt)

Delete-list : MACH-PT(DOCK, "unload", PT, t)

Resource : Dock

Duration : δt

by the following operations:

1) Generate a linearly-sequenced plan for each subgoal; these plans are called "subplans".
2) Identify problematic interactions between the actions of parallel subplans.
3) Linearize the conflicting actions; construct precedence constraints between the pairs of conflicting actions to avoid harmful interactions.

3.4.2.3.1 Plan Generation for Subproblems

The generation of linear plans can be carried out by any STRIPS-like plan generation system. For example, the recursive OPERATOR-SEARCH algorithm introduced in Section 3.2 can be used for this purpose.

The planning system uses a backward-chaining method in its searching for the best actions, it works backward from the goal state to 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, instances of operators defining the branches. The solution path, which starts with the root (the goal state) and leads to the leaves (the initial state), defines the plan. The search trees generated by the planning system for part 1 and part 2 are depicted in the Appendix (see Figures A.1 and A.2). The resulting linearly sequenced plans are shown in Figures 3.3 and 3.4.

```
operation : op1, op2, op3

p1    ENTER
p2    EXECUTE(LOAD,DOCK)
p3    TRANSFER(DOCK,M1)
p4    EXECUTE(M1,op1)
p5    TRANSFER(M1,M2)
p6    EXECUTE(M2,op2)
p7    TRANSFER(M2,M3)
p8    EXECUTE(M3,op3)
p9    UNLOAD(M3,DOCK)
p10   EXIT
```

Figure 3.3 Linearly Sequenced Plan for PT 1

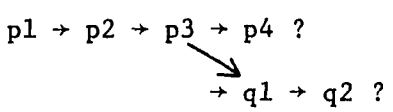
```
operation : op1, op3

q1   ENTER
q2   EXECUTE(LOAD,DOCK)
q3   TRANSFER(DOCK,M1)
q4   EXECUTE(M1,op1)
q5   TRANSFER(M1,M3)
q6   EXECUTE(M3,op3)
q7   UNLOAD(M3,DOCK)
q8   EXIT
```

Figure 3.4 Linearly Sequenced Plan for PT 2

3.4.2.3.2 The Conflict-Detection Mechanism

After a linear plan is constructed for each subgoal, the next step is used to identify problematic interactions between parallel actions. The primary cause of such interactions is the potential conflicts in using resources. There are two possible approaches in this step. The first approach is based on the method used in NOAH (Sacerdoti [1977]) and DCOMP (Nilson [1980]); the interaction-detection mechanism, called the "critic", of the planning system identifies potentially harmful interactions between planning steps by checking the effects of the operators involved. If the preconditions of an operator to be applied is deleted by an operator previously applied by the planner, the current operator has to be delayed until its deleted preconditions are added back by some other operator applied later. Thus to test the potential conflicts for an operator, two kinds of information are crucial: those operators in the plan that can delete its preconditions, and those operators in the plan that can result in its preconditions; the former is recorded in an "adder list", the latter is recorded in a "deleter list". These two lists are parts of a table kept by the planner, referred to as the table of multiple effects (TOME), as shown in Table 3.9.

This table can be used to detect conflicts systematically; for instance, if the plan developed so far is as follows:



According to the linearly-sequenced plan already built, the next operator to apply is q3. However, by checking with Table 3.9, one of

Table 3.9 The Table of Multiple Effects (TOME)

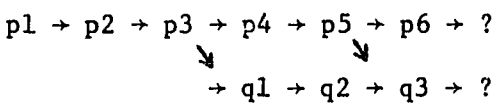
	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
A	--	--	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M2)	--	IDEL (M3)	IDEL (DOCK)
D	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M2)	--	IDEL (M3)	--	IDEL (DOCK)	--
P	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M2)	--	IDEL (M3)	--	IDEL (DOCK)	--
DL	q1,q7	--	q3	--	--	--	q5	--	q1,q7	--
AL	q3,q8	--	q5	--	--	--	q7	--	q3,q8	--

	q1	q2	q3	q4	q5	q6	q7	q8
A	--	--	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M3)	IDEL (DOCK)
D	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M3)	--	IDEL (DOCK)	--
P	IDEL (DOCK)	--	IDEL (M1)	--	IDEL (M3)	--	IDEL (DOCK)	--
DL	p1,p9	--	p3	--	p7	--	p1,p9	--
AL	p3,p10	--	p5	--	p9	--	p3,p10	--

A: Add list
D: Delete list
P: Precondition

DL: Deleter List
AD: Adder list

the elements in its deleter list, p3, is in a parallel branch; thus, there is a risk of conflict between these two operators' actions if we do not restrict their invocations. By checking with Table 3.9 again, we find that p5, which is on q3's adder list, will result in the pre-condition which q3 requests. Therefore, q3 must not be applicable until p5 is finished; otherwise, there would be a conflict. This can be explicitly represented by imposing a precedence constraint stating that q3 be a successor of p5, denoted by p5 → q3. Thus, the plan has a new precedence ordering between p5 and p3:



By observing the table of multiple effects shown in Table 3.9, we find that in the manufacturing environment such as the one in this example, the major cause of conflict is machine usages. We also find the predominant constraint used in the conflict-resolution step to be the restriction that the operators requiring the same machine cannot be in parallel branches. The preceding approach using TOME is just one way to construct this constraint. Next, we shall attempt to use resources as a basis to recognize and avoid conflicts between actions. As will be seen, resource information makes the task of conflict detection easier.

3.4.2.3.3 Reasoning About Resources

In a broader sense, a resource can be defined as the object used by the corresponding operator during its application and cannot be shared by more than one operator. Declaration of a resource by an operator imposes the condition that the resource must be available for

the operator to be applicable and that the resource will be occupied by the action - as represented by that operator - during its application.

Using the resource information, the first step to detect problematic interactions is to identify critical sections of each subplan. A critical section is defined to be a set of consecutive operators that must be executed as an indivisible planning step. When consecutive operators in a subplan declare the same resource for their actions, these operators form a critical section.

A critical section in a plan has exclusive access to the resources its operators require; operators in another section that require the same resource may not have access until the resource is released. This mutually exclusive execution of critical section is called mutual exclusion. This concept is the essence of our approach to detect conflicts. In the manufacturing domain, critical sections and mutual exclusions can be used to regulate that consecutive operations in a machine cannot be interleaved, which appears to be appropriate.

By observing the resources used by each planning step, as shown in Table 3.10, there are seven critical sections:

[p1,p2], [p3,p4], [p5,p6], [p7,p8,p9]

[q1,q2], [q3,q4], [q5,q6,q7] .

The operators in the set listed above are consecutive operators that use the same machine.

Next, those critical sections in parallel subplans that require the same resource must be declared mutually exclusive. The set of mutually exclusive pairs are:

Table 3.10 Resource Declarations of Planning Steps

Action	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
resource	DOCK	DOCK	M1	M1	M2	M2	M3	M3	M3	-

Action	q1	q2	q3	q4	q5	q6	q7	q8
resource	DOCK	DOCK	M1	M1	M3	M3	M3	-

[p1,p2] : [q1,q2]

[p3,p4] : [q3,q4]

[p7,p8,p9]: [q5,q6,q7].

If the conflict-detection mechanism in Section 3.4.2.3.2 is used, an operator is blocked if one of its deleter is in a parallel branch of the plan. The operator is reactivated when an operator in its adder list is applied. Similarly, by using resource information, the potential harmful interaction of an operator is detected if another operator in a mutually-exclusive critical section is in a parallel branch of the plan. The current operator then has to wait until the resource is released.

This mutual exclusion between critical sections that require the same resource can be enforced by semaphors as used for concurrency management in operating systems. A semaphor is an integer variable shared by subplans; each resource is associated with one semaphor. The value of a semaphor, either zero or one, is used to signal the status of the resource. When the semaphor is one, the resource is available; when the semaphor is zero, the resource is occupied. Using the concept of this semaphor mechanism, a conflict-detection procedure based on resource reasoning can be as follows:

Procedure CONFLICT-DETECTION(P)

/* to check if an operator P can be applied */

Begin

R := RESOURCE(P)

```
If S(R) = 1  Then  /* resource is available */
    Applicable := TRUE
    S(R) = 0
Else   /* resource is busy */
    Applicable := FALSE
Return(Applicable)
End
```

To implement mutual exclusion in critical sections of the subplans, each critical section is preceded by a P operation and followed by a V operation on the same semaphor. Given a semaphor S, P(S) delays until S > 0 and then executes S := S - 1; V(S) executes S := S + 1; the semaphor is always initialized to one. Since the mutually exclusive sections share the same semaphor, only one of the sections can get access to the critical region. The blocked operator will wait in a queue associated with that semaphor until the resource is released. Then the semaphor turns to one again and the waiting operators can start using the resource. The parallel subplans of the problem using semaphors to regulate resource accesses are depicted in Figure 3.5.

The advantages of using the semaphor mechanism to regulate the usage of resources are its simplicity and correctness in keeping mutual exclusion. The primary usage of this kind of synchronization is to monitor the execution of a plan. The insertion of P, V operations in the plans can block an operator from using an occupied resource during the execution of the plan. However, if we want to construct a partially ordered plan in advance, so that we can apply

$P(S_0)$ → p1 → p2 → $V(S_0)$ → $P(S_1)$ → p3 → p4 → $V(S_1)$ → $P(S_2)$ → p5 →
p6 → $V(S_2)$ → $P(S_3)$ → p7 → p8 → p9 → $V(S_3)$ → p10

$P(S_0)$ → q1 → q2 → $V(S_0)$ → $P(S_1)$ → q3 → q4 → $V(S_1)$ → $P(S_3)$ → q5 → q6 → q7 →
$V(S_3)$ → q8

Figure 3.5 Parallel Plans with Synchronization Operations

analytical models based on the partially ordered network, other techniques are needed.

In the following section, we shall discuss a procedure for deciding the precedence ordering between mutually exclusive operators. A partially ordered network of operators/actions is produced by this procedure. Because parallelism is maximized during the construction, this partially-ordered network, together with the duration information, provides us with a schedule that, for the example, has the shortest total duration.

3.4.2.4 Constructing the Partially-Ordered Plan

In Section 3.4.2.3, two approaches to detect interactions are explored; one is based on the use of a table of multiple effects in identifying potential conflicts, the other is based on resources. Once the problematic interactions are identified, these conflicts should be avoided by imposing a sequential ordering between the conflicting operators/actions; the sequential ordering thus imposed is referred to as the "precedence constraint" between the two actions.

After a problematic interaction between two actions is identified, a decision is needed regarding the direction of the precedence constraint. As far as the "feasibility" of the plan is concerned, any one of the two actions can precede the other, as long as there is no overlap in the resource utilization. However, to achieve the objective of maximizing parallelism, or, equivalently, minimizing the total duration, the planner needs to identify the action which will take place earlier than the other.

A procedure similar to the discrete simulation method is used to dynamically decide the precedence relationship between two conflicting actions. The underlying principle - based on the least commitment strategy - is not to impose any precedence constraint unless it is absolutely necessary, so that the parallelism among the subplans is maximized. A plan generator, called PLAN-AHEAD, is used to decide the ordering of actions in parallel plans. The information about resources and duration of actions is crucial to the inference engine in making these decisions.

The basic idea of the plan generator is as follows. The plan generator has a global clock and an EVENT-LIST. The global clock, represented by TNOW, is updated discretely to the time when the next event occurs. All the future events that are to occur are scheduled in a "calendar", represented by a list called EVENT-LIST.

The EVENT-LIST is ordered by the occurrence time of its element; each element in the EVENT-LIST is an operator to be applied next in one of the subplans. In other words, the first element on the EVENT-LIST contains the information of what the next applicable operator is, and when it will become applicable. After TNOW is updated to that time, the CONFLICT-DETECTION routine is used to check if there's any interaction that prevents this operator, p*, from being applied. Two situations can happen:

1) If the resource requested by p* is occupied, then p* is put into a queue corresponding to that resource. p* will stay in the queue until the current occupant, p', leaves the resource. p* will become applicable again at that time.

2) If the resource is available, p* is applied; its successor in the linear subplan is then scheduled on the EVENT-LIST to be applicable when p* is finished. The event-occurrence time of the successor is calculated by the sum of TNOW and the duration of p*. The queue corresponding to the resources used by p* is checked to see if any operator is blocked by p*. If there is, say a r, waiting for the resource, then r becomes applicable and a precedence constraint is imposed between p* and r. p* must precede r.

The plan generator can be implemented in a PASCAL-like procedure, as follows.

```
Procedure PLAN-AHEAD
INPUT:  N linearly-sequenced plans; the first operator of plan i is
        O_i
OUTPUT: Plan: a partially ordered network
Begin
(1)  Initialization:  TNOW := t0; Plan := Nil;
(2)  Put (O_i,TNOW) on the EVENT-LIST, i=1,...,N
(3)  While EVENT-LIST <> Empty Repeat
     Begin
(4)  (p^i,T) := the first entry on EVENT-LIST
(5)  w := PREDECESSOR(p^i)
(6)  If (RESOURCE(p^i) <> RESOURCE(w)) Then
        Begin  /* the resource used by plan i is released */
(7)     If queue(RESOURCE(w)) is nonempty Then
           Begin  /* reactivate the blocked operator */
(8)        r := the first operator on the queue
```

```
(9)         Plan := SEQUENCE(w,r,Plan)
(10)        Plan := SEQUENCE(PREDECESSOR(r),r,Plan)
(11)        Put (SUCCESSOR(r),TNOW+DURATION(r)) on the EVENT-LIST
            End
        End
(12) Call CONFLICT-DETECTION(p^i)
(13) If applicable Then
        Begin  /* p^i is applicable */
(14)    TNOW := T
(15)    p* := p^i
(16)    Plan := SEQUENCE(w,p*,Plan)
(17)    Put (SUCCESSOR(p*),TNOW+DURATION(p*)) on the EVENT-LIST
        End; Else  /* p^i is blocked */
(18)       Put p^i on a queue(RESOURCE(p^i));
        End  /* Repeat */
End  /* PLAN-AHEAD */
```

In the PLAN-AHEAD procedure, several built-in functions are used by the simulator and they are explained below:

PREDECESSOR(p^i): the operator which precedes p^i in subplan i

SUCCESSOR(p^i): the operator which follows p^i in subplan i

SEQUENCE(w,r,Plan): Plan is a partially ordered network of operators. w precedes r in Plan when r is added to Plan; if r is already in Plan, only the predence constraint is added.

QUEUE(S): a FIFO list which contains operators waiting for the resource S to be available.

DURATION(P): the duration specified in operator P.

RESOURCE(P): the resource used by operator P.

The flow-chart of PLAN-AHEAD is shown in Figure A.3 in the Appendix.

To illustrate how PLAN-AHEAD works, we shall now use the procedure to solve the example problem.

As shown in Figures 3.3 and 3.4, the linearly-sequenced subplans for parts 1 and 2 are the following.

Part 1: → p1 → p2 → p3 → p4 → p5 → p6 → p7 → p8 → p9 → p10

Part 2: → q1 → q2 → q3 → q4 → q5 → q6 → q7 → q8

Now, we want to use the plan generator to decide the necessary precedence ordering between the actions in the two parallel plans. The objective is to avoid any potential conflicts in machine accesses. Also, the total duration should be minimized.

The planning cycles generated by PLAN-AHEAD are illustrated in Table 3.11; the global clock, the contents of EVENT-LIST (E-L), the queues for resources, and the ordering of the plan constructed up to that moment is shown in each cycle to verify the correctness of the plan generator.

The plan-generation process can also be represented by a bar chart, as depicted in Figure 3.5. The plan is placed on a one-dimensional chart with time as the only dimension. The duration of each operator is explicitly represented. The advantage of using this chart is that, by examining it, the concurrency or the lack of it among the subplans can be easily observed.

There is one more thing about the plan generator that needs to be pointed out. In step(3) of the procedure PLAN-AHEAD, the first

Table 3.11 Planning Cycles Generated by PLAN-AHEAD

1. TNOW = 0 E-L : (p1 , 0) , (q1 , 0)

   Plan : Nil

2. TNOW = 0 E-L : (q1 , 0) , (p2 , 0)

   Plan : p1

3. TNOW = 0 E-L : (p2 , 0) ; queue(DOCK) : q1

   Plan : p1

4. TNOW = 0 E-L : (p3 , 1) ; queue(DOCK) : q1

   Plan : p1 → p2

5. TNOW = 1 E-L : (p4 , 3) , (q1 , 3)

   Plan : p1 → p2 → p3

6. TNOW = 3 E-L : (q2 , 3) , (p5 , 6)

   Plan : p1 → p2 → p3 → p4
   ↓
   → q1

7. TNOW = 3 E-L : (q3 , 4) , (p5 , 6)

   Plan : p1 → p2 → p3 → p4
   ↓
   → q1 → q2

8. TNOW = 4 E-L : (p5 , 6)

   Plan : p1 → p2 → p3 → p4
   ↓
   → q1 → q2

   queue(M1) : q3

Table 3.11, continued.

9. TNOW = 6 E-L : (p6 , 8)

Plan : p1 → p2 → p3 → p4 → p5
↓
→ q1 → q2

queue(M1) : q3

10. TNOW = 8 E-L : (q4 , 10) , (p7 , 14)

Plan : p1 → p2 → p3 → p4 → p5 → p6
↓ ↓
→ q1 → q2 → q3 →

11. TNOW = 10 E-L : (q5 , 13) , (p7 , 14)

Plan : p1 → p2 → p3 → p4 → p5 → p6
↓ ↓
→ q1 → q2 → q3 → q4

12. TNOW = 13 E-L : (p7 , 14) , (q6 , 15)

Plan : p1 → p2 → p3 → p4 → p5 → p6
↓ ↓
→ q1 → q2 → q3 → q4 → q5

13. TNOW = 14 E-L : (q6 , 15)

Plan : p1 → p2 → p3 → p4 → p5 → p6
↓ ↓
→ q1 → q2 → q3 → q4 → q5

queue(M3) : p7

14. TNOW = 15 E-L : (q7 , 19)

Plan : p1 → p2 → p3 → p4 → p5 → p6
↓ ↓
→ q1 → q2 → q3 → q4 → q5 → q6

queue(M3) : p7

Table 3.11, continued.

15. TNOW = 19 E-L : (q8 , 22)

Plan : p1 → p2 → p3 → p4 → p5 → p6
→ q1 → q2 → q3 → q4 → q5 → q6 → q7

queue(M3) : p7

16. TNOW = 22 E-L : (p8 , 24)

Plan : p1 → p2 → p3 → p4 → p5 → p6 ----------→ p7
→ q1 → q2 → q3 → q4 → q5 → q6 → q7 → q8

17. TNOW = 24 E-L : (p9 , 28)

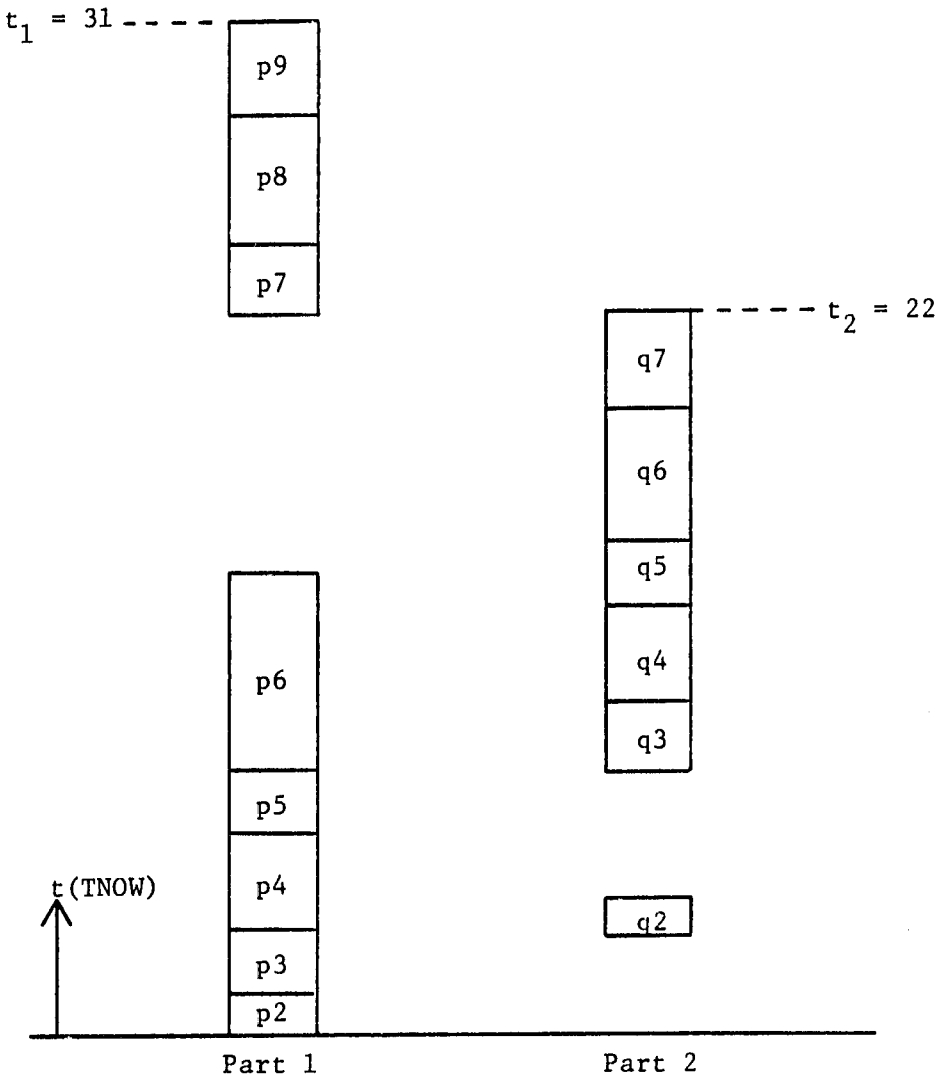
Plan : p1 → p2 → p3 → p4 → p5 → p6 ----------→ p7 → p8
→ q1 → q2 → q3 → q4 → q5 → q6 → q7 → q8

18. TNOW = 28 E-L : (p10 , 31)

Plan : p1 → p2 → p3 → p4 → p5 → p6 ----------→ p7 → p8 → p9
→ q1 → q2 → q3 → q4 → q5 → q6 → q7 → q8

19. TNOW = 31

Plan : p1 → p2 → p3 → p4 → p5 → p6 ----------→ p7 → p8 → p9 → p10
→ q1 → q2 → q3 → q4 → q5 → q6 → q7 → q8



Total Duration = 31

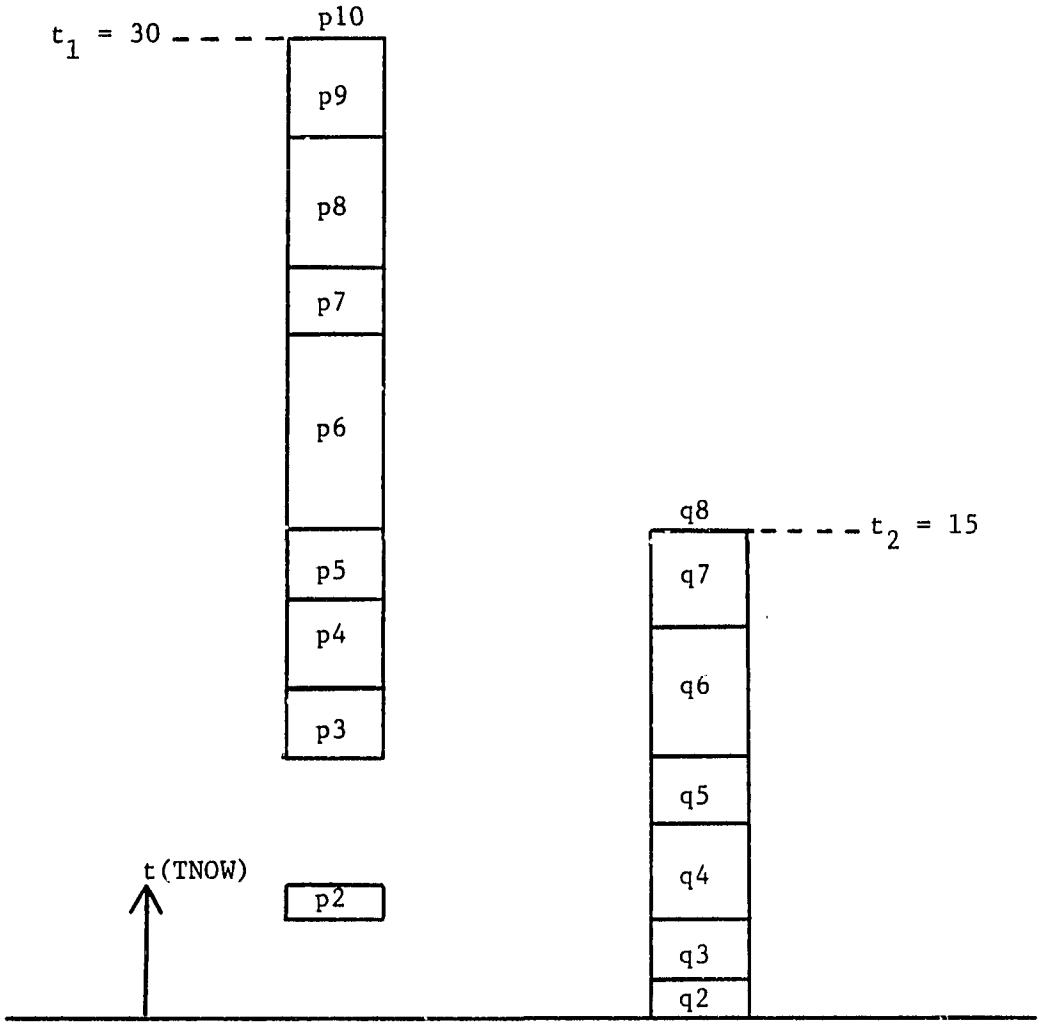
Average Duration = 26.5

Figure 3.6 The Schedule which Starts with Part 1

operator to be applied is chosen arbitrarily among the subplans. One possible heuristic for this step could be to begin with the plan with the longest total duration, since it is likely to be the one that affects the total duration. Well, this heuristic proves to be questionable. In the present example, the longest subplan is the subplan for Part 1, which takes 23 time units as opposed to 15 time units taken by Part 2. However, as shown in Figures 3.6 and 3.7, the plan beginning with part one turns out to perform worse than the plan beginning with part two, primarily because there are more machine access conflicts. This nondeterminism in plan generation causes suboptimism in some cases. PLAN-AHEAD can be used to test various alternatives in deciding the best plans. The functional diagram of the planning system is shown in Figure 3.8.

3.4.3 Plan Revisions

During the first phase of the proposed planning method, a linearly-sequenced subplan is generated for each subgoal. Because these subplans are constructed independently, each of them seeks to include its preferred actions and resources in the subplan without considering the requirements of other subplans - it is a greedy approach. However, a fundamental problem of this approach arises: if several subplans all request a highly capable resource which does everything well, the savings in processing time may not be worth the waiting time as a result of the long queue. In other words, in some situations, a subplan may be better off if it uses an alternative resource instead of waiting for the originally more preferred one.



Total Duration = 30

Average Duration = 22.5

Figure 3.7 The Schedule which Starts with Part 2

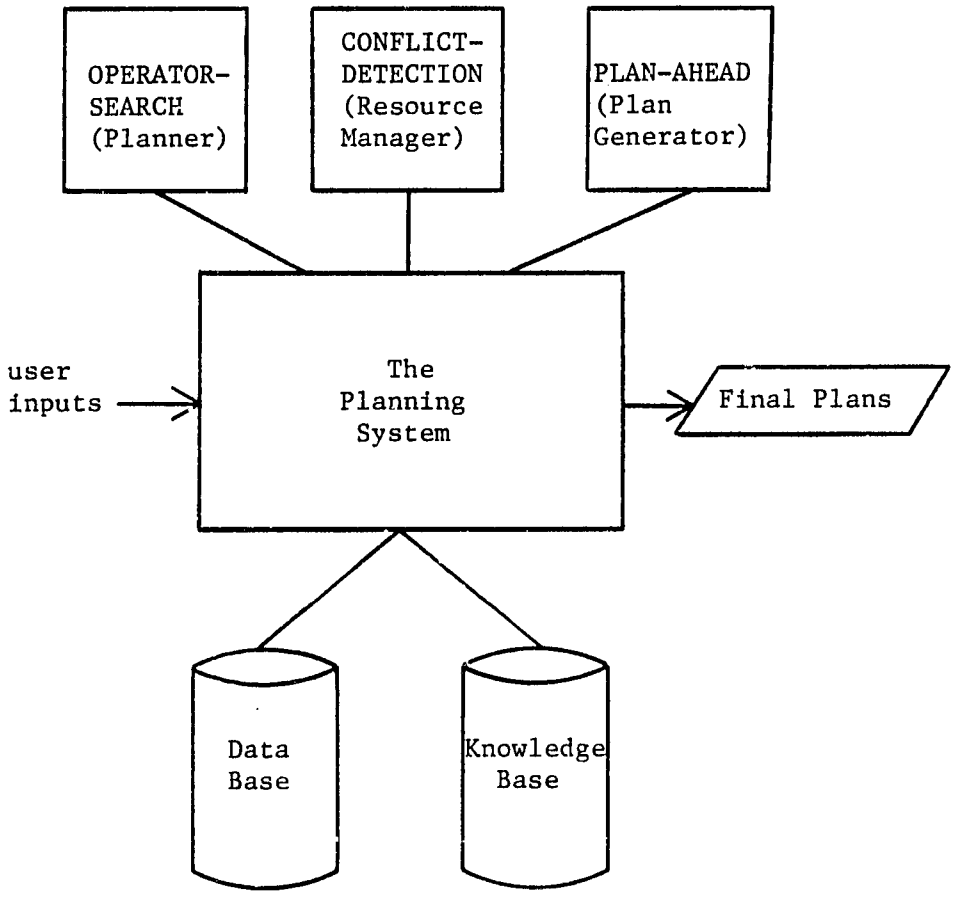


Figure 3.8 The Planning System

The same problem also occurs in planning systems like DCOMP, whose primary objective is "to construct a noninteractive partial ordering plan" (Nilson [1980]). There is no attempt to optimize any objective function. However, to perform scheduling, it is always important to minimize the duration of the schedule.

This section proposes a method that reassigns waiting jobs to an alternative resource so as to achieve better utilization and performance. The strategy can be described as follows:

The Plan Revision Scheme

step 1. Identify the resource for which this job is waiting.

step 2. Locate the section of the subplan that would use this resource.

step 3. Evaluate the expected waiting time vs. the additional processing time by an alternative, idle machine.

step 4. Find out the initial conditions and the ending conditions of this section.

step 5. Generate a plan that can transform the initial conditions to the goal conditions, using another idle resource.

step 6. Modify the subplan by replacing the section identified in step 2 with the newly generated plan from step 4.

As an example, according to the plan shown in Figure 3.5, the subplan for Part 2 waits for four time units for the machine M1. By using the plan-revision scheme, the planning system may find an alternative resource for Part 1 and thus eliminate the wasted waiting time. We shall follow the scheme step by step as follows:

(step 1.) the wanted resource is M1;

(step 2.) the section of subplan requesting M1 consists of q3 and q4;

(step 3.) the expected waiting time for M1 is four time units, while additional processing time by using m3 instead is five time units (Table 3.5); since the expected waiting time is less than the additional processing time of an alternative machine, the plan remains the same.

In the same plan, the subplan for Part 1 waits, at p6, for eight time units for machine M3. The same plan-revision scheme is used to determine whether the plan can be improved. This time an alternative machine is taken and the total duration is reduced by the scheme. This can be shown by the following steps:

(step 1.) the wanted recource is M3;

(step 2.) the section of subplan requesting M3 consists of p7, p8, and p9;

(step 3.) the expected waiting time for Part 1 is eight time units, while the additional processing time by using an idle, alternative machine - identified as M1 - is four time units. Thus the subplan should be modified such that M3 is replaced by M1 for planning steps in the section;

(step 4.) the initial and the ending conditions are shown in Figure 3.9;

(step 5.) a plan is generated that can transform the initial conditions to the goal conditions; this is shown in Figure 3.10;

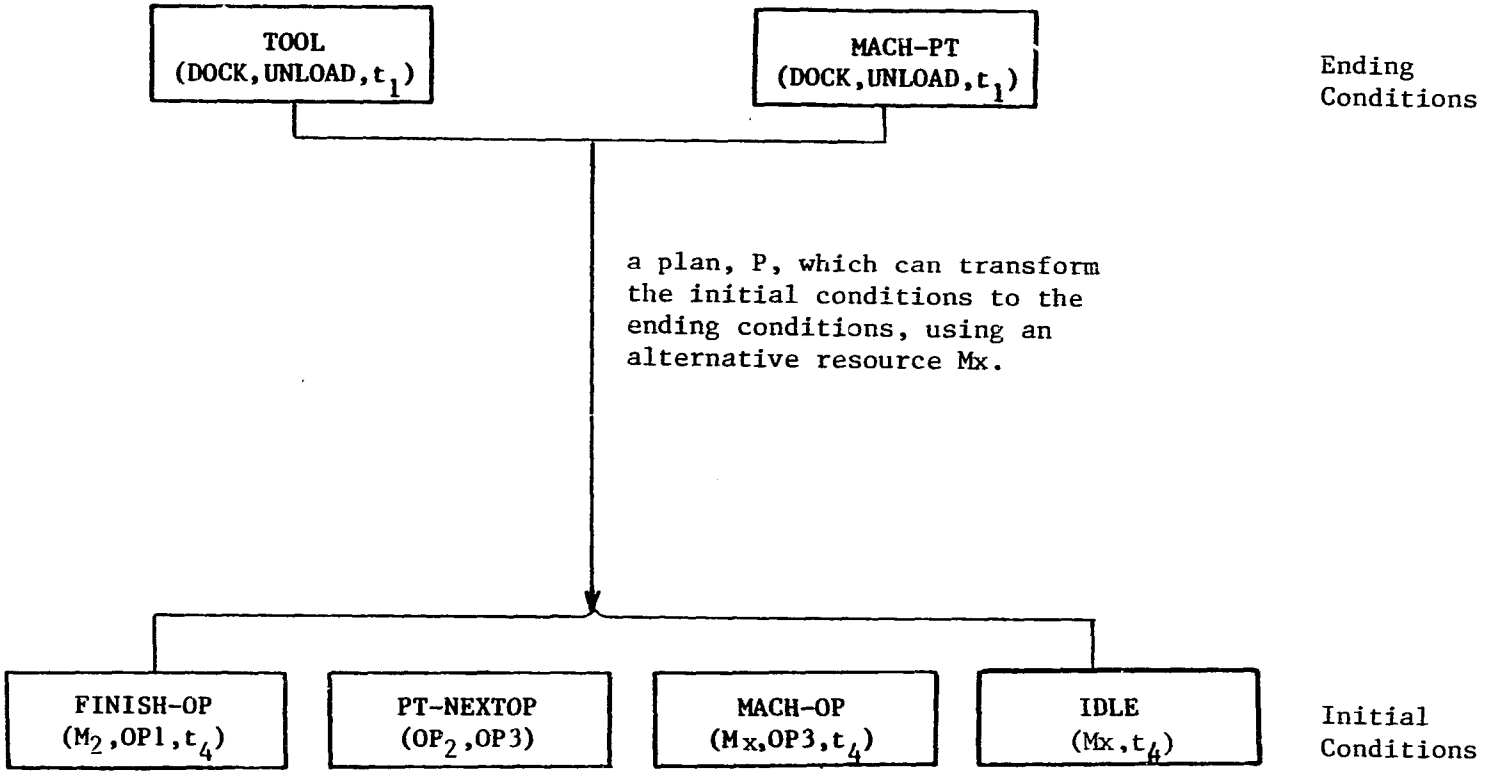


Figure 3.9 The Initial Conditions and the Ending Conditions for the Plan-Revision Steps

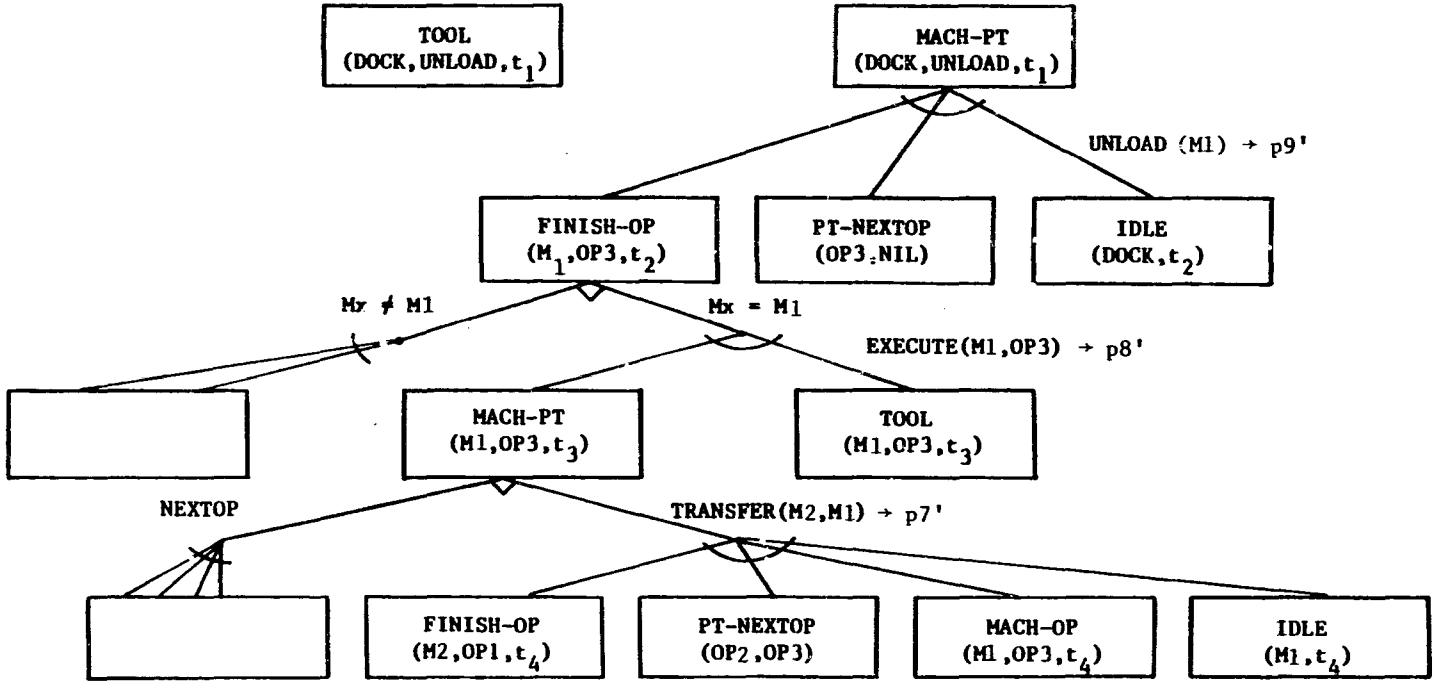


Figure 3.10 The Search Tree for a Plan Using an Alternative Resource

(step 6.) the revised plan is constructed by replacing p7, p8, and p9 with the new operators derived by step 5 above (Figure 3.11); the total duration of the schedule is improved by five units (Figure 3.12).

3.4.4 Characteristics of the Scheduling System

It has been shown that the planning system can function as a scheduler in a distributed system and efficiently assign sharing resources. We shall now characterize such a scheduling system based on the planning approach and compare it with the more conventional approaches. The scheduling problem in the flexible manufacturing cell we have been dealing with has the following characteristics:

(1) Jobs consist of linearly ordered operation sequences.

(2) A given operation can be performed on several alternative machines with different processing durations.

(3) Each machine, while capable of performing a variety of operations, can process only one operation at a time.

This scheduling problem can be formulated as an integer programming problem which captures all the characteristics listed above. Formulating the scheduling problem mathematically enables us to formally describe the system under study and to contrast our problem with similar problems treated in the scheduling literature. The integer programming problem is described as follows.

Decision Variables:

X_{ijk} : the completion time of operation j of job i on machine k.

Y_{ijpq} = 1 if operation q of job p precedes operation i of job j;

= 0 otherwise.

p1 ENTER

p2 EXECUTE(LOAD , DOCK)

p3 TRANSFER(DOCK , M1)

p4 EXECUTE(M1 , OP1)

p5 TRANSFER(M1 , M2)

p6 EXECUTE(M2 , OP2)

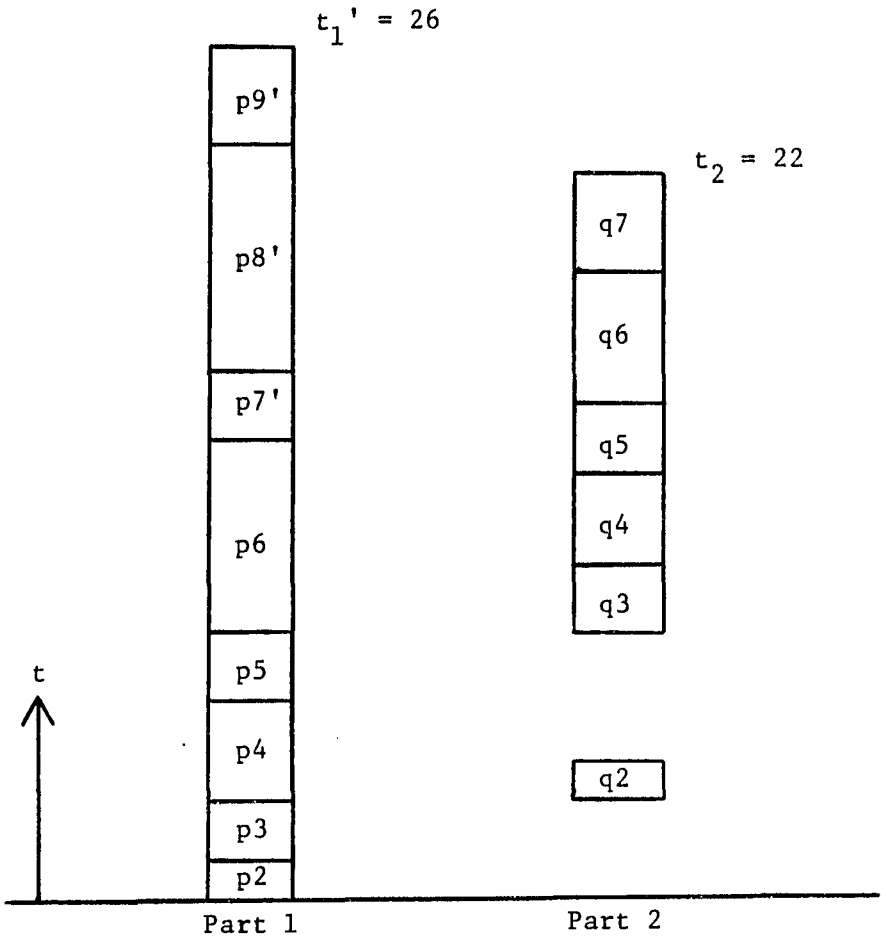
p7' TRANSFER(M2 , M1)

p8' EXECUTE(M1 , OP3)

p9' UNLOAD(M1 , DOCK)

p10 EXIT

Figure 3.11 The Revised Plan for Part 1



Total Duration = 26

Average Duration = 24

Figure 3.12 The Schedule after Plan Revision

Z_{ijpq} = 1 if operation j of job i precedes operation q of job p;

= 0 otherwise.

Constants:

t_{ijk} : the processing duration of operation j of job i on machine k.

M : a very large positive number.

$\ell(i)$: the last operation of job i.

(A) Minimize $\sum_{i=1}^{n} X_{il(i)k}$

Subject to

(B) $X_{ijk} - X_{i,j-1,k} \geq t_{ijk}$

(C) $X_{pqk} - X_{ijk} + M \cdot Y_{ijpq} \geq t_{pqk}$

(D) $X_{ijk} - X_{pqk} + M \cdot Z_{ijpq} \geq t_{ijk}$

(E) $Y_{ijpq} + Z_{ijpq} \geq 1$

(F) $X_{ijk} \geq 0$; Y_{ijpq}, Z_{ijpq} = 0 or 1

$1 \leq i, p \leq N$; $1 \leq j, q \leq n$; $1 \leq k \leq M$

The objective function described in (A) is to minimize the average duration needed for each job to complete. An alternative objective is to minimize the total duration of the schedule, i.e., the time when all the jobs are completed. This objective function can be described as:

$$\underset{i}{\text{Minimize}} \{X_{i,\ell(i),k}\}.$$

The constraint set (B) represents the linear ordering of operations in a job. These constraints also impose the condition that an operation may not begin until its predecessors are completed.

Constraints (C), (D), and (E) are used to regulate the mutually exclusive condition of machine sharing. For a pair of manufacturing

operations consisting of operation j of job i and operation q of job p, one of the following three situations may occur:

(i) they are performed on the same machine; operation j of job i precedes operation q of job p.

(ii) they are performed on the same machine; operation q of job p precedes operation j of job i.

(iii) they are performed on different machines. In this case, there is no constraint on their precedence ordering.

In the formulation, condition (i) is represented by $Z_{ijpq} = 1$ and $Y_{ijpq} = 0$; similarly, condition (ii) is represented by $Z_{ijpq} = 0$ and $Y_{ijpq} = 1$; lastly, condition (iii) is represented by $Z_{ijpq} = Y_{ijpq} = 1$.

The scheduling problem modeled by the formulation (A) to (F) is more complicated than conventional scheduling problems discussed in the literature. When having to solve similar n-part-m-machine scheduling problems, most existing job shop scheduling methods (e.g., Baker [1974], p. 207) make the assumption that each job has the same number of operations, one on each machine. However, as described by (A) to (F), scheduling in the flexible manufacturing cell has the characteristics that a job may have any number of operations to be done and that a job may be performed on a machine more than once while skipping less preferred machines. The additional complexity of scheduling can be attributed to the versatilities of machines in such an environment.

Another characteristic of scheduling in flexible manufacturing cells is that jobs arrive at the system randomly over time and the scheduling method should capture this dynamic characteristic in

constructing schedules. Using traditional methods, scheduling in such a dynamic environment is usually carried out by means of myopic dispatching rules - decision rules for assigning jobs when a machine becomes idle. These rules are myopic in the sense that a job will be selected from the set of jobs currently waiting for the machine, without considering future jobs. Most dispatching rules assign jobs based on information local to the machine. For example, the rule can be based on the length of the processing time (the "shortest processing time" rule) or the amount of total processing remained to be done (the "least work remaining" rule). Although simple to calculate, these rules result in suboptimal schedules because they are myopic and local. Simulation studies are widely used to determine the performance of different rules in various environments (e.g., Moore [1967]).

By contrast, the proposed planning system can perform two types of scheduling: static and dynamic. The static scheduling problem is to construct a schedule for a given number of parts (this is the case for the problem solved in the preceding sections). The dynamic version of the scheduling problem, where jobs arrive randomly over time, requires the modification of the current schedule for those jobs already in the system while taking into account the operations required by the newly arrived jobs. In this context, our method is extremely flexible in accommodating new jobs and dynamically changing environment. First, the actions required to complete the new jobs are derived by the planner. These actions are then scheduled on the EVENT-LIST in a linearly-sequenced manner, intermingled with the remained actions scheduled for other jobs. The resulting schedules,

which assign jobs dynamically, perform better than those using myopic dispatching rules because considerably more global information has been taken into consideration.

Prior scheduling methods for concurrent processes (e.g., Baker [1974], and Brinch Hansen [1973]) have been focused on the problems in which the task system - which specifies the tasks and their precedence ordering - is known in advance. By contrast, the scheduling procedure performed by the planning system described in this paper is "goal-directed." Only the goal of the scheduling problem (e.g., the completion of manufacturing jobs) needs to be specified. The sequences of actions that can achieve the goal are selected by the planning system. The partial ordering of the actions are also established to correctly allocate resources, while minimizing the total duration and maintaining good utilization of machines.

Using the EVENT-LIST, the planning system possesses the "event-driven" capability. Events that are beyond the control of the planning system can be included in the plan. Examples of these events, in the manufacturing domain, include: the regular maintenance time, the closing and opening of the system between shifts, and tool changes of particular machines. These events can be scheduled on the EVENT-LIST together with the other planned actions. The other actions in the plan will then accommodate the effects of these events automatically by the plan generator.

EVENT-LIST can also be implemented as a priority queue - that is, the elements in the queue are ordered by their priority - so that the jobs can be prioritized. At each moment of time, each planned action

has a priority number. Whenever a resource becomes idle, PLAN-AHEAD will assign the resource to the action with the highest priority. An application of this priority scheduling is in the environments where meeting the due-date is a critical factor in deciding the final plan. After a preliminary plan - which is a partially ordered network of actions - is determined for the parts involved, the "latest starting time" for each action can be derived backwardly from the due-date information. The actual starting time of an action is compared with this calculated latest starting time. The priority number assigned to the job reflects how tight it is for the job to meet its due-date. In general, the more urgent is a job, the higher is the priority assigned to the job, so that it can be completed as early as possible. The machine will only start the operation of a low priority job if no jobs of higher priority are present. Under the non-preemptive assumption, once the machine is assigned a job, it is committed to serve that job to completion even if jobs of higher priority arrive during its service.

In the multiprogramming environment, where jobs can be swapped back and forth between processors and memory devices, the preemptive scheduling policy is widely used. The job in service in a preemptive system is interrupted and returned to the queue upon the arrival of a job with higher priority. The reason for using preemptive scheduling is the resulting fast response to urgent jobs. In the computer aided manufacturing environment, however, it is extremely difficult to interrupt an operation of a part, put the part into the queue, and resume the operation later when no parts of higher priority are

present. The physical irregularities of the parts and of the fixtures (which hold the parts to the machine) make manufacturing operation uninterruptable for the sake of precision. Thus, scheduling systems in the manufacturing environment are primarily non-preemptive.

3.5 Distributed Planning

3.5.1 Planning by Multiple Agents

Multi-agent planning is concerned with the situation where a group of agents cooperate to achieve common goals. There are two approaches in handling multi-agent planning, the centralized and the decentralized approaches, which are classified in terms of the planning responsibilities of the agents.

The centralized approach is planning for multiple agents; a single planning agent generates a plan to be carried out by a group of agents. In general, the centralized multiple-agent planning problem can be viewed as a special case of nonlinear planning problems, with each subplan carried out by a different agent. The sequencing and scheduling problem in Section 3.4 is thus a centralized, multi-agent planning problem; the partially-ordered plan is centrally produced, containing a subplan for each manufacturing process of a part. As an extension, the plans may be carried out by two different agents, each of which moves a part between machines according to the plan generated for that particular part.

Another example of the centralized approach occurs in the operating system domain: the problem of coordinating intelligent agents at different sites of the ARPANET to achieve certain networking

tasks specified by users. For instance, a user may tell the agent at CMU that he wants a file at Stanford to be transferred to MIT, stored on a disk, and printed there. The CMU agent will construct a complete plan to accomplish this task, figure out the courses of actions for all three agents, and then inform Stanford and MIT what to do.

In contrast, multi-agent planning can also be accomplished in a decentralized manner. This is referred to as distributed planning. The goals of the problem are decomposed into subgoals and distributed among the agents. Each agent will then construct a plan for the subgoal it is assigned and execute its share of the final, synthesized plan. For example, the mobile robots in a computer aided manufacturing system can play the role of planning agents; each carries a part around the system to complete necessary operations. According to the manufacturing requirements of the part it carries, the robot's control system generates a plan for the part and then executes the plan by carrying the part to the selected machines. The important issue here is: how can a robot make sure that its plan does not conflict with that of the other robots, while completing the required operations efficiently.

The identification of conflicts and subsequent resolution of conflicts needs additional consideration in distributed planning systems. Since the control is decentralized, the potential harmful interactions to a subplan caused by other agents must be detected purely based on local information. From this standpoint, two kinds of activities become important in distributed planning: communication and synchronization. Communication activities are used to exchange

the necessary information between the agents. Synchronization activities are used to establish desired interactions among agents' plans so that their actions are well-coordinated.

3.5.2 Communication and Synchronization

After each agent has constructed a (sub)plan for the subgoals it is assigned, the potential conflicts among all the subplans need to be recognized and then avoided. The communication and synchronization capabilities are essential for an agent to direct its activities in concert with that of the other agents, resolving any conflict that may occur.

When all the subplans are generated by a single agent, conflicts can be detected by reasoning about resources - actions of different subplans that use the same resource are potentially conflicting. A sequential ordering constraint is imposed between every pair of conflicting actions to ensure the correctness of their accessing the resources. This is the method used in the preceding section.

Since different subplans are under the control of different agents in distributed planning, the detection and the avoidance of conflicts become more complicated in the decentralized environment. Unless the agents exchange their information about their subplans, an agent could not possibly know whether its planned actions are conflicting with those of other agents, nor how to resolve the conflicts.

Thus, concurrently executed subplans among the agents must communicate and synchronize in order to cooperate. Cooperation is achieved through the effort to resolve all the potential conflicts. The idea is to intermix communication and synchronization activities

with other actions in the plan, so that one agent may exercise influence upon the plan of other agents. The issue of how to properly integrate the communication and synchronization activities into the plans in order to coordinate the actions of different agents will be subsequently addressed.

Coordinating multi-agent planning is conceptually similar to managing the concurrent processes in a multi-processor operating system, where numerous processes contend for accesses to the processors and other system resources. The job of the operating system is to provide mutual exclusion and synchronization among the processes, so that none of them will interfere with one another.

Based on the techniques of concurrent processing, communication between agents can be achieved by one or two methods: by shared variables or by message passing. The semaphors used for resource reasoning in Section 3.4 are an example of the shared-variable approach. This approach is used primarily for centralized multi-agent planning, where the subplans share a database during their development. Semaphors (which are basically shared variables) are used to indicate the status of resources; critical regions are identified in the sub-plans to ensure orderly access to shared resources. However, the shared-variable approach is not appropriate for distributed planning where the group of agents are loosely-coupled and do not share any common database. In such an environment, message passing is more appropriate.

When the message-passing approach is used for communication and synchronization, the processes send and receive messages instead of

reading and writing shared variables. Message passing is a form of communication between the sender and the receiver of the message. Synchronization can be accomplished by such communication operations because the action of sending a message must take place before the action of receiving the same message; one action is delayed until the other is ready for the communication. Thus, a precedence ordering between the sender and the receiver is enforced by the operation of message passing. This property of message passing can be used to maintain mutual exclusion in critical sections of agents' plans.

The techniques of concurrency management developed for multiprocessor operating systems can be incorporated into distributed planning in the following manner.

(1) Since the major cause of conflicts between planned actions of different agents is the accessing of shared resources, the consecutive actions in an agent's plan that use the same resource should be in a critical section.

(2) If several agents include the use of the same resource in their plans, their actions must be coordinated to avoid any potential conflicts. This coordination is accomplished through the proper placement of synchronization activities in each agent's plan. These synchronization activities will then ensure the mutual exclusion between critical sections of different agent's plan. This is equivalent to enforcing that every resource should be exclusively used by one agent at any given time. (3) and (4) below show how to achieve this synchronization in a decentralized fashion.

(3) Synchronization is achieved by message passing between agents. When an agent reaches a communication operation in its plan, the agent waits for the corresponding agent (designed as the receiver or the sender) to reach the matching communication operation. At that point, the communication is performed as directed by the input/output commands and both agents resume their plan executions for subsequent actions.

(4) Potential conflicts are avoided by enforcing the mutual exclusion of critical regions. When an agent is ready to enter a critical region, it sends a signal to a "coordinator" to ask for permission. The coordinator checks on the status of the resource and grants the permission if no other critical region is using the same resource. Otherwise, the agent is suspended until the resource is free again.

Since the formalism we shall use to denote message-passing activities in distributed planning is based on the Communication of Sequential Processes (CSP) formalism defined by Hoare [1978], this synchronization mechanism is briefly reviewed in the following section.

3.5.3 Synchronization Based on Message Passing

CSP is a formalism for concurrent processing in distributed environments. A program in this formalism is a collection of sequential processes, each of which can include interprocess communication operations. Communication is achieved by the use of input/output commands in processes.

An output command in CSP has the form

receiver ! value

where receiver is the name of the process that is the designated receiver and value contains the information intended to be transmitted. The counterpart of the output command, the input command, is in the form

sender ? x

where sender is the name of the process which is expected to send the message. When the message is received, the value in the message is copied into x.

Communication is invoked when one process names another as the receiver for the message and the second process names the first as the sender for the message. In this case, the value is copied from the first process to the second process. In the CSP formalism, message passing is synchronous because an input or output command is delayed until the other process is ready with the corresponding output or input. Thus, the insertion of input and output commands in processes can be viewed as a way to impose a precedence ordering constraint between a pair of operations from different processes.

Hoare also adopts the guarded commands as a means of introducing the pattern-matching feature in the CSP formalism. When the input commands are coupled with guarded commands in the receiver, input messages that do not match the patterns of the guarded commands are inhibited.

A pattern-matching statement using the guarded command is in the following form:

[if $G_1 \rightarrow S_1$
□ $G_2 \rightarrow S_2$
...
□ $G_n \rightarrow S_n$
fi]

where G_i is guard containing a Boolean expression and S_i is the corresponding statements that are executed when G_i is chosen. This pattern-matching statement works as follows. A guard G_i is chosen for execution if its Boolean expression is true and the sender named in G_i is ready to execute the corresponding output command. If several guards can be chosen, only one is selected nondeterministically. If none of the guards can be chosen, the statement aborts. The guarded commands provide a receiver with the "selective communication" capability - only those messages that match the guards will be received.

Using the CSP formalism in distributed planning, the coordination scheme is as follows. When an agent is about to enter a critical section in its plan, it sends a message to a coordinator to notify the entry. The coordinator checks on the status of the resource and grants the entry if no other critical section is using the same resource. Otherwise, the coordinator will delay the entry until the resource is free again.

The communication between the planning agents and the coordinator is accomplished by using the CSP formalism. Specifically, when an agent enters a critical section of its plan, it sends a message to the

coordinator indicating the resource. This can be accomplished by the output command:

M ! enter(i)

where i is the index of the resource. When the agent, who is currently in the critical section using resource i, exits the critical section and releases resource i, it signals the coordinator again by an output command

M ! exit(i) .

For the coordinator, it uses BUSY(i) to represent the availability of resource i (BUSY(i) = false when resource i is available). An agent may use resource i only if it signals to the coordinator about the attempt and resource i is available. This condition can be represented by a guarded command:

□ not BUSY(i); A1 ? enter(i) → BUSY(i) := true

where BUSY(i) is initialized to be false. This statement can be interpreted as follows. If BUSY(i) is false and the agent A1 sends a message "enter(i)", then A1 is granted the entry to resource i; and BUSY(i) becomes true.

3.5.4 The Application of Distributed Planning to a Machine Loading Problem

An example of the application of distributed planning in the computer integrated manufacturing environment, which is originally discussed in Georgeff [1982], is now presented using the resource reasoning and synchronization approaches. It is concerned with a machine loading problem with two mobile robots, each of which plays the role of a planning agent. Without the loss of generality, we only

consider a two-part-one-machine problem, where the major focus is the use of synchronization mechanisms to coordinate the planning of each agent.

Assume that each agent has first generated a plan using a STRIPS type linear planning method as follows:

robot1 (part1):

A1 MOVE(part1,MACH,robot1)

A2 LOAD(part1,MACH,robot1)

A3 EXECUTE(part1,OP1)

A4 UNLOAD(part1,MACH,robot1)

A5 MOVE(part1,warehouse,robot1)

robot2 (part2):

B1 MOVE(part2,MACH,robot2)

B2 LOAD(part2,MACH,robot2)

B3 EXECUTE(part2,OP2)

B4 UNLOAD(part2,MACH,robot2)

B5 MOVE(part2,warehouse,robot2)

The planning steps of robot 1 and robot 2 consist of similar actions. Both robots need to move a part to a machine, denoted by MACH, and load the part onto the machine. After executing a required operation - OP1 for part 1 and OP2 for part 2, the robots unload the parts and move them to the warehouse. Since only one robot can have access to the machine at a time, consecutive actions that use MACH as the resource form a critical section. Different critical sections of the same resource are kept mutually exclusive by means of synchronization operations.

To detect potential conflicts between the two agents, the first step is to identify the critical sections of each agent's plan. A critical section is a set of consecutive actions that require the same resource. Thus

[A2,A3,A4] and [B2,B3,B4]

are two critical sections. Furthermore, these two critical sections are mutually exclusive because they use the same resource.

This mutual exclusion is enforced by inserting CSP operations into the two plans. When an agent is ready to enter a critical region, it outputs a message to a resource manager to ask for permission. The resource manager, M, using guarded commands, will check on the availability of the resource and grant the permission if the resource is free. Otherwise, the agent is delayed until the resource is released by the current user.

```
robot1(R1):
MOVE(part1,MACH,robot1)          /* move to the machine */
M! enter(MACH)                   /* ask for permission */
LOAD(part1,MACH,robot1)          /* load the part */
EXECUTE(part1,OP1)               /* machining */
UNLOAD(part1,MACH,robot1)        /* unload the part */
M! exit(MACH)                    /* release the resource */
MOVE(part1,warehouse,robot1)     /* move to warehouse */
```

```
robot2(R2):
MOVE(part2,MACH,robot2)          /* move to the machine */
M! enter(MACH)                   /* ask for permission */
LOAD(part2,MACH,robot2)          /* load the part */
EXECUTE(part2,OP2)               /* machining */
UNLOAD(part2,MACH,robot2)        /* unload the part */
M! exit(MACH)                    /* release the resource */
MOVE(part2,warehouse,robot2)     /* move to warehouse */
```

The resource manager keeps track of the status of machines. Its knowledge is modeled by the pattern-matching guarded commands:

```
[if not BUSY2; R1 ? enter(MACH) → BUSY2 := true
□  not BUSY1; R2 ? enter(MACH) → BUSY1 := true
□     R1 ? exit(MACH) → BUSY2 := false
□     R2 ? exit(MACH) → BUSY1 := false
  fi]
```

The above statements can be interpreted as follows. A robot can enter its critical section if the other robot is not executing its critical section; when a robot finishes using the machine, the status of the machine becomes "not BUSY."

3.6 Conclusion

Mechanisms have been described for using a knowledge-based planning system to manage the computer integrated manufacturing system characterized as a distributed environment. Two kinds of information in the action formalism are emphasized: resource and duration. The planning system uses a "reasoning about resources" mechanism to maintain the correctness of machine usages when several manufacturing jobs

are competing for the machine. A plan-generator, called PLAN-AHEAD, determines the precedence ordering between conflicting actions by means of the duration information; the final plan - a partially ordered network - has maximized parallelism. A plan-revision mechanism is used to reassign a job to an alternative machine if the job is delayed in a machine queue. We have shown that the planning system can play the role of an on-line scheduler. The scheduler is characterized by being goal directed, its ability to cope with the dynamic environment, and being event driven. Operating system techniques have been used to provide appropriate synchronization and communication in coordinating concurrent manufacturing activities. Planning steps are grouped into critical sections according to the resource they need and thus maintain the mutual exclusion of shared resources.

CHAPTER 4
TASK SHARING AND PLANNING IN CELLULAR FLEXIBLE MANUFACTURING SYSTEMS

4.1 Introduction

4.1.1 Overview

This chapter attempts to incorporate a decentralized allocation mechanism in the information system for the computer integrated manufacturing environment. The cellular organization is used to embody a distributed control structure into the manufacturing system - machines are grouped into manufacturing cells which are the modular units in the system. A cell can communicate with other cells through a communication subsystem (e.g., a local area network). Because of the autonomous nature of the cells and the lack of a central control unit, this cellular system is characterized as a loosely-coupled, decentralized-control system. In each cell there is a knowledge-based planning system, similar to the one described in Chapter 3, that manages and controls the execution of manufacturing tasks within a cell. In this context, the information system in the cellular manufacturing environment is a distributed knowledge-based system.

Since an incoming job may consist of a set of tasks to be assigned to several manufacturing cells, the information system needs to supply a mechanism that permits careful matching of tasks to

available machines, ensuring appropriate uses of resources. Also, this meachnism should enable efficient communication between manufacturing cells working on related tasks, or should be able to migrate subtasks dynamically in case of machine overload. Accordingly, there are three primary issues that need to be addressed:

(1) the interface language that enables communication among cell hosts;

(2) the problem-solving process which, utilizing the communication network, enables effective cooperation among cells to perform jobs; and

(3) the automation of this problem-solving process in each cell in a decentralized manner.

A negotiation protocol based on the works in distributed problem solving (DPS) (e.g., Smith [1978], Davis and Smith [1983]) will be developed in this chapter to fulfill these requirements. It is a high-level protocol used to ensure orderly interactions between asynchronous, cooperating cells; it also prompts proper actions in a cell based on the messages received. The underlying idea is to structure the interactions between modules - the manufacturing cells - as a process of negotiation.

It is important to have a good representation of the contract net protocol to capture the dynamic, concurrent nature of the protocol. In addition, this representation should be integrated into executable programs, controlling the interactions between cells and coordinating the assignments of tasks to machines.

In this chapter we use the augmented Petri nets - an integration of production rules and Petri nets - to model the negotiation protocol. The automation of this augmented Petri nets provides the basis for a task-sharing algorithm to dynamically assign tasks to appropriate cells. By implementing the algorithm in a knowledge-based system, we adopt a unified representation for the planning process and the cooperation process.

4.1.2 The Structure of a Cellular Flexible Manufacturing System

The introduction of Flexible Manufacturing Systems (FMS) into the manufacturing industry has been an important new step in the development of the fully automated manufacturing systems. An FMS is designed to manufacture a variety of products in small volume, while simultaneously possessing the efficiency of a transfer line and the flexibility of a job-shop (Groover [1980]). The focus of this chapter, however, is a distributed version of the FMS, as defined subsequently.

Cellular Flexible Manufacturing Systems (CFMS) consist of a collection of manufacturing cells, each of which is an independent module of a group of machines, robots and material handling devises (Figure 4.1). The idea is to let each manufacturing cell specialize in a specific family of part types and thus further improve efficiency from the reduced set-up changes between consecutive jobs (Cutkosky [1983]). Several cells may be assigned to a single job if that job comprises sub-components of different part families; a mechanism is therefore required to properly assign jobs and sub-components to various cells. On the other hand, since a manufacturing cell may

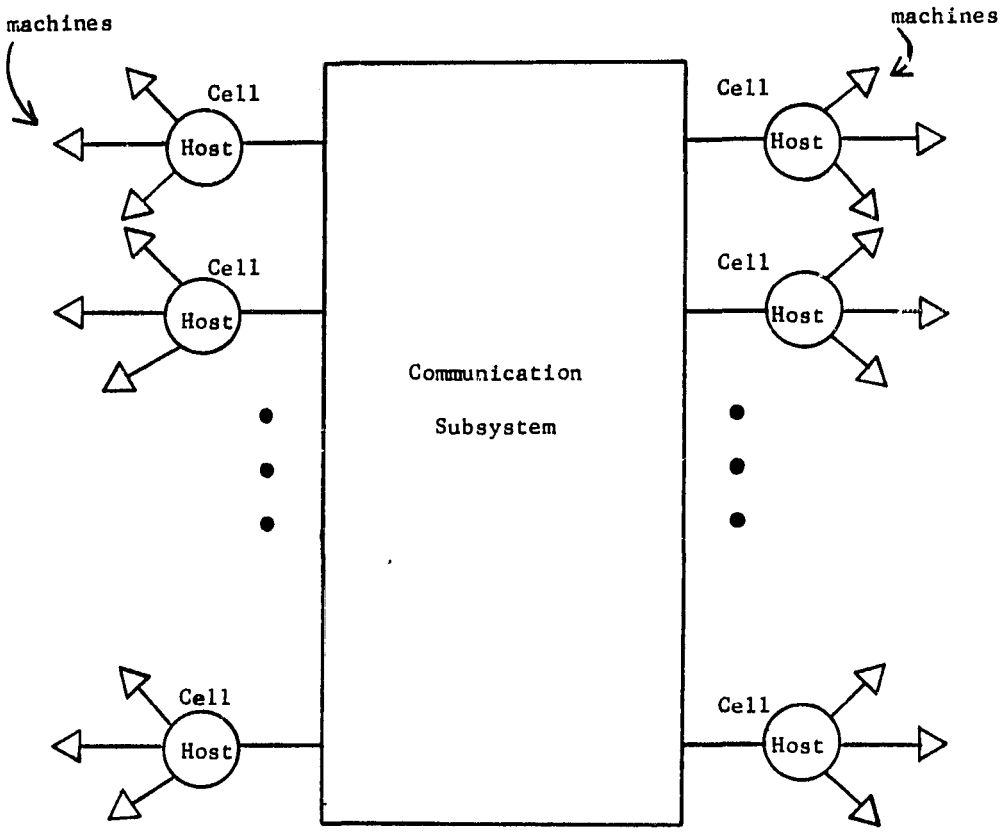


Figure 4.1 Cellular Flexible Manufacturing System

perform operations for different jobs in a multiprocessing fashion, the planning system within a cell also needs to resolve conflicts among these jobs. An example of the organization of a manufacturing cell is shown in Figure 4.2.

Thus there is a control hierarchy in the CFMS, as shown in Figure 4.3. Sensors are used to provide the system with information about the process environment such as part locations and robot-arm manipulator movements. The machine controllers act as interfaces between the cell host and the machining processes while controlling the individual machines. The cell host computer is responsible for the control of part flows and the scheduling of multiple tasks within a cell. The CFMS monitor collects management information from the cells; this information includes machine status, cell loading and in-process inventories, etc. Upon taking a new job, the cell host will request the corresponding automation programs and other relevant information on that job from the CFMS monitor computer.

Besides savings on the set-up time, using manufacturing cells to form a loosely-coupled CFMS with decentralized control has several additional advantages over the centralized FMS.

1. Modularity and Extensibility: new cells can be easily added into CFMS without having to modify the other components or the control structure.
2. Graceful Degradation and Reliability: faulty cells can simply be ignored and reroute the workpiece to other cells of the same product family.

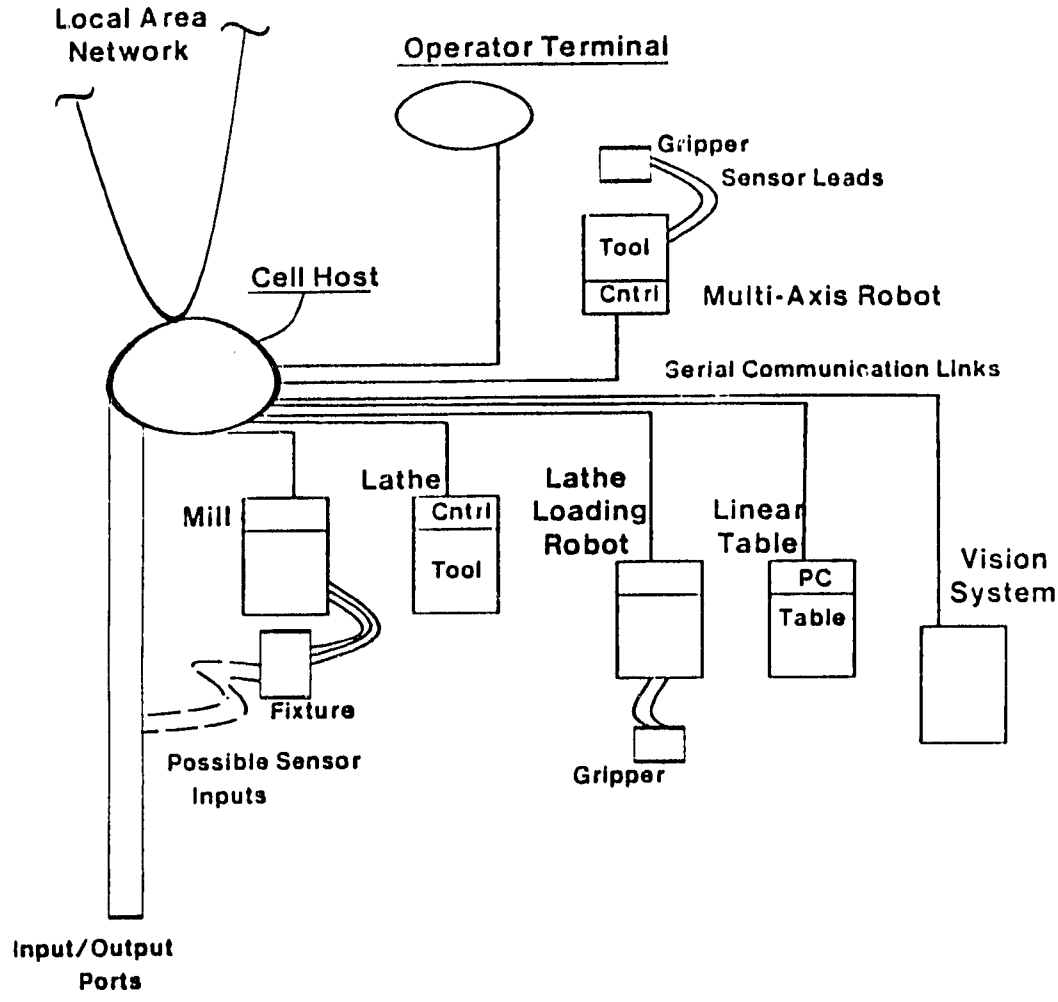


Figure 4.2 The Organization of a Manufacturing Cell (from Cutkosky, et. al. [1983])

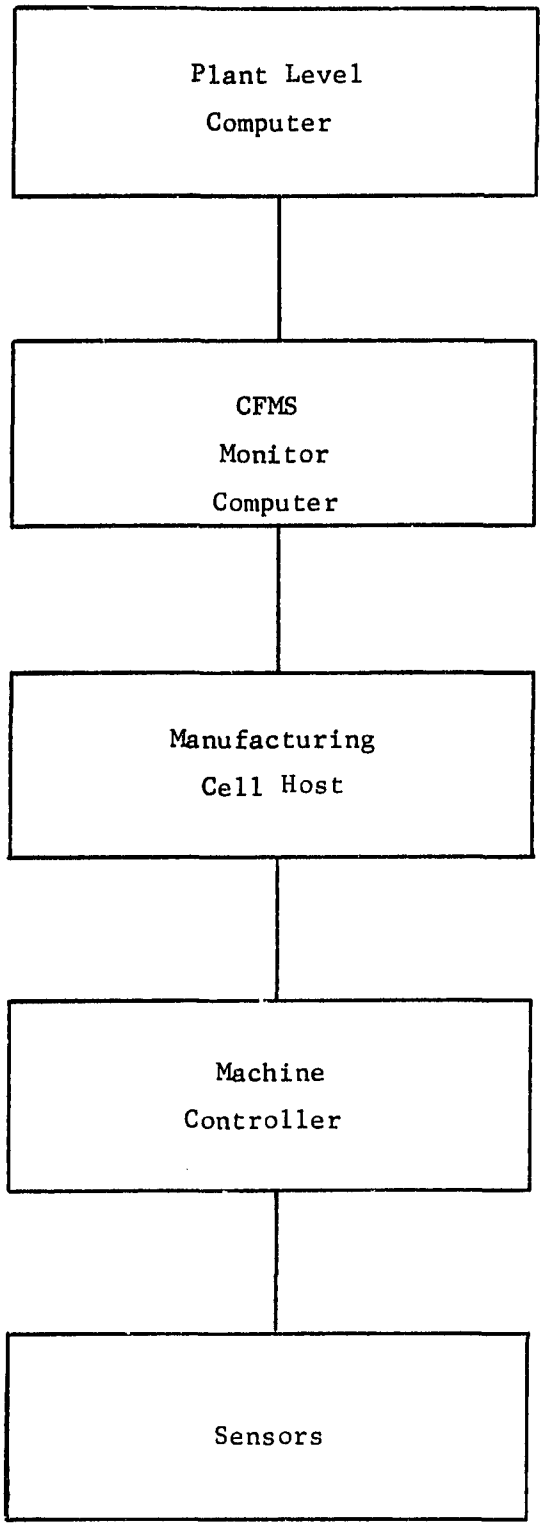


Figure 4.3 The Control Hierarchy

3. Signal Accuracy: distributing control permits the placement of control computers (cell hosts in CFMS) near the mechanical devices with which they interact, thus eliminate the weakening signal problem associated with centralized systems where the signals have to pass through long links.

Despite all the inherent advantages, using a distributed control structure does introduce additional overhead and complexities in the planning system. The major causes of these complexities are the extra coordinating activities required to maintain global coherence and to achieve optimality in task allocations. Therefore, good designs on the organization and coordination are required to justify the use of decentralized control.

The remainder of this chapter is organized as follows. The second section describes the characteristics of the distributed problem solving system in a CFMS environment and proposes a planning model to tackle the problem. The next section proposes the use of contract net protocol for coordinating the tasks; this protocol is then represented by an augmented Petri net model. The fourth section illustrates the implementation of the contract net protocol as well as the issue of algorithmic complexities, and the final section summarizes the contributions of our approach. Possible extensions are also discussed.

4.2 Cooperative Planning in the CFMS Environment

4.2.1 Reconfiguration of CFMS and the Task Sharing Problem

In a CFMS, when a job consists of sub-components of different part types, several manufacturing cells are assigned to execute the operations required by sub-components collectively. Thus the planning system needs to select the most appropriate cells to perform tasks for the sub-components. This is called the task-sharing problem. Task sharing also occurs when a cell is overloaded and wants to distribute some of the tasks to other cells. In the CFMS environment, task sharing serves as the basis for cooperation between cells; the result of an agreement on task sharing is the binding of manufacturing cells to perform the job in a coordinated fashion.

Since a CFMS by definition has multiple jobs which coexist in the system, whereas a cell may be shared by several jobs, a configuration of CFMS is defined as the assignment of cells to tasks. Whenever there is a change of jobs, the reconfiguration of the CFMS can be achieved by solving the task sharing problems for the new set of jobs. For a fixed CFMS configuration, a manufacturing cell acts like a work station in a transfer line: high efficiency is achieved by pipelining the workpieces through the fixed sequence of cells connected as a result of task sharing. This pipelining continues until the required batch size of the job is completed. The CFMS has the flexibility to accommodate different job combinations by reconfiguring the manufacturing cells. The reconfiguration of the CFMS is realized by distributing the new tasks among the cells through the task sharing procedure.

4.2.2 Modelling the Coordinating Cells: The Society of Experts Metaphor

A classical analogy to investigate the problem solving process in a distributed environment is that of teaming up a society of experts to solve a problem cooperatively (Minsky [1979], Kornfield and Hewitt [1981], Davis and Smith [1983]). Viewing the host computer of each cell as a problem solving agent (an "expert") in charge of the planning activities, the planning system of a CFMS then consists of a group of planning agents; the knowledge or expertise is distributed among the experts, while each expert has only a partial view on the whole problem domain. This society of experts metaphor can contribute to our designing a distributed problem solving system by providing a model to address the following issues:

1. The information about what the experts need to communicate with each other; this information can provide us with insight on the possible types of messages, and the proper contents the messages should carry when they are passed between manufacturing cells.
2. The way the experts cooperate in performing a task; this knowledge can help us in designing a similar algorithm to distribute a job among the manufacturing cells and to maintain proper interactions between cells if necessary.
3. The way a set of experts reach agreements, this information is of central importance to the configuration/reconfiguration of CFMS, since the grouping of manufacturing cells to perform a job can be interpreted as forming an agreement between these cells.

One implication resulting from the society of experts metaphor is that the processes in the cellular manufacturing system - i.e., task

sharing, resource allocation and information transactions - are analogous to those of human organizations. The related research in economics and control theories concerning team behavior and decentralized decision making may therefore serve as vehicles to our modeling the distributed system.

We have stated that the primary problem-solving activities in a manufacturing cell is the planning of tasks within the cell; however, the term "planning" was not rigorously defined. In the following section, a formalism for the planning process as well as a formalism for the multi-task planning process will be presented.

4.2.3 A DPS Formalism for the CFMS

In our modeling the control system for CFMS, cell hosts are problem solving units that interact with the manufacturing environment. They must be able to aggregate individual steps into sequences to achieve desired goals. This process is referred to as the planning process. A planning system has two components:

1. The world model, containing a symbolic description of the real world. This world model is represented by the collection of first-order predicates in a database. An instance in the database is called a state-description in the world model.
2. The action model, describing the transformational effects of actions that map from states to states. Such mappings are usually referred to as operators, as the STRIPS operators defined in Nilson [1980].

Operators specify the form of the individual steps which make up a plan. All possible ways of applying the operators, both to initial states and to intermediate states, determine the search space for the set of predicates in the database. An inference engine is the control unit of the planning system that directs the plan generation process to achieve a desired goal state from a given initial state. The sequence of actions that are generated is called a linear plan.

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 $S \rightarrow S$; IS is the initial state and G is 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, 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 state IS and to achieve the goal G. The coordination between planning agents may be accomplished through messages passed between agents. The key issues then are how to automate this coordination and how to regulate orderly interactions.

The configuration of a distributed problem solving system and the effects of interactions between the agents can be visualized better if we represent the system as a directed graph

$$G = (E, I)$$

The graph G defines the information structure of the distributed system. The problem solving activities in node i may impact on the problem solving activities in node j through the interactions I_{ij}. Every node in the graph represents a problem-sovling agent E_i.

A task T is decomposed into subtasks t_1, t_2, ..., t_n which are assigned to experts E_{e_1}, E_{e_2}, ..., E_{e_m} ($e_i \in R$ is the index of the corresponding expert). If the collection of subtasks assigned to expert e_i is denoted by T_{e_i}, then

$$\bigcup_{i \in [1,m]} T_{e_i} = T$$

and

$$T_{e_i} \bigcap_{i \neq j} T_{e_j} = \phi.$$

We are mainly concerned with problems which can be sufficiently decoupled and the effects of one agent are largely independent of other agents. This is the case in the CFMS environment where machining operations in different cells are mostly independent. The primary coordinating activities, then, are the assignments of subtasks to appropriate agents. The process of DPS can be algorithmically represented as follows:

Procedure DPS (T)

Input:

T: the task to be achieved.

E: the set of expert nodes.

I: the set of interactions.

Output:

P: a distributed plan to achieve goal T.

```
Begin
    T' ← -- DECOMPOSE(T)
         {T' is a partition of T}
    A ← -- DISTRIBUTE(T',E)
         {A is the set of pairs (e_i, T_e_i)}
    For all i Do
         Begin
         If (OVERLOAD(e_i)) Then DPS(T_e_i)
         P_i ← -- EXECUTE(e_i,T_e_i)
         End
         P ← result(e_i, P_i)
End
```

This DPS strategy is consistent with that proposed by Kieburtz [1979], Tenney and Sandell [1981], and Davis and Smith [1983]. (In Section 4.4.3, we shall see an implementation of this strategy based on the contract net approach.)

In the decomposition phase, the aim is to identify subtasks that can be carried out concurrently. It is important to analyze the dependency relation between the subtasks, so that necessary synchronization can be applied accordingly. For a manufacturing process, the primary dependency relation between subtasks is the precedence constraints between manufacturing operations.

The second phase, the distributing of subtasks among nodes, is the major concern of this chapter. We use a task-sharing strategy based on a "negotiation process" to schedule the subtasks. The details of this procedure will be discussed in the next section.

The execution phase is applied locally at each node; in a CFMS, it is handled by the cell hosts. The primary task of this phase is coordinating the multiple jobs in the cell and resolving conflicting requests.

Methods developed in the artificial intelligence field suitable for multitask planning, such as NOAH (Corkill [1979]), DCOMP (Nilson [1980]), SIPE (Wilkins [1982]) and Deviser (Vere [1983]) are possible tools in this phase. The planning activities of these methods can be broken down into four stages:

1. linear planning;
2. interaction analysis;
3. conflict resolution; and
4. plan synthesis.

In the manufacturing environment, additional consideration should be given to the possibility of collision between moving parts or robot arms. This problem has been solved by posting constraints in the planning process (Bourne, et. al. [1982], Lozano-Perez [1982]).

Finally, the completed subtasks are collected and synthesized into the finished job. In the manufacturing process, this phase can be included in the final task of assembling the components.

4.3 The Contract Net Strategy

4.3.1 Negotiation Procedure for the Dynamic Reconfiguration of CFMS

The view of treating cooperation as the task sharing problem is directly inherited from Davis [1983]. The coordinating mechanism primarily aims at assigning sub-tasks to the most appropriate expert who is both capable and willing to do the tasks. To design a similar mechanism to coordinate the planning activities of manufacturing cells in CFMS, we solve the task sharing problem by an algorithm analogous to the contract negotiation between cells. This procedure consists of announcement-bid-award sequences to distribute a task to appropriate cells; it is characterized as a mutual selection process: a manager cell with tasks to be distributed attempts to select the most suitable cell to handle the tasks, while a cell with idle machines is also selecting among the announced tasks and submitting bids on those tasks it prefers. A contract is established when a bidding cell is selected by the manager cell. The announcement-bid-award cycle is detailed as follows:

When a manufacturing cell has a task it is not capable of handling, the cell may decide to announce 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 to other announced tasks.

c) 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. When a machine becomes idle, the cell selects a task at the top of the list and submit a bid to the cell originating 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 appropriate cell based on some ranking criteria (e.g., distance between the two cells, the loading factor of the bidder, etc.). The successful bidders are informed of the award through award messages from the manager cells. Thus the contract negotiation is an interactive process where both parties have evaluated all the alternatives before they reach the final agreement to establish the contract.

4.3.2 The Design of Communication Protocol for the CFMS

To enable processes at various manufacturing cells to communicate with each other, a set of rules must be established to regulate the interactions between the cells and to ensure that they proceed in an orderly fashion. This set of rules is called protocol of the communication. The requirements of protocol for establishing cooperation between manufacturing cells are more complicated than merely deciding on the communication paths. The protocol also has to carry out the negotiation algorithm for task sharing, relying heavily on the

interpretation of the messages contents. Because of the problem-oriented nature, we call this protocol a problem-solving protocol. Thus, a formal model for the CFMS protocol should contain three components:

1. A formalism for the message contents. There is a task dependent language to describe the information detailed in the messages; since this language is common to all of the cells, this language is also called the intercell language. The primary function of the intercell language is to describe the characteristics of the task in the message.
2. A formalism for the message format. This formalism, being independent of the task domain, is used to regulate the format of the message given a message type. For instance, the message for task announcement is formatted as (Smith [1980])

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

   where each component enclosed in "{ }" is to be filled with information encoded in the intercell language.
3. A formalism for the negotiation process. This formalism is used to describe the proper sequencing of actions to carry out contract negotiation among cells. Since several contracts may coexist in

a CFMS, this formalism needs to be able to describe asynchronous, parallel processes. The augmented Petri net model (Zisman [1978]) is used to model the negotiation process for its descriptive power in modeling concurrent problem solving systems.

The first two formalisms for regulating information in the message can be realized by context free grammar in the form of BNF expressions. This is a common treatment in modeling protocols (e.g., Smith [1980], Teng and Lui [1978]) and will not be elaborated here. In the following section, the formalism for contract negotiation is discussed.

The network on which the negotiation protocol is implemented can be modelled as a three-layer structure (Figure 4.4). The negotiation protocol is a high-level, problem-oriented protocol governing the communication between cell hosts for task-sharing. The host-to-host protocol, or the transport protocol, is used to provide reliable communication between processes in host computers. This layer is often implemented by the program called transport stations (TS) which is part of the host's operating system. The lowest level of the protocol, the transmission protocol, is responsible for the 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 multi-layer protocol model, as defined in Tanenbaum [1981].

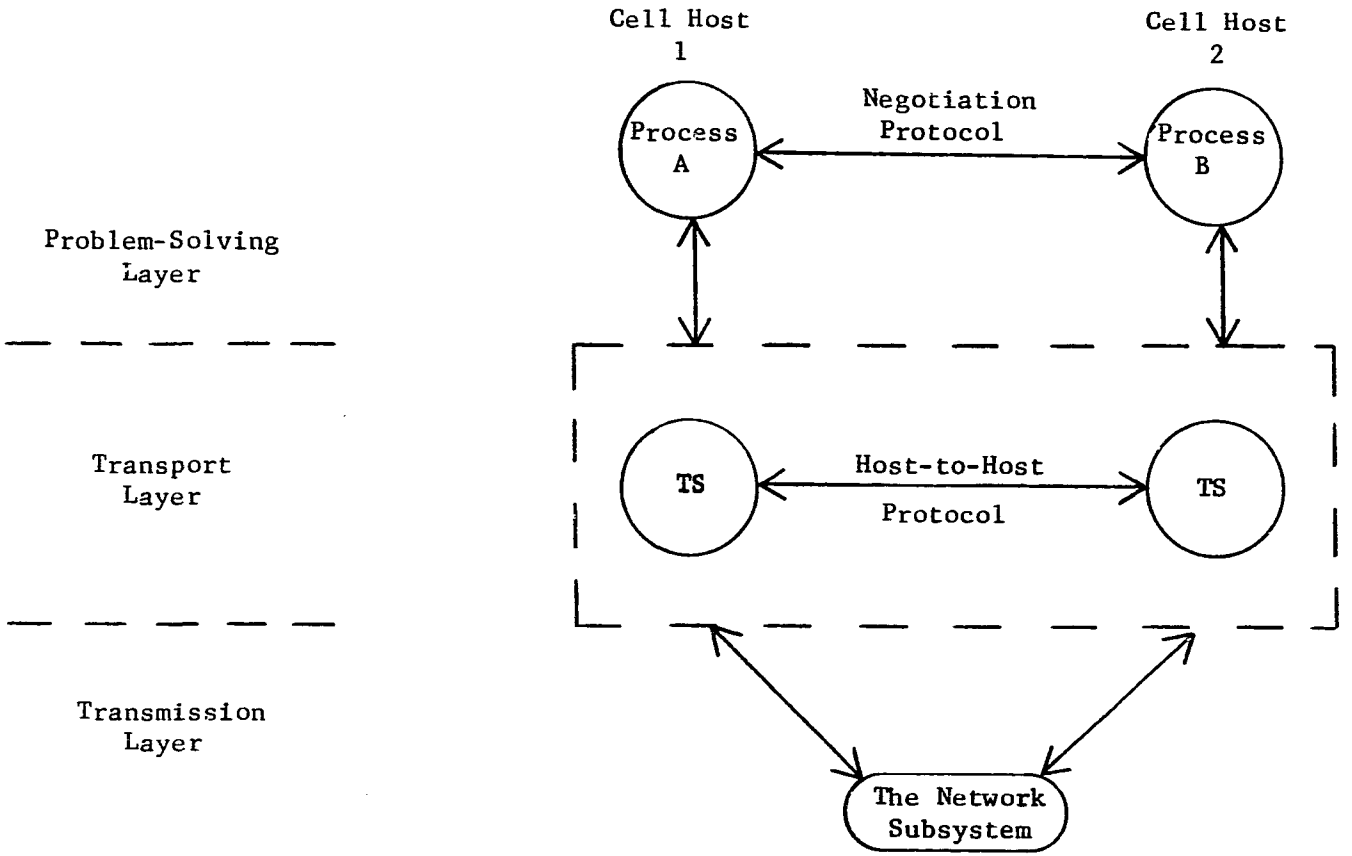


Figure 4.4 A Structure of the Protocol

4.4 Implementing the Contract Net

4.4.1 A Model for the Negotiation Process

During the contract negotiation process, there are strict requirements for communication and coordination between cells, while each cell is an independent, autonomous entity in the process. Thus a good formal model should ably describe two aspects of the negotiation:

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

Here we use the augmented Petri net model to represent the negotiation process. 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 grow exponentially. In the augmented Petri net model, 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 aspects 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 (Peterson [1981], Danthine [1980]). The definition of the Petri net follows:

Definition 1 (Petri Net)

A Petri net, W, is a four-tuple, $W = <P,T,I,O>$, where P is the set of places, T is the set of transitions, $I:T \rightarrow P^*$ is the input function, and $O:T \rightarrow P^*$ is 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.

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

Definition 2 (Petri Net Language)

If there exists a Petri net, $W = <P,T,I,O>$, a labelling function for the transition $1:T \rightarrow Z$, and an initial marking λ, then the possible sequences of transition firings generate the Petri net language:

$$L(1) = \{1(\beta) \in Z^* \mid \beta \in T^* \text{ and } \delta(\lambda,\beta)\}$$

where δ is the next-state function. For sequence of transitions $t_{j1}, t_{j2}, \ldots, t_{jk}$, $\delta(\lambda, t_{j1}t_{j2}t_{j2}\ldots t_{jk})$ is the result of firing t_{j1}, then t_{j2} and so on until t_{jk} is fired.

We now proceed to formally define an augmented Petri net:

Definition 3 (Augmented Petri Nets)

An augmented Petri net is composed of seven elements:

APN = <P,T,I,O,λ,AP,D>

where <P,T,I,O> is a Petri net 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. AP is the set of active productions and D is the set of database elements in the production system.

A transition t in T is firable iff

(1) t ∈ 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.

The execution of task-sharing negotiations in the CFMS demands well-governed interactions and cooperations between cells, exhibiting concurrency and parallelism. 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.

The concurrent nature creates difficulties in modeling the negotiation process. The transfer of messages (e.g., task-announcements, bids) from one cell to another requires that the activities of the involved cells be synchronized.

Adopting the augmented Petri net model, the negotiation process can be represented by two subsets: one (Figure 4.5) is constructed from the viewpoint 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 4.6) is constructed from the viewpoints of the cells who receive the task-announcement (the contractor cells). This sub-net deals with the decision on submitting bids.

Each event in the process is modeled as a production rule, and the interactions between these events are represented by the Petri net. Each transition in the Petri net (denoted by a bar in the figures) maps to a production rule. When a transition is enabled (all input places have at least one token), its firing will be determined by the rules corresponding to that transition.

Table 4.1 lists the set of production rules that determine the transitions in the two augmented Petri nets in Figure 4.5 and Figure 4.6. 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 contract negotiation process of all cells in deciding the proper actions, and they execute the actions in appropriate sequences.

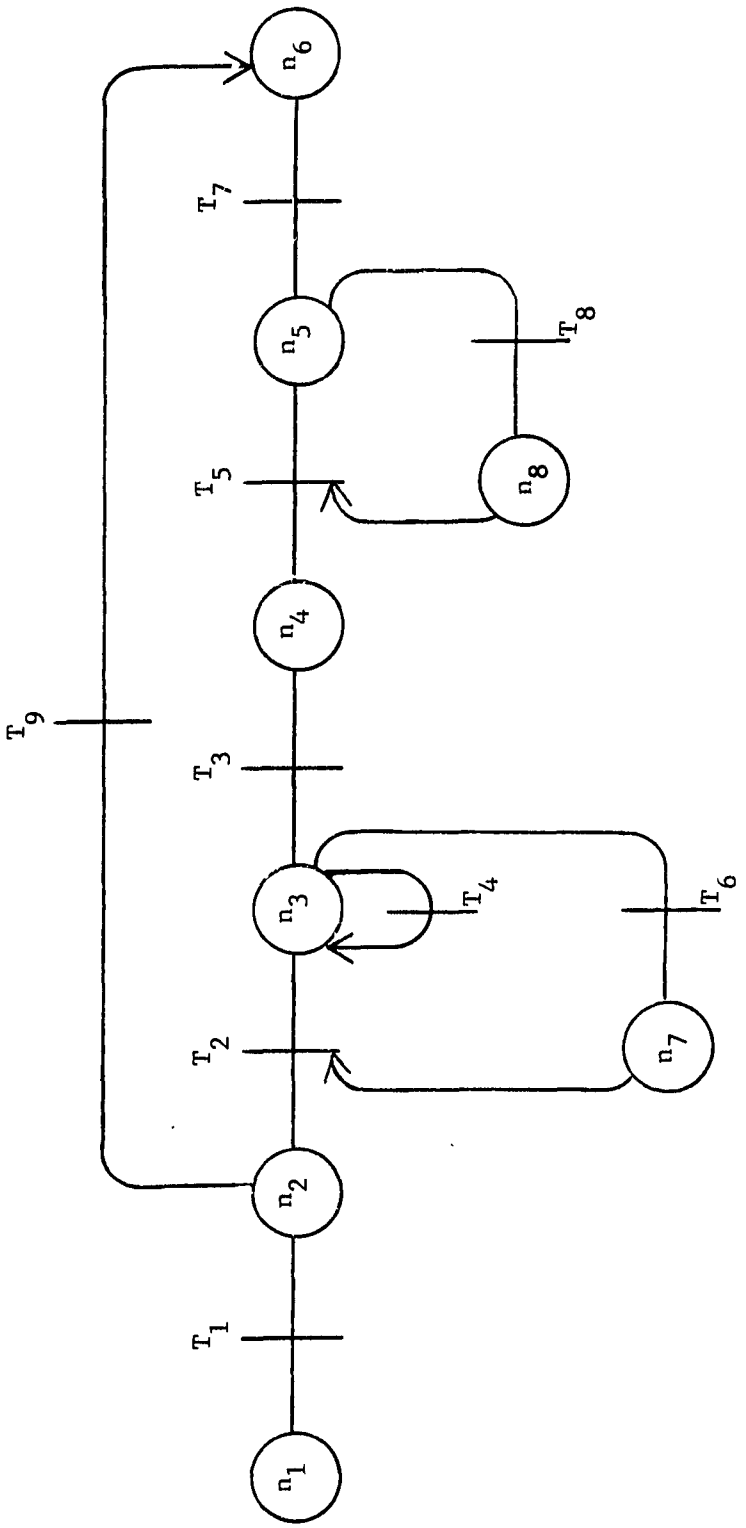


Figure 4.5 The Task-Announcement Process

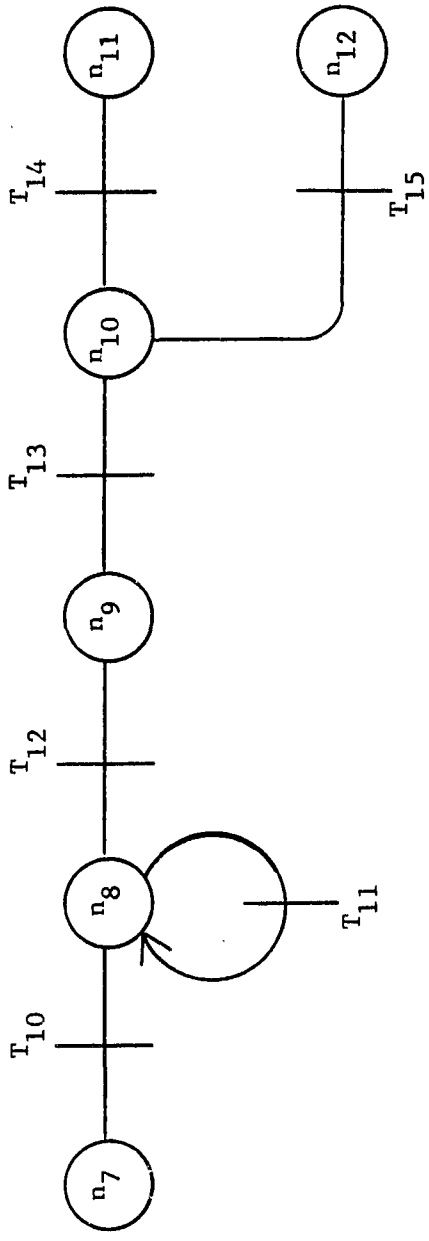


Figure 4.6 The Bidding Process

Table 4.1 The Production System

T_1... If a new task is generated
Then specify the information about the task in a task template.

T_2... If the cell cannot handle the task
Then initiate the task-announcement procedure.

T_3... If a bid is received
Then initiate the bid-processing procedure.

T_4... If the deadline is not reached
Then retain a ranked "bid list" on all the bids received so far.

T_5... If (the deadline is reached)
AND (the bid list is nonempty)
Then (Select the task on the top of the list)
AND (initiate the bid-award procedure).

T_6... If (the deadline is reached)
AND (the bid list is empty)
Then call reannounce procedure.

T_7... If the bidder accepts the award
Then list the cell in the "assignment-list."

T_8... If the bidder rejects the award
Then select the next highest ranked task.

T_9... If the cell can handle the task itself
Then list the task in the Agenda of the cell.

T_{10}... If (a task announcement is received)
AND (the cell is qualified for the task)
Then (initiate the task-ranking procedure).

Table 4.1, continued.

T_{11}... If the required processor is busy

Then put the task on the "active-task-announcement list" corresponding to that processor.

T_{12}... If the processor is idle

Then select the task of highest ranking in the active-task-announcement list, whose deadline is not reached.

T_{13}... If the deadline is not reached

Then (compute the information to fill out bid specification in the message)
AND (send the bid message)
AND (store the task information temporarily).

T_{14}... If (the bid is accepted)
AND (the cell decides to accept the award)

Then (store the task information into the Agenda of the cell)
AND (send an acceptance message to the manager cell).

T_{15}... If (the bid is granted an award)
AND (the cell decides to reject the award)

Then (send a reject message to the manager cell).

T_{16}... If the bid is rejected

Then (select the next highest ranked task in the active-task-announcement list).

4.4.2 Implementing the Protocol

Note that the negotiation protocol represented by the production system listed in Table 4.1 is specified in abstraction. Necessary services to carry out the implementation are provided by programs within the host and by protocols of lower layers; there may be several steps until the lowest level implementation of a given protocol layer structure is achieved. Specifying the protocol by abstraction and subject to hierarchical refinement in this manner is consistent with conventional protocol design principles (Bochman and Sunshine [1980]).

Table 4.2 lists a program which implements the rules of Table 4.1 in a LISP-like language. Three kinds of literals are used in the program:

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

The Petri nets for the negotiation process can be directly translated into a language called PNL (Petri Net Language), as illustrated in Table 4.3. 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 then translated, in turn, into a procedure language which can be the input language for an existing compiler

Table 4.2 Specifying the Production System in Program Form

T_1 if (NEW-TASK task)
then (TASK-INITIALIZATION task)

T_2 if (TASK-EVALUATE task)
then (TASK-ANNOUNCEMENT task)

T_3 if (BID-RETURN bid)
AND (LEQ time-now deadline)
then (BID-PROCESSING bid)

T_4 if (LEQ time-now deadline)
then

T_5 if (GT time-now deadline)
AND (NE bid-list blank)
then (BID-AWARD bid-list)

T_6 if (GT time-now deadline)
AND (EQ bid-list blank)
then (REANNOUNCE task)

T_7 if (REPLY-TO-AWARD accept)
then (LIST-ASSIGNMENT task)

T_8 if (REPLY-TO-AWARD reject)
then (RE-AWARD task)

T_9 if (NOT(TASK-EVALUATE task))
then (LIST-AGENDA task)

T_{10} if (TASK-ANNOUNCED task)
AND (BID-EVALUATE task)
then (TASK-RANKING task)

Table 4.2, continued.

T_{11} if (EQ(PROCESSOR-FOR-TASK task)busy)
then (LIST-ACTIVE-TASK-ANNOUNCEMENT task)

T_{12} if (EQ(PROCESSOR-FOR-TASK task)idle)
then (BID-REPLY(BID-SELECT a-t-a-1))

T_{13} if (LEQ time-now deadiine)
then (BIDDING task)

T_{14} if (BID-REPLY accept)
AND (CELL-CONDITION normal)
then (LIST-AGENDA task)
AND (REPLY-TO-AWARD accept)

T_{15} if (BID-REPLY accept)
AND (CELL-CONDITION not-normal)
then (REPLY-TO-AWARD reject)

T_{16} if (BID-REPLY reject)
then (RE-BIDDING(BID-SELECT a-t-a-1))

Table 4.3 The PNL Description for Petri Nets of the Negotiation Process

T_I:	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)		

(Nelson, et. al. [1983]). Thus we are able to have Petri nets as a high-level representation which can trigger the corresponding Petri net language when required; the description and processing of PNL require that unique names be assigned to the places and transitions.

4.4.3 A Task-Sharing Algorithm Based on Controlled Production System

By the aforementioned definition, the Petri net language (firable sequences of production rules in the augmented Petri net) can be used as a "control language" to regulate the invocation of production rules in the production system during the problem-solving process. In general, a production system whose control structure is described by a control language is called a controlled production system where the control language can be expressed as context-free languages, finite state language, or Petri nets.

The utilization of control language in the production system offers the following merits: first, the interpretation is more efficient because the control language in essence puts constraints on the applicable set of production rules and eliminates the irrelevant production rules from consideration; second, the control structure can be explicitly represented by proper modeling languages which can be subject to future modification without having to change the contents of the program, i.e., the production system. This separation of control with programs has been advocated by some researchers (e.g., Georgeff [1982], Kowalski [1979]).

A production system consists of a set of production rules and a database. Each of the production rules is an if-then conditional

statement. Whenever each condition element in a production is matched by database elements, the production is said to be invocable. The set of all invocable production rules is called the conflict set; a conflict resolution strategy is used to select one production from the conflict set. The actions of this production are executed, resulting in modifications of database elements (McDermott and Forgy [1978]).

An interpreter for production systems is called an inference engine (Davis, et. al. [1977]). The basic control cycle of the inference engine is the recognition-action cycle consisting of three phases: selection, conflict resolution, and action. In the selection phase all invocable productions are identified to form the conflict set. A conflict resolution method is applied in the second phase to obtain the production rule for execution. Finally, each action element of that production is executed.

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, expressed in the form

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

with q being 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.

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 $u \in C$ and $X_1 \in D$. A production rule p is said to be applicable if $up \in C$; a state $\langle up,X_2 \rangle$ is said to be derivable from the state $\langle u,X_1 \rangle$, denoted

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

iff p is applicable at $\langle u,X_1 \rangle$ and the interpretation of p, h(p), results in a pair $\langle q,r \rangle$ satisfying $q(X_1) = TRUE$ and $r(X_1) = X_2$.

In other words, the database elements in X_1 satisfy the preconditions of p, the actions of p change X_1 to X_2, and the control language accepts the production symbol p. The resulting new state after applying p on $\langle u,X_1 \rangle$ is $\langle up,X_2 \rangle$.

Thus, in the controlled production system, 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 to 5, we can now propose a theorem which formally proves that the augmented Petri net model in essence is a production system controlled by the Petri net language.

Theorem 1:

For any augmented Petri net

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

there exists a controlled production system

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

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

Proof

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

Now, if we let the L(1) in APN correspond to the control language C in M, then:

(1) If a production 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) t is applicable in the production system <T,D,h> and (b) t is accepted by the language C. Therefore, t is applicable in M. (Definition 5)

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

Q.E.D.

This isomorphism between the augmented Petri net model and 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 4.1 while being guided by the Petri nets shown in Figure 4.5 and Figure 4.6. The algorithm is similar to an inference procedure in the production systems (e.g., the OPS5 interpreter by Forgy [1981]) except that in the beginning of each cycle the algorithm decides on the "applicable" production set by using the control language. The algorithm is illustrated as follows:

Procedure - Task-Sharing {executed by a manager cell}
Input: a task T, consiting of a set of decomposable subtasks (t_i)
Output:
(1) Assignment of subtasks to cells
(2) Addition of subtasks to the agenda

Begin
Repeat {Based on Controlled Production System}
(Step 1) <Control> --
Apply the control language to decide the set of applicable productions → p'
(Step 2) <Selection> --
Select the productions among p' which are invocable → 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 RHS of the selected production

Until the goal condition.
end {task-sharing}

The inference engine adopts the data-directed search scheme in executing the task-sharing algorithm (Buchanan and Duda [1982]). In the first step, 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 consitute 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 the set of all invocable rules forms the conflict set CF; a conflict resolution strategy is used in Step 3 to select a single rule from the set CF. Sophisticated conflict resolution strategies have been devised (e.g., McDermott and Forgy [1982]), while the simplest scheme may be to select the first production rule that is invocable. Step 4 makes changes in the database according to the specification of the action part of the selected rule. The cycle continues until either of two conditions occur.

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 fails; an exception handling routine will be called upon to check the error.

The production system, the inference engine, and the database for task-sharing are integrated into a knowledge-based system called the contract processor, implemented in every cell host (Figure 4.7). The functions performed by the contract processor are similar to that by communication knowledge sources in the DPS network proposed in Lesser and Corkill [1983]. In terms of the distributed problem solving strategy presented in Section 4.2.3, the primary function of the contract processor is to distribute subtasks among the cells. A complete distributed scheduling algorithm based on the DPS strategy is shown schematically in Figure 4.8. Note that this is but a simplified version, since the decomposition of tasks into subtasks can be implemented recursively (for example, see Kieburtz [1979]).

The configuration of the knowledge-based system for executing the negotiation protocol is shown in Figure 4.9. The rules in Table 4.1 are stored in the knowledge base, the predicates for state descriptions are stored in the database. The control language is a procedure language translated directly from the PNL program in Table 4.3

4.4.4 Issues of Efficiencies

The efficiency of the task-sharing algorithm is affected by two factors: the efficiency of the controlled production system and the efficiency of the negotiation process. The former is related to the organization and search strategies of the production system, while the latter is related to the parameters of the negotiation protocol as set by the user. This section will explore both aspects.

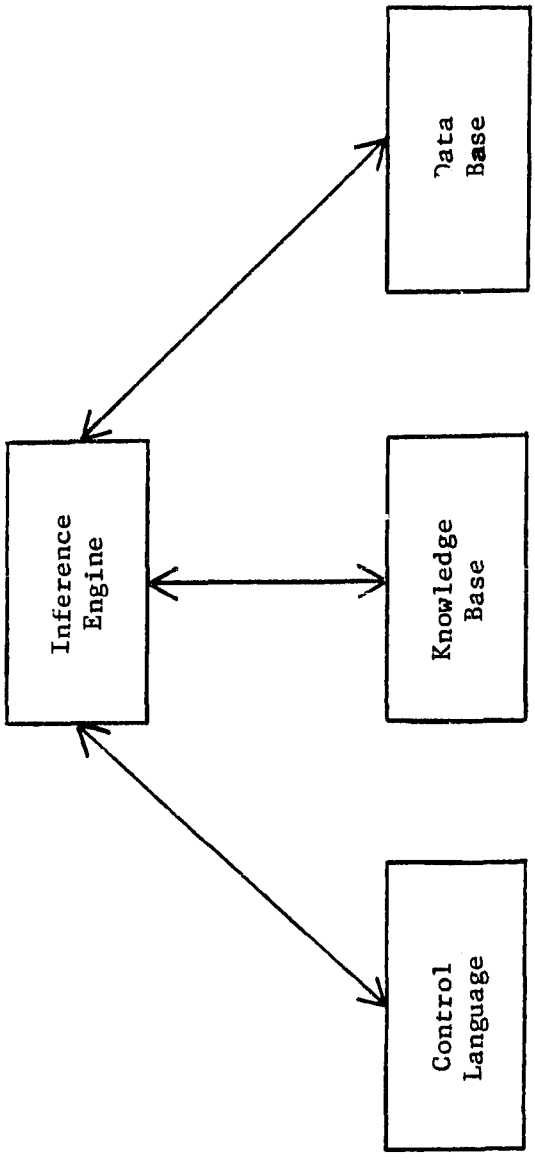


Figure 4.7 The Knowledge Base for Negotiation Protocol

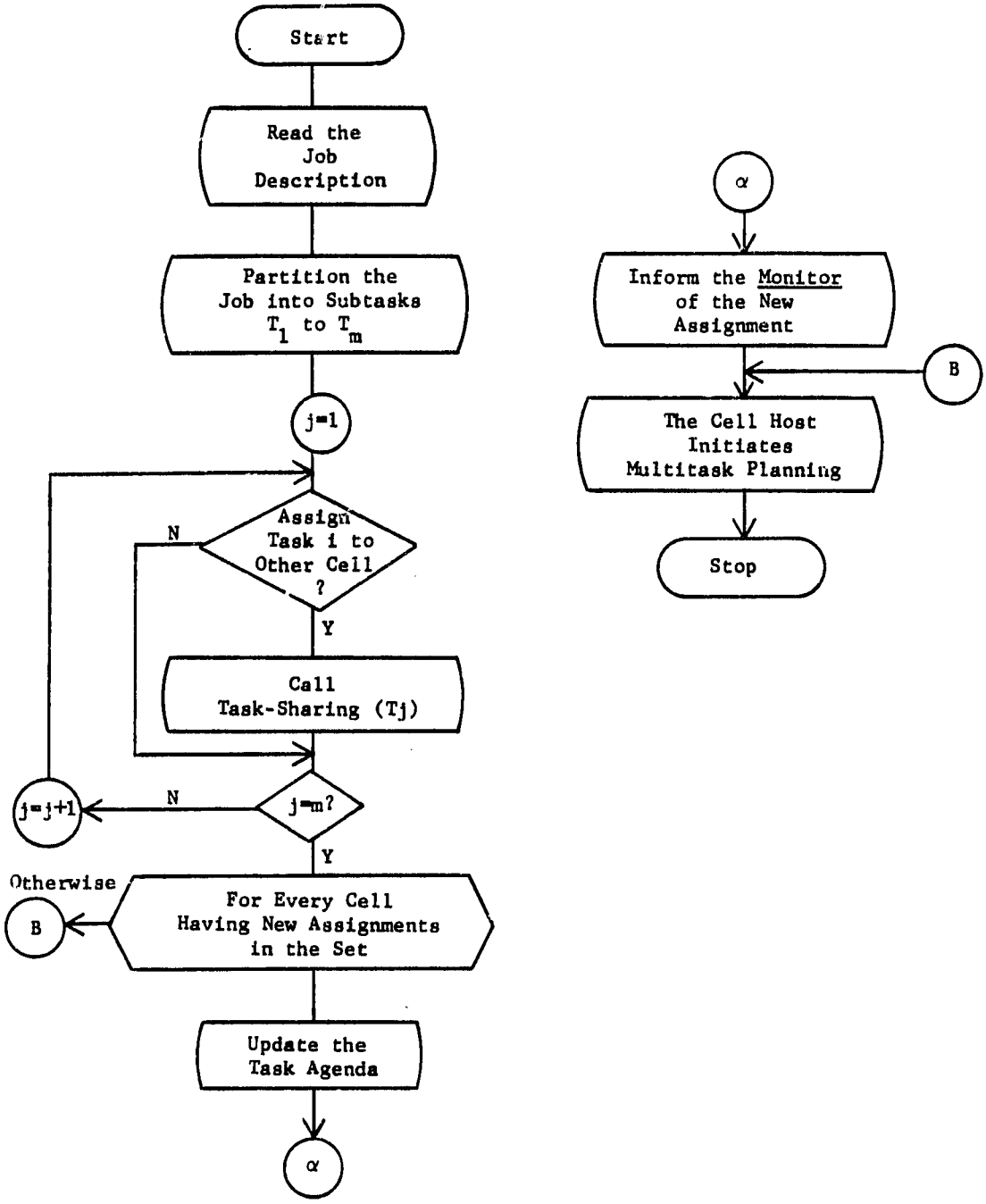


Figure 4.8 The Flow Chart of the Task-Sharing Algorithm

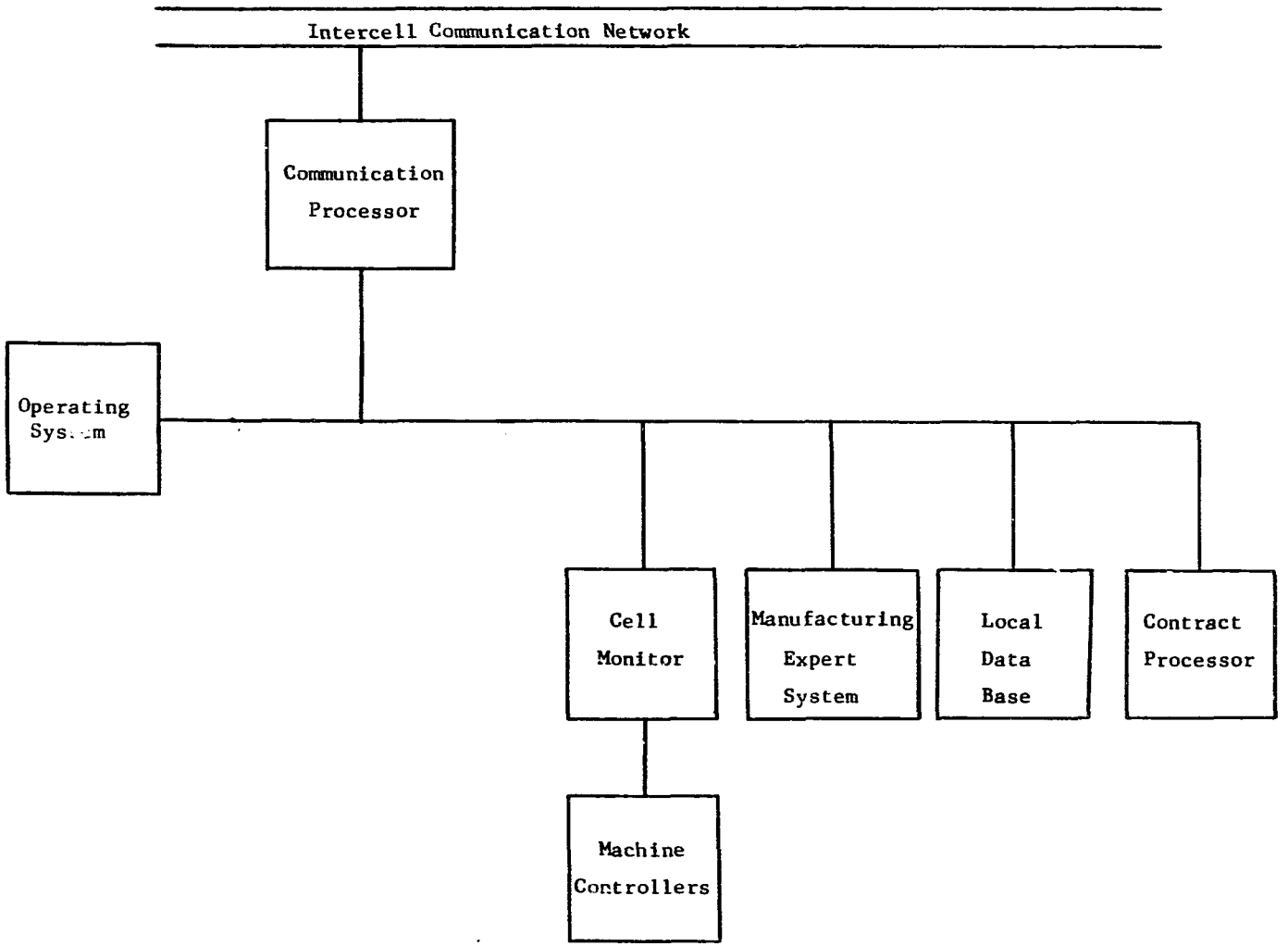


Figure 4.9 The Organization of a Cell Host

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

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

where P = size of the production system

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

T.match = Average time to find a match b
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

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 five elements:

task-announcement time: T_1

bidding time: T_2

bid-evaluation time: T_3

award time: T_4

accept time: T_5.

For the sake of simplicity, we shall assume that transmission speed of the communication subsystem is constant, i.e., is unaffected by the traffic load. Then the throughput time is

$$T = (T_1 + T_2 + T_3) \times m_1 + T_4 \times m_2 + T_5$$

where

m_1: average number of reannouncements

m_2: average number of reawards.

Simulation can be conducted to investigate the appropriate values of T_2 and T_4, i.e., the deadline for submitting bids and accepted awards.

The contract processor in essence serves as a scheduler that dynamically assigns tasks to other cells. In this regard, there are several other works which address similar problems in the distributed processing literature. Tilborg and Wittie [1981] proposed the use of a hierarchical high-level operating system as the control structure of the network; tasks are recursively subdivided and assigned to the processor which is the manager node of a subtree with sufficient

capability. Bryant and Finkel [1981] and Efe [1982] all used the loading factor as the criterion for subtask migrations to achieve load balancing. The cooperation mechanism is implemented in a pairing algorithm: pairs that differ greatly in load are connected, tasks in the more loaded processor are migrated to the other processor to reduce the load difference. All of these distributed scheduling methods assume that the processors are homogeneous; however the negotiation approach in this chapter does not need this assumption. Our method is also more intelligent and versatile in making dynamic assignments of tasks to processors by utilizing negotiation protocol as the coordinating mechanims: assignments are accomplished by the mutual selections between the manager nodes and the contractor nodes. The coordination itself is treated as a problem-solving process.

4.5 Concluding Remarks

Smith proposes a conceptual framework for organizing cooperation among decentralized and loosely-coupled problem-solving agents; he uses the contract net protocol as a high-level problem-oriented equivalence to the communications protocol of standard networks.

In this chapter we have attempted to design an intelligent manufacturing planning system based on the contract net framework, a model based on the augmented Petri nets is developed to specify the negotiation protocol. We feel that this model can better represent the dynamic and concurrent nature of the protocol in coordinating planning activities of manufacturing cells.

The mathematical properties of the production system and the Petri nets - safeness, deadlock-detection, reachability, and liveness- can be used to verify and analyze the protocol. Further, treating the distributed planning system by the community-of-experts metaphor enables us to take advantages of theories in economics and management science in modeling the contract net system.

The implementation of the protocol by a knowledge-based system helps realize 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 enforces the view that the contract net protocol is a problem-solving protocol, enabling us to adopt unified representations for both the planning activities and the cooperation process.

CHAPTER 5
SUMMARY AND CONCLUSIONS

The information system in the computer integrated manufacturing environment is a decentralized, loosely-coupled system connected by the communication network. To perform on-line planning and control of manufacturing processes, the information system is organized as a distributed knowledge-based system in this thesis. Each manufacturing cell uses a local knowledge-based planning system to manage the jobs in the cell, while interacting with the planning systems of other cells through the communication network. The system is decentralized because both knowledge and data are logically and geographically distributed: there is neither a centralized controller nor global data stroage. The system is also loosely-coupled as a result of the fact that, for each cell-host processor which contains the local knowledge base, a greater percentage of the processing capacity is devoted to problem solving than communication.

In the knowledge-based planning system within each cell, the plans which regulate the sequence of manufacturing operations are constructed in three steps. In the first step, a linearly-sequenced plan is generated for each job independently by a search procedure (this is also referred to as an inference process or rule interpretation procedure). Then, in the second step, a plan generator, called PLAN-AHEAD, is used to detect potential conflicts of machine

assignments for the multiple jobs and establish necessary precedence constraints to avoid any possible conflicts between different jobs. Conflict detection can be achieved either by a "critic" mechanism or a "reasoning about resource" mechanism. The best precedence ordering among the potential conflicting actions can then be determined by examining the duration information.

In the final step of the planning, step three, a plan-revision scheme is used to improve the plan constructed by the first two steps. The necessity of plan revision comes about from the fact that the operations for each job are generated independently in step one, without considering the requirements of other jobs. The underlying idea of plan-revision is to detect the situations when several jobs are waiting for the same machine and then consider reassigning waiting jobs to other machines in order to shorten the total planning duration.

An example concerning the on-line scheduling of multiple jobs in a flexible manufacturing cell has been used to demonstrate the planning method. It has been shown that the planning system can function as a scheduler in a multi-processor environment - a manufacturing cell, for instance - and efficiently assign sharing resources. Because of the additional capability of the environment, the scheduler needs to be more flexible than the traditional job-shop scheduler; the planning system is used to fulfill this requirement. Furthermore, the planning system can also handle the dynamic scheduling problem where jobs arrive at the system randomly over time. The scheduling is characterized as goal-directed; only the goal of the scheduling problem needs to be specified and the course of actions that can achieve

the goal are selected by the planning system. The total duration of the schedule is minimized by the planning method.

To incorporate decentralized control structure in the system, we use a "society of experts" metaphor as the basis to organize the information system. Each manufacturing cell is viewed as a problem solving agent with a certain area of expertise. The knowledge required to do a job is distributed among the agents. Thus, the decentralized task allocation problem is treated in the context of distributed problem solving with multiple agents. In the problem under study, each agent is a local knowledge-based system in charge of the planning and control of the manufacturing jobs as well as cooperating with other agents.

The major issue in this kind of distributed problem solving is two-fold: on one hand the plans produced by individual agents must be locally good, achieving their assigned tasks efficiently; on the other hand the aggregation of these local plans should result in overall solutions that are globally acceptable. Accordingly, coordination mechanisms are required to enable each agent to direct its own activities in concert with the activities of the other agents based on local decision and information.

Communication is the only way to coordinate the activities of individual agents because of the fact that the system is decentralized and loosely-coupled. However, instead of using distributed processing approaches which focus primarily on correctly transmitting bit streams of data through the communication network, we consider the communication activities at the problem solving level: the level that

determines the content of the message and the effective way for the agents to interact in order to achieve common goals. To this end, the information system should specify three things: the interface language for the agents, the process that regulates the proper interactions and the program that can automate the interactions.

A problem solving protocol, the negotiation protocol, is used to fulfill these requirements. The interactions among manufacturing cells are organized as the process of negotiation. Tasks to be distributed are broadcast through the communication network; each cell will evaluate the tasks and submit bids on those tasks suitable for local execution. Tasks are then assigned to the best bidders. By means of bid specifications and task abstractions in the interface language, the negotiation protocol helps to reduce message traffic and message processing overhead for using the communication network.

We have used the augmented Petri net model to represent the negotiation process. The augmented Petri net is an integration of two representation models: production rules are used to model the individual events and the Petri net is used to model the interactions and temporal relationships between these events. This augmented Petri net model can be implemented in a rule-based system with a specified control language, the Petri net language. The automation of the negotiation process is accordingly accomplished by the inference engine of the rule-based system and can be used to dynamically assign tasks to appropriate cells. By implementing the algorithm for task allocation in a knowledge-based system in this way, we in effect have a unified approach to the planning process and the cooperation process.

In a sense we have incorporated a market structure into the information system by means of the negotiation protocol, with tasks viewed as commodities and the cells as the bidders with varying preferences. Just like the way commodities are allocated to economic agents through the market, the manufacturing tasks can be allocated to appropriate cells by using the negotiation protocol in the information system.

As for the future extensions of this research, we need to expand the individual knowledge-based systems to fully represent the manufacturing problem domain and to increase the capability for dealing with complicated problems. More judgemental rules and heuristics should be incorporated into the knowledge base so that the planning system can include other functional areas in the manufacturing systems, e.g., process selection, inventory planning and control, exception handling, material requirement planning, etc. In addition, the knowledge-base system should provide an interface with on-line users, enabling question answering during the problem solving process.

Furthermore, the knowledge representation of the world model can use the frame-based representation for the invariant properties of the world, thus increase the efficiency of the pattern matching. Since time has been an important factor in the planning, proper representation of the time domain in the world model should also be formalized. The planning system can adopt the hierarchical approach, which describes actions by various levels of abstractions to facilitate the inference process for plan generation; constraints can also be established to reduce the search space.

Although the term "problem" in this thesis is defined in the context of manufacturing processes, it may very well be generalized to other distributed environments; e.g., a business procedure in the office automation system or a computing job in a computer network. Analytical models for the negotiation procedure will be developed in future research in order to identify formal bidding and award strategies in the specified problem domain.

LIST OF REFERENCES

LIST OF REFERENCES

Baker, K.R. Introduction to Sequencing and Scheduling, John Wiley & Sons, New York, 1974.

Bochman, G. and Sunshine, C. "Formal Methods in Communication Protocol Design," IEEE Transactions on Communications, 1980, pp. 624-631.

Bourne, D. and Fussell, P. "Designing Programming Languages for Manufacturing Cells," The Robotics Institute, CMU-RI-Tr-82-5 Carnegie Mellon, 1982.

Brinch Hansen, P. Operating System Principles, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

Bryant, R. and Finkel, R. "A Stable Distributed Scheduling Algorithm," Proceedings of the IEEE Distributed Computing Systems, 1981.

Buchanan, B. and Duda, R. "Principles of Rule-Based Expert Systems," Computer Science Department, HPP-82-14, Stanford, 1982.

Chandrasekaran, B. "Natural and Social System Metaphors for Distributed Problem Solving," IEEE Trans. on Systems, Man and Cybernetics, Vol. 11, No. 1, Jan. 1981, pp. 1-5.

Chen, P.P.S. and Akoka, J. "Optimal Design of Distributed Information Systems," IEEE Trans. Computers, Vol. C-29, No. 12, Dec. 1980, pp. 1068-1080.

Corkill, D.D. "A Framework for Organizational Self Design in Distributed Problem Solving Networks," COIN Tech. Rep. 82-33, Dept. of Computer and Information Science, University of Massachusetts, 1982.

Cutkosky, et. al. "Precision Machining Cells within a Manufacturing System," The Robotics Institute, Carnegie-Mellon, 1983.

Danthine, A. "Protocol Representation with Finite State Models," IEEE Transactions on Communications, 1980, pp. 632-642.

Davis, et. al. "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence, Vol. 8, 1977, pp. 15-45.

Davis, R. and Smith, R. "Negotiation as a Metaphor for Distributed Problem Solving," Artificial Intelligence, Vol. 20, 1983, pp. 63-109.

Efe, K. "Heuristic Models of Task Assignment Scheduling in Distributed Systems," IEEE Computer, 1982, pp. 50-56.

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

Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R. "The Heresay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," Computing Surveys, Vol. 12, 1980, pp. 213-254.

Forgy, C. OPS5 User's Manual, Dept. of Computer Science, CMU-CS-81-135, Carnegie-Mellon, 1981.

Fox, M.S. "An Organizational View of Distributed Systems," IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-11, No. 1, Jan. 1981, pp. 70-80.

Fox, M.S., Allen, B.P., Smith, S.F. and Strohm, G.A. "ISIS: A Constraint-Directed Reasoning Approach to Job Shop Scheduling," CMU-RI-Tr-83-8, The Robotics Institute, Carnegie-Mellon University, 1983.

Galbraith, J. Organization Design, Addison-Wesley, Reading, Mass., 1977.

Georgeff, M. "Procedural Control in Production Systems," Artificial Intelligence, Vol. 18, 1982, pp. 175-201.

Georgeff, M. "Communication and Interaction in Multi-Agent Planning," Proc. AAAI, 1983, pp. 125-129.

Groover, M. Automation, Production Systems and Computer-Aided Manufacturing, Prentice-Hall, Englewood Cliffs, N.J. 1980.

Hewitt, C. "Viewing Control Structures as Patterns of Passing Messages," Artificial Intelligence, Vol. 8, 1977, pp. 323-364.

Hoare, C.A.R. "Communicating Sequential Processes," Comm. ACM, Vol. 21, 1978, pp. 666-677.

Kieburtz, R. "A Hierarchical Multicomputer for Problem-Solving by Decomposition," Proceedings of the IEEE Distributed Computing Systems, 1979, pp. 63-71.

Kornfeld, W.A. "ETHER - A Parallel Problem Solving Systems," IJCAI, 1979.

Kornfield, W. and Hewitt, C. "The Scientific Community Metaphor," IEEE Systems, Man and Cybernetics, 1981.

Kornfeld, W.A. "Combinatorially Implosive Algorithms," CACM, Vol. 25, Oct. 1982, pp. 734-738.

Kowalski, R. "Algorithm = Logic + Control," Communications of the ACM, Vol. 22, pp. 424-436.

Lesser, V.R. and Corkill, D.D. "Functionally Accurage, Cooperative Distributed Systems," IEEE Trans. on Systems, Man and Cybernetics, Vol. 11, No. 1, Jan. 1981, pp. 81-96.

Lesser, V. and Corkill, D. "The Distributed Vehicle Monitoring Testbed: a Tool for Investigating Distributed Problem Solving Networks," The A.I. Magazine, Vol. 4, No. 3, 1983, pp. 15-33.

Lesser, V.R. and Erman, L.D. "An Experiment in Distributed Interpretation," IEEE Transaction on Computers, C-29 (12), Dec. 1980, pp. 1144-1163.

Lozana-Perez, T. "Robot Programming," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A.I. Memo 698, 1982.

Malone, T.W., Fikes, R.E. and Howard, M.T. "Enterprise: A Market-Like Task Scheduler for Distributed Computing Environments," Working Paper, Xerox Palo Alto Research Center, 1983.

March, J.G. and Simon, H.A. Organization, Wiley, New York, 1958.

Marschak, J. and Radner, R. Economic Theory of Teams, Yale Univ. Press, New Haven, CT, 1972.

McDermott, D. and Forgy, C. "Production System Conflict Resolution Strategies," in Pattern Directed Inference Systems, D. Waterman and F. Hayes-Roth, Eds., Academic Press, 1978.

McLean, C., Mitchell, M. and Barkmeyer, E. "A Computer Architecture for Small-Batch Manufacturing," IEEE Spectrum, May 1983, pp. 59-64.

Minsky, M. "The Society Theory of Thinking," in Artificial Intelligence - an MIT Perspective, P. Winton, Ed., MIT Press, 1979.

Moore, J.M. "A Review of Simulation Research in Job Search Scheduling," Production and Inventory Management, Vol. 8, No. 1, 1967.

Nau, D.S. and Chang, T.C. "Prospects for Process Selection Using Artificial Intelligence," Computer in Industry, Vol. 4, No. 3, 1981.

Nelson, R., Haibt, L. and Sheridan, P. "Casting Petri Nets into Programs," IEEE Trans. on Software Engineering, Vol. SE-9, No. 5, Sept. 1983, pp. 590-602.

Nilson, N. Principles of Artificial Intelligence, Tioga, Palo Alto, CA, 1980.

Nof, S.Y., Whinston, A.B. and Bullers, W.I. "Control and Decision Support in Automatic Manufacturing Systems," AIIE Trans., Vol. 12, No. 2, 1980, pp. 156-169.

Peterson, J. Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Rosenschein, J.S. "Synchronization of Multi-Agent Plans," PNCAI-82, Pittsburgh, PA, 1982, pp. 115-119.

Sacerdoti, E.D. A Structure for Plans and Behavior, Elsevier, North-Holland, New York, 1977.

Shaw, J.P. and Whinston, A.B. "Distributed Planning in Cellular Flexible Manufacturing Systems," Technical Report, Management Information Research Center, Krannert School, Purdue University, 1983.

Simon, H.A. The Sciences of the Artificial, Second Edition, The MIT Press, Cambridge, Mass., 1981.

Smith, A. Wealth of Nations, 1776.

Smith, R.G. "A Framework for Problem Solving in a Distributed Environment," Ph.D. thesis, Computer Science Dept., Stanford Univ., 1978.

Smith, R.G. "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computer, Vol. 29, 1980, pp. 1104-1113.

Tanenbaum, A. Computer Networks, Prentice Hall, New Jersey, 1981.

Tantawi, A.N. and Towsley, D. "Optimal Load Balancing in Distributed Computer Systems," IBM Thomas J. Watson Research Center, 1984.

Tate, A. "Generating Project Networks," Proc. IJCAI, 1977. pp. 888-893.

Teng, A. and Liu, M. "A Formal Approach to the Design and Implementation of Network Communication Protocol," Proceedings of COMPSAC, 1978, pp. 114-128.

Tenney, R. and Sandell, N. "Structures for Distributed Decision-making," IEEE Transactions on Systems, Man and Cybernetics, 1981, pp. 517-527.

Tilborg, A. and Wittie, L. "Wave Scheduling: Distributed Allocation of Task Forces in Network Computers," Proceedings of the IEEE Distributed Computing Systems, 1981, pp. 337-347.

Vere, S. "Planning in Time: Windows and Durations for Activities and Goals," Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, May 1983, pp. 246-266.

Von Hayek, F., "The Use of Knowledge in Society," The American Economic Review, 35, Sept. 1945, pp. 519-530.

Wah, B.W. "File Placement on Distributed Computer Systems," IEEE Computer, Jan. 1984, pp. 23-32.

Wilkins, D. "Parallelism in Planning and Problem Solving: Reasoning about Resources," Tech. Note 258, AI Center, SRI Inter., 1982.

Zisman, M. "Use of Production Systems for Modelling Asynchronous Concurrent Processes," in Pattern Directed Inference Systems, D. Waterman and F. Hayes-Roth, Eds., Academic Press, 1978.

Zisman, M. "Representation, Specification and Automation of Office Procedures," Ph.D. dissertation, Wharton School, University of Pennsylvania, 1977.

APPENDICES

Appendix A

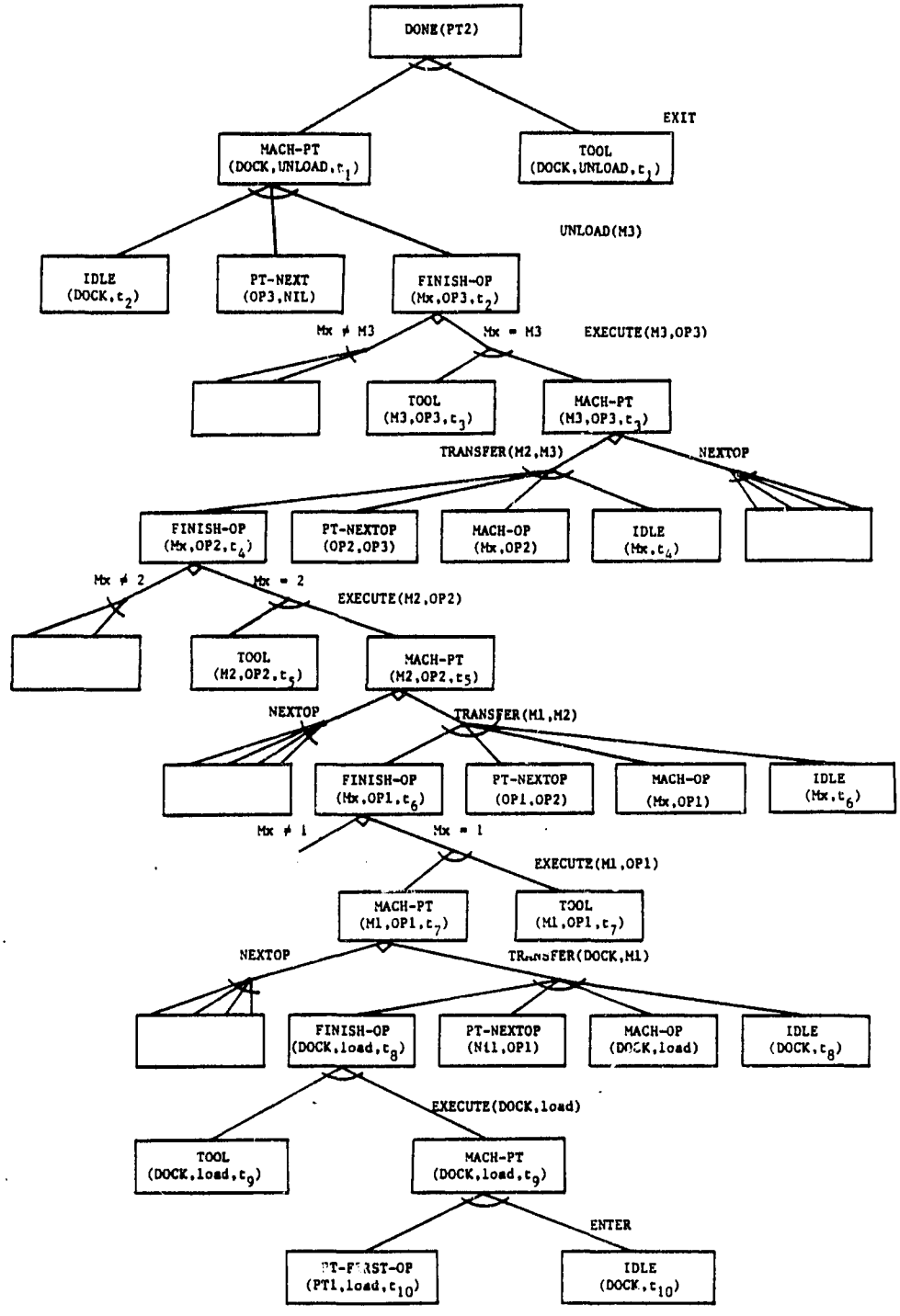


Figure A.1 The Search Tree Generated for the Planning of PART 1

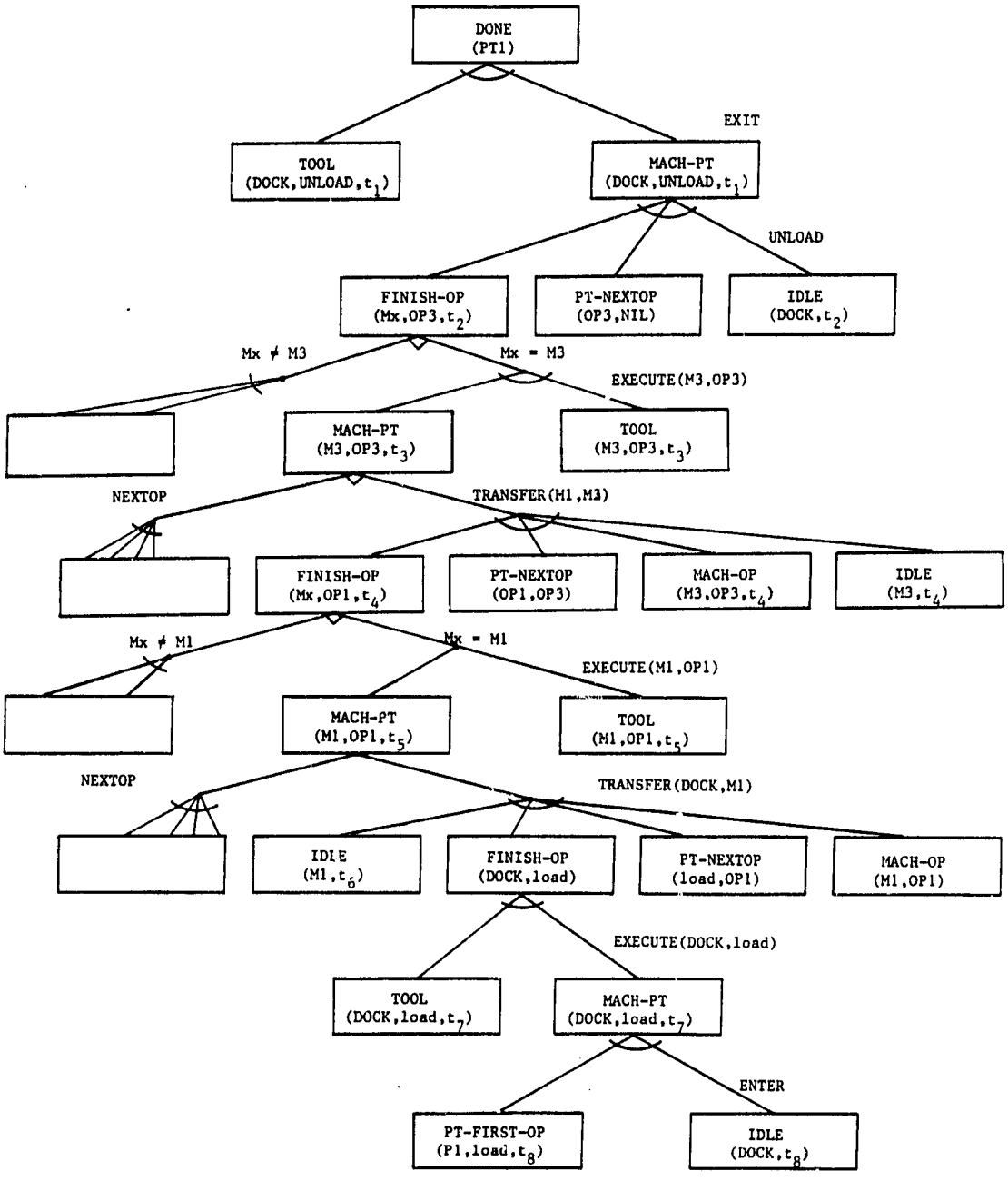


Figure A.2 The Search Tree Generated for the Planning of PART 2

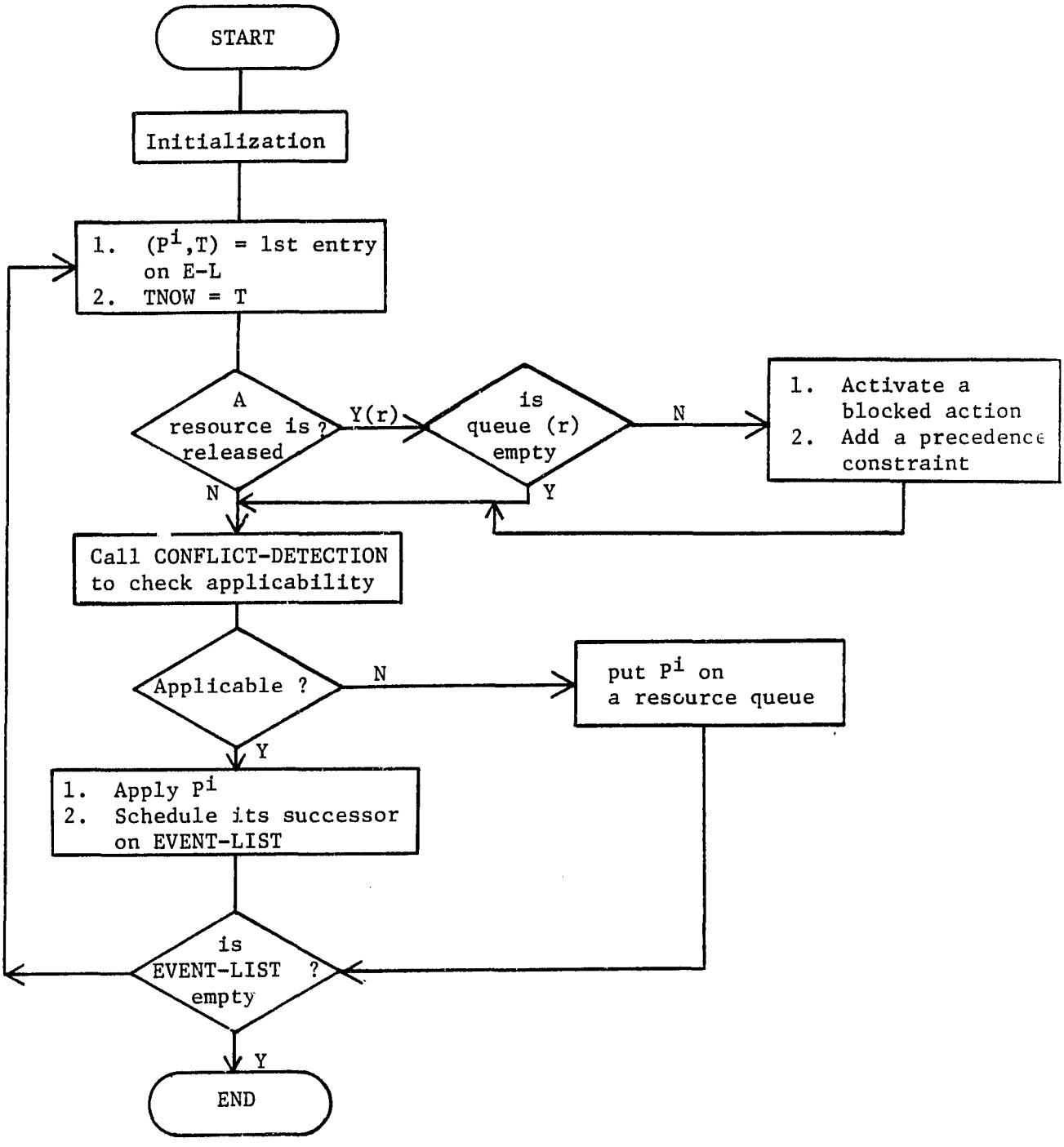


Figure A.3 The Flow-Chart for PLAN-AHEAD, a Plan Generator

Appendix B

Table B.1 Predicate Literals Used in the Negotiation Protocol

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 B.2 Functions 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 B.3 Procedures 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 B.4 Lists Used in the Negotiation Process

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.

VITA

VITA

Jeng-Ping Shaw was born in Taipei, Taiwan on July 13, 1956. He received a B.S. in Industrial Engineering from Hsing-Hwa University at Hsinchu, Taiwan in June, 1978. Subsequently, he earned an M.S., also in Industrial Engineering, from the State University of New York at Buffalo in August, 1980. He had been in the doctoral program at the School of Industrial Engineering at Purdue University for one year before entering the Krannert Graduate School of Management in the Fall of 1981. He received the Ph.D. degree in Management Information Systems from Purdue University in August, 1984.

Jeng-Ping Shaw is married to Crystal J. Shaw. He will be Assistant Professor of Business Administration at the University of Illinois at Urbana-Champaign in the Fall of 1984.