KNOWLEDGE BASED DYNAMIC RESTRUCTURING OF FLEXIBLE PRODUCTION SYSTEMS

By

Jianhua Gu

B.A.Sc. (Electrical Engineering) Tsinghua University, China, 1985

M.Sc. (Electrical Engineering) Chongqing University, China, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September, 1994

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Mechanical Engineering

The University of British Columbia
Vancouver, Canada

Date October 7, 1994

DE-6 (2/88)

Abstract

This thesis addresses high level automation in flexible production systems (FPS). In a flexible production system, for example in a dedicated FPS for fish processing, the process components are typically organized as distinct workcells where each workcell is responsible for a special class of production activity. Changes in production demand, variation in raw material supply, and malfunction or failure of workcell components can change the workload on some components in the workcell. A technique that is termed dynamic restructuring, which enables an FPS to change its configuration automatically to suit new situations, thereby achieving a near-optimal load distribution among system components is developed in the thesis.

Characteristics of the restructuring problem are known to be fuzzy (both restructuring goal and FPS status), complex, and non-analytic. A knowledge-based method is employed for solving this problem rather than the conventional mathematical methods. Different types of knowledge sources are organized in a three-level hierarchy of decision making:

- The first (highest) level determines whether there should be a restructuring operation according to the current FPS status.
- The second level determines a restructuring method, in which, due to the fuzziness of the restructuring goal, general heuristics are used.
- The third level selects an optimal action based on the current situation, employing fuzzy logic to handle system uncertainty.

Various approaches of knowledge representation and reasoning are used, as appropriate in different levels of the hierarchy. A blackboard architecture is implemented to coordinate associated knowledge sources, which makes the entire system easily expandable. Heuristics are organized in the order of their priorities, in the planning of restructuring actions. Techniques of fuzzy associative memory are developed to evaluate a set of possible restructuring actions from which a decision maker could pick an optimal one. Therefore, the system is intelligent in the sense that the decision maker always selects a proper action based on the current system status.

The FPS restructuring system is implemented in Prolog. The approach is applied to an automated fish processing plant, in computer simulation. Several special cases are studied through simulation experiments, to demonstrate the practical use of the approach.

Table of Contents

List of Figures

Acknowledgment

I would like to take this opportunity to express my deep gratitude to my supervisor Dr. Clarence de Silva for his encouragement, support, and delightful guidance throughout the entire course of my M.A.Sc. program.

My acknowledgment should also go to my colleagues in the Industrial Automation Laboratory for their help, discussion and comments on my research project.

The financial support by grants to Dr. C.W. de Silva from the *Advanced Systems Institute of B.C.*, and from the *Natural Sciences and Engineering Research Council of Canada* is highly appreciated.

Chapter 1

Introduction to System Restructuring

This chapter introduces the work that is described in the thesis. It addresses the concept of dynamic restructuring of flexible production systems, and the background information that is needed to study this concept. The characteristics of system restructuring is discussed. Based on this backdrop, machine intelligence is considered for the problem solving, along with appropriate control techniques, and fuzzy logic is proposed here to deal with complexity, uncertainty and the non-analytic nature of the problem.

Section 1.1 starts with a discussion of the demand for high level automation in flexible production systems; Section 1.2 describes a hierarchical structure that is suitable for monitoring and control of an automated system; Section 1.3 studies the feasibility of dynamic restructuring in flexible production systems (FPS); and Section 1.4 discusses some important characteristics of the restructuring problem in an FPS. According to these characteristics, an approach is proposed in Section 1.5 for solving the restructuring problem. Section 1.6 outlines the objectives of the thesis. Finally, Section 1.7 gives concluding remarks for this chapter, which will include a brief overview of the entire thesis.

1.1 Demand for High-Level Automation

Due to the market competition, modern production systems are developed focusing on the two objectives: flexibility and minimum inventory, especially minimum intermediate inventory [43, 45]. Efficient management and high level control are required for achieving

these objectives.

Fluctuating supply and demand levels and the popularity of small-batch production call for high-level flexibility and adaptability in a production system. Systems with these characteristics can better cope with increasing quality and throughput rate demands of the market. To respond to this challenge, highly automated systems have to be developed. Such systems should be intelligent with the ability to adapt quickly to different production operations autonomously and self-organize their structure to make them operate efficiently and cost-effectively [10].

In a conventional production system [45, 43, 29], workcells are linked into a fixed architecture. Workload of components in a workcell changes correspondingly only in response to the processing demand on the workcell, which may fluctuate due to market factors. Some components in a particular workcell may work at a fraction of their full capacity, while similar components in another workcell are overloaded. In order to achieve efficient and optimal operation of the overall system, the workloads of the same type of components should be redistributed. This "balanced" operation may be realized by means of restructuring the workcell configurations. In order to achieve this, the workcell boundaries should be flexible and the overall system structure variable, resulting in a dynamic structure control system [10] (see Figure 1.1).

In Figure 1.1, suppose that the production demand for $workcell_j$ is increased, which would make some components in the workcell overloaded. Meanwhile, suppose that there exists a surplus in the component capacity in $workcell_i$; for example, components A, B in $workcell_i$, which are of the same types as the overloaded ones in the $workcell_j$, operate below their full capacities. Therefore, component B may be reassigned to $workcell_j$, so as to fill its processing capacity requirement. Note that the original workload of component B may be transferred to other components of the same type in $workcell_i$ before assigning B to $workcell_j$, which is known as component releasing. Component A shows another

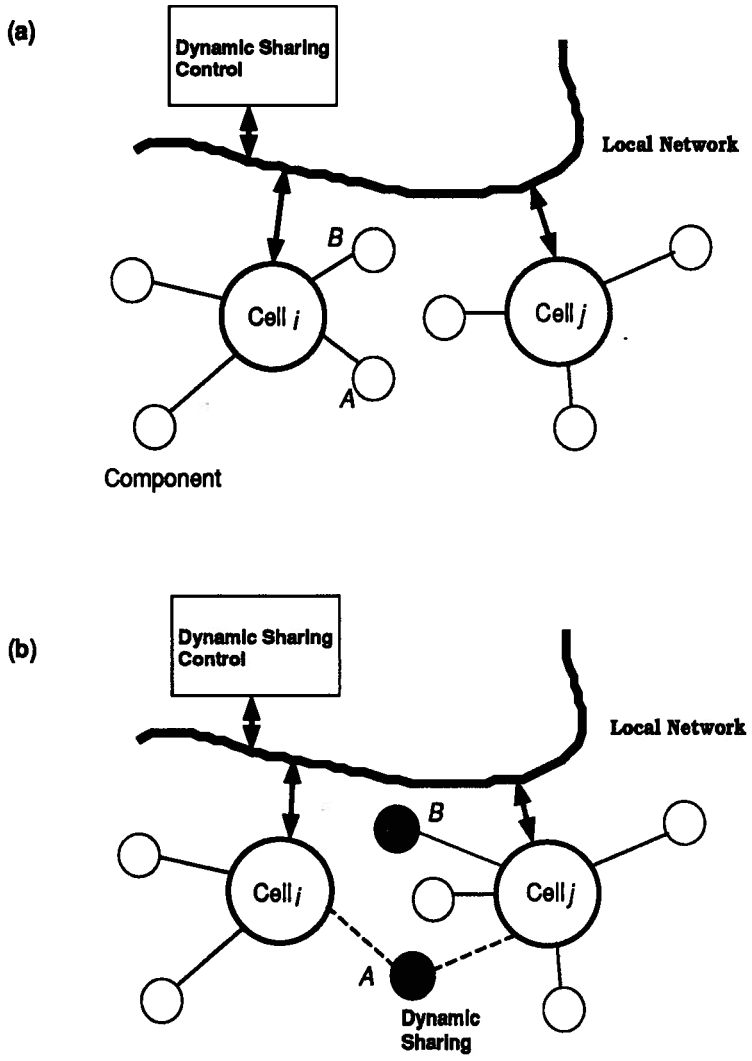

Figure 1.1: A Multiple Cell Production System and Its Dynamic Reconfiguration: (a) Before Reconfiguration; (b) After Reconfiguration.

operation mode. It is shared between the two workcells. In this latter case, the workcells should be scheduled cooperatively for sharing the components.

It is important to mention how the shared components work. They could be scheduled in different operation modes as illustrated in Figure 1.2. Here, case (a) is considered as

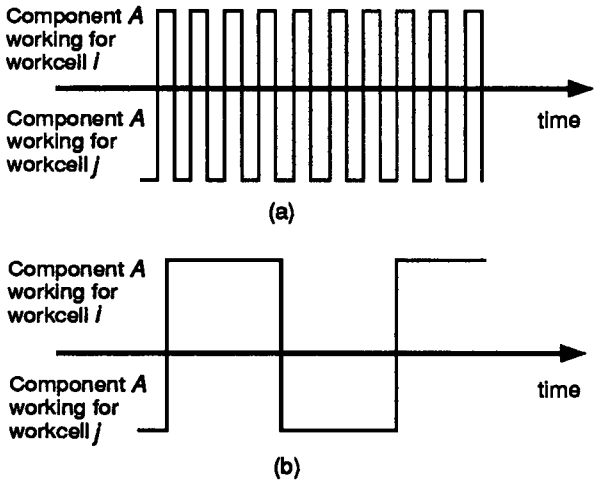

Figure 1.2: The Component Sharing Modes: (a) Run-Time Component Sharing (b) Static Sharing

that where A works for both $workcell_i$ and $workcell_j$, which is somewhat analogous to time sharing of a computer CPU for multiple processes, resulting in slow processing speed, but without extra intermediate inventory. It should be noted that, the processing speed (fast or slow) or workcell activity level (high or low) are of significance. In case (b), component A works for a workcell for a considerable long time, and then shifts for another workcell, and then returns to the first workcell, and so on. In the second case, there would be significant extra inventory, provided that the processes of the two workcells are related and are constituents of a complete production activity, which is the usual case in production. This extra inventory will degrade the system performance.

Consequently, dynamic sharing of a component, which requires switching of components in order for them to work for more than one workcell in runtime, as shown in case

(a) discussed above, for example, is required in order to achieve system flexibility and zero/minimum inventory.

1.2 Structure of a Hierarchical Automation System

For better understanding of dynamic restructuring from the point of view of information exchange and control, the system in Figure 1.1 can be organized into a control hierarchy as schematically illustrated in Figure 1.3.

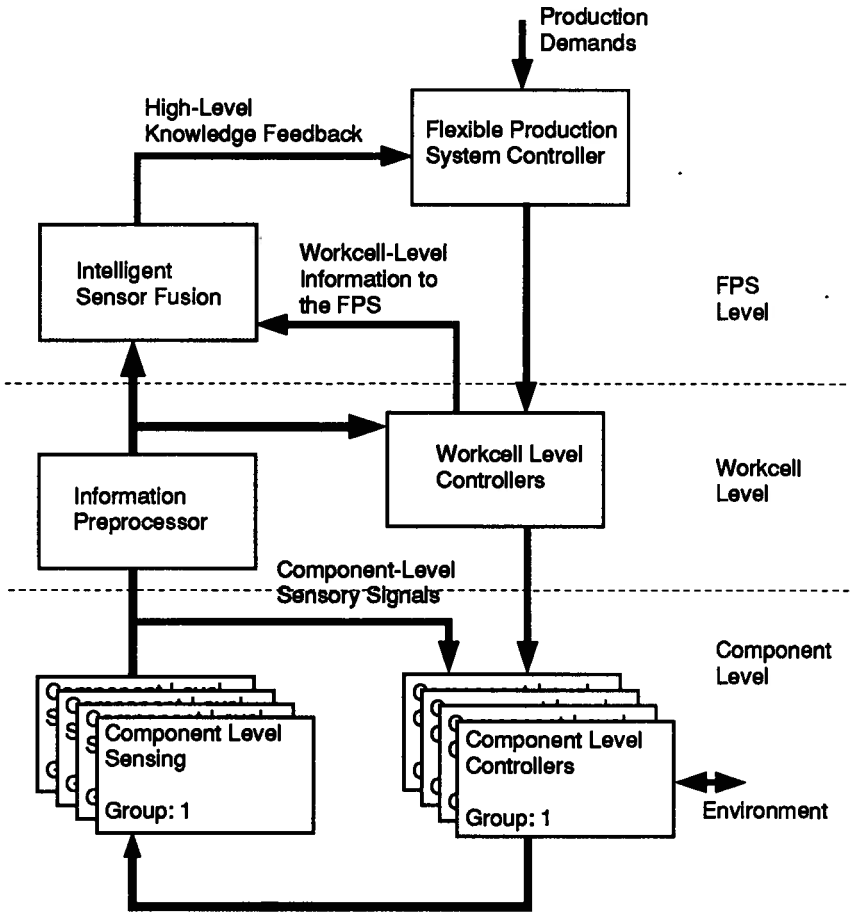

Figure 1.3: The Hierarchy of an FPS Control System

The top level FPS controller plays the important role of management of the entire

system. It responds to the changes of task demands and the feedback information; makes decisions on FPS reconfiguration; and commands the low-level controllers to employ proper components in carrying out the required taks. The task demands considered here are assumed independent and assigned to individual workcells, which may actually be part of a complete production task. For example, product grading and packaging may be considered independent for separate workcells, but they may be (usually are) part of a complete production activity, as an example, fish processing, which may include fish head cutting, grading, and packaging. Task demand management, which is to decide task demands for individual workcells according to the overall production requirements, is not included in this system. However, it is important to note that the tasks assigned to different workcells are related, and for achieving zero intermediate inventory, the workcells should operate at a desired processing speed. The major objective of an FPS controller is to realize the desired processing rates under the condition of current available facility resources. In particular, if the workload of the overall system is not properly assigned to its components, say, some are overloaded, while some of same type possess significant capacity surplus, the system should be restructured so as to achieve uniform or balance operation, which will result in increased system performance and reduced operating cost.

A workcell controller is the executor of an assigned task. It performs the given task in a desired rate using designated components under control of the FPS controller, and may include some components in a sharing mode. Components are scheduled and controlled according to the task type and processing rate. Task planning is discussed in Vollmann (1992) [45]. In particular, the shared components should be scheduled in coordination with other workcells. In dedicated production systems, a fish processing plant, for example, task planning is standardized, and the component workloads will vary almost proportionally with the task demand, which may be represented by a processing rate. Of course, for different tasks, component load factors, (*workload/unit-task*), may

be different. For example, suppose a vision system is required for fish head cutting, and the load factor is $0.4s/$*unit-cutting*. As a result, for a unit cutting (1 *unit-cutting/sec*), a standard[1] vision system should work $0.4s$. If the same vision system is used for fish grading, the load factor may be lower, say, $0.25s/$*unit-grading*.

Components physically belong to workcells, but are logically grouped by their functions, and assigned to groups. Components in the same group are assumed to be interchangeable and shareable (The feasibility aspect will be considered in the next section). Even human workers could be grouped into one or more groups when they are considered as part of a workcell in this manner. A component controller responds to commands from the workcell level. For example, an image processing system will execute a particular job as required, for example, to measure the gill position of a fish or to analyze the firmness of an object such as herring roe.

The low-level sensory signals are fed back to component controllers as well as workcell controllers for closed loop control. The component signals, together with reports from the workcell controllers, are pre-processed by an intelligent sensing system, which will generate information for the FPS, and will provide knowledge to the FPS controller in a proper form, as fuzzy sets, for example.

The information between the production system and outside world takes two forms: updating of the task demands, and interactions between components and their enviroment, for example in material handling and processing. Both types of information may trigger the system for restructuring. The change of task demands, which results in the change of component workloads, is a direct trigger for restructuring. Also an FPS may call for a restructuring action during processing, when some components are damaged or partially damaged and have a degraded performance. Such a damage may cause a reduction of the component capacity, may reduce the speed of communication, and can

[1]This concept is used for component capacity measurement, which will be discussed in Chapter 2.

bring about other undesireable effects.

This thesis will be centered around the restructuring controller of a flexible production system (FPS), which will be designed to make decisions based on task demands and FPS information, to assign proper components to a given production activity.

1.3 Feasibility of FPS Restructuring

Three points are presented here to support the concept of system restructuring. Firstly, well developed low level automation provides the possibility of dynamic restructuring of a system. Especially, programmability and the availability of sophisticated component level controllers that have the necessary flexibility make some components easily be shiftable to other jobs. Then, it will be possible for a component to shift from one task to another, or work in a sharing mode.

As an example, an image processing system can function differently using different program modules. It may calculate the mean brightness to measure temperature from an infrared image; firmness from an ultrasonic image, and colors from the separate Red-Green-Blue (RGB) channels of a color image. It may also recognize and "measure" the position and shape of an object. In fish processing industry, such functions are needed in many different workcells, with different processing time requirements. Therefore, some image systems, if they operate below full capacity, could be used for more than one workcell to save operating costs.

Secondly, local network communication should be highly developed, which can effectively support the high-level hierarchical control; provide information feedback; and enable cooperation of shared components. For example, it is well known that automatic guided vehicles (AGV) usually operate in a completely sharing mode.

Thirdly, one should take into consideration that some types of components in working

site are actually mobile. Human beings, when considered as components in workcells, can walk from one site to another. Mobile robots [19] also fall into this category. In this case, travelling time of a component may be included as a part of the working time in scheduling. The shifting frequency, therefore, needs to be optimized by considering the ratio of working time and travelling time. However, this is not a major topic here. Some robots (or human beings) can perform work, such as loading and unloading, in a sharing mode. To generalize the feasibility representation in dynamic restructuring, the concept of feasibility index is introduced. Two situations can be considered:

- **Load sharing.** In this case, a load is shared by more than one component of a same type. The feasibility index of load sharing is determined by the type of load and attributes of the sharing pair of components. Also, some geographical factors, such as FPS layout, and network communication capacity should be considered as well. A system expert may recommend a value or an algorithm to calculate the feasibility index. This feasibility index is taken into account when an overload is to be shared by another component, the load sharing feasibility should be considered between the pair of components, one of which is overloaded and the other is at below capacity.

- **Component sharing.** Here, a component is assigned to more than one load. The component may work for several workcells which carry out different tasks. The feasibility index of component sharing is slightly different from that of load sharing. The former case happens when a component is to be released from the system, its workload should be transferred to another component of the same type. The second component is therefore in a mode of component sharing: it works for more than one job. In this case, the feasibility index of component sharing is primarily determined by the component itself, rather than a pair of components.

In both the cases, a basic property of components which is associated with the capability of sharing determines the feasibility index. In this research, it is defined as "component shareability". The shareability index of a component is considered as the capability of this component to share loads, the basic factors that affect the index being the capability of communication, its mobility, etc, of the component. The index takes a real number in the interval $[0, 1]$, and is usually assigned by a system expert. This number indicates the degree of shareability of a component; in particular, "1" means the component is completely shareable, and "0" means the component is not shareable at all. This, then, together with geographical layout and task type, will be used for evaluating component sharing. The feasibility of load sharing, which involves pairs of components, also takes into consideration such factors as geographic layout and repair state of the components.

1.4 Characteristics of Dynamic Restructuring

To develop an approach for solving the present problem, it is helpful to understand the characteristics of the problem. From the point of view of control, the task and the performance requirements of an FPS controller are greatly different from that of a low-level control system [21]. It has the common characteristics of a high-level control system as well as its own specific features.

Generally, a high-level control system is different from a low level control system in the following aspects [7, 8, 11, 40].

- **Control bandwidth and dynamic response.**

  The control cycle of high-level restructuring is sufficiently long that the system may be considered to be in a steady state (normal operation) most of the time. The steady state, therefore, is the most important consideration. What we optimize is

the FPS operation rather than the dynamic response during the transient stage of performing a restructuring operation.

- **Intelligence and accuracy.**

  In general, required intelligence is high and the associated precision may be low for decision making in a high-level system. The restructuring system should be designed to make intelligent decisions and to deal with uncertainty of information and knowledge.

What should be particularly stated with regards to a dynamic restructuring controller are the following features:

- **The fuzziness of the performance requirement itself (fuzzy goal).**

  This feature makes the goal of the control system fuzzy, resulting in the possibility of multiple solutions. The general requirement of restructuring is to make an FPS operate uniformly with respect to the load distribution among components, and also to make the workcell components work close to their full, design capacity. An example is given now to indicate how fuzziness may enter into the decision making process of system restructuring. Suppose two components A and B are identical and shareable. In one case, component A works at 50% of its full capacity and, component B works at 90% of its full capacity. In another case, component A works at 70%, and component B works at 70% as well, in which 20% workload of B is shared by A. It is difficult to precisely determine which case is better, and fuzziness may enter the decision making process.

- **The non-analytic nature of the restructuring problem.**

  It is the fuzziness of a restructuring goal that leads to this second feature of the problem. In the absence of fuzziness, if there exists an analytic cost function, for

instance, a linear programming method could be used for solving the associated problem. Such a method, like an optimal control design, is actually driven by a requirement of control performance. Usually, the algorithm can be deduced from the cost function and system constraints, such as a mathematical model. In a restructuring problem, both the control performance and the constraints may be fuzzy, and the problem is generally non-analytic. Therefore past experience would be very valuable to solving the associated problem. Heuristics are very helpful in the design of control actions, which may indirectly respond to the restructuring requirements. The conventional, performance-index-oriented design is no longer suitable here. Instead, a data-driven method incorporating heuristics may apply.

- **The complexity of knowledge sources needed for problem solving.**

  As a consequence of the feature mentioned above, dexterious knowledge would be required to support the decision making in selecting a proper restructuring action. This feature needs an open structure, in order to cope with different kinds of knowledge sources.

1.5 Implementation of Dynamic Restructuring Using Machine Intelligence

For the characteristics that have been discussed in the previous section, it is clear that, to implement dynamic restructuring technology in an automated process, human expertise and past experience would be extremely useful. Specifically, a knowledge-based system is desirable for problem solving. Therefore, in system design and implementation, one has to primarily consider formulation of the restructuring problem, represention of the problem solving knowledge, and the design of reasoning strategies [31].

Problem formulation is the first step toward machine intelligence [32]. Here, the problem should be represented in a form that would enable a computer to "understand"

it. Human beings can understand a language through semantics, but a computer cannot. Computers recognize syntax. Everything must be represented in a certain form which is recognizable to a computer. One such form of representation is propositional calculus [2], in particular, the first order predicate calculus. For example, $father(tom, david)$ is a predicate which means that "*tom* is *david*'s father". This is understood through the semantics of the word $father$ to a human being. A computer simply recognizes the symbol $father$, regardless of the meaning it holds. This is illustrated in Figure 1.4.

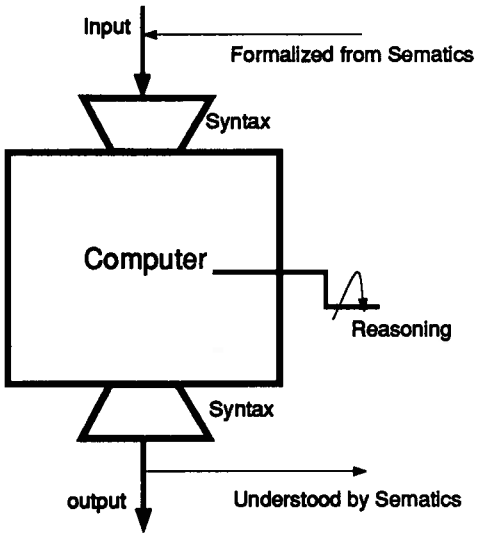

Figure 1.4: Semantics and Syntax in Machine Intelligence

Suppose that the facts

$$father(tom, david)$$

and

$$father(david, catherine),$$

and a definition:

$$grandfather(X, Y) \longleftarrow father(X, Z), father(Z, Y),$$

are given. Then if the query "who is *catherine*'s grandfather" is made, its logic consequence will be

$$grandfather(tom, catherine).$$

The predicate *grandfather* could be understood by human beings since it has an accepted meaning, but it is only a symbol to a computer which only recognizes the relation definition of *grandfather* and *father*.

A simple logic programming language dealing with operation of the first order predicates, which is to be used throughout the thesis, is explained in Appendix A.

Knowledge representation and reasoning are the core in system design and realization. Due to the complexity of the system, the problem solving knowledge is organized into a hierarchy. The programmatic knowledge and detailed knowledge are considered in different level, so as to implement hierarchical decision making. The agent then could not be tangled into unnecessary detail at first.

Some important conditions have to be satisfied in system design. They are usually met, but are explicitly discussed here to avoid confusion, with respect to subsequent design procedures.

- Full knowledge of actions that change the FPS is known. This will provide the possibility of planning the next step restructuring action, since each decision making should be made based on the current situation. In this manner, an agent will be able to completely predict the effects of an action.

- The agent has complete knowledge about the FPS. All the related and necessary information should be given to the system. This includes the assumable and askable knowledge, which, by indicating as askable, the agent may ask the user to provide.

Therefore, "negation as failure" could be applied to the restructuring system. For example, if there is no information about a component, it is assumed to be non existent.

1.6 Objectives of the Research

Dynamic restructuring is a new research area, with many aspects to be studied, from component sensing to final system integration. It is not possible to cover all the topics in this thesis. The major objectives of the research are as follows:

- To study and develop a methodology of dynamic restructuring of a flexible production system;
- To formulate and represent the restructuring problem and associated knowledge, and to obtain and abstract restructuring heuristics and rules on the basis of a fuzzy measure of FPS performance;
- To develop an architecture and reasoning strategies for problem solving;
- To design a fuzzy decision making procedure for the optimization problem associated with selecting proper actions in restructuring;
- To design and implement a simulation system for further research on this area, and to use it for the simulation of a realistic FPS.

Although many other related topics are mentioned and briefly discussed in the thesis for the sake of completeness of the design procedure, and also for providing a somewhat complete view of solving the problem, they are not considered major objectives of the research.

1.7 Summary

This chapter introduced the problem of dynamic restructuring of an FPS; and discussed the role it plays in a high level FPS automation system. The dynamic restructuring technology is actually developed to meet the needs of system flexibility, minimum inventory, and cost-effective operation.

Modularity, multitude working modes and mobility of components, together with the highly developed local-network communication techniques, provide the feasibility of FPS restructuring in a practical setting.

Restructuring problem is generally characterized as fuzzy, complex, non-analytic and heuristic-oriented. These features require the use of knowledge-based techniques and machine intelligence.

The remainder of the thesis is arranged as follows: Chapter 2 gives a theoretic framework for the problem solving method. Chapters 3 develops a blackboard architecture for the restructuring problem, giving particular attention to the knowledge representation and processing frameworks. Chapter 4 investigates the heuritics associated with the problem. In Chapter 5, the use of fuzzy logic is treated as a suitable approach in taking into account the incompleteness and non-analytic nature of the knowledge and associated decision making. Chapter 6 discusses some implementation problems of the approach and gives simulations of the application of the technique to an automated fish processing plant. Chapter 7 provides some concluding remarks on the research and proposes possible future work.

Chapter 2

A Theoretical Framework for Restructuring

In Chapter 1, the dynamic restructuring problem for a flexible production system (FPS) is proposed as a reorganizing system that will enable near-optimal or uniform operation of the FPS. This chapter will formulate the associated problem and discuss the methodology to achieve the goal.

Section 2.1 introduces the representation of system configuration; and the attributes of workcells and components. Section 2.2 formulates the restructuring problem. Then a short literature review is given in Section 2.3. Section 2.4 discusses the possibility of interpreting the problem of restructuring as conventional planning. Section 2.5 presents a method of knowledge based restructuring, which includes both load sharing and component releasing. Section 2.6 concludes the chapter.

2.1 Flexible Production Systems

From the point of view of restructuring (see Figure 1.3), a flexible production system (FPS) can be organized in a hierarchy that includes a system level, a workcell level and a component level. Attributes related to restructuring in each level are discussed and represented in this section.

2.1.1 FPS Configuration and Layout

Consider the multiple-workcell FPS that is schematically shown in Figure 2.1. The ith workcell is denoted by W_i and its jth component is denoted by c_j^i. For simplicity of

notation the subscript and superscript of this component notation is dropped in the sequel, except when they are explicitly needed.

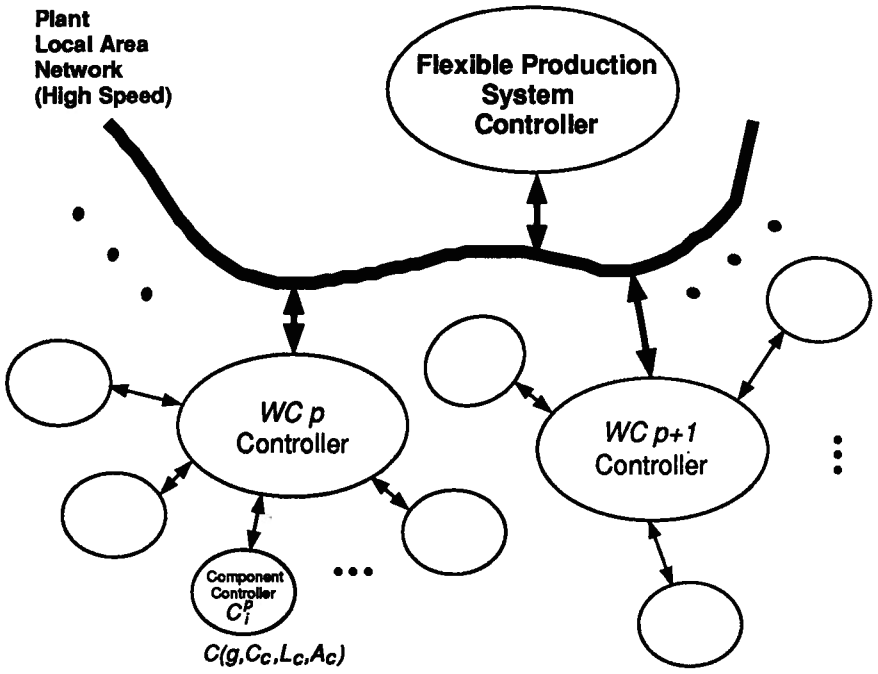

Figure 2.1: A Flexible Production System

Two attributes should be noted in the system level of an FPS: workcell configuration and layout. Configuration of workcell W can be expressed by

$$CON_W = \{c^W\} \tag{2.1}$$

Suppose that the task demand of a particular workcell is D_W. The corresponding workloads L_c of the individual components of this workcell may be determined using a standard task planning procedure, as expressed by the relationship

$$L_c = L_c(D_W) \tag{2.2}$$

Note that, if a component is shared between two workcells, whose status is denoted by c^{W_i,W_j}, then the component is considered as a component in both workcells W_i and W_j.

Geographic location of a workcell is also an important factor that affects restructuring. Usually, restructuring system favours in establishing component sharing between workcells that are conveniently located with respect to each other. Two coordinates for the position of workcell W on the workshop floor are denoted by X_W and Y_W.

2.1.2 Workcell Activity Level

Suppose that the processing capacity of a workcell component c is C_c. Also, at a given instant, suppose that the production level of a workcell is P_W and the activity level of component c of this workcell is A_c. Then the following relations will hold:

$$\begin{aligned} If \;\; & L_c \leq C_c \;\; \forall c \in W \\ then \;\; & A_c = L_c \end{aligned} \tag{2.3}$$

$$\begin{aligned} If \;\; & \exists c \in W \; such \; that \; L_c > C_c \\ then \;\; & A_c \leq L_c \\ and \;\; & P_W < D_W \end{aligned} \tag{2.4}$$

Here, W denotes the set of components in a particular workcell.

2.1.3 Component Capacity

The processing capacity of the components is the resource of the system that will be rearranged in restructuring in the FPS. Since components could be shared for carrying out different tasks, capacities of the components of the same type should be standardized so as to make the load transfer conveniently comparable. Using this standard component representation, load planning in equation (2.2) can be represented by referring to the standard component. As an example, a linear representation for load planning or capacity

requirement planning is illustrated as below:

Consider the calculation of the capacity requirement for image processing in a cutting workcell of a fish processing system. The workcell may consist of vision workstations, robots and automated guided vehicles (AGV), in addition to other necessary equipment. Given the cutting demand as x *units/sec*, in which the unit may be a processed item of fish; and the standard capacity requirements for processing a unit task, which are called planning factors, as follows:

standard vision: v sec/unit,

standard robot: r sec/unit,

standard AGV: a sec/unit,

we can calculate the capacity needed for the cutting process as:

standard vision: $vx \times 100\%$,

standard robot: $rx \times 100\%$,

standard AGV: $ax \times 100\%$.

For instance, if the speed of vision processing for a cutting task is 0.3 *sec/unit*, and the processing rate demanded for cutting is 2 *units/sec*, then the vision capacity needed (i.e. the load) is 60% of full capacity of a standard vision facility, say an IBM PC^{TM} based vision system with a SHARP GPB3^{TM} image processing board. Different vision systems may be expressed as a percentage referred to the IBM PC^{TM} system as the baseline unit. A Sun-SPARC^{TM} station based vision system with Datacube MaxVideo 20^{TM} image processing system (which requires a host computer with VME Bus), could be faster, and may be assigned a capacity factor 2, with respect to the baseline unit (IBM^{TM} PC-based). Then the above vision requirement applied to a Sun-SPARC^{TM} station based system corresponds to a load of 60%/2, or 30%, a significantly low percentage of its full capacity. In other words since the capacity of the more powerful vision system is

double that of the baseline system, the load is halved for the same task, which should be intuitively clear as well.

2.1.4 Component Workload Status

Next consider a group g of "similar" components which are shareable or interchangeable within the group. The characteristics of a component c may be represented by the ordered set $[g, C_c, L_c, A_c]$, as shown in Figure 2.1. Here C_c L_c A_c are the capacity, workload and activity level of component c respectively. Suppose that, at a given instant of workcell operation, the workcell components are monitored and some are determined to operate below capacity (under-capacity) while some others are overloaded. Then, the following conditions hold:

$$If \quad c \in g \ and \ L_c < C_c \ then \ c \in g_u \tag{2.5}$$

$$If \quad c \in g \ and \ L_c > C_c \ then \ c \in g_o \tag{2.6}$$

with

$$g_u \wedge g = g_u \quad and \quad g_o \wedge g = g_o \tag{2.7}$$

where g_u= subgroup which contains all components that operate below capacity (under-capacity) within the component group g; g_o= subgroup which contains all components that are overloaded within g.

Note that, in general, $g_u \vee g_o \subseteq g$ because for some components within g, it is possible that $L_c = C_c = A_c$. Also, even when $L_c = C_c$ for a particular component, it is possible that $A_c < C_c$ for that component as a result of gross reduction of the production level P_W of the workcell that contains the component, due to an overload in some other component within the same workcell [12].

2.2 Formulation of Restructuring Problem

The restructuring problem may be formulated as one of parameter optimization. For this purpose, a performance index has to be defined.

2.2.1 Performance Index

The optimal restructuring problem may be expressed as

$$minimize \quad J = J(\underline{\alpha}, \underline{C}, \underline{A}, \underline{L}) \tag{2.8}$$

$$subject\ to \quad P_w(\underline{A}) = D_w(\underline{L}) \quad \forall\, w \in FPS \tag{2.9}$$

where $\underline{\alpha}$ is a vector of weighting parameters.

For example, a quadratic cost function could be used; thus

$$J = \sum_c \alpha_c (C_c - A_c)^2 \tag{2.10}$$

with nonsymmetric weighting parameters such that

$$\alpha_c = \alpha_{co} \; if \; L_c > C_c \tag{2.11}$$

$$\alpha_c = \alpha_{cu} \; if \; L_c \leq C_c \tag{2.12}$$

$$\alpha_c = 0 \quad if \; L_c = 0 \tag{2.13}$$

It should be noted that no activity (zero activity) is considered a desirable condition. In other words, rather than having, say, two undercapacity components of the same type, it is desirable to transfer the load of one component into the other and completely release the first component. This is advantageous for reasons such as reduction in operating costs and wear and tear. Furthermore, the component that is released of activity in this

manner will be available for absorbing overloads from other similar components in the FPS. It is clear that, zero-activity components do not contribute to the cost function J, as expressed by equation (2.10). Accordingly, the optimization strategy inherently favours the existence of zero-activity components.

2.2.2 Restructuring Requirements for Knowledge Based System

Due to the presence of fuzziness, as discussed in Chapter 1, the cost function given in Section 2.2.1 provides only a guideline for restructuring and a rough evaluation of the restructuring performance, but shouldn't be considered as the entire goal of restructuring. There are still many other factors that should be considered, such as, sharing penalty, sharing feasibility, the geographical possibility of sharing, and so on. As an example, sharing within a workcell is usually considered better than sharing between workcells. Because of these reasons, in view of fuzziness of a restructuring goal, heuristics from experienced operators could be considered as well in order to complete the requirements for modeling a restructuring problem. This may provide an easy way to design restructuring strategies using past experience. In this context, some requirements that have to be satisfied by a restructuring procedure may be stated.

These requirements are itemized below:

1. Overloads are not allowed. The load limitation of a component is its full (design) capacity. A main purpose of a restructuring controller may be considered as the elimination (or reduction) of overloads.

2. Reduce the operating cost of an FPS by releasing components which are greatly under their full capacity. (For employees, hiring and firing may be considered in this context). The judgement associated with problem is fuzzy in general. There

might not exist a crisp standard to judge whether a component should be released. Therefore, fuzzy decision making will be needed here.

3. Maintain the number of sharings (both load sharing and component sharing) as low as possible, for the system reliability may weaken with increased activity of sharing.

4. Selection of components for sharing or releasing cannot be done arbitrarily. Some evaluation should be made in choosing an action for sharing or releasing.

5. Use any available past knowledge and experience, which may be helpful in improving the knowledge base.

Thus, restructuring of components in a multiple-workcell FPS should not be carried out using purely analytical optimization criteria alone. Certain factors or conditions may favour some structures while objecting to some others. The factors that determine the desirability of a particular structure of component sharing may be based on non-analytical criteria such as what has been mentioned above. Under such conditions, a knowledge-based approach to making the restructuring decisions would be desirable.

2.3 Literature Review on Background Material

In this section, we will present a literature survey of the background work of the present research.

2.3.1 Dynamic Restructuring

Dynamic restructuring of flexible production system originated in the work of de Silva (1992) [8]. It was proposed as an important concept of soft automation [9]. A framework for the problem solving is presented in de Silva (1993b) [10]. Knowledge based system is

developed in Gu an de Silva (1994a, b, c) [20, 21, 22] and in de Silva and Gu (1994) [12]. The problem solving architecture was established in Gu and de Silva (1994a) [20] and the heuristics and conflict resolution method are developed in Gu and de Silva (1994b, c) [21, 22]. An analytical framework is presented and a new evaluation method for actions, using priority fuzzy sets, was introduced in de Silva and Gu (1994) [12]. Also, in that paper, the concept of restructuring-feasibility index was formalized and utilized for the optimization of restructuring.

2.3.2 Knowledge-Based Design of Flexible Systems

There are three major areas of research related to this work: 1. Integrated system design; 2. Decision support systems; 3. Expert control systems. Pertinent work in these three areas is outlined in this section.

Computer Integrated Technology

The main research activities in production automation are concentrated in integrated mechanical design; planning and scheduling of flexible production systems; and decision support systems for plant design.

Leong *et al.* (1991) [30] described an integrated knowledge-based system for mechanical design using a blackboard architecture. In solving a typical mechanical design problem, they viewed the design process as being composed of a set of tasks that can be progressively subdivided into a hierarchy of smaller tasks in the form of a tree structure, (see Figure 2.2). These tasks are solved by a team of specialists using a distributed problem-solving approach.

Two kinds of communication were proposed for the hierarchy; specifically, broadcasting using a blackboard, and message passing directly between two members in the same team. In Figure 2.2, the detail mechanical design is composed of coupling design,

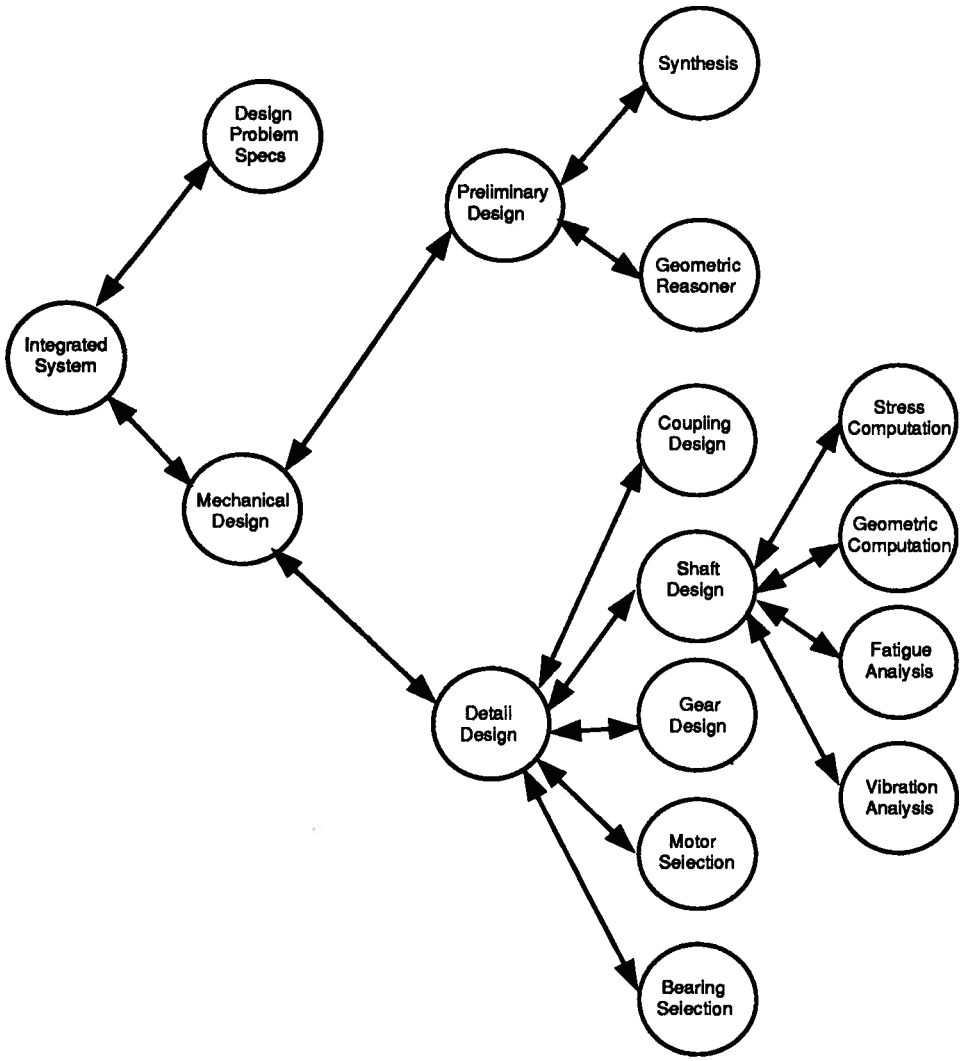

Figure 2.2: An Integrated System for Mechanical Design

shaft design, gear design, motor selection, bearing selection, etc. There are natural links between these various knowledge sources. Leong *et al* designed a horizontal communication structure for this system, which may be necessary but makes the entire blackboard system not so modular and opportunistic reasoning.

The planning and scheduling system for a flexible manufacturing workcell (FMC), described by Kovacs and Mezgar [29] is closer to the problem in the present research. In Kovacs (1991) [29], designed FMC configurations, layouts and schedules are put into modules of simulation and analysis for testing. System will return to carry out reconfiguration if the design is not satisfactory. The design procedure is illustrated in Figure 2.3, in which several modules have been identified. They are described below.

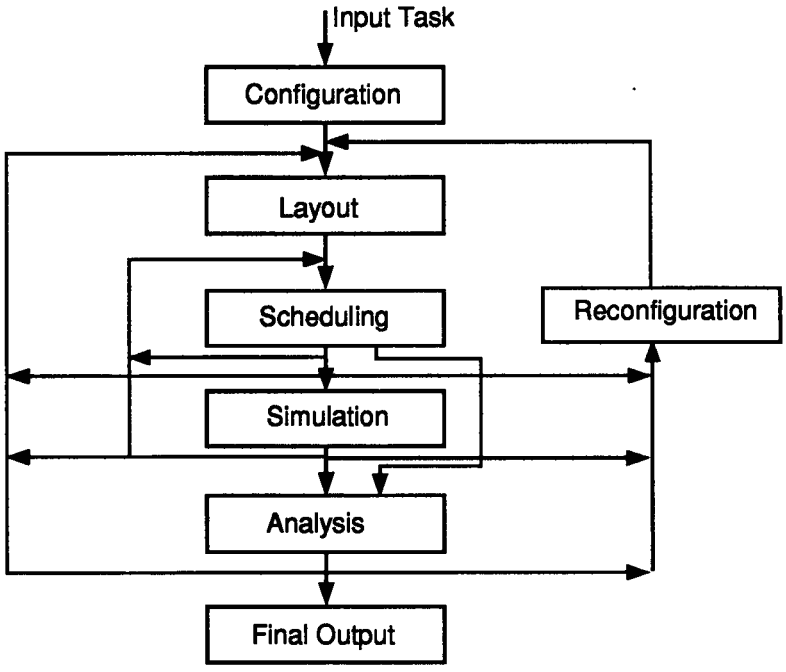

Figure 2.3: A Reconfiguration and Scheduling System for an FMC

- **Configuration module**

The cell configuration design has two separate steps, which can be carried out by

iteration: the first step is to get a pre- or basic configuration which is an "average" of several possible configurations, and the second step is the reconfiguration from this average configuration according to the actual production tasks and manufacturing situations.

Reconfiguration may be necessary in certain situations, for example,

- During normal operation, better adaptation to changing workloads can be achieved by rearrangement of system elements, or regrouping of some of them.
- In the case of component failures, the operation has to be continued without using the defective element, and with a minimal loss of production.
- New elements can be added to the system, and old elements can be disconnected, for the purpose of system maintenance and upgrade.

Flexibility in design and operation can be guaranteed under these circumstances through the capability of configuration and reconfiguration.

- **Layout planning module**

  In this module the relative geographic positions of the manufacturing machines within the workcell are specified, while the useful aspects of the material-handling system are taken into consideration.

- **Scheduling module**

  Once the configuration and the layout of the system are designed, the processing time needed for each item that is produced will be known. Consequently, a task can be scheduled. If the schedule is unsatisfactory, then modification to machines and transportation procedure, layout changes and reconfiguration can be considered to

improve the situation. The accepted schedule is forwarded to both the simulation and the analysis modules.

Another configuration example is described by Iizuka and Tsuji (1988) [23]. This is an expert system for the design of a computer system configuration. It focuses on the following two points: 1. Discussions on the effectiveness of Prolog for representation of the interconnections between various pieces of equipment; for representation of the system components; and for implementation of the functions of knowledge base retrieval, and 2. System structure which emphasizes on improved maintenance of expert system. The paper gives a good example of using Prolog to represent knowledge, especially that of interrelationship between components, and also describes the implementation of an inferencing mechanism.

Decision support system

Decision making is traditionally supported by operations research techniques. Its basic purpose is to predict future events and choose an optimal one according to some performance index and constraints.

The knowledge to support a decision, based on real-time data, must be able to provide answers to the following questions [3]:

- "What happens?": Ability to analyse a situation and identify advantageous and problematic aspects.
- "What can happen?": Ability to reason on the evolution of the system. It might be possible to identify the possible descriptive patterns of the situation in future times.

- "What should be done?": Ability to reason on the control actions most convenient for improving the system operation.

Fayyad and Kass (1991) [17] described a part of the PAVE (Plant Assembly Verification Environment). PAVE is intended to support industrial engineers during the design process and help manage the complexity of the problems which they encounter.

Expert control system

Usually a decision making system is an open loop system. Given a situation and a task, it makes a decision. In a feedback control system, the system situation is updated automatically and a control decision is made periodically. Due to system delays (inertia delay and pure delay), these decisions will interact with each other and cause the system behave as a dynamic system response. Many authors have discussed expert control. Some emphasize on the representation of system situation, such as system error characteristics[48],and fuzzy representation of system error [6]. Some emphasize on system structure and parameter knowledge, such as self-tuning expert system [1]. Some others emphasize on control policy, such as, robust control, wait and see control etc.[34]. Information feedback is useful in any type of control system.

Although dynamic restructuring is a new technique in automation, its development is more or less related to the above mentioned areas. Its application domain is associated with flexible production systems; and its technology is based on knowledge engineering which incorporates decision making techniques and information feedback.

2.4 Restructuring as Planning

A naive method of restructuring is to represent it as a simple planning procedure [18, 38] of restructuring actions. Basically, the central activity of restructuring planning will

require the following:

- An initial world (here the term "world" is used to denote a certain FPS situation) in which overloads may exist;
- A restructuring goal (fuzzy) that typically states "There should be no component overloads and components should operate near their design capacity,with minimal cost and near the specified production demand "; and
- A set of actions which includes establishing or terminating a load sharing process and releasing of an undercapacity component, or moving of a component from one workcell to another.

The planning problem then is to find a sequence of actions which can transfer an initial world into one in which the desired goal has been achieved [38]. Every action changes the world to some extent, and eventually, by completing the entire set of actions, the system will reach the final world, in which no component overloads or capacity shortages would exist. If such a final world does not exist after all the actions have been tried, then the restructuring planning would have failed.

2.4.1 State Space Method

One popular method of representing planning knowledge is the state space representation. The concept is illustrated in Figure 2.4. In a state space, each node represents a list of properties of a world; in short, every node is a certain world. An arc corresponds to an action which changes the status of a world. Each action provides a complete specification of how to obtain the new world from the world where this action was originated; for example,

$$move(robot1, from(canning), to(cutting)) : world1 \Rightarrow world2$$

where, $world1$ and $world2$ are expressed as,

$$world1 : \left\{ \begin{array}{l} holds(robot1, position, canning, world1); \\ \ldots \ \ldots; \\ (other\ properties) \\ \ldots \ \ldots \end{array} \right\}$$

$$world2 : \left\{ \begin{array}{l} holds(robot1, position, cutting, world2); \\ \ldots \ \ldots; \\ (other\ properties) \\ \ldots \ \ldots \end{array} \right\}$$

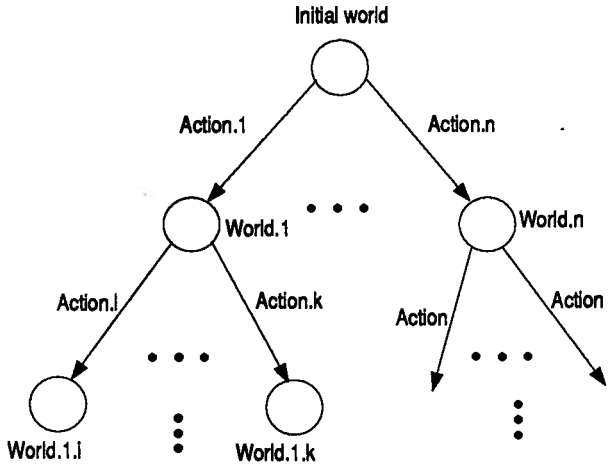

Figure 2.4: The State-Space Representation of Planning

The predicate $move(robot1, from(canning), to(cutting))$ corresponds to an action which moves $robot1$ from the *canning* workcell to the *cutting* workcell. Assuming that the effect of this action is only to change the position of $robot1$, then the new world denoted by $world2$ will keep all properties of $world1$ except for the position of $robot1$. The properties are represented by the predicate $hold(O, P, V, World)$ which has the intended interpretation that object O holds the property P at value V in world $World$.

Reasoning through a state space is simple. A search can be made starting from the initial world to a final world in which desired goal is true. A plan is just the list of actions on the path that reaches the final world (desired goal).

The major problem with the state space representation is that everything about the world must be explicitly listed, which will require a long list of properties. Even more serious, in the fact that, every effect of an action must be anticipated and explicitly listed, in this approach.

Suppose that we have n groups of components which could be rearranged in an FPS, and that each group has the same number (m) components. Then the total number of sharing or releasing pairs of components is given by $n \cdot C_2^m = n \cdot m(m-1)/2$. The number of graph nodes (ignoring other restructuring actions such as moving a component) is given by $[n \cdot m(m-1)/2]^d$ approximately, where d is the number of stages of action in a plan.

Consider a small system with $n = 3$ (e.g., robot, vision unit, AGV), $m = 5$ (i.e., 5 components in each group), and $d = 10$ (i.e., 10 actions in a plan), the size (in number of nodes) of the search space for breadth first or A^* search is of the order of $[n \cdot m(m-1)/2]^d = (\frac{3 \times 5 \times 4}{2})^{10} = 6 \times 10^{14}$. It is therefore almost impossible to explicitly list all the situations and effects of actions in a graph of this size.

2.4.2 Action Calculus

Now consider reformulating the problem in order to deal with the complexity of the state space representation for planning. We represent the world in the form of either *init* (the initial world) or $do(A, W')$; which means a new world that is derived by applying the action A on world W'. This is called action calculus.

Therefore, the planning problem is to find a world W in which the desired goal is satisfied. A typical goal for the restructuring problem may be represented as,

?- $\{holds(vision1, work\_load, 100\%, W);$

$\cdots\ \cdots,$

$(for\ all\ components),$

$\cdots\ \cdots,$

$\{holds(robot5, work\_load, 100\%, W);$

where all the component loads are required to be at 100% of its full capacity.

Two kinds of knowledge are required in the reasoning process that achieves the goal: the specifications of the initial world and the effects of all actions on various worlds. A complete specification of each action will, in general, require at least one axiom or postulate for each relation of the domain (i.e., each property). There are $n{\cdot}m$ components in the FPS, and suppose that each component has the six properties (*group*, *position*, *work_for*, *capacity*, *work_load*, *shareability*). Hence the total number of domain relations is $6{\cdot}n{\cdot}m$. The number of actions may be estimated as $n{\cdot}m(m-1)/2$. Accordingly, we will need at least $6nm \cdot nm(m-1)/2$ axioms, because all the relations have to be stated for each action. Clearly, it is a difficult job to specify all these axioms, but it is better than the state space method, where it is required to specify everything (state and action) for every world.

Goal regression[38] may be employed as the reasoning procedure for system restructuring. In an actual situation, each action affects only a very small part of the world. Much of the world will remain unchanged, which means most of the axioms will just state that the properties in the new world are the same as those of the old world [39]. In this thesis, a blackboard-based planning method will be developed to cope with the practical problems associated with an extensive action space. It avoids the problem of large dimensionality by assuming that if a property is not changed by an action, it remains unchanged in the new world. In this manner, we will need only to specify the effects of

the actions. Heuristics play a very important role in that method.

2.5 Knowledge-Based Restructuring

One may observe that the goal represented for planning the restructuring process, as given in the last section, is somewhat artificial. The problem is that the actual goal is fuzzy, and it cannot be represented by a crisp description of component loads, for example, 90% of full capacity. Furthermore, expertise and past experience should be used to decide the search direction rather than employing a non-heuristic search. A more realistic framework for knowledge-based restructuring is developed in this section.

Consider a group g of shareable or interchangeable components which are associated with a set of workcells $\underline{w}$ in a flexible production system (FPS). Each workcell in this set has an associated set of components $\underline{c} \in g$. Then the knowledge-based restructuring process may be expressed as

$$g(\underline{w}^*(\underline{c}^*)) = R(g) \otimes g(\underline{w}(\underline{c})) \qquad \forall\, g \in\ FPS \tag{2.14}$$

Here $R(g)$ may be interpreted as a knowledge system which evaluates the present association of components $\underline{c}$ with workcells $\underline{w}$, within a group g and then modifies these associations appropriately so as to reduce component overloading, according to some performance criterion. The resulting new association of components $\underline{c}^*$ within g, now with a set of modified workcells $\underline{w}^*$, is denoted by $g(\underline{w}^*(\underline{c}^*))$. Note that the flexible production system is said to be restructured when the configuration of the workcells changes due to such a modified association of components. The decision making process given by equation (2.14) has to be executed for all groups of component within the FPS.

Basically, the changes in the association of components constitute two types of actions: A component joining a workcell, or a component leaving a workcell, which will be discussed in detail in the remainder of this section.

2.5.1 Workload Sharing

Again, consider a group of shareable or interchangeable component g with m overloaded components represented by the subgroup g_o and r undercapacity components represented by the subgroup g_u.

In knowledge based restructuring, each overloaded component (OC) has to be matched with an appropriate undercapacity component (UC) for load sharing. This may be accomplished using the rulebase given in Figure 2.5. Note that a rule represented here has the format:

IF *situation* **THEN** *action* **WITH** *bf*

The *situation* is the current status (or context) of the components, which could be represented by a fuzzy descriptor. Therefore, the context of situation could be represented in a Cartesian product space with dimensions equal in number to the context levels of components. The *action* is a crisp action set, where each composed proposition in the situation context may lead to one or more actions. Each implication from a situation to an action is assigned a belief value bf, which is a representation of the corresponding to human confidence on the particular rule.

A rulebase is a mapping from a situation space S to an action space A.

$$R : S \Longrightarrow A \tag{2.15}$$

In particular, $R(a)$ represents a sub-rulebase of R associated to a specific action a.

Consider the rule base given in Figure 2.5 for component matching with the objective of load sharing. The states that a component may take are given by the set {no activity, undercapacity, balanced, overloaded}. Even though, the condition of "no activity" (na) may be treated as a special case of "highly undercapacity" (hu), it is desirable to define a separate state for this condition, for example, as in the rulebase given in Figure 2.5, for

reasons that should be clear later, in the context of component releasing. In particular, the state of *na* is far more desirable than that of *hu*. A balanced state is said to exist in a component when its activity is equal to its capacity (i.e., $A_c = C_c$). This is the optimal state of component operation. The membership functions of the fuzzy resolution levels for the two context variables *OC* and *UC* should be known in using fuzzy logic to represent and process [7] the knowledge base given in Figure 2.5.

Context Levels	
Overloaded Component *OC*	Undercapacity Component *UC*
	No activity *na*
Highly overloaded *ho*	Highly undercapacity *hu*
Moderately overloaded *mo*	Moderately undercapacity *mu*
Lightly overloaded *lo*	Lightly undercapacity *lu*

If	*OC* **is** *ho*	**and**	*UC* **is** *na*	**then**	sharing(OC, UC)	**with**	$bf = 1.0$.
If	*OC* **is** *ho*	**and**	*UC* **is** *hu*	**then**	sharing(OC, UC)	**with**	$bf = 1.0$.
If	*OC* **is** *ho*	**and**	*UC* **is** *mu*	**then**	sharing(OC, UC)	**with**	$bf = 0.8$.
If	*OC* **is** *ho*	**and**	*UC* **is** *lu*	**then**	sharing(OC, UC)	**with**	$bf = 0.6$.
If	*OC* **is** *mo*	**and**	*UC* **is** *na*	**then**	sharing(OC, UC)	**with**	$bf = 0.8$.
If	*OC* **is** *mo*	**and**	*UC* **is** *hu*	**then**	sharing(OC, UC)	**with**	$bf = 0.8$.
If	*OC* **is** *mo*	**and**	*UC* **is** *mu*	**then**	sharing(OC, UC)	**with**	$bf = 0.6$.
If	*OC* **is** *mo*	**and**	*UC* **is** *lu*	**then**	sharing(OC, UC)	**with**	$bf = 0.6$.
If	*OC* **is** *lo*	**and**	*UC* **is** *na*	**then**	sharing(OC, UC)	**with**	$bf = 0.4$.
If	*OC* **is** *lo*	**and**	*UC* **is** *hu*	**then**	sharing(OC, UC)	**with**	$bf = 0.4$.
If	*OC* **is** *lo*	**and**	*UC* **is** *mu*	**then**	sharing(OC, UC)	**with**	$bf = 0.6$.
If	*OC* **is** *lo*	**and**	*UC* **is** *lu*	**then**	sharing(OC, UC)	**with**	$bf = 0.6$.

Figure 2.5: The Rulebase of Component Matching for Load Sharing

In evaluating the load sharing decisions, the factors considered in the rulebase of

Figure 2.5 are the level of overload $L_c - A_c$ of the overloaded components and the level of undercapacity $C_c - A_c$ of the undercapacity components. This information is "crisp" as obtained from sensory data of the workcell components, and has to be fuzzified using the corresponding membership functions.

A reasoning procedure [22] using the techniques of fuzzy associative memory (FAM) [27] is designed in prioritizing an action by considering the membership grade values of the situations as well as the belief values of the fuzzy logic rules. The procedure is illustrated in Figure 2.6.

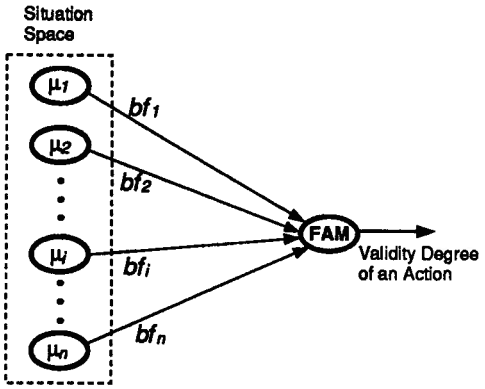

Figure 2.6: The Fuzzy Associative Memory

The priority value of an action is then expressed by:

$$PV(a) = fam(R(a))|OC, UC. \tag{2.16}$$

It follows that, the priority value PV of an action a is a function of a sub-rulebase $R(a)$ associated with action a under the condition of the fuzzy status of OC and UC of two sharable components.

Apart from the priority as evaluated from the overload level $L_{OC} - A_{OC}$ and the undercapacity level $C_{UC} - A_{UC}$ along with the rule beliefs, the feasibility of load sharing should also be considered. The feasibility may be expressed as a matrix; thus

$$F = [f_{ij}]_{m \times r} \tag{2.17}$$

The priority value for a matching pair of components being considered for load sharing, has to be multiplied by the corresponding feasibility index as given by equation (2.17) in order to arrive at a total assessment of the load sharing decision. The feasibility values have to be suitably updated following each decision of load sharing.

Finally, the action with the highest assessment value based on the current system situation will be selected as the correct decision for restructuring purpose.

2.5.2 Component Releasing

As noted before, an undercapacity component may be released by transferring its load to another undercapacity component of the same type to save operation cost. But, an undercapacity component need not be released in every load transfer process between two undercapacity components. For example, a lightly undercapacity (lu) component might be converted into either a moderately undercapacity (mu) component or a highly undercapacity (hu) component in this cycle, and would be ready for sharing with an overloaded component (regardless of whether ho, mo, or lo) in the next planning cycle of restructuring.

The rule-based approach to load transfer between two undercapacity components is somewhat similar to that between an overloaded component and an undercapacity component. But the rulebase that is used would be quite different from what is given in Figure 2.5. Consider an undercapacity component (UR) that is expected to be released of its load and a second undercapacity component (UL) that is expected to receive the load of the first component. An appropriate rulebase for determining the priority of component releasing actions is given in Figure 2.7.

It should be noted that the no-activity components should not be involved here, since it will not benefit the system operation. Specifically, an inverse action in a subsequent step could make the decision system cycle in an endless loop.

Context Levels	
Releasing Undercapacity Component UR	Loading Undercapacity Component UL
Highly undercapacity *hu*	Highly undercapacity *hu*
Moderately undercapacity *mu*	Moderately undercapacity *mu*
Lightly undercapacity *lu*	Lightly undercapacity *lu*

If	UR **is** *hu*	**and**	UL **is** *hu*	**then**	*releasing*(UR, UL)	**with**	$bf = 1.0$.
If	UR **is** *hu*	**and**	UL **is** *mu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.8$.
If	UR **is** *hu*	**and**	UL **is** *lu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.6$.
If	UR **is** *mu*	**and**	UL **is** *hu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.6$.
If	UR **is** *mu*	**and**	UL **is** *mu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.4$.
If	UR **is** *mu*	**and**	UL **is** *lu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.4$.
If	UR **is** *lu*	**and**	UL **is** *hu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.4$.
If	UR **is** *lu*	**and**	UL **is** *mu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.4$.
If	UR **is** *lu*	**and**	UL **is** *lu*	**then**	*releasing*(UR, UL)	**with**	$bf = 0.4$.

Figure 2.7: The Rulebase of Component Matching for Component Releasing

Here too, a feasibility matrix of load transfer is used, and the application of the fuzzy-associative memory (FAM) reasoning procedure would be given by a counterpart of equation (2.16), as

$$PV(a) = fam(R(a))|UR, UL \tag{2.18}$$

In the same manner as for load sharing, the decision maker in this case picks an action with the highest assessment value.

2.6 Summary

This chapter presented a theoretical framework for restructuring a flexible production system (FPS). Basic properties of an FPS were discussed first. In particular, the status of a component is determined by the load (or capacity requirement) which is given by the load planner based on task demands, and the available capacity of the component. Usually, the component status is represented by a fuzzy descriptor, Next the restructuring problem was formulated. The restructuring performance index, or the goal, was studied. A quadratic cost function could be used for evaluating the restructuring results, but for knowledge based system design, expertise and past experience would be much more helpful, in order to cope with the fuzzy performance requirements.

A brief literature review was given. Since the dynamic restructuring is a new research area, some related work was also presented.

The basic idea of restructuring is the planning of actions. Due to the complexity and fuzziness of the system, conventional planning methods do not apply in general. Heuristic-based restructuring was suggested and presented. However, in a practical design, the knowledge required to complete a restructuring process is complex and found in various forms. Knowledge representation and reasoning will be the main topics that

will be addressed in the following three chapters.

Chapter 3

Problem Solving Architecture

The concept and a theoretic framework of dynamic restructuring have been discussed in chapters 1 and 2. In this chapter, a problem solving architecture will be introduced and applied to the dynamic restructuring problem.

In Section 3.1, the complexity and variety of knowledge sources required for FPS restructuring are discussed. A hierarchy is considered to organize knowledge of different levels of complexity and resolution. Knowledge sources which may be represented in a variety of forms and in need of different reasoning strategies, will be organized in an open architecture,using a blackboard model. Section 3.2 presents the architecture of the blackboard model. In Section 3.3, the restructuring system is organized into the blackboard structure. Some specific knowledge sources are discussed in Section 3.4.Some concluding remarks are made in Section 3.5 of the chapter.

3.1 Knowledge Organization

Knowledge associated with restructuring of a flexible production system (FPS) deals with various aspects and may be treated at different levels. In particular, the following knowledge sources are needed:

- Knowledge about load planning, which determines the component loads corresponding to a given task demand. The component capacity requirements will be determined according to appropriate relations between the task demands and component

loads.

- The operating data about the FPS which is an important source of information for decision making. Specifically the following information should be updated:
  - component type and capacity;
  - component shareability;
  - information associated with load sharing feasibility, such as the FPS layout;
  - current component load, and the component associations with workcells.
- Knowledge needed for information pre-processing. This knowledge is needed to evaluate the FPS data and determine values for component status and sharing feasibility index, for the the decision making associated with system restructuring. Component status is evaluated by comparing the required load and the available capacity of a component. Also the updating of the index of sharing feasibility is usually a knowledge-based process.
- Knowledge about the restructuring decision itself. This will include heuristics in different levels of detail and resolution. Detailed selection of sharing pairs of or releasing pairs of components should not be made before deciding whether a sharing or releasing action is actually needed. The decision as a whole should be made after the current FPS situation has been correctly recognized. In general, in view of the complexity and variety of knowledge sources and due to the presence of different levels of knowledge, a well organized structure would be strongly recommended.

3.1.1 Organization of Knowledge in a Hierarchy

In the restructuring system, knowledge of different levels is organized in a hierarchy as shown in Figure 3.1.

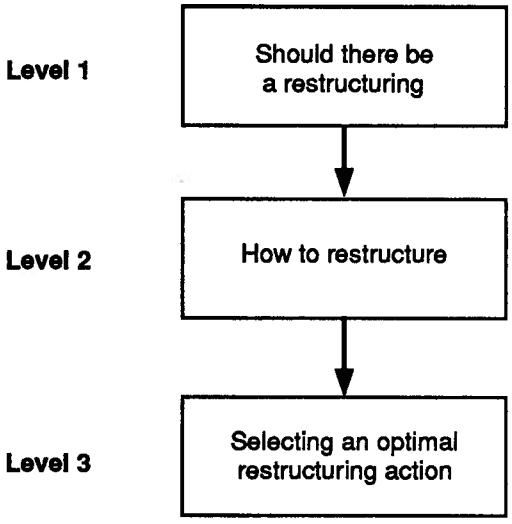

Figure 3.1: The Hierarchy for Organization of the Restructuring Knowledge

Level 1 decides whether there should be a restructuring action. The FPS situation, which could be changed by the restructuring actions, should be updated and recognized before a decision is made.

Level 2 decides how to restructure the system. Some general heuristics about load sharing, component releasing, and some other restructuring actions are vital in restructuring planning, since the search space is extensive by large. Without the heuristics, it may not be practical to find a solution. Particularly, in a data driven system, such as an FPS restructuring system, search space could be narrowed by using heuristics for optimization.

Level 3, which is the lowest level in a restructuring system, deals with more precise information. A more accurate method can be used to decide the optimal restructuring action under the guidance of level 2. It is still a fuzzy decision procedure, however, since the restructuring system as a whole is a high level one in an automation system of the type shown in Figure 1.3. Level 1 is developed in this chapter. Level 2 and Level 3 will be discussed in more detail in the next two chapters.

3.1.2 Organization of Different Types of Knowledge Sources

Since the level 1 uses different knowledge sources for deciding whether to carry out a restructuring action, it may be organized into a blackboard [16, 25] architecture. This architecture consists of a global (shared) data region called blackboard (BB), several knowledge sources (KS) or intelligent modules that interact with this data region, and a control unit. The knowledge sources are not arranged in a hierarchical manner and will cooperate as equal partners (specialists) in making a knowledge-based decision. The knowledge sources interact with the shared data region under the supervision of the control unit. When the data in the blackboard changes, which corresponds to a change in context, the knowledge sources would be triggered by that in an opportunistic manner and an appropriate decision would be made. That could result in further changes to the blackboard data and subsequent triggering of other knowledge sources. Note that the data may be changed by external means (for example, through a user interface) as well as due to knowledge-sources actions.

The decision making process of the restructuring problem that was presented in the previous sections is essentially an opportunistic one where an appropriate knowledge source would be acting at a given time in carrying out the most appropriate action (e.g., information updating, load planning, restructuring decision making). It follows that a blackboard model would be suitable for the implementation of the associated knowledge based system. Furthermore, the following charcteristics of this level of decision making should be noted:

- The knowledge sources may take different forms. They may include rule based decision making; planning procedures; information processing; or even look-up-table updating. In a blackboard architecture, the knowledge sources can be implemented as independent modules, so as to employ different reasoning strategies.

- The practical control mechanism of this level can be quite complex. For example, after the system makes a sharing decision, the control may continue with another decision if an overload remains; or may determine component loads if task demands changed; or may establish the system activity level if some components become disabled. The blackboard architecture provides a mechanism of opportunistic control, which can carry out such steps in the present application.

- The FPS information should be accessible by every knowledge source, in which the integrated data of the overall operation of the FPS is needed for the reasoning process. Such data may be posted on a blackboard which would be visible to all knowledge sources.

3.2 Blackboard Model

Blackboard model is known as a powerful achitecture for constructing knowledge based systems. As a model, blackboard system is basically modular, and employs opportunistically reasoning. As a result, it provides the dexterity and flexibility to deal with diverse kinds of knowledge, and different inferencing mechanisms. Based on this model (or concept), one can establish a customized blackboard framework, although there are some common and general consideration. In order to understand the various blackboard structures, it is helpful to trace the intellectual history of blackboard concepts.

3.2.1 Blackboard System History

"Blackboard" as a technical term used in the AI literature first appeared in Newell (1962) [33]. Newell concerned with the organizational problems of programs that existed at the time. He tried to organize subroutines independently and in a hierarchical manner. Even though this organization had many advantages, it also had difficulties: First, it was

found difficult to communicate between subroutines; Second, the ordered subroutine calls fostered the need for doing things sequentially; Third, the total program was organized to do only one thing at a time. For these difficulties, it was noted [33] that they "might be alleviated by maintaining the isolation of routines, but allowing all the subroutines to make use of a common data structure". This should be the rudimentary idea of a blackboard system.

The first real blackboard system is known to be Hearsay project [16] which was designed and implemented for speech understanding. This blackboard structure was proposed by H. Simon (1977) [41]. Before that, Simon published a paper [16] in which he mentioned the term "blackboard" in a slightly different context from Newell. There his focus was on the problem-solving method: In the typical organization of a problem-solving program, the solution effort is guided and controlled by goals and subgoals [41]. Together with Newell's common data structure, we can visualize a prototype framework for a blackboard system; which consists of a common data structure and employs hierarchical problem solving.

In the design of Hearsay [16], such system characteristics as hierarchically organized data levels and opportunistic reasoning, which we now accept as integral parts of a blackboard system, were derived from needs and constraints that were different from those of Newell and Simon. (Accordingly, the structure of a blackboard system may be modified slightly in order to meet different needs and constraints).

Presently, BB techniques are widely used for problem solving [25, 35, 36] and will be increasingly popular in the future because of their attractive characteristics.

3.2.2 Blackboard System Structure

Now we consider the structure of a blackboard system at the model level. The blackboard model is a conceptual, high-level organization of information and knowledge which are

needed to solve a problem. It is a general prescription for dynamic control of a system, and for the use of knowledge for incremental, opportunistic problem solving. Organizationally, the blackboard model consists of three components.

The knowledge sources (KS). The knowledge needed to solve the problem is partitioned into *knowledge sources*, which are kept separate and independent.

The blackboard data structure. The data on the system state which are needed for problem-solving (represented as objects in the solution space) are kept in a global data store known as the *blackboard.* Knowledge sources can activate changes in the blackboard, which will lead incrementally to a solution to the problem. Communication and interaction among the knowledge sources take place solely through the blackboard.

Controller. What knowledge source should be applied, when, and to what part of the blackboard, are problems handled by the blackboard controller.

The reasoning process associated with a blackboard has the following characteristics: The solution to a problem is progressed one step at a time. At each control cycle any type of reasoning step (data driven, goal driven, forward chaining, backward chaining, etc.) can be used. The selection and the application of knowledge sources are dynamic and opportunistic rather than fixed and preprogrammed.

The data objects on the blackboard are organized into hierarchical levels. Different data associated with different concepts can be made distinct and organized. Kowledge sources can be made to span two levels, with the information on one level serving as input and the information on another level as the output. Such a system is illustrated in Figure 3.2.

The control unit also gets information from the knowledge source (KSs), because a KS consists of not only the domain knowledge (for problem-solving), but also the knowledge of using that knowledge.

Therefore, a blackboard system is quite suitable for handling restructuring problems

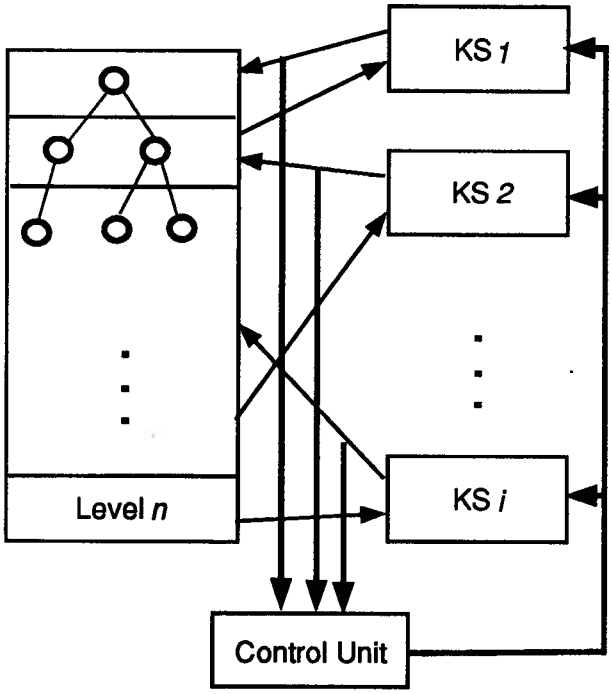

Figure 3.2: A Model of a Blackboard System

as in the present research, as it has some beneficial characteristics for handling systems of: 1. large solution spaces; 2. noisy and unreliable data; 3. a variety of input data and a need to integrate diverse information; 4. many independent pieces of knowledge which should cooperate in forming a solution; 5. need to use multiple reasoning methods; 6. need of an evolutionary solution. These are considered as characteristics of the dynamic restructuring system.

3.2.3 Characteristics of a Blackboard

- **Complexity Management.** A common technique for dealing with complexity in a system is to divide the system into subsystems and make the relationship between subsystems lessextensive, and implement the interaction between subsystems through a higher level system.

In blackboard systems, a problem is decomposed so as to maximize the independence of the subsystems (KSs). Specificaaly no direct interaction between KSs is allowed, and all interaction should take place through the blackboard. But we should note that a KS may interact with more than one blackboard. It can get the problem from a domain blackboard, and communicate theoutput to another blackboard.

- **Easy problem formulation.** In a blackboard structure, both the problem and the knowledge are partitioned, which makes the problem formation relatively easy.

  Knowledge representation is separated into two parts: domain knowledge, and the knowledge of using the domain knowledge. (how, when, or where it is to be used). We can view a KS as a big rule,with the knowledge of using the knowledge as the condition part and the domain knowledge as the action part of the rule.

- **Flexible reasoning strategy.** Ill-structured problems are characterized by poorly defined goals and an absence of a predetermined decision path from the initial state to a goal. For example, in a dynamic restructuring problem, given a task, we do not have a clear goal that represents the workcell configuration for performing the task. For example, two components of the same type could be interchanged without affecting the system performance. The opportunistic problem-solving approach in a blackboard system provides the capability to deal with unpredicted or unenumerated events.

In next section, the restructuring problem will be formulated in a blackboard architecture for the first level decision making.

3.3 Problem Formulation in a Blackboard Architecture

The blackboard architecture that is used in the implementation of the FPS restructuring system is illustrated in Figure 3.3. There are seven knowledge sources (KS) which represent the problem solving knowledge and four blackboards representing system information which can be accessed by the knowledge sources. A control unit supervises the system and activates a triggered KS according to its priority. There may be data changes in the blackboard in executing of a KS. Such a change may trigger other KSs, which may lead to more execution of KSs. The procedure will end once the data in the blackboard satisfy the restructuring goal.

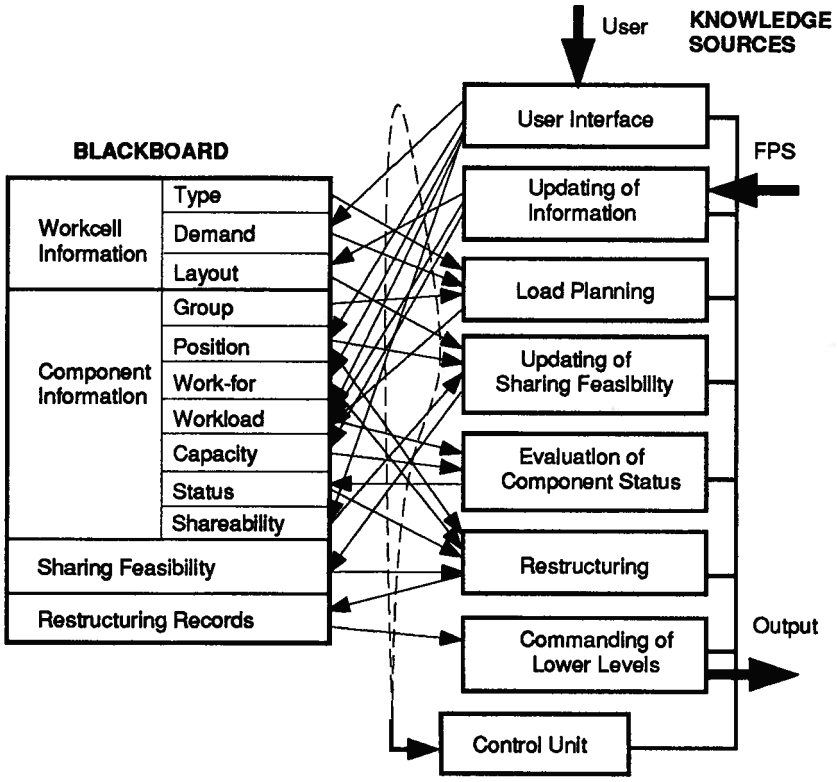

Figure 3.3: The Blackboard Structure of the Restructuring System

3.3.1 Knowledge Sources

Knowledge is organized into seven modules with information encapsulated, except the data from the blackboard. The modules are independent knowledge sources, each being responsible for solving a particular problem. In particular, the KSs are:

- a user interface for interaction between man and machine;
- a load planning system;
- an FPS information updating system;
- a component status evaluating system;
- an updating procedure for load transfer feasibility;
- a restructuring decision maker; and
- an output module.

The functions of these knowledge sources will be discussed in the next section. The KSs are represented by a frame structure as follows:

KS:

trigger-status;

priority;

input-blackboards;

function-procedure.

Here, "trigger-status" which has values "on" and "off", indicates whether the KS could be activated. The "priority" value resolves conflicts if more than one KS is triggered. The priority value is not given arbitarily, but according to the problem solving mechanism. For example, if both the KSs of "status evaluation" and "restructuring" are

triggered, the priority should be given to "status evaluation" for it should be executed before restructuring. This is the case because restructuring module should make decisions based on current status of the components which is evaluated by the KS "status evaluation".

The "input-blackboards" provide information that could cause the triggering of the KS "status evaluation" through the data change in one or more of these blackboards. Hence, KS triggering could be done automatically when a blackboard is modified. The "function procedure" is actually the action part of the KS. It is designed with encapsulated information so as to avoid parameter passing when it is called.

In the KS frame, each entity represents a property of a KS. From the point of view of object-oriented design, this structure defines a template or "schema" of which a KS is an instance. In logic representation (for Prolog), a predicate is designed for the KS implementation: $ks(K, P, V)$ which is intended to mean that the KS K has the property (attribute) P at the value of V. The template is then represented by

$ks(KS, trigger\_status, T).$

$ks(KS, priority, R).$

$ks(KS, input\_blackboard, B_1).$

.....

$ks(KS, input\_blackboard, B_n).$

$ks(KS, function\_procedure, F).$

where, KS is the name of the knowledge source, T is the value of the trigger-status (either "on" or "off"); R is the number of the priority value; and $B_1, \cdots, B_n$ are the input-blackboards of this KS. (The advantage having separated clauses of input blackboards should be clear in the later design of automatic triggering.) F is the functional procedure of the KS to be called when the knowledge source KS is activated.

There are three methods of data operation related to this template: "trigger_KS", "remove_trigger" and "execute_KS", which could be designed as member functions in an object-oriented system. Their Prolog clauses are as follows:

$$trigger\_KS(KS) \longleftarrow$$
$$retract(ks(KS, trigger\_status, off)),$$
$$assert(ks(KS, trigger\_status, on)).$$

$$remove\_trigger(KS) \longleftarrow$$
$$retract(ks(KS, trigger\_status, on)),$$
$$assert(ks(KS, trigger\_status, off)).$$

$$execute\_KS(KS) \longleftarrow$$
$$ks(KS, function\_procedure, F),$$
$$call(F).$$

where $trigger\_KS(KS)$ triggers the knowledge source KS on; $remove\_trigger(KS)$ is the inverse action of $trigger\_status$; and $execute\_KS(KS)$ executes the functional part of the knowledge source KS.

The particular knowledge sources in Figure 3.3 can be represented using the KS template as follows:

1. User interface

   $ks(user, trigger\_status, off).$
   $ks(user, priority, 6).$
   $ks(user, body, user).$

   where the *user* is the entry of this KS.

2. Updating of FPS information.

   $ks(fps, trigger\_status, off)$.

   $ks(fps, priority, 5)$.

   $ks(fps, body, fps)$.

   where fps is the entry predicate.

3. Load planner.

   $ks(load\_distributing, input, demand)$.

   $ks(load\_distributing, trigger\_status, off)$.

   $ks(load\_distributing, priority, 4)$.

   $ks(load\_distributing, body, load\_distributing)$.

   where *load_distributing* is the entry predicate.

4. Evaluation of the component status.

   $ks(set\_status, input, workload)$.

   $ks(set\_status, input, capacity)$.

   $ks(set\_status, trigger\_status, off)$.

   $ks(set\_status, priority, 3.5)$.

   $ks(set\_status, body, update\_workstatus)$.

   where *update_workstatus* is the entry predicate.

5. Updating of feasibility.

$ks(feasibility, input, sharability)$.

$ks(feasibility, input, position)$.

$ks(feasibility, trigger\_status, off)$.

$ks(feasibility, priority, 3)$.

$ks(feasibility, body, update\_feasibility)$.

where $update\_feasibility$ is the entry predicate.

6. Restructuring.

$ks(restructuring, input, position)$.

$ks(restructuring, input, work\_for)$.

$ks(restructuring, input, workstatus)$.

$ks(restructuring, input, feasibility)$.

$ks(restructuring, trigger\_status, off)$.

$ks(restructuring, priority, 2)$.

$ks(restructuring, body, restructuring)$.

where $restructuring$ is the entry predicate.

7. Output.

$ks(output, input, plan)$.

$ks(output, trigger\_status, off)$.

$ks(output, priority, 1)$.

$ks(output, body, output)$.

where $output$ is the entry predicate.

It is worth noting that this representation simply gives descriptive knowledge about a KS. The restructuring procedure which will be implemented in the control unit is

separated. Without doubt, knowledge sources themselves could also adopt a blackboard structure.

3.3.2 Blackboard

Common data including workcell level information and component level information are located in the blackboard (BB), and represented by objects of workcells and components using simple frames. The structure of a component frame is shown below:

component:

group;

position;

work-for;

workload;

capacity;

workstatus.

shareability

and a workcell frame has the following structure:

workcell:

task-type;

production-demand;

layout.

Similar to the representation of knowledge sources, the predicates $component(C, P, V)$ and $workcell(W, P, V)$ are defined for the representation of information of a component and a workcell respectively. Here, $component(C, P, V)$ means that the component C holds the property P at the value V, and $workcell(W, P, V)$ has the same interpretation except W representing a workcell rather than a component. Consequently, the component template is given by:

$component(C, group, G)$

$component(C, position, W)$

$component(C, work\_for, WF)$

$component(C, workload, WL)$

$component(C, capacity, Cc)$

$component(C, workstatus, S)$

$component(C, shareability, N)$

where C denotes a component, G is the name of a component group, say, *vision*, *robot*, *AGV* etc; W indicates the workcell in which the component C is located; WF is a list of workcells for which the component C works, and WF has the form $[W1, W2, \cdots, Wn]$; WL is a list of workloads corresponding to WF, say $[30, 20, \cdots, 40]$, in which load 30 (30% of the full capacity of a standard component of the group G) is assigned to workcell $W1$, and load 20 is assigned to $W2$ and so on; Cc is the capacity of the component C as a percentage with reference to a standard component; S which is a fuzzy variable, represents the load status of component C; and N which is a real number in the interval [0, 1] represents the shareability index of the component. For example,

$component(robot1, group, robot)$.

$component(robot1, position, cutting\_cell)$.

$component(robot1, work\_for, [cutting\_cell])$.

$component(robot1, workload, [80])$.

$component(robot1, capacity, 100)$.

$component(robot1, workstatus, [(mu, 0.4), (lu, 0.6)])$.

$component(robot1, shareability, 0.9)$.

Note that the same property of all objects is put in a specific sub-blackboard. For example, a sub-blackboard of capacity contains the capacity information of every component of the system. (See Figure 3.4). The KSs are sensitive to the properties of objects.

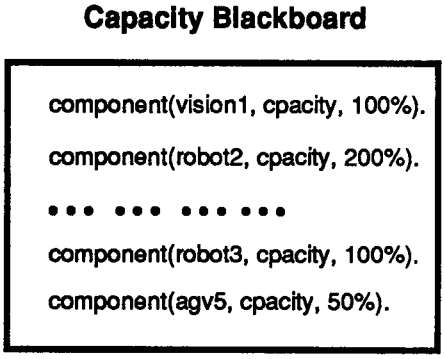

Figure 3.4: A Sub-Blackboard Representing Component Capacity

Actually, they are designed to handle object properties, rather than individual objects.

The template of workcell representation is given by:

$$workcell(W, task\_type, T).$$

$$workcell(W, demand, D).$$

$$workcell(W, layout, (X, Y)).$$

where W is a workcell; T is the task type that workcell W can process; D is the task demand; and (X, Y) pair describes its geographic position on the workshop floor. An instance is:

$$workcell(cuttin\_cell, task\_type, cutting).$$

$$workcell(cuttin\_cell, demand, 2item/s).$$

$$workcell(cuttin\_cell, layout, (150, 30)).$$

Also, the properties of workcells are divided into separate sub-blackboards.

There are two methods associated with the blackboard operations *get_property* and $modify\_property$ which could be considered as member functions in an object-oriented

design, such as an implementation in C++[24]. The two methods also incorporate some incomplete knowledge treatments and automatic triggering of KSs.

get_property(O, P, V) reports the property (P) value V of object O, and can be implemented as follow:

$$
\begin{aligned}
&\textit{get\_property}(O, P, V) \longleftarrow \\
&\qquad \textit{holds}(O, P, V), !. \\
&\textit{get\_property}(O, P, V) \longleftarrow \\
&\qquad \textit{holds}(O, \_, \_), \\
&\qquad \textit{assumable}(O, P, V). \\
&\textit{get\_property}(O, P, V) \longleftarrow \\
&\qquad \textit{holds}(O, \_, \_), \\
&\qquad \textit{ask}(O, P, V).
\end{aligned}
$$

$$
\begin{aligned}
&\textit{holds}(O, P, V) \longleftarrow \textit{component}(O, P, V). \\
&\textit{holds}(O, P, V) \longleftarrow \textit{workcell}(O, P, V). \\
&\textit{holds}(O, P, V) \longleftarrow \textit{ks}(O, P, V).
\end{aligned}
$$

If there exists a fact that the object O holds the property P at the value V, then the *get_property* returns this fact. Otherwise it uses the assumable default value or asks user whether the object exists. Here, object O can be either a component, a workcell or a knowledge source. Therefore, if our database is incomplete, the system will not hang.

modify_property(O, P, NV) modifies the property P of object O to a new value NV. If the property does not exist, the new value will be asserted to the database. An important side effect is that when a property BB named P is changed, associated KSs with input from the blackboard should be triggered.

$modify\_property(O, P, NV) \longleftarrow$

$component(O, P, V), !,$

$retract(component(O, P, V)),$

$assert(component(O, P, NV)),$

$trigger\_KS(P).$

$modify\_property(O, P, V) \longleftarrow$

$assert(component(O, P, NV)),$

$trigger\_KS(P).$

$trigger\_KS(P) \longleftarrow$

$findall(KS, ks(KS, input\_blackboard, P), KSs),$

$trigger(KSs).$

where $trigger(KSs)$ triggers a list of knowledge sources KSs.

$trigger([]).$

$trigger\_KS([H|T]) \longleftarrow$

$trigger\_KS(H),$

$trigger(T).$

Therefore, the knowledge sources interact with the blackboard through the methods given above. Conversely, The modification of blackboards will trigger certain agents through the methods associated with the template of "ks", for example, $trigger\_KS$ and $remove\_trigger$.

3.3.3 Control Unit

The control unit of a blackboard system functions as an inference engine for the descriptive knowledge that is represented by KSs. It monitors the system by looking for the KSs

with their *trigger_status* "on"; and selecting the KS of the highest priority; and finally executing that KS.

The whole system is designed to possess opportunistic reasoning and is data driven. A KS will be triggered when any of its input blackboards are modified, and the trigger-status can be removed only after the KS has been executed. Several KSs may be triggered at the same time, but the KS with the highest priority will be called first. This reasoning strategy actually provides an open architecture when several knowledge sources are working together, especially when the problem solving process is not very clear or the final goal is ill defined.

A simple reasoning procedure is given below:

$control \longleftarrow$

$collect\_triggered\_KS(KSs),$

$select\_highest\_PR(KSs, KS),$

$execute(KS),$

$control.$

$control \longleftarrow$

$trigger([user, fps]),$

$control.$

Here, the operation $collect\_triggered\_KS(KSs)$ collects all triggered knowledge sources. It may fail in case none of the knowledge sources are triggered, which will cause the controller turn to the second clause, which will then trigger the knowledge sources of "User Interface" and "Updating of FPS Information ". As an example of system operation, suppose that the blackboard region the "Component Information" has been changed. Then the knowledge sources "Updating of Sharing Feasibility " and "Restructuring" will be both triggered, and the former which has a higher priority, will be executed first. This execution may change the blackboard (BB) of "Sharing Feasibility", and can in turn

trigger the KS of "Restructuring" again. The execution of the KS of "Restructuring" may cause some other blackboard regions to change. This will trigger some more knowledge sources and consequently change the data in the associated blackboards, and so on. The procedure will end when an execution of the knowledge sources does not result in further changes of the BB data, and the trigger status of all the knowledge sources are off. Then, the system will idle.

3.4 Functions of Knowledge Sources

In the architecture shown in Figure 3.3, there are senven knowledge sources (KS) which deal with user interface, information updating, load planning, evaluating of component status, updating of sharing feasibility, restructuring, and output. Major functions of these knowledge sources are outlined below:

KS of user interface. This handles the input of production demands. It also provides a function that enables a user to change the information in the blackboard. It has the highest priority without any precondition.

KS of updating of information. This KS updates the operating information of the FPS, including the available capacity resources, workcell activity levels, and the system layout. It obtains data from the component level through information preprocessors and from the workcell level through intelligent preprocessors as well. This KS has the second priority.

Techniques of sensor fusion [5, 37, 13] could be used for this purpose. A multiple sensor approach which is similar to the method a human would use to monitor a production process is considered. The approach of a human is characterized by lack of accurate sensors and process models. However, the human considers a number of different sensors (his/her own senses), and processes the information about a variety of process variables.

Similarly, one can consider a system for the monitoring of production processes whereby the measurement of process variables is performed by several sensing devices which feed their signals into a processing model. A knowledge based system can be employed to handle these possibly inaccurate and incomplete signals, and provide information to the decision maker of FPS restructuring. (see Figure 3.5)

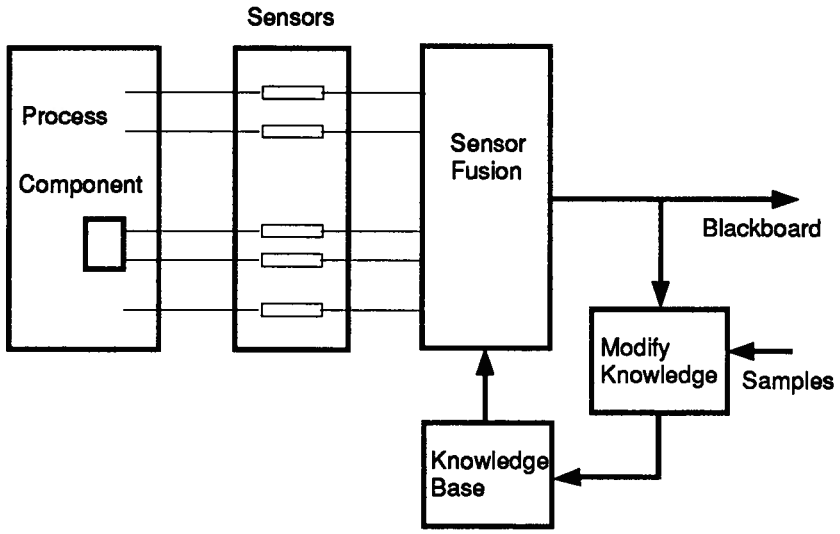

Figure 3.5: An Intelligent Sensing System

This part in itself is a wide topic, and will not be studied in detail in this thesis. It is assumed that there is such a system (KS) for later use in the simulation.

KS of load planning. This KS has a knowledge base for capacity planning [45, 4] methods and associated transformations that determine component loads for a specified demand (see equation (2.2)). It assigns the workload to the workcell components according to the processing demand and the capacity planning method, and then sets the component status according to component capacity, load, and the activity level. This KS has the third highest priority.

KS of evaluation of component status. This KS uses information, such as capacity C_c, load L_c and activity level A_c of a component, and evaluates the status

of the component using equation (2.5). The degree of component overload or capacity surplus is measured by fuzzy descriptors OC and UC, whose membership functions are illustrated in Figure 3.6.

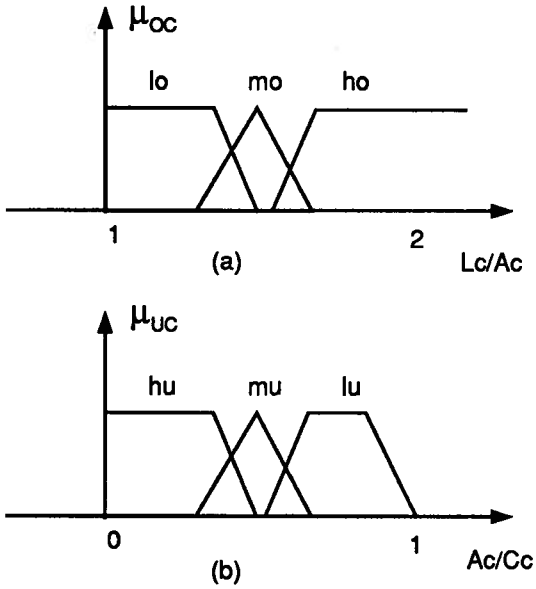

Figure 3.6: The Membership Functions for Fuzzy Descriptors of Overload (a) and Undercapacity (b) of Component

KS of updating of sharing feasibility. This KS obtains information about component shareability and geographic factors, and updates the sharing feasibility index. Since this should be done before restructuring, it has a higher priority than that for KS of restructuring.

KS of restructuring. This KS performs a procedure of deciding the most appropriate action against the badness of current system. Knowledge based methods employing fuzzy logic are used in the decision making. Actually, levels 2 and 3 in Figure 3.1 will implement this knowledge source, and is central to restructuring decision making.

KS of output (commanding of lower levels). This sends restructuring commands to the workcell controllers which are located at a lower level. Some coordination of the

shared components should be done here. In addition, the restructuring commands might have to be approved by system experts. This KS has the lowest priority.

3.5 Summary

Basic considerations about knowledge organization were presented in this chapter. A three level decision making hierarchy was used for restructuring system.

1. The first level will determine whether a restructuring should be called.
2. The second level provide some general guidance about how to restructure the system.
3. The third level will select an optimal restructuring action among all the possibilities.

The first level will make the blackboard model to cooperate with different kinds of modules for information updating, data processing, and decision making associated with restructuring. The characteristics of a blackboard architecture, such as a common data area and opportunistic reasoning, make it particularly suitable for organizing a high level FPS restructuring system.

Domain knowledge is represented by objects of workcells and components. Each type of property of the objects will occupy a sub-blackboard. Data change in the blackboard will trigger related knowledge sources. A control unit will monitor the entire system and repeatedly select a triggered KS that has the highest priority for execution.

This structure is suitable for bottom-up proof, particularly, when the restructuring goal is fuzzy.

Chapter 4

Heuristics and Actions

The concept and the framework of dynamic restructuring have been presented in chapters 1 and 2, and a problem solving architecture has been developed in Chapter 3. This chapter will study the heuristics for solving the problem. Actions, are also discussed in detail, which change data in the blackboard system that has been described, so as to make the blackboard evolve from the initial world to the final world in which the restructuring goal is true.

Section 4.1 analyzes the blackboard-based restructuring procedure; Section 4.2 develops the heuristics and Section 4.3 specifies the actions of restructuring. Section 4.4 gives a summary.

4.1 Analysis of Restructuring Procedure

Referring to the decision making hierarchy given in Figure 3.1, we note that the second level will be called after the first level has decided to have a restructuring action. It is very helpful to have a clear picture of the restructuring procedure in building the general restructuring heuristics.

4.1.1 Planning Using Blackboard

As discussed in Section 2.3, general planning methods are not suitable for the restructuring problem which is addressed in this thesis. A practical flexible production system

(FPS) is too complex to be represented in a reasonable size by classical planning methods. As a remedy, in blackboard-based restructuring planning, the use of expertise from domain experts and practical operators is exploited. The basic idea of blackboard-based planning is as follows: Represent the initial world in a blackboard; and then change the world by performing some actions until the final world (goal) is achieved, in which the desired goal is true. This approach is illustrated in Figure 4.1.

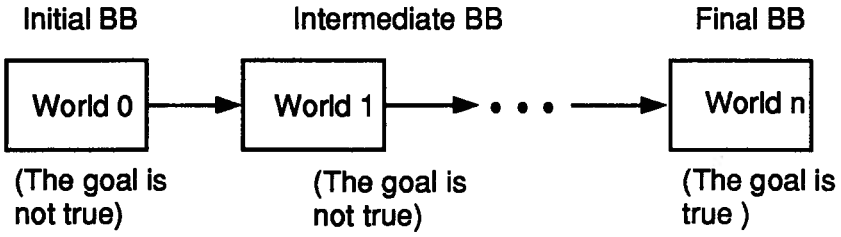

Figure 4.1: The Blackboard-Based Planning

Note, however, that we do not specify object properties for every world in the reasoning procedure. The system (workcells, components) situation is still represented in the general property predicate, but without specifying the world:

$$holds(Object, Property, Value),$$

such as

$$component(O, P, V)$$

or

$$workcell(O, P, V).$$

If we assume that complete knowledge about an FPS exists and is represented in the blackboard, then we can use "failure as negation"; that is if a particular property cannot be found, that means the property does not exist. (i.e., $not(holds(O, P, V))$ is true if $holds(O, P, V)$ fails). Therefore, it is important that all the object properties are

placed in the blackboard (or indicated as assumable and askable), and all changes of properties are applied on the blackboard. Then an action will amount to changing the corresponding blackboard (BB).

The present change of representation does not mean that we treat an FPS as a static world. Instead, we implicitly assume that if an object property is not changed by an action, then the object will still keep the old property value in the new world. Clearly, we no longer need axioms for all situations of every action. We rather need only those rules in which an action really changes the world. This significantly reduces the extent of the representation space of a problem during solution.

The effects of all actions should be anticipated and specified. The corresponding blackboard should be modified when an action is applied. It is this modification that makes the blackboard evolve from an initial world to the final world where the the goal is achieved.

4.1.2 Heuristics for Search

The search space of restructuring planning is potentially large, and heuristics would be necessary for practical solution of the problem. Also, due to the fuzziness of a restructuring goal, multiple solutions may exist. Thus, optimization should be considered as well. A decision for selecting a restructuring action is made on the following basis:

1. Trimming the search space by using heuristics;

2. Selecting an optimal action in the trimmed space.

Note that without the first step, the second step is not usually feasible for it is hard to evaluate all actions in a global search space. This idea is illustrated in Figure 4.2

Here, for example, if only the actions pertaining to "component releasing" should be considered first according to some heuristics, then only those actions in the sub-space

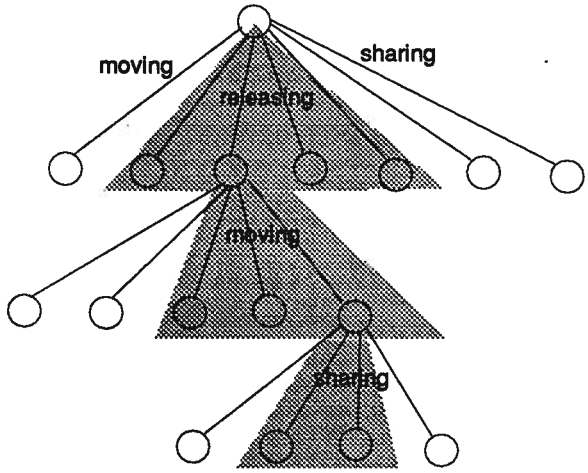

Figure 4.2: The Heuristic Search and Optimization

indicated by the shaded area (of releasing) are of interest for further consideration. This is what is termed second level decision making, in the present problem, the third level is a optimization procedure that selects an action that has the highest assessment from the sub-space. (The third level will be discussed in Chapter 5.) The same procedure as before can be applied in all level of planning, as shown in Figure 4.2.

The next two sections will develop heuristics for restructuring and specify all possible actions.

4.2 Heuristics for Problem Solving

In restructuring, some general guidelines should be taken into account before selecting a detailed action. This is very helpful in the search for an optimal action in a particular case, for example, as shown in Figure 4.2. Human expertise and past experience are assets in developing heuristics. Note that, each heuristic will be developed for only one case; separate cases use separate heuristics even though the conclusions might be the same. Union is not allowed. This would be necessary for a clear logic expression and according to the convention of Prolog.

4.2.1 Heuristics

Termination of a Load Sharing Action

Firstly, if an FPS does not work properly, the existing load sharing states should be checked. In case a previously-overloaded component is no longer overloaded due to changes in operating conditions,(say, the associated task demand has been dropped or the component has been upgraded), the sharing would not be necessary and should be terminated. Consider the following three heuristics:

Heuristic 1A: Terminating a load sharing action due to load reduction. If there exists a load sharing state $shared(OC, UC, Load)$, **and** the OC would no longer be overloaded without this sharing, **then** this load sharing action should be terminated. This may be implemented as

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad plan(shared(OC, UC, Load)), \\
&\qquad component(OC, workload, L), \\
&\qquad not\_overloaded(OC, L, Load), \\
&\qquad A = terminating(shared(OC, UC, Load)).
\end{aligned}
$$

Here $plan(R)$ records an action R which is either $shared(OC, UC, Load)$ a record of action $sharing(OC, UC)$ or $released(VC, C, Load)$ a record of action $releasing(UR, C)$ where VC is a virtual component reflecting the properties of UR. $not\_overloaded(OC, L, Load)$ is true if OC is not overloaded given loads L and $Load$. If all the subgoals (conditions) are true, a command of sharing termination will be issued.

In another case, if the index of sharing feasibility between a sharing pair of components reduces below a threshold value, the sharing should be ceased. This may happen, for example, when the communication system of a sharing component is damaged completely or partially, or the moving path for a shared component is no longer available.

Heuristic 1B: Terminating a load sharing action due to feasibility reduction. If there exists a load sharing state $shared(OC, UC, Load)$, **and** the sharing feasibility index drops below a threshold value, **then** this load sharing action should be terminated. This may be implemented as

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad plan(shared(OC, UC, Load)), \\
&\qquad feasibility((OC, UC), I), \\
&\qquad I < 0.6, \\
&\qquad A = terminating(shared(OC, UC, Load)).
\end{aligned}
$$

where I is the index of sharing feasibility between the pair OC and UC; and 0.6 is the threshold value. If the sharing feasibility value is less than 0.6, the sharing should be terminated.

Furthermore, as given in the next heuristic, an existing sharing state should be terminated if either component which is being shared becomes overloaded. Then, a new, more appropriate sharing strategy should be sought in place of the current strategy, in order to remove the overload.

Heuristic 1C: Terminating a load sharing action due to a load increase. If there exists a load sharing state $shared(OC, UC, Load)$, **and** either OC or UC is currently overloaded, **then** $shared(OC, UC, Load)$ should be terminated. This may be implemented as follows:

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad plan(shared(OC, UC, Load)), \\
&\qquad overload(OC, or, UC), \\
&\qquad A = terminating(shared(OC, UC, Load)).
\end{aligned}
$$

where $overload(OC, F, UC)$ checks the workload status of OC and UC according to the flag F. If either of the components is overloaded, the sharing should be disabled.

Termination of a Component Releasing Action

A component releasing action also should be checked to determine whether a component is shared with more than one load, due to that action. Similarly, if the shareability index of the loaded component on a "component releasing" action is below a certain threshold, or if the component has been overloaded, the component releasing action should be terminated.

Heuristic 2A: Terminating a component releasing action due to reduction of the component shareability index. If there exists a component releasing state "$released(VC, C, Load)$", **and** the component C is no longer suitable for receiving the released load, **then** this component releasing action should be terminated.

Here $released(VC, C)$ is a record of action $releasing(UR, C)$ which released a component UR by transferring its load to component C, (Note that C would occupy in its original task as well, and therefore, would be shared for the loads of both components). A virtual component VC records all the properties of VC except capacity. It is expressed by:

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad plan(released(VC, C, Load)), \\
&\qquad component(C, shareability, I), \\
&\qquad I < 0.6, \\
&\qquad A = terminating(released(VC, C, Load)).
\end{aligned}
$$

where I denotes the shareability of component C. If I is less than a threshold value (0.6 in this case), the component C should not be shared for the both loads. Note that the "releasing" action actually results in a component sharing, in the sense that the record $released(VC, C, Load)$ represents an existence of a component sharing state for C.

Heuristic 2B: Terminating a component releasing action due to an increase

of load. If there exists a component sharing state "$released(VC, C, Load)$", **and** the component C is currently overloaded, **then** this component releasing state should be terminated. This may be implemented as follows:

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad plan(releasing(VC, C, Load)), \\
&\qquad component(C, workload, L), \\
&\qquad component(C, capacity, Cc), \\
&\qquad sum\_up(L, LD), \\
&\qquad LD > Cc, \\
&\qquad A = terminating(releasing(VC, C, Load)).
\end{aligned}
$$

Here LD is the sum of all the loads assigned to component C. If LD is greater than its capacity C_c, then component C should not be shared, and hence the component releasing state should be terminated.

Component Moving

A component may be moved from a workcell to which it is assigned to another workcell. Usually, for proper control and communication within the system, the geographic position of the component has to be convenient for the workcell for which it now works. This situation takes place when a component is released within its original workcell, and is reassigned to another workcell in order to absorb an overload. One thing which should be noted here is that, the moving action should be done preferably prior to the load reassignment, in the practical situation, although the moving is actually caused by the need for a reassignment. Consequently, in the restructuring report, a moving action could be placed before the reassignment action.

Heuristic 3: Moving a component. If component C is located in workcell W, **and** is working solely for another workcell $W1$, **then** this component could be moved to

workcell $W1$ from workcell W. The implementation of this heuristic is given below:

$$
\begin{aligned}
&\mathit{heuristic}(A) \longleftarrow \\
&\qquad \mathit{component}(C, \mathit{position}, W), \\
&\qquad \mathit{component}(C, \mathit{work\_for}, [W1|[]]), \\
&\qquad W \neq W1, \\
&\qquad A = \mathit{move}(C, \mathit{from}(W), \mathit{to}(W1)).
\end{aligned}
$$

Component Releasing

Component releasing should be considered before component sharing, because the released components may be used later, in sharing. Based on human expertise, sharing of components within a workcell is better and easier than sharing between workcells. Hence, component releasing within a workcell is considered to be of higher priority than effecting a component releasing action between workcells. Also, the component shareability should be considered in effecting a releasing action.

Heuristic 4: Releasing a component within a workcell. If there exists an undercapacity component, **and** there exists another undercapacity component of the same type in the same workcell, **then** one of them could be released by transferring its load to the other one. This heuristic may be implemented as follows:

$$
\begin{aligned}
&\mathit{heuristic}(A) \longleftarrow \\
&\qquad \mathit{group}(G), \\
&\qquad \mathit{group\_components}(\mathit{undercapacity}, G, UCs), \\
&\qquad \mathit{best\_action}(UCs, A, PD, \mathit{within}).
\end{aligned}
$$

where G is the group name. Here, $\mathit{group\_components}(F, G, UCs)$ groups all components in group G into a list UCs according to the flag F which may take values "overloaded" and "undercapacity"; and $\mathit{best\_action}(UCs, A, PD, FL)$ is a decision making procedure,

which will be discussed in Chapter 5, for selecting the best action A which has priority degree PD, according to flag FL which has values "within" (a workcell) and "between" (workcells). The action A takes the form "$releasing(UR, UL)$" here.

Heuristic 5: Releasing a component between workcells. **If** there exists an undercapacity component, **and** there exists another undercapacity component of the same type in an other workcell, **then** one of the two components could be released by transferring its load to the other. This heuristic may be implemented as follows:

$$
\begin{aligned}
&heuristic(A) \longleftarrow \\
&\qquad group(G), \\
&\qquad group\_components(undercapacity, G, UCs), \\
&\qquad best\_action(UCs, A, PD, between).
\end{aligned}
$$

Here, the only difference from the previous one is that the flag "within" (a workcell) is changed to "between" (workcells). This difference reflects an important consideration of priorities of actions within a workcell or between workcells.

Load Sharing

Component overloads are considered now. For the same reasons as in releasing a component, a load sharing action within a workcell is considered to have a higher priority than an action of component sharing between workcells.

Heuristic 6: Sharing a component within a workcell. **If** there exists a component overloaded, **and** if there exists another component of the same type that is operating at undercapacity in the same workcell, **then** share the undercapacity component with the overloaded one. The implementation of this heuristic may be as follows:

$$heuristic(A) \longleftarrow$$
$$group(G),$$
$$group\_components(overloaded, G, OCs),$$
$$best\_action(OCs, A, PD, within).$$

Note that this heuristic is different from **Heuristic 4** only by the flag "overloaded" in place of "undercapacity". The decision-making procedure will select a proper action A by checking the status of the components. The action A takes the form $sharing(OC, UC)$ here.

Heuristic 7: Sharing a component between workcells. **If** there exists a component that is overloaded within a workcell, **and** if there exists another component of the same type in another workcell that is operating at undercapacity, **then** share the undercapacity component with the overloaded one. This heuristic may be implemented as follows:

$$heuristic(A) \longleftarrow$$
$$group(G),$$
$$group\_components(overloaded, G, OCs),$$
$$best\_action(OCs, A, PD, between).$$

It is clear in this heuristic that, the flag is "between" instead of "within" in the previous heuristic, so as to call for sharing actions between workcells.

4.2.2 Usage of Heuristics

The heuristics discussed in the previous sections are given different priorities in practice. They are considered in the order of their priorities, in the actual operation of

the knowledge-based decision making system. By reviewing the process of restructuring planning, it should be noted that the heuristics are actually used by a mechanism that is schematically shown in Figure 4.3.

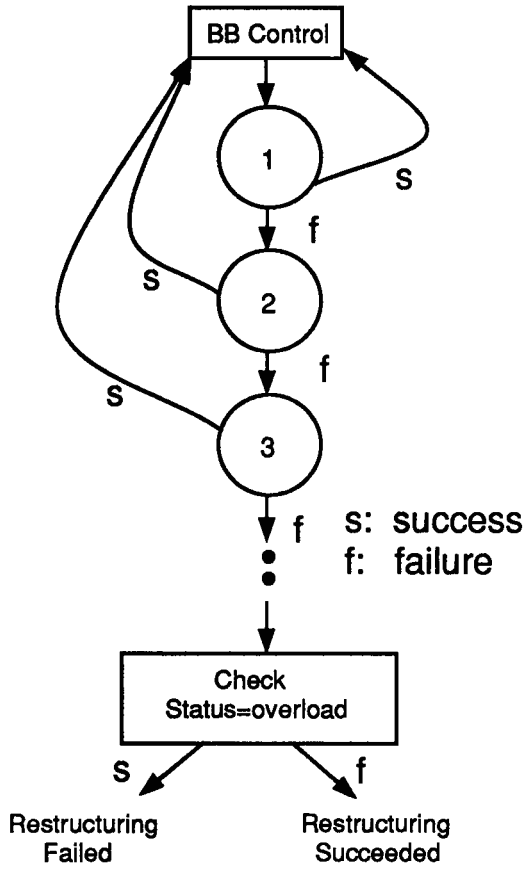

Figure 4.3: The Reasoning Sequence

Here, the heuristics denoted by 1, 2, 3, $\cdots$ etc. are arranged in the order of the priority of execution of the associated rules. Note that only when the heuristics with higher priority have failed to solve the problem, a lower priority one can be fired. When a heuristic succeeds, which means the blackboards may receive new data by the actions issued by the heuristic, the first-level decision maker will review the entire problem in the blackboard all over again. This process will end when none of the knowledge sources

are triggered, which actually means no data change has been made in the last step of the restructuring process. In another words, no heuristic succeeded and no action was issued.

4.3 Actions

Although the heuristics return various actions, they actually fall into two kinds of physical actions:

$$transfer\_load(OC, UC, L)$$

and

$$move\_component(C, W1, W2)$$

where $transfer\_load(OC, UC, L)$ removes load L from OC and assigns it to UC; $move\_component(C, W1, W2)$ moves component C from workcell $W1$ to workcell $W2$. This section will specify the restructuring actions by means of practical physical actions.

Load Sharing

In "load sharing" an undercapacity component shares the load of an overloaded component. Consequently, the load originally assigned to the second component will be shared by the two components (of the same type). As a result, the originally undercapacity component is now shared between its original load and the new load. This action is represented as follows:

$$\begin{aligned} sharing(OC, UC) \longleftarrow \\ & determin\_load(sharing(OC, UC), Load), \\ & transfer\_load(OC, UC, Load), \\ & record(shared(OC, UC, Load)). \end{aligned}$$

where, $determin\_load(sharing(OC, UC), Load)$ determines the load "*Load*" to be transferred from OC to UC; and $record(A)$ records the information of this action for further reference in restructuring.

Component Releasing

Component releasing represents a load transfer from an undercapacity component to another undercapacity component, with the complete load of the first component being absorbed by the second one thereby releasing the first component. This action may be implemented as follows:

$$
\begin{aligned}
&releasing(UR, UL) \longleftarrow \\
&\qquad determin\_load(releasing(UR, UL), Load), \\
&\qquad transfer\_load(UR, UL, Load), \\
&\qquad make\_virtual(UR, VC), \\
&\qquad record(released(VC, C, Load)).
\end{aligned}
$$

Here, the predicate $make\_virtual(UR, VC)$ records the properties of UR, and in case this component releasing action should be terminated, the virtual component could be used for the intermediate state of load transfer. Note that, a virtual component is a copy of the released component except its capacity is always set to zero. In the component releasing process, it is a copy of UR in the record, and makes UR free for other use.

Terminating a Sharing or Releasing Action

Terminating a load sharing action is the inverse operation of component sharing. It may be implemented as below:

$$terminating(shared(OC, UC, Load)) \longleftarrow$$
$$transfer\_load(UC, OC, Load),$$
$$remove(shared(OC, UC, Load)).$$

Note that the termination of a component releasing operation is not as straightforward, because it is not just the inverse operation of component releasing, since the released component might already be used for other purposes. In fact, the inverse operation of component releasing does not always exist. In such a case, a virtual component could be used, which has all the properties (recorded when making the releasing action) of the formerly released component UR, but with zero capacity (i.e., does not exist physically). Thus, a virtual component represents a pure shortage of capacity, and it always seeks a solution for its overload. The implementation is and follows:

$$terminating(released(VC, C, Load)) \longleftarrow$$
$$transfer\_load(C, VC, Load),$$
$$remove(released(VC, C, Load)).$$

Moving a Component

Moving a component needs a specific type of action, which is denoted as *move_component*. Moving a component is specified as:

$$move(C, from(W1), to(W2)) \longleftarrow$$
$$free(C),$$
$$move\_component(C, W1, W2).$$

That the component C must be a non-activity component is a precondition for actually moving a component.

The *move_component*/3 action changes only the position of a component. The *transfer_load*/3 action changes the component workloads, and also the property of

work_for. If the load of a component that is dedicated to a workcell, is zero, it is released from that workcell, at least in principle.

4.4 Summary

Restructuring of a flexible production system (FPS) was analyzed in this chapter. The core activity in restructuring was identified as a planning procedure. A plan is a sequence of actions from the initial world to a goal world. Heuristics play a very important role in searching for this sequence.

Heuristics include: 1. checking whether the existing sharing states are still suitable; 2. moving a component; 3. selecting a component-releasing action and; 4. selecting a load-sharing action. They are fired in the order of their priority.

The actions in a restructuring process include: terminating a sharing, moving a component, releasing a component and sharing a load. The main physical actions involved are: *transfer_load* and *move_component*, which modify the status of a component.

Chapter 5

Fuzzy Decision Making

In previous chapters, the architecture of the knowledge-based restructuring system has been established. In particular, the problem solving heuristics have been developed in Chapter 4. Clearly, there may be more than one action that could be available at a restructuring stage when using heuristics for solving the problem. This chapter addresses the resolution of this situation, which is known as conflict resolution. Fuzzy logic will be used in evaluating the available actions in a stage of decision making.

Section 5.1 discusses the conflict resolution problem in a restructuring process, which requires a decision on selecting an optimal action. Section 5.2 gives a fuzzy decision-making structure. According to this structure, Section 5.3 presents a method for recognition of the status of a flexible production system (FPS), using fuzzy descriptors. Section 5.4 presents a decision-making method with emphasis on techniques of fuzzy associative memory. Section 5.5 gives concluding remarks.

5.1 Conflict Resolution

As indicated in Figure 4.2, heuristics lead the problem solver to a sub-solution space (shaded area), in the restructuring process. The need for conflict resolution arises in the subsequent steps of problem solving in the sub-solution space. Since the final goal is fuzzy, there may be more than one solution in the trimmed solution space. Optimization should be considered for this reason. Note that, for different types of actions, the appropriate conflict resolution method might be different, which is actually determined by the

restructuring mechanism, and will be discussed in detail in the sequel.

For actions about terminating a sharing mode, the first match method [13, 14] could be used. An action of terminating a sharing state will affect only the pair of shared components (including virtual components), and the other components in the FPS will not be affected. Thus, component sharings can be released in any order. In addition, the associated heuristics have the highest priority, which means, only after all unnecessary component sharings have been released, that other actions such as component releasing and load sharing should be considered.

For an action of moving a component also, the first match method is suitable. Almost the same reasons apply here, specifically, 1. moving a component does not affect other moving actions; and 2. all moving actions will be executed before making further decisions on releasing and sharing of components. Therefore, the sequence of executing the moving actions will not affect the entire decision making process for restructuring.

For component releasing and sharing actions, however, the sequence of actions will affect the decision making process of restructuring. As an example, consider four components of the same type: $C1$, $C2$, $C3$, and $C4$. Suppose that, $C1$ is lightly overloaded, $C2$ is moderately overloaded, $C3$ has a high undercapacity, and $C4$ has a light undercapacity. If $C1$ is considered first, $C3$ should be suitable for sharing the overload in $C1$. As a result, $C2$ will not be able to find a proper partner to share its overload. The proper match would be $C4$ for $C1$ and $C3$ for $C2$.

Generally, releasing and sharing actions do affect other releasing and sharing decisions. It follows that an optimal match should be used for the associated conflict resolution. In particular, the following important factors should be considered:

- The load status of the components which will be subjected to a releasing or sharing action;

- The belief degree of knowledge of executing such an action between this pair of components;
- Feasibility of load transfer between the pair.

In the following sections, a fuzzy decision making system will be developed for selecting the optimal actions.

5.2 A Fuzzy Decision-Making Structure

Two steps are considered for the fuzzy decision making process as illustrated in Figure 5.1.

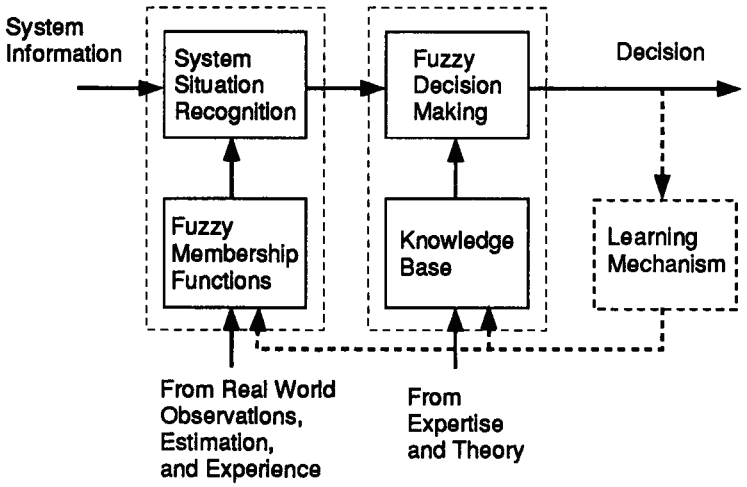

Figure 5.1: Schematic Representation of Fuzzy Decision Making

Firstly, the system status should be recognized. The recognized conditions should match the control knowledge of experts. Particularly, in a knowledge-based restructuring system, component workload status is expressed in terms of linguistic descriptors, such as "highly overloaded" or "lightly undercapacity". Naturally, it may be useful then, to measure and recognize the component status as well in a qualitative manner. Secondly, based on the current status of the system, a set of rules may be fired to make decisions for

selecting proper restructuring actions. A learning mechanism could be used in a feedback loop to adjust fuzzy membership functions and the knowledge base.

Fuzzy logic [46, 28, 26] has been employed to deal with qualitative and non-crisp concepts that are present in decision-making problems of the real world. Applications are found in dynamic systems and control [8, 27]. The present practice in fuzzy logic control is to consider the confidence level of the associated fuzzy-logic rules to be 100 percent. In this thesis, however, both the situation facts and human knowledge are considered to have varying degrees of belief. A fuzzy decision is made by considering all possibilities of a system status, with different degrees of belief. The corresponding rules are also assumed to have different degrees of belief.

"Fuzzy membership functions" describe a mapping from real-world "physical" measurements to human interpretations of linguistic terms in a fuzzy descriptor. The concepts are specific to a particular use, but should be valid for a generally recognized measurement range corresponding to a given attribute. The knowledge base of "decision making" consists of rules such as those given in figures 2.5 and 2.7. The fuzzy decision maker fires those rules whose condition parts match the present FPS state, as observed, and synthesizes the conclusions of those rules with associated belief values, so as to generate a restructuring decision.

In the following two sections, the representation of system information and human knowledge associated with the present problem will be discussed. Then, a fuzzy decision-making procedure will be designed and implemented in Prolog, which uses this representation.

5.3 Representation of Situation

5.3.1 Fuzzy Representation of Component Status

Component status is expressed by a fuzzy set [15] rather than using crisp logic, which is based on the following reasons:

- A "crisp" sensing signal of a component indicating its status may not be accurate at a given time, and may not be reliable. A fuzzy representation might take subjective and qualitative considerations and past experience into account, and would be better.
- Some information on components may be directly as fuzzy indicators, such as the load status of a humanbeing, or quality of a piece of processed herring roe, as provide by "intelligent" sensors.
- Human knowledge describing a component status is fuzzy by itself. In order to use human knowledge to make decisions, the system conditions should be recognized in a compatible manner.

The component status can be described by a fuzzy descriptor such as {highly overloaded (*ho*), moderately overloaded (*mo*), lightly overloaded (*lo*), just ok (*ok*), lightly undercapacity (*lu*), moderately undercapacity (*mu*), highly undercapacity (*hu*), and no activity (*na*)}. Each value of the variable in the "universe of discourse" will have a membership grade which represents the degree to which that value belongs to the set. A component status is represented by the membership grades at which it belongs to a group of fuzzy sets. For example,

$$S_{OC} = \{lo(0.8),\ mo(0.2)\}.$$

where S_{OC} is a fuzzy variable representing the status of the component OC. It states that the membership of this quantity in the fuzzy set *lo* is 0.8, and in *mo* is 0.2, and it does not belong to any other fuzzy sets. As a result, this component is eligible to use the rules for both states *lo* and *mo*.

Typical membership functions of component status are shown in Figure 5.2. For example, a component of status *ok* means that when it is working within the crisp load range 85 $\sim$ 100% it operates in an "ok" state, with any possibility of displaying another state, and when it is operating within the non-crisp edge beyond this range, there is a possibility of displaying the characteristics of little *lu* or *lo* as well. Note that the "capacity" and "load" are both standardized for the convenience of comparison in later decision making.

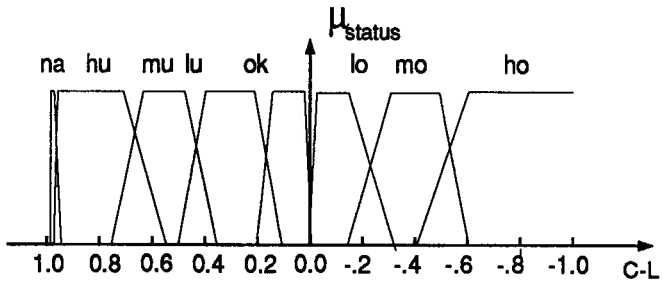

Figure 5.2: The Membership Functions of Component Status

To provide an analytical foundation for our developments, some definitions of fuzzy operators are given next. Using them, a "symbolic language" can be introduced, which may facilitate further developments in the area of fuzzy-logic decision making.

Definition 1. Fuzzy operators are denoted as follows, with the given order (increase) of precedence:

proposition $\rightarrow$;

$$\begin{aligned} \textit{negation} \quad & \neg; \\ \textit{intersection} \quad & \cap; \\ \textit{union} \quad & \cup; \end{aligned} \tag{5.19}$$

For example, $S_{OC \to lo}$ means the component status is "lightly overloaded" (lo). This statement may be interpreted in two ways. First, the statement may be assigned a "validity value" (say, 0.8). Then, under that particular circumstance, the level of validity of the statement is 80%. Alternatively, one could think about of a specific "measurement", say e_1 of the variable S_{OC}. This measurement may represent a membership grade of 0.8, as read off the membership function of lo. (i.e., $\mu_{lo}(e_1) = 0.8$). Under the same circumstances, the statement $S_{OC \to mo}$ may be assigned another validity value (say 0.2). As before, then, the specific measurement e_1 of S_{OC} has a membership grade of 0.2 (or $\mu_{mo}(e_1) = 0.2$). In summary, a given fuzzy-logic statement may possess a particular validity value under a specific circumstance, or alternatively, the membership grade at which the measurement belongs to the fuzzy set in that statement also would be equal to the validity value. In this manner, a given circumstance will satisfy different fuzzy states at different validity values, or equivalently, a given measurement will have different membership grades in different state sets (the latter being known as "fuzzification" of a measurement). As an example, a component $robot1$ whose workload is 60% of its full capacity can be fuzzified to:

$$S_{robot1} = \{lu(0.4), mu(0.6)\}$$

A general representation for component status S_c is

$$S_c = \{m_i(\mu_i)\} \tag{5.20}$$

The ith proposition "S_c is m_i" has a membership grade value μ_i.

$$\mu_{m_i}(S_c) = \mu_i \tag{5.21}$$

The proposition set is expressed by:

$$\mathcal{S}_c = \{w_c \rightarrow m_i(\mu_i)\} = \{p_i(\mu_i)\} \tag{5.22}$$

with

$$p_i = w_c \rightarrow m_i \tag{5.23}$$

Note that the notation of calligraphic letters represent proposition sets. Using this convention, the status of *robot*1 can be expressed as

$$\mathcal{S}_{robot1} = \{w_{robot1} \rightarrow lu(0.8), w_{robot1} \rightarrow mu(0.2)\}$$

Members in the above set are propositions of the status of the component *robot*1 with associated validity values.

5.3.2 Compound Propositions

For both sharing and releasing of components, at least two components should be considered. For example, component status of *robot*1 and *robot*2 are factors that should be considered for load transfer between the two components. The compound proposition "status of *robot*1 and status of *robot*2" is therefore taken into account, which is expressed by

$$\mathcal{S}_{robot1} \cap \mathcal{S}_{robot2}.$$

It may be expressed in a Catesian product space with dimensions of $\mathcal{S}_{robot1}$ and $\mathcal{S}_{robot2}$. The elements in the space represent a status of the FPS by considering only the status of *robot*1 and *robot*2.

Generally a fuzzy state is given by an expression of fuzzy variables, which represent system-status factors. A fuzzy-status expression may have the following four forms:

$$\begin{aligned} &1.\ Expr\ :\ Single\ fuzzy\ variable;\\ &2.\ Expr\ :\ \neg Expr;\\ &3.\ Expr\ :\ Expr \cap Expr;\\ &4.\ Expr\ :\ Expr \cup Expr. \end{aligned} \tag{5.24}$$

These are elaborated below:

Case 1: Here, the expression is a single fuzzy variable X, given by the fuzzy set

$$X = \{s_i(\mu_i)\}_{i=1,\cdots,r} \tag{5.25}$$

Then, the fuzzy status $\mathcal{X}$ is expanded as a set of propositions with the validity values,

$$\mathcal{X} = \{m_i(\mu_i)\}_{i=1,\cdots,r} \tag{5.26}$$

where, the statement

$$m_i = X \rightarrow s_i \tag{5.27}$$

has a validity value μ_i in a particular situation. Equivalently, if the situation is given by a "measurement" $\overline{m}$, then its membership grades in the various fuzzy state variables m_i are given by $\mu_{m_i}(\overline{m}) = \mu_i$; $i = 1, 2, \cdots, r$. (μ_i rather than $\mu_{m_i}(\overline{m})$, is used in later representations for simplicity. Note that μ_i is defined in the interval [0, 1]). Here, r is the resolution [7] of a fuzzy set representing variable X; s_i is the ith fuzzy state of this variable; and μ_i may be interpreted as either a validity value of m_i, or the membership grade of a measurement within a fuzzy state set as discussed before.

Case 2: The negation $\neg$ is a unitary operator. It sets every proposition of the operand to its "complement of 1", specifically, if

$$\mathcal{X} = \{m_i(\mu_i)\}_{i=1,\ldots,r} \tag{5.28}$$

then,

$$\neg\mathcal{X} = \{m_i'(\mu_i')\}_{i=1,\ldots,r}, \tag{5.29}$$

where,

$$m_i' = \neg m_i \tag{5.30}$$

with,

$$\mu_i' = 1 - \mu_i \tag{5.31}$$

Case 3: The intersection $\cap$ is a binary operator. Its result is represented in the Cartesian-product space having the dimensions of the two operands. Specifically, if

$$\mathcal{X} = \{m_i(\mu_i)\}_{i=1,\ldots,r} \tag{5.32}$$

and,

$$\mathcal{Y} = \{m_j'(\mu_j')\}_{j=1,\ldots,p} \tag{5.33}$$

then,

$$\mathcal{X} \cap \mathcal{Y} = \{m_{ij}''(\mu_{ij}'')\}_{i=1,\ldots,r;j=1,\ldots,p} \tag{5.34}$$

where,

$$m_{ij}'' = m_i \cap m_j' \tag{5.35}$$

with,

$$\mu_{ij}'' = min\{\mu_i, \mu_j'\} \quad or \quad \mu_{ij}'' = \mu_i \cdot \mu_j' \tag{5.36}$$

depending on whether the *min* or the *dot* interpretation is used.

Case 4: The union $\cup$ is also a binary operator. Its result is also represented in the Cartesian-product space having the dimensions of the two operands as in case 3. Specifically, if

$$\mathcal{X} = \{m_i(\mu_i)\}_{i=1,\ldots,r} \tag{5.37}$$

and,

$$\mathcal{Y} = \{m'_j(\mu'_j)\}_{j=1,\ldots,p} \tag{5.38}$$

then,

$$\mathcal{X} \cup \mathcal{Y} = \{m''_{ij}(\mu''_{ij})\}_{i=1,\ldots,r;j=1,\ldots,p} \tag{5.39}$$

where,

$$m''_{ij} = m_i \cup m'_j \tag{5.40}$$

with,

$$\mu''_{ij} = max\{\mu_i, \mu'_j\} \quad or \quad \mu''_{ij} = min(\mu_i + \mu'_j, 1) \tag{5.41}$$

depending on whether the *max* or the "bounded sum" interpretation is used.

For example, consider the two standard components *robot*1, *robot*2. Suppose that their workloads are about 125% and 50% of their full capacity, respectively. They are evaluated to have the following status, according to the membership functions shown in Figure 5.2.

$$S_{robot1} = \{lo(0.75),\ mo(0.25)\};$$
$$S_{robot2} = \{lu(0.85),\ mu(0.85)\};$$

Then the proposition sets are:

$$\mathcal{S}_{robot1} = \{S_{robot1} \rightarrow lo(0.75),\ S_{robot1} \rightarrow mo(0.25)\};$$
$$\mathcal{S}_{robot2} = \{S_{robot2} \rightarrow lu(0.85),\ S_{robot2} \rightarrow mu(0.85)\};$$

The compound proposition of intersection of the two components is:

$$S_{robot1} \cap S_{robot2} =$$
$$\{S_{robot1 \to lo} \cap S_{robot2 \to lu}(0.75),$$
$$S_{robot1 \to lo} \cap S_{robot2 \to mu}(0.75),$$
$$S_{robot1 \to mo} \cap S_{robot2 \to lu}(0.25),$$
$$S_{robot1 \to mo} \cap S_{robot2 \to mu}(0.25)\}.$$

Here, the elements tell the membership degrees of the composed status. For example, the first element means the validity degree of the statement of "*robot*1 is lightly overloaded and *robot*2 is at light undercapacity", or the membership grade of the compound proposition for the specific measurements of S_{robot1} and S_{robot2}, at that particular status, is 0.75. The entire space represents the status of an FPS by considering the factors S_{robot1} and S_{robot2}. Elements in this space represent all possibilities of the system status. Any situation can be represented by this base using different degrees, which actually is a hypersurface in an $n+1$ dimensional space (n variables and an extra dimension for the validity value or membership grade).

To develop a procedure for the computation of compound propositions in Prolog, the following operators are defined for use with fuzzy logic operations.

```
op(300, xfy, ~).        %attribute of an object.
op(350, xfy, →).        %fuzzy proposition.
op(400, fy, ¬).         %fuzzy negation.
op(450, yfx, ∩).        %fuzzy intersection.
op(500, yfx, ∪).        %fuzzy union.
op(550, xfx, ⟶).        %fuzzy implication.
op(600, xfx, with ).    %rule belief.
```

Here, $op(P, R, N)$ defines an operator N at the level of precedence P and associated operand relation R. In R, f stands for the operator, and x and y stand for the operands, where x means it may contain only operators of lower precedence than that of f, and y may contain operators of the same level of precedence of f. Note that, operator symbols may be changed in actual coding. Here only the logic relations are shown.

Then, a compound situation of component status may be implemented by the predicate $situation(E, \mathcal{S})$ in which E is a fuzzy expression and $\mathcal{S}$ is the evaluated situation set.

$$
\begin{aligned}
&situation(O \sim X, S) \longleftarrow \\
&\qquad holds(O, X, FS), \\
&\qquad expand(O \sim X, FS, S). \\
&situation(\neg X, S) \longleftarrow \\
&\qquad situation(X, S1), \\
&\qquad expand\_not(S1, S). \\
&situation(X \cap Y, S) \longleftarrow \\
&\qquad situation(X, S1), \\
&\qquad situation(Y, S2), \\
&\qquad expand\_and(S1, S2, S). \\
&situation(X \cup Y, S) \longleftarrow \\
&\qquad situation(X, S1), \\
&\qquad situation(Y, S2), \\
&\qquad expand\_or(S1, S2, S).
\end{aligned}
$$

The predicates: *expand*, *expand_not*, *expand_and*, *expand_or*, are implementations of the four cases discussed in equations (5.26), (5.29), (5.34), and (5.39).

The usage of this predicate is illustrated by considering the following example: Suppose that, there are two components with workloads given by:

$$component(agv1, workstatus, [(mo, 0.8), (lo, 0.3)]).$$
$$component(agv2, workstatus, [(hu, 0.5), (mu, 0.7)]).$$

A query is issued for the situation by considering component status of $agv1$ and $agv2$.

$$?\text{-}situation(agv1 \sim workstatus \cap agv2 \sim workstatus, \mathcal{S}).$$

The answer would be:

$$\begin{aligned}\mathcal{S} = &[(agv1 \sim workstatus{\rightarrow}mo \cap agv2 \sim workstatus{\rightarrow}hu, 0.5),\\ &(agv1 \sim workstatus{\rightarrow}mo \cap agv2 \sim workstatus{\rightarrow}mu, 0.7),\\ &(agv1 \sim workstatus{\rightarrow}lo \cap agv2 \sim workstatus{\rightarrow}hu, 0.3),\\ &(agv1 \sim workstatus{\rightarrow}lo \cap agv2 \sim workstatus{\rightarrow}mu, 0.3)]\end{aligned}$$

where, $\mathcal{S}$ is the expanded space with elements representing all the possibilities of the situation.

5.4 Knowledge Representation and Reasoning

After recognizing the system status, a reasoning procedure has to be carried out through a set of rules (knowledge) in order to make a decision. A typical fuzzy-logic rule is of the form:

IF situation **THEN** action

A rule base gives a mapping from a situation set to an action set, as denoted by

$$\mathbf{R} : \mathcal{S} \Longrightarrow A \tag{5.42}$$

Then, for a given context (or situation) $\mathcal{S}$, the corresponding action set is obtained as

$$A = \mathbf{R} \circ \mathcal{S} \tag{5.43}$$

where $\mathbf{R}$= rule base; $\mathcal{S}$ = system situation represented in a situation space; $\Longrightarrow$ = mapping relation; $\circ$ = composition operator; and A= an action set.

An action set for restructuring is a set of crisp logic actions. The basic idea of fuzzy decision making is to evaluate all possible actions in the set and select the optimal one. For example, an action set of "load sharing" may contain:

$\{sharing(vision1, vision2);$
$sharing(vision1, vision4);$
$sharing(vision3, vision4)\}.$

Note that only one action can be chosen at a time. The fuzzy decision maker evaluates the priority degree of every action according to the current system situation, and then chooses the action with the highest assessment value, which comes from the priority degree and load transfer feasibility index.

5.4.1 Rule Belief

Given a particular situation, different people may make different decisions. It should be clear that different people have different belief degrees in their decision-making rules. It is the belief degree that provides the possibility of learning, so as to make a better decision.

In dealing with uncertainty, the membership grades of a fuzzy set and the belief factors of a rule could be considered similarly. They represent the same concept: every statement has a degree of belief; a membership grade represents a validity level of a proposition (fact) and a rule belief factor represents a confidence level of a rule. There is

no absolutely exact measure of situations in the world, and as a result, the recognition of the world is fuzzy. Also, there is no absolute truth; human knowledge evolves through learning and modification. When a rule is used for making a decision, its belief degree should be considered so as to give a confidence degree to the decision.

Consider the fuzzy-logic rule

$$\textbf{IF } A \textbf{ THEN } B \tag{5.44}$$

The decision based on this rule will vary due to several factors [14].

1. If the situation of A is different, then the decision will be different even if the membership functions of A and B are unchanged and the rule itself is unchanged.

2. If the definition of the fuzzy variable A changes, then the decision will also change even if the system data that is measured by A is unchanged and also B and the rule itself are unchanged.

3. If the definition of the fuzzy variable B changes, then the decision will also change even if the other aspects (as in 1 and 2 above) are unchanged.

4. If the validity of the rule itself changes, then the decision based on the rule should change even if other aspects (as in 1, 2, and 3 above) remain unchanged.

Note that in case 1, it is the "existing fact" (say $\overline{A}$) about the situation that has changed. This is not a case of "learning" however. In case 2, it is our understanding of the fuzzy variable A that has changed. This may be modeled by changing the membership function of A (i.e., $\mu_A(a)$). This may result from learning, new knowledge, expert opinion, etc. In case 3, it is our understanding about the fuzzy variable B that has changed. This may be modeled by changing the membership function of B (i.e., $\mu_B(b)$). This too may result from learning, etc. Finally, in case 4, it is the "level of belief" of the rule itself

that has changed. The definitions of A and B and also the measurement of A may remain unchanged. Here as well, the change may be a result of the learning of new facts, expertise, experience, etc. This may be modeled by assigning a "belief degree" to the rule, which may be adjusted on the basis of new knowledge.

Situation recognition ($\mu_A(a)$), and action set B (crisp logic) have been discussed for a restructuring system in the previous sections. The rule beliefs will be further addressed by using an illustrative example. Consider again the sharing match example in Section 5.1. Suppose we have a rule base of the form:

If X **is** *lo* **and** Y **is** *hu* **then** *sharing*(X, Y).

If X **is** *mo* **and** Y **is** *hu* **then** *sharing*(X, Y).

If X **is** *lo* **and** Y **is** *lu* **then** *sharing*(X, Y).

We should be able to match $C1$ (*lo*) with $C3$ (*hu*) since we have such a rule in the rulebase (rule 1). In actual decision making, however, it is better to match $C1$ with $C4$ (rule 3), which is at light undercapacity, and $C2$ should be matched with $C3$ (rule 2). Clearly, the three rules do not have the same weights, and they actually have different degrees of belief. We know that there would be a wastage of capacity if a lightly overloaded component is matched with a highly undercapacity component, and hence, this rule (rule 1) should be given a low belief degree. The other two rules will have higher priority in the decision making process. Consequently, the correct result can be deduced from the modified rulebase as shown below:

If X **is** *lo* **and** Y **is** *hu* **then** *sharing*(X, Y) with belief 0.7.

If X **is** *mo* **and** Y **is** *hu* **then** *sharing*(X, Y) with belief 0.95.

If X **is** *lo* **and** Y **is** *lu* **then** *sharing*(X, Y) with belief 0.9.

To facilitate the use of the concept of "rule belief", the following definition is given.

Definition 2. A fuzzy rule is an implication from a composed fuzzy situation to an action with a belief factor;

$$R : p(EXP) \longrightarrow a \textbf{ with } bf \tag{5.45}$$

where, **with** is defined as an operator having a precedence level higher than that of implication; p is a compound situation of a fuzzy expression EXP (a value of EXP); a is an action; and bf is the belief degree of the rule.

For example, the rule base given above can be expressed as

$X\text{→}lo \cap Y\text{→}hu \longrightarrow sharing(X, Y)$ **with** 0.7.

$X\text{→}mo \cap Y\text{→}hu \longrightarrow sharing(X, Y)$ **with** 0.95.

$X\text{→}lo \cap Y\text{→}lu \longrightarrow sharing(X, Y)$ **with** 0.9.

The belief degrees might be changed consequently, as the operators or experts find that the rules need to be modified. A learning procedure may be implemented to automatically change the belief values based on new knowledge or information. (see Figure 5.1)

Next, to elaborate how rule beliefs affect decision making, let us explore the combination of the membership degrees of facts (context) and rule beliefs in the reasoning process to make a decision.

5.4.2 Reasoning

As discussed before, fuzzy decision making requires a combined consideration of the membership grades of the context variables (which depend on their membership functions and current measurements) and rule beliefs. The objective is to select an action with highest validity degree. The key point here is to decide the "validity degrees" of the action set.

In general, each composed proposition in a situation context is associated with one or more actions (in the rule base). All associations with an action are counted together in determining the "validity degree" of this action.

Definition 3. Fuzzy associative memory: All rules associated with an action are fired simultaneously, and will contribute their beliefs toward the overall validity degree of the action. This is expressed as

$$VD(a) = FAM(\mathbf{R}(a) : EXP \longrightarrow A) \tag{5.46}$$

where, $VD(a)$ means the "validity degree" of an action a which is a specific value of the action variable (action set) A; FAM stands for "fuzzy associative memory", which is a function that computes the "validity degree" from the rule base according to the current situation expressed by EXP (which provides the membership grades of the situation); and $\mathbf{R}(a)$ is the instantiation of the rulebase $(\mathbf{R} : EXP \Longrightarrow A)$, for the specific action a.

FAM could be designed in two ways: "validity degree" (or "priority degree") of an action is either the maximum value or sum of contributions from all situation elements through corresponding rules (See Figure 2.6). If the summation is used for FAM, the "max-min" operation in equations (5.36) and (5.41) should be replaced by the "plus-dot" operation, for consistency. Figure 5.3 gives a decision making procedure for selecting an optimal component pair for sharing or releasing.

Note that, in Figure 5.3, the long arrows with bfs may be chains of rules obtained by a proof procedure, as well as direct rules. The decision making procedure is implemented by the predicate *decision*(E, A) which selects an action A with the highest priority degree from all possible actions according to the situation expressed by E and the installed decision-making knowledge (rules):

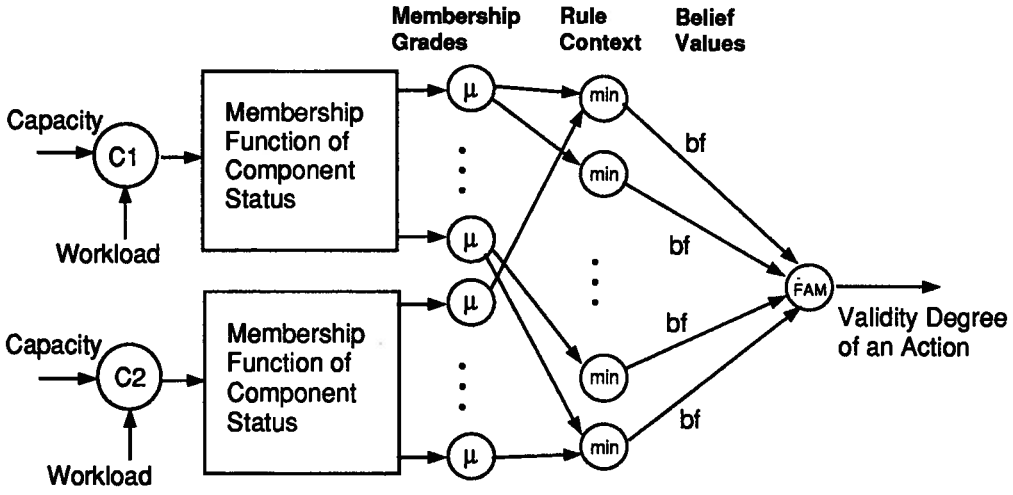

Figure 5.3: Computation of the Validity Degree of Actions

$decision(E, \_) \longleftarrow$

$situation(E, S)$,

$fam(S, A, VD)$, % computation of validity level.

$assessment(A, VD, V)$,

$V > 0.5$, % set threshold value at 0.5.

$assert(action(A, V))$, % store a might-be action.

$fail$. % find all possible actions.

$decision(\_, A) \longleftarrow$

$optimal\_action(A)$, % use the stored actions.

$retractall(action(\_, \_))$.

where $V > 0.5$ sets a threshold value for the assessment level of acceptable actions. $fam(S, A, VD)$ is the implementation of equation (5.46); $assessment(A, VD, V)$ evaluates the action A by the assessment value V from its validity level and the feasibility index; and the $optimal\_action(A)$ selects an action A corresponding to the highest priority degree. These procedures are shown below:

$fam([], \_, 0).$

$fam([(EX, G)|T], A, VD) \longleftarrow$

$EX \longrightarrow A$ **with** $BF, !,$ % find a rule.

$fam(T, A, VDT),$

VD *is* $max(VDT, G * BF).$ % "max" may be replaced by plus.

$fam([\_|T], A, PDT) \longleftarrow$

$fam(T, A, PDT).$ % no rule, no contribution.

Here, if there is a rule that matches a situation proposition, than the rule is fired. Its belief value BF is multiplied by the validity value G of the situation proposition, and contributed to the validity degree VD of action A according to the fuzzy inference method. If there is no rule for the current situation proposition, the VD of the action is retained for further firing rules. The end condition is that the situation space is empty, which corresponds to zero validity value.

$optimal\_action(A) \longleftarrow$

$findall(X, action(X, \_), As),$

$best(As, A, \_).$

Now an example is given to illustrate the decision making process. Suppose that we have a database in the blackboard of FPS component status.

$component(vision1, workstatus, [(hu, 0.8), (mu, 0.3)]).$

$component(vision2, workstatus, [(lu, 0.9), (mu, 0.1)]).$

$component(vision3, workstatus, [(mu, 1.0)]).$

$component(vision4, workstatus, [(hu, 0.8), (mu, 0.3)]).$

$component(robot1, workstatus, [(hu, 0.6), (mu, 0.4)]).$

$component(robot2, workstatus, [(lu, 0.9), (mu, 0.1)]).$

$component(robot3, workstatus, [(mu, 1.0)]).$

$component(robot4, workstatus, [(mu, 1.0)]).$

$component(agv1, workstatus, [(lu, 0.8), (mu, 0.2)]).$

$component(agv2, workstatus, [(hu, 0.7), (mu, 0.3)]).$

$component(agv3, workstatus, [(mo, 0.7), (lo, 0.3)]).$

$component(agv4, workstatus, [(mo, 0.7), (lo, 0.3)]).$

$component(agv5, workstatus, [(hu, 0.7), (mu, 0.3)]).$

$component(agv6, workstatus, [(lu, 0.8), (mu, 0.2)]).$

Here, for example, $agv3$ is overloaded, and a component is sought for sharing. The action set is:

$$\{sharing(agv3, C)\}_{C \in AGV}$$

where AGV is a group name. This can be accomplished by the query:

$?\text{-}decision(agv3 \sim workstatus \cap C \sim workstatus, sharing(agv3, C)).$

$C = agv5$

Note that the "plus-dot" operation (of fuzzy logic) is applied in this decision maker. All possible actions can be observed by examining the dynamic database "action/2". For example:

$action(sharing(agv3, agv1), 0.60).$

$action(sharing(agv3, agv2), 0.65).$

$action(sharing(agv3, agv5), 0.65).$

$action(sharing(agv3, agv6), 0.60).$

The data in the blackboard will be modified once the selected action is executed. A new decision will be made based on a new database.

5.5 Summary

This chapter presented a decision making method for selecting an optimal action in the third level of the restructuring system. A structure was discussed in which two steps were considered: situation recognition and rule synthesis.

Fuzzy logic was used for representing the component status. A general method for situation representation was presented and implemented in Prolog. In decision making, all rules associated with an action were fired simultaneously. The combined consideration of situation validity level and rule belief have been employed in the assessment of an action. The action with the best assessment value (which is a product of the priority value and the feasibility index) was selected as the final decision.

Chapter 6

Implementation and Case Study

Thus far in the thesis, the methodology of knowledge-based restructuring of a flexible production system has been developed. The present chapter will give some practical applications of the developed approach.

Section 6.1 will describe a model of an automated fish processing plant, which is introduced here as a case study. Section 6.2 will discuss some implementation problems, and give a panorama of problem solving. In the remainder of the chapter, several restructuring cases will be given, in which the simulation results will be graphically illustrated. Finally, a short summary of the chapter will be given.

6.1 A Fish Processing System

Figure 6.1 shows a model of an automated fish processing plant. There are three workcells in the system: a fish head cutting workcell, a fish grading workcell, and a packaging workcell which have three types of interchangeable component: vision stations, robots, and AGVs (Automated Guided Veicles). It is assumed that the processing activity level of each workcell is determined by the components of these types. In other words, the capacity of the above mentioned components is a bottleneck for the production activity of the plant. Under normal production conditions, the components are assigned to the workcells as follows:

- Cutting workcell: Vision1, Vision2, Robot1, AGV2, AGV5;

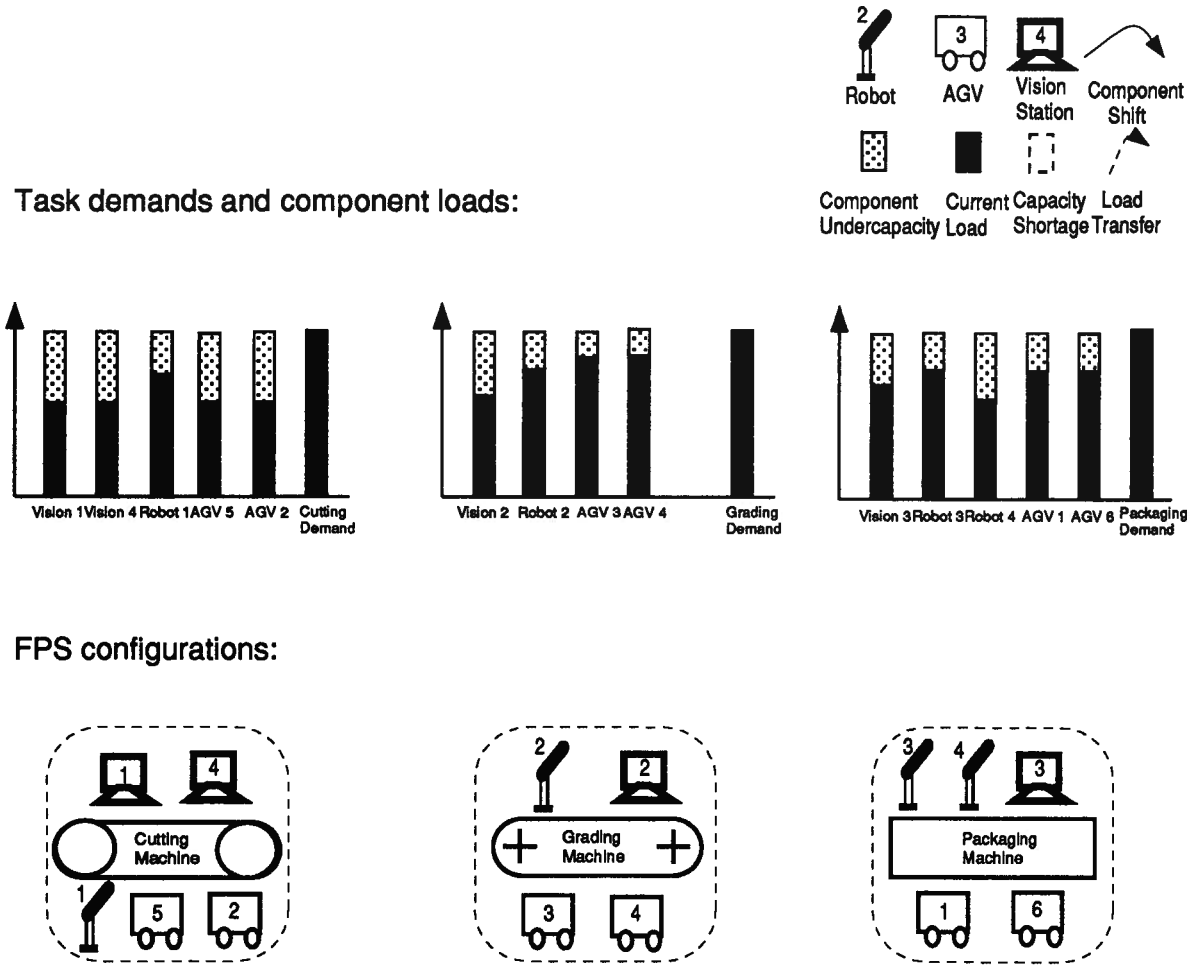

Figure 6.1: A Model of a Flexible Production System (FPS)

- Grading workcell: Vision4, Robot4, AGV1, AGV6;
- Packaging workcell: Vision3, Robot2, Robot3, AGV3, AGV4.

The normal operating conditions, which may be considered as the initial stage for restructuring, is illustrated by the chart of Figure 6.1, in which, the dotted and the solid areas indicate the capacity and the load, respectively, of a component. The component loads are assigned according to the process demand (task demands) by the load planner. The demand levels are assigned to be 100% in this stage of normal operation, as shown in the figure. Therefore, the loads in the normal production stage are actually representative of the load planning factors.

6.2 Implementation and Operation of the Restructuring System

6.2.1 Rule Bases

In the fuzzy decision making level, rulebases of load sharing and component releasing are given in an object level, which means that the rules are designed separately from the decision making procedure. The rule base of load sharing is represented as:

$OC{\rightarrow}ho \cap UC{\rightarrow}na \longrightarrow sharing(OC, UC)$ **with** 0.9.

$OC{\rightarrow}ho \cap UC{\rightarrow}hu \longrightarrow sharing(OC, UC)$ **with** 1.0.

$OC{\rightarrow}ho \cap UC{\rightarrow}mu \longrightarrow sharing(OC, UC)$ **with** 0.9.

$OC{\rightarrow}ho \cap UC{\rightarrow}lu \longrightarrow sharing(OC, UC)$ **with** 0.7.

$OC{\rightarrow}mo \cap UC{\rightarrow}na \longrightarrow sharing(OC, UC)$ **with** 0.9.

$OC{\rightarrow}mo \cap UC{\rightarrow}hu \longrightarrow sharing(OC, UC)$ **with** 0.9.

$OC{\rightarrow}mo \cap UC{\rightarrow}mu \longrightarrow sharing(OC, UC)$ **with** 0.7.

$OC{\rightarrow}mo \cap UC{\rightarrow}lu \longrightarrow sharing(OC, UC)$ **with** 0.7.

$OC{\rightarrow}lo \cap UC{\rightarrow}na \longrightarrow sharing(OC, UC)$ **with** 0.6.

$OC{\rightarrow}lo \cap UC{\rightarrow}hu \longrightarrow sharing(OC, UC)$ **with** 0.6.

$OC{\rightarrow}lo \cap UC{\rightarrow}mu \longrightarrow sharing(OC, UC)$ **with** 0.7.

$OC{\rightarrow}lo \cap UC{\rightarrow}lu \longrightarrow sharing(OC, UC)$ **with** 0.7.

Note that there are two preconditions for the rules: 1. $OC \neq UC$; 2. OC and UC are of a same type. There are several methods to check the preconditions, one of which is to check the two components before calling the rule, another one is to add the preconditions as bodies of the rules in Prolog. The rulebase for component releasing is shown below:

$UR{\rightarrow}hu \cap UL{\rightarrow}hu \longrightarrow releasing(UR, UL)$ **with** 1.0.

$UR{\rightarrow}hu \cap UL{\rightarrow}mu \longrightarrow releasing(UR, UL)$ **with** 0.9.

$UR{\rightarrow}hu \cap UL{\rightarrow}lu \longrightarrow releasing(UR, UL)$ **with** 0.7.

$UR{\rightarrow}mu \cap UL{\rightarrow}hu \longrightarrow releasing(UR, UL)$ **with** 0.7.

$UR{\rightarrow}mu \cap UL{\rightarrow}mu \longrightarrow releasing(UR, UL)$ **with** 0.6.

$UR{\rightarrow}mu \cap UL{\rightarrow}lu \longrightarrow releasing(UR, UL)$ **with** 0.6.

$UR{\rightarrow}lu \cap UL{\rightarrow}hu \longrightarrow releasing(UR, UL)$ **with** 0.6.

$UR{\rightarrow}lu \cap UL{\rightarrow}mu \longrightarrow releasing(UR, UL)$ **with** 0.6.

$UR{\rightarrow}lu \cap UL{\rightarrow}lu \longrightarrow releasing(UR, UL)$ **with** 0.6.

The same preconditions apply here. Note that, the component property in the above rules is implicitly "the component status".

6.2.2 Separation of Knowledge from Reasoning Procedure

As the rule bases show, the knowledge about dynamic restructuring is represented independently of the use of the knowledge which is implemented by the inference engine, for example, fuzzy associative memory (fam). The same is true of the knowledge sources and the control unit in the first-level decision making; and also the heuristics and the search drive *restructuring* in the second level. This kind of separation has the advantages that the system could be expanded easily, and the reasoning procedure could be modified for different requirements. The fact that the knowledge is represented in a descriptive manner makes the knowledge-based system quite different from conventional programs.

6.2.3 Panorama of Restructuring

Restructuring begins at the first level (Figure 3.1) by checking the system status and deciding whether to call for a restructuring action. Basically, the blackboard system is activated by the changes of data which typically are a change in the task demand or a change in the status of FPS operation. These changes usually trigger the load distributing and system evaluating knowledge sources, so as to update the current component loads, load-transfer feasibility indices and component status. Any of these data changes will definitely call for a check to determine the possibility of restructuring.

Selecting an action is a procedure of search. Due to complexity of the problem and fuzziness of the final goal, normal search methods are not practical here. Heuristics are developed, which trim the search space, and limit the search space to only one kind of actions at a restructuring stage. For actions (of the same kind) in a trimmed space, a fuzzy decision maker provides the optimal decision.

After an action has been selected, the load to be transferred in the action will be determined. Then the action is executed, and the corresponding sharing and releasing relations will be recorded. This execution may make further changes in the blackboard, and in turn trigger some more knowledge sources, and the control will return to the blackboard (the first level).

This cycle repeats until the decision maker failed to pick an acceptable action. (A threshold of assessment value should be set for actions.) In this case, a reporting procedure (KS of "output") will be called. If there is no component overloaded, the system is desirable. This may mean to satisfy two conditions: 1. no component is overloaded; 2. no undercapacity pair could provide a component release — components work close to their full capacity. Consequently, the system restructuring succeeds. If there is still an overload, clearly, the restructuring has failed. To show the practical application of the approach, some restructuring case studies will be considered in the following sections.

6.3 Restructuring Due to Change of Demand

Any change to the demands causes the KS of "Load Planning" to trigger. This KS will be executed in the subsequent control cycle. It will then change the blackboards of "Workcell Information" and "Component Information", which will trigger the knowledge sources of "Updating of Sharing Feasibility" and "Restructuring". Due to the preset priorities, the KS of "Updating of Sharing Feasibility" will be fired first to modify the BB of "Sharing Feasibility". After all the presettings are made, the FPS will be restructured according to the information in the blackboard. The KS of "Restructuring" may change the blackboards of "Workcell Information" and "Component Information", and also may post restructuring commands on the blackboard. Restructuring feasibility will be checked. The change in the information about the workcells and components may

trigger some knowledge sources to confirm that the load has been properly distributed. The change of restructuring commands will trigger the output KS. Finally, the system will become idle again, for steady operation of the FPS.

As an example, suppose that the cutting demand drops by 40% due to a reduction in the raw material supply, (say at the end of a fishing season), and the grading demand is increased by 50% in order to reduce an existing backlog. Only the fish of high grade are packaged, and the low grade fish may be used for canning. Therefore, suppose that according to the current market demand, the packaging load is maintained at the previous level.

The conditions due to these changes in task demand are shown as "Phase II Transition" in Figure 6.2. Clearly, *Robot*2, *AGV*3, and *AGV*4 will be overloaded, resulting in a capacity shortage. Also *Vision*1, *Vision*4, *AGV*2, and *AGV*5 now operate well below their full capacity. The component status is updated as follows:

```
component(robot4, workstatus, [(lu,1)]).
component(vision1, workstatus, [(mu,0.8),(hu,0.6)]).
component(vision4, workstatus, [(mu,0.8),(hu,0.6)]).
component(robot1, workstatus, [(lu,0.8),(mu,1)]).
component(agv2, workstatus, [(mu,0.8),(hu,0.6)]).
component(agv5, workstatus, [(mu,0.8),(hu,0.6)]).
component(vision3, workstatus, [(ok,1)]).
component(robot3, workstatus, [(ok,1)]).
component(agv1, workstatus, [(ok,1)]).
component(agv6, workstatus, [(ok,1)]).
component(vision2, workstatus, [(ok,1)]).
component(robot2, workstatus, [(lo,1)]).
```

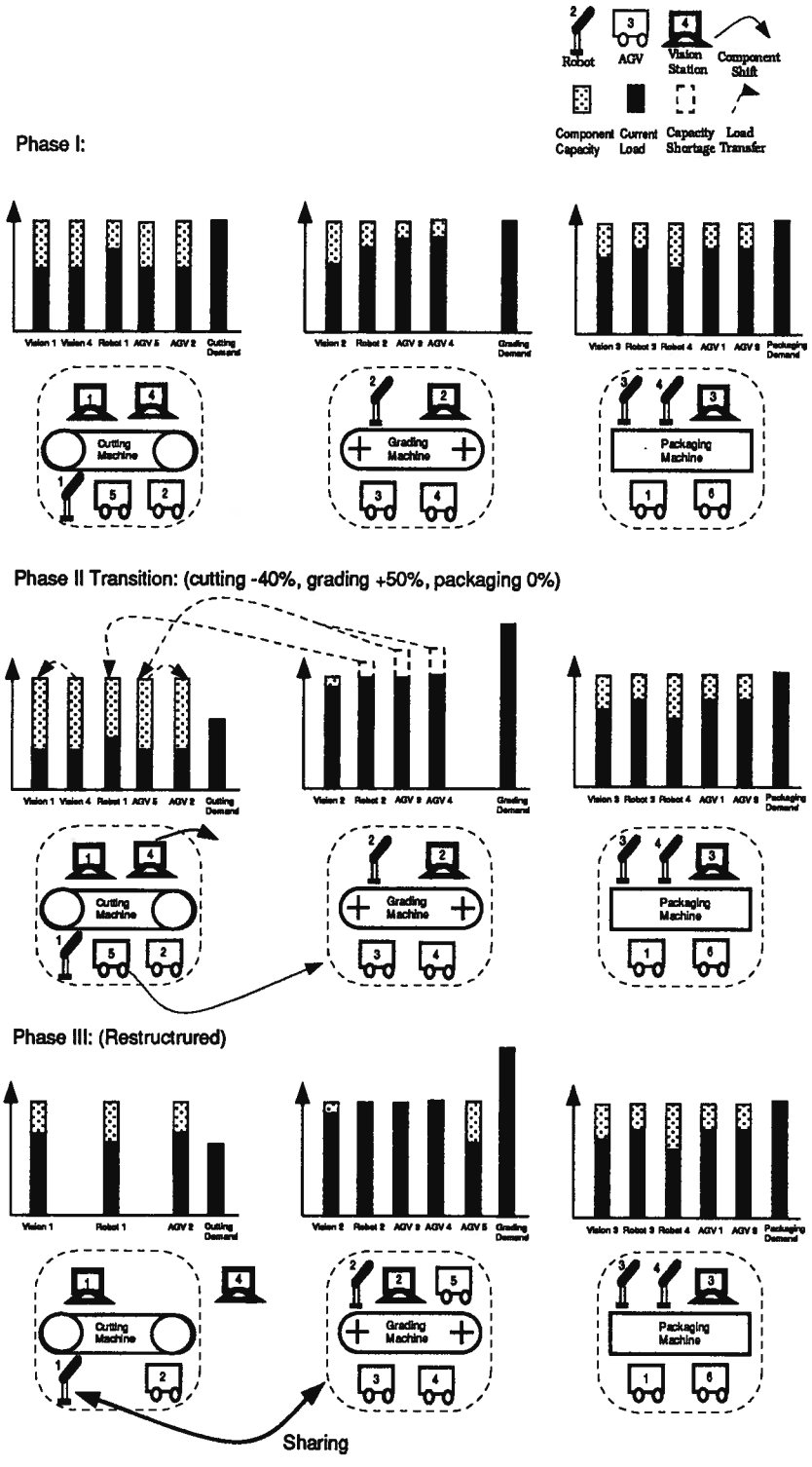

Figure 6.2: A Case Study: the Simulation of a Demand Change

```
component(agv3, workstatus, [(mo,0.75),(lo,0.25)]).
component(agv4, workstatus, [(mo,0.75),(lo,0.25)]).
```

This representation uses the predicate *component* which is declared in Section 3. The value of the *workstatus* is a fuzzy set. For example, $[(mo, 0.75), (lo, 0.25)]$ in $component(agv4, workstatus, [(mo, 0.75), (lo, 0.25)])$ means $agv4$ is moderately overloaded (mo) with a belief degree of 0.75, and lightly overloaded (lo) with a belief degree of 0.25.

Sharing feasibility is checked for pairs of component in a group of sharable components. Here, factors such as component reliability, geographical position of the components, and operating cost, will be considered. The following feasibility matrix is used here:

	robot1	robot2	robot3	robot4
robot1	0.00	0.92	0.90	0.81
robot2	0.92	0.00	0.95	0.85
robot3	0.90	0.95	0.00	0.90
robot4	0.81	0.85	0.90	0.00

The matrix gives the feasibility values of sharing workloads between pairs of robot. An example is that the feasibility value of sharing a workload between *robot*1 and *robot*2 is 0.92 in a scale of [0, 1].

Now, the KS of "Restructuring" is triggered. The following are some intermediate results showing the decision making procedure:

(1). Component releasing action.

```
FIND undercapacity components of vision: [vision1,vision2,vision3,vision4]
 To find all possible actions for vision1 within workcells.
  action: releasing(vision1,vision4)
```

```
      priority= 0.538
      feasibility = 0.9
      assessment = 0.48
 To find all possible actions for vision2 within workcells.
 To find all possible actions for vision3 within workcells.
 To find all possible actions for vision4 within workcells.
  action: releasing(vision4,vision1)
      priority= 0.538
      feasibility = 0.9
      assessment = 0.48
The best action is releasing(vision4,vision1);  assessment = 0.48
This action can be executed!
Load (transferred) for action releasing(vision4,vision1) is: 36.0
*** execute action:releasing(vision4,vision1)
```

Here, the procedure shows how the restructuring planner makes a decision using the heuristic of releasing a component within a workcell. A group of undercapacity vision stations is found, and then the possible actions of releasing a vision station are assessed. This is done by mutipling the sharing feasibility value by the priority value, which is obtained by using the technology "fuzzy associative memory". A threshold value that is appropriate for this type of actions is 0.4. Note that, the threshold value depends on the definition of membership function of priority. Finally, an action will be selected for executing, if its assessment value is higher than the treshold. In this case, the $releasing(vision4, vision1)$ is chosen. The load of the to-be-released component will be transferred to the other one.

(2). Component sharing action.

```
FIND all overloaded components of robot: [robot2]
 To find all possible actions for robot2 between workcells.
  action: sharing(robot2,robot1)
       priority = 0.4
       feasibility = 0.92
       assessment = 0.368
  action: sharing(robot2,robot4)
       priority = 0.4
       feasibility = 0.855
       assessment = 0.342
The best action is sharing(robot2,robot1);  assessment = 0.368
This action can be executed!
Load (transferred) for action sharing(robot2,robot1) is: 20.0
*** execute action:sharing(robot2,robot1)
```

This is similar to the decision making process that was used for component releasing. The differences are: 1. The threshold is set to a much lower value (here, it is 0.2), because any overloaded component should be shared; 2. The load to be transferred is determined by the work status of both components.

Similar procedures are used for other actions. At the end, none of the actions will be executed since their assessment values would below the threshold value. Then the planner will check the blackboard, calculate the cost function, and report the restructuring result:

```
PLANNING RESULT
plan:releasing(vision4,vision1)
plan:releasing(agv5,agv2)
plan:sharing(robot2,robot1)
```

```
plan:shifting(agv5,from(cutting),to(grading))
plan:sharing(agv4,agv5)
plan:sharing(agv3,agv5)
Restructuring succeeds;    The cost function is: J=0.729.
```

The final result of restructured system is illustrated as Phase III in Figure 6.2.

6.4 Restructuring Due to Change of Sharing Feasibility

An example is given now to show how the feasibility values can affect the restructuring results. The same three-workcell fish processing system as in the previous case study, and the same initial (normal) operating conditions are used here. This time, however, suppose that *Robot*1 is partly damaged (say, its network-communication system is operating at a reduced speed). Consequently, its sharability is assumed to be reduced, as shown in the feasibility matrix below:

	robot1	robot2	robot3	robot4
robot1	0.00	0.28	0.27	0.24
robot2	0.28	0.00	0.95	0.85
robot3	0.27	0.95	0.00	0.90
robot4	0.24	0.85	0.90	0.00

The feasibility change is considered when the planner selects a robot for sharing the load of *Robot*2. The decision making procedure is outlined below:

```
FIND all overloaded components of robot: [robot2]
 To find all possible actions for robot2 between workcells.
  action: sharing(robot2,robot1)
       priority = 0.4
```

```
        feasibility = 0.276
        assessment = 0.11
  action: sharing(robot2,robot4)
        priority = 0.4
        feasibility = 0.855
        assessment = 0.342
The best action is sharing(robot2,robot4);  assessment = 0.342
This action can be executed!
Load (transferred) for action sharing(robot2,robot4) is: 20.0
*** execute action:sharing(robot2,robot4)
```

The above procedure is quite similar to the previous one, but with a different result, since the sharing feasibility of the pair $(robot2, robot1)$ has been reduced. As a result, the second action, $sharing(robot2, robot4)$, is chosen instead of $sharing(robot2, robot1)$.

The final result is shown in Figure 6.3, and the output is given below:

```
PLANNING RESULT
plan:releasing(vision4,vision1)
plan:releasing(agv5,agv2)
plan:sharing(robot2,robot4)
plan:shifting(agv5,from(cutting),to(grading))
plan:sharing(agv4,agv5)
plan:sharing(agv3,agv5)
Restructuring Succeeds;     The cost function is: J=0.777.
```

The higher value of the cost function for this second case is justifiable in view of the degraded performance of a workcell component. Also, note that the final outcome (Phase III of Figure 6.3) now is somewhat different from what was realized in the previous case

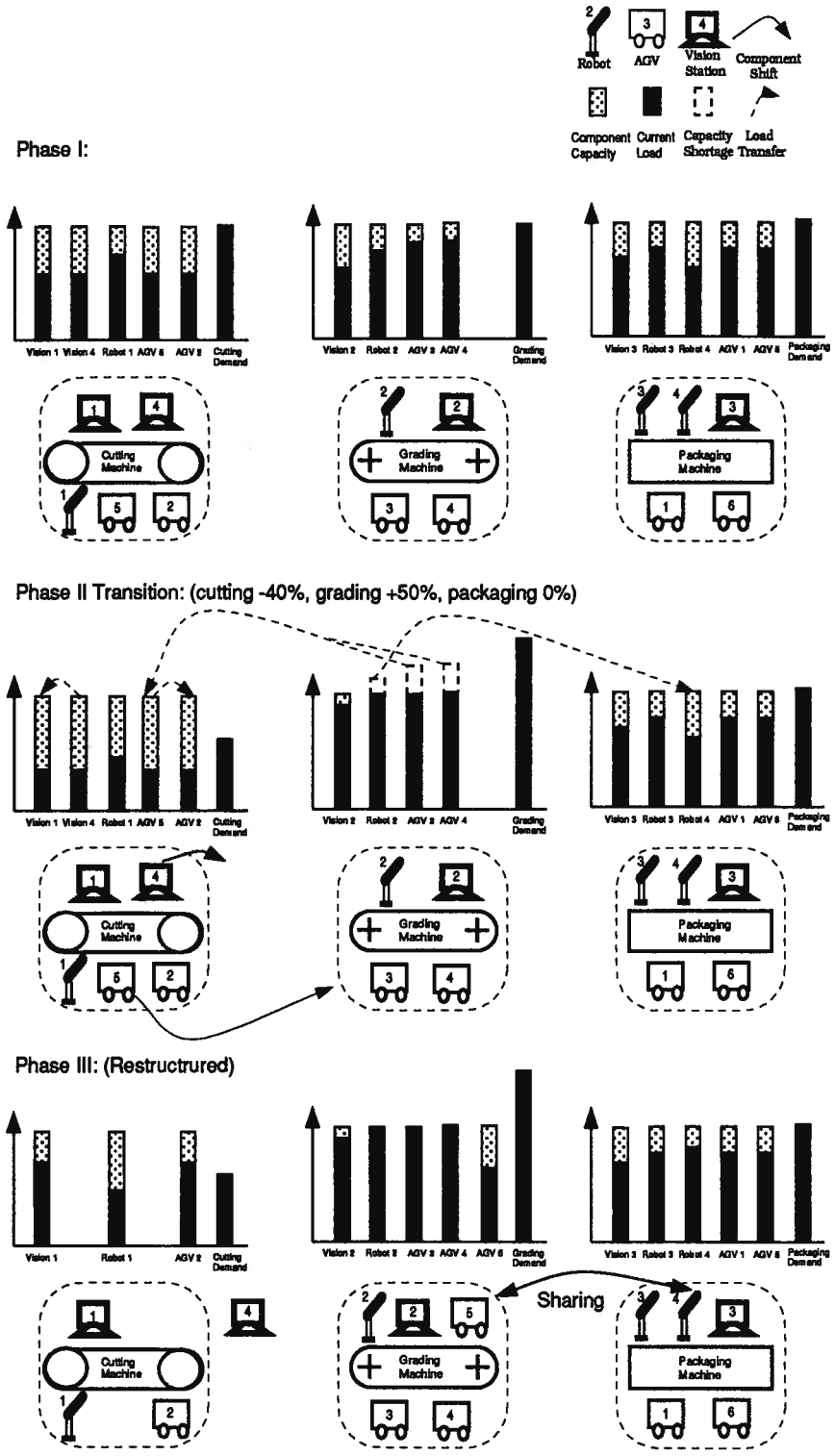

Figure 6.3: A Case Study: the Simulation of a Change of Sharing Feasibility

(Phase III of Figure 6.2) for the same starting and transition conditions, which is a direct result of the change in feasibility parameters of component sharing.

6.5 Summary

A full view of restructuring of a flexible production system was given, which explained the mechanism of decision making of dynamic restructuring. Computer simulations were given to demonstrate the application of the method to a fish processing system. What should be particularly mentioned is that, since the fuzziness of a goal was introduced, the restructuring system acquired some tolerance to non-desirable situations, so as to avoid too frequent restructuring.

Chapter 7

Conclusions and Future Work

This chapter will give a summary of the thesis. The main contribution of the research will be highlighted. It will also indicate some possible work for future study on the present topic.

7.1 Conclusions

A novel method for use in the high-level automation of production systems — dynamic restructuring of flexible production systems — has been developed in this thesis. The problem has been formulated as one of reassociating components to workcells, in order to deal with changes in system operating conditions. Uniformity and optimality of process operation is targeted in this manner, through system restructuring. The fuzziness of a restructuring goal, which results in multiple solutions, is particularly emphasized. This feature determines the use of data-driven proofs and the requirement of optimization in problem solving.

A three level decision making system is proposed by considering fuzziness, complexity and non-analytical nature of the problem. The problem solver makes programmatic decisions and detailed decisions in different levels, thereby making the system direct, reliable, somewhat modular, and capable of utilizing different methods of knowledge representation and reasonings.

The first level adopts a blackboard architecture, in which different types of knowledge sources are employed for updating of system information, evaluation of system status

and decision making for restructuring. The independence of knowledge sources provides a dexterity or flexibility in knowledge representation, and enables the use of various reasoning methods. Therefore, the system is readily extendable and maintainable. The mechanism of opportunistic reasoning is quite helpful in coping with complexity and variety of system operating conditions.

Heuristics for different cases are developed in the second level in which an inference engine directly applies heuristics, according to their order of priority.

The third level makes the most detailed decisions on selecting a specific action using fuzzy sets and associated technologies. Both the system status and the restructuring knowledge (rules) are evaluated with respect to a validity degree or belief value. The fuzzy decision making is based on the assessment of actions which uses techniques of fuzzy associative memory.

Knowledge is represented separately from the reasoning procedure which uses that knowledge. Also, different reasoning strategies are employed for different cases. Blackboard system provides opportunistic reasoning so as to adapt the system to various, unexpectable events. In the second level of decision making for restructuring, an ordered search is quite suitable. This method, together with the third-level decision making, gives a practical and somewhat optimal approach in selecting actions for system restructuring.

The dynamic restructuring system has been implemented in Prolog. Case studies have been presented that applies the technique to a flexible fish processing system, in computer simulation. The results were found to be quite encouraging.

7.2 Main Contributions

The main contribution of the research is in the development of a new technique that will assist in high-level automation of flexible production systems. In particular, a technique

for dynamic restructuring of flexible production systems has been developed. From the methodology development to detailed implementation, the thesis emphasizes on emulation of human problem solving through the means of advanced technologies in artificial intelligence, control engineering, and fuzzy logic. In particular, the following contributions have been made:

1. Formulated and represented the dynamic restructuring problem;
2. Developed or identified proper problem solving methods, by considering special characteristics of the problem;
3. Developed a problem solving architecture;
4. Successfully adopted blackboard techniques to the dynamic restructuring problem;
5. Abstracted effective heuristics for the restructuring problem;
6. Designed a fuzzy decision making system for the restructuring; and
7. Implemented the developed techniques through computer simulation of an industrial process, demonstrating their use in high-level automation.

7.3 Future Work

Not addressing the complement research on related knowledge sources (KS), such as sensing and capacity planning, the developed approach for dynamic restructuring still needs to be improved in several ways.

A learning mechanism could be used for training rule belief values for better decision making in selecting a proper action. A basic idea would be to give a full set of rules in which those practically impossible (to human) rules are initially assigned zero belief

degrees. Therefore, adding rules and retracting rules may be incorporated as changes in belief values. Fuzzy neural networks could be employed for this purpose.

Since fuzzy membership functions are represented as trapezoids (see figure B.1) by specifying the four corners in the form of a list in Prolog, it is difficult to have a universal representation of arbitrary membership functions in this format. Fuzzy logic operations could be represented in C code to avoid this limitation of Prolog.

System integration is another major project which has to be carried out in any practical implementation of the approach. Component sharings should be scheduled properly, and hand-shaking signals and other communication should be well planned and designed. The transfer of the developed technology to appropriate industries is also an important aspect of this research, which needs to be carried out.

Bibliography

[1] Åström, K.J. and B. Wittenmark, (1989), *Adaptive Control*, Addison-Wesley, Reading, Mass.

[2] Bench-Capon, T.J.M., (1990), *Knowledge Representation : an Approach to Artificial Intelligence*, Academic Press, San Diego, CA.

[3] Campbell, J.A. and J. Cuena, (1989), *Perspective in Artificial Intelligence*, Ellis Howwood Ltd, GB.

[4] Chang, T.C., (1990), *Expert Process Planning for Manufacturing*, Addison-Wesley Publishing Company.

[5] Chryssolouris, G., M. Dpmroese, and P. Beaulieu, (1992), Sensor synthesis for control of manufacturing processes, *ASME Trans. Engineering for Industry*, Vol. 114, pp158.

[6] de Silva, C.W., A.G.J., MacFarlane, (1989), *Knowledge-Based Control with Application to Robots*, Springer-Verlag.

[7] de Silva, C.W., (1991a), Fuzzy information and degree of resolution within the context of a control hierarchy, *Proc. IEEE IECON'91*, Vol.2, pp1590-95.

[8] de Silva, C.W., (1992), Research laboratory for fish processing automation, *International Journal of Robotics and Computer-Integrated Manufacturing,* Vol. 9, pp49-60.

[9] de Silva, C.W., (1993a), Soft automation of industrial processes, *Engineering Applications of Artificial Intelligence*, Vol. 6, pp. 87-90.

[10] de Silva, C.W., (1993b), Knowledge-based dynamic structuring of process control systems, *Proc. Fifth International Fuzzy Systems Association World Congress*, Seoul, Korea, Vol.II, pp1137-1140.

[11] de Silva, C.W., (1993c), Hierarchical processing of information in fuzzy logic control applications, *Proc. 2nd IEEE Conference on Control Applications*, Vancouver, BC, Vol.1, pp. 457-461.

[12] de Silva, C.W., and J. Gu (1994), An intelligent system for dynamic sharing of workcell components in process automation, *Engineering Applications of Artificial Intelligence* (In Press).

[13] de Silva, C.W., (1994a), *Intelligent Control: Fuzzy Logic Applications*, CRC Press, Boca Raton, FL. (To be published).

[14] de Silva, C.W., (1994b), Automation Intelligence, *Engineering Applications of Artificial Intelligence*, Vol.7, 1994 (In Press).

[15] Dubois, D. and H. Prade, (1980), *Fuzzy Sets and Systems*. Academic Press, Orlando, FL.

[16] Engelmore, R.S. and T. Morgan (1988) (eds), *Blackboard System*, Addison-Wesley, Reading, Mass.

[17] Fayyad, K.E. and R. Kass, (1991), Task dependency modeling to support assembly plan design, *Proc. IEEE 7th Conference on AI Application*, pp212-216.

[18] Georgeff, M.P. (1987), Planning, in J. Allen, J. Hendler and A. Tate (eds), *Readings in Planning*, Morgan Kaufmann, San Mateo, CA, 2:359-400.

[19] Gruver, W.A., (1994), Intelligent robotics: an applications overview, *IEEE Trans. on Industrial Electronics*, Feb. 1994.

[20] Gu, J. and C.W., de Silva, (1994a), A procedure for dynamic sharing of components in multiple-workcell process plants. *Proc. 1994 American Control Conference*, Baltimore, MD, Vol. 1, pp. 289-93.

[21] Gu, J. and C.W. de Silva (1994b), Dynamic restructuring of flexible production systems. *Proc. IFAC Symp. on Intelligent Components and Instruments for Control Applications* , Budpest, Hungry, pp.71-76.

[22] Gu, J. and C.W. de Silva (1994c). Fuzzy decision making in variable-structure control systems. *Proc. ASME'94 Congress*, Chicago, IL. (In press).

[23] Iizuka, Y. and H. Tsuji, (1988), A computer system configuration design expert system: IDEA/C, *Proc. International Workshop on AI for Industrial Application*, pp442-447.

[24] Jaeschke, R., (1993), *C++ : An Introduction for Experienced C Programmers*, CBM Books, Horsham, PA.

[25] Jagannathan, V., R. Dodhiawala and L.S. Baum (eds) (1989), *Blackboard Architecture and Applications*, Academic Press, Boston.

[26] Janko, W.H., M. Roubens, and H.J. Zimmermann, (eds) (1989), *Progress in Fuzzy Sets and Systems*, Kluwer Academic, Boston.

[27] Kosko, B., (1992), *Neural Networks and Fuzzy Systems: A Dynamic Systems Approach to Machine Intelligence*, Prentice Hall, Inc., NJ.

[28] Kosko, B., (1993), *Fuzzy Thinking : The New Science of Fuzzy Logic*, Hyperion, New York.

[29] Kovacs, G. and I. Mezgar, (1991), Expert systems for manufacturing cell simulation and design, *Engineering Application of Artificial Intelligence*, Vol.4, pp417-424.

[30] Leong, K.T., S.K. Sim, and Y.W. Chan, (1991), BESMED: a blackboard knowledge-based approach to integrate mechanical design, *Engineering Application of Artificial Intelligence*, Vol.4, pp205-220.

[31] Levesque, H.J., (1986), Knowledge representation and reasoning, in *Annual Review of Computer Science*, 1:255-87.

[32] Luger, G.F. and W. A. Stubblefield, (1992), *Artificial Intelligence : Structures and Strategies for Complex Problem Solving*, Benjamin/Cummings Pub. Co., Redwood, CA.

[33] Newell, A., (1962), Some problems of the basic organization in problem-solving programs, *Proceedings of the Second Conference on Self-Organizing Systems*, pp393-423, Spartan Books.

[34] Nicholson, H., B.H., Swanick, (eds) (1985), *Self-Tuning and Adaptive Control: Theory and Applications*, Peter Peregrinus Ltd, New York.

[35] Pang, G.K.H, (1989), A blackboard system for the off-line programming of robots, *Journal Intelligent Robotic Systems.* Vol. 2, pp. 425-44.

[36] Pang, G.K.H, (1991), A framework for intelligent control, *Journal Intelligent Robotic Systems.* Vol. 4, pp. 503-12.

[37] Pau, L., (1989), Knowledge representation approaches in sensor fusion, *Automatica*, Vol. 25, pp. 207-14.

[38] Poole, D., A. Mackworth, and R. Goebel, (1993), *Computational Intelligence : A Logical Approach*, Department of Computer Science, University of British Columbia, (draft).

[39] Reiter, R., (1991), The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression, in V. Lifschitz (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, pp. 359-380.

[40] Saridis, G.N., (1977), *Self-Organizing Control of Stochastic Systems*, M. Dekker, New York.

[41] Simon, H.A., (1977), Scientific discovery and the psychology of problem solving, *Models of Discovery,* Reidel, Boston, MA.

[42] Sterling, L. and E. Shapiro, (1986), *The Art of Prolog: Advanced Programming Techniques*, The MIT Press.

[43] Tempelmeier, H., H., Kuhn, (1993), *Flexible Manufacturing Systems : Decision Support for Design and Operation*, Wiley, New York.

[44] Turner, R., (1984), *Logics for Artificial Intelligence*, Halsted Press, New York.

[45] Vollmann, T.E., L.B. William, and D.C. Whybark, (1992), *Manufacturing Planning and Control Systems*, Business One Irwin, 3rd Edition.

[46] Zadeh, L.A. (1965). Fuzzy sets. *Information and Control*, Vol. 8, pp. 338.

[47] Zadeh, L.A. and J. Kacprzyk, (eds) (1992), *Fuzzy Logic for the Management of Uncertainty*, Wiley, New York.

[48] Zhou, Q.J. (1985). The robustness of an intelligent controller and its performance. *Proc. IEE International Control Conference*, Vol. 1, pp. 429-33.

Appendix A

Logic Programming

Prolog which stands for PROgramming in LOGic is an implementation language of logic representation and reasoning of knowledge using first order predicates. Essentially, Prolog allows the writing of programs as sets of "Horn Clauses" [42, 44] (see next section for a definition), and executes these programs by means of resolution, applied in a top-down manner, the effect of which is similar to the operation of a goal driven system, using a depth-first search strategy.

A.1 Horn Clauses

Horn Clauses are a subset of first-order predicate calculus. They are sentences in clausal form but which contain at most one literal on the left hand side (LHS). In particular, the following are all Horn Clauses:

Clause 1: $P \longleftarrow Q \cap R$

Clause 2: $P \longleftarrow$

Clause 3: $\square \longleftarrow Q$

Clause 4: $\square \longleftarrow$

Horn Clauses are important to logic programming because they are much more computationally tractable than full clausal forms. Consequently, there is a strong incentive to sacrifice the expressiveness of full clausal forms to achieve these computational gains.

A.2 Syntax of Prolog

1. Conjunction of predicates is written as a comma ',';
2. Disjunction of predicates is written as a semicolon ';';
3. All clauses end with a period '.';
4. Constants and predicate names begin with a lower case letter;
5. Variables begin with an upper case letter;
6. Parameters to predicates are enclosed in brackets and separated by commas;
7. Lists of terms are written in square brackets and the terms are separated by commas;
8. The symbol '|' is used to separate the tail of a list.

A.3 Resolution and Unification

When a query is issued, the system tries to prove the goal (the query) by a process of search and unification. Prolog selects clauses by trying them in the order they appear in the database, and evaluates the clauses in the body of the Horn Clause from left to right.

For example, suppose that we have the database:

$$father(tom, david).$$
$$parent(david, mary).$$
$$grandfather(X, Y) \longleftarrow father(X, Z), parent(Z, Y).$$

If the query is ?-$father(Who, david)$, the first clause will match the query by unifying Who to *tom*, so as to give an answer $Who = tom$. If "who is *mary*'s grandfather" is the query, i.e., ?-$grandfather(Who, mary)$, the third clause will be called to achieve the goal by instantiating Y to *mary* and unifying X to Who. Note that X and Who are still free variables, but if either of them is fixed to a constant, the other one should also be fixed to the same value. Now the predicates in the body of the third clause will be verified from left to right: $father(X, Z)$ will match $father(tom, david)$ by instantiating X to *tom* and Z to *david*. Thus, all variables will be instantiated. The second predicate in the body of the third clause will be instantiated to $parent(david, mary)$ which will succeed since it exists as a fact. Finally, the goal will be achieved by $Who = tom$.

In general, unification has the following forms:

1. A free variable unifies another free variable;
2. A free variable unifies a constant (numeric or symbolic), which is known as instantiating a variable to the constant;
3. Two identical constants always unify.

A.4 Prolog for AI Programming

Prolog provides a means to represent knowledge independent of the use of that knowledge. Only the logic relations are represented by a set of Horn clauses, whereas deductive reasoning is implemented by a built-in inference engine. As an example, a member of a list can be described as follow:

$$member(X, [X|T]).$$
$$member(X, [H|T]) \longleftarrow X \neq H, member(X, T).$$

The first clause gives an instance that if X is a head of a list, it is a member of the list; the second clause completes the description: if X is not the head of a list, but is a member of the tail, it is still a member of the entire list.

A.5 Non-logic Features

As a practical programming language, Prolog has some non-logic features. Specifically, it provides the following:

1. Input and output, which are necessary for user interface;
2. Database operations, which assert new knowledge to a database or retract old knowledge from the database;
3. Controls, which may cut (!) a branch in the search path or force a search to fail (fail);
4. Negation, which can be implemented as a failure of proof using "cut" and "fail".

$$not(P) \longleftarrow P, !, fail.$$
$$not(P).$$

Non-logic features make Prolog powerful in practical programming. However, abuse of these features may weaken the advantages of logic programming.

Appendix B

Intended Interpretation of Prolog Predicates

Some important Prolog predicates developed in the dynamic restructuring system are listed below with intended interpretations.

X→S1∩Y→S2⟶A: This is an objective level rule, which means A is the conclusion under conditions that component X has status $S1$ and component Y has status $S2$.

R with B: True if the rule R has the belief degree B.

ask(O, P, V): Asks the user the value V of property P of Object O.

assessment(A, VD, V): Provides an assessment value V of action A where VD (validity value) is one factor, and another one being the feasibility index is obtained from the blackboard.

assumable(O, P, V): True if object O holds the default value V for property P.

belief_calculate(X, S, P): True if P is the membership grade of X belonging to a fuzzy set S, which has the shape defined by $[A, B, C, D]$, as explained in the predicate **mem_function/2**.

best(As, A, PD): True if A is the best (in terms of priority or validity value) in the set As, where PD is the priority value of A.

best_action(CL, A, PD, Flag): To find the best action A for all components in a list CL according to the flag $Flag$, where PD is the priority degree of action A, and $Flag$ takes values "within" a workcell or "between" workcells.

centeroid(P, MF, A, M): True if A and M are area and the first moment of a fuzzy quantity P, where MF is the corresponding membership functions of P.

collect_triggered_KS(KSs): True if KSs is a list of all triggered knowledge sources.

combine(X, L, LN): True if LN is the combined list of X and a list L, where X and the elements of L and LN take the form (*fuzzySetName, membershipGrade*). If X already exists in L, use "max" interpretation to merge X into L. Otherwise, append X to L.

component(O, P, V): True if V is the value of property P of component O. (a blackboard)

control: Blackboard controller.

decision(E, A): To select an action A with the highest assessment value from all possible actions under the situation expressed by E, and using the installed decision making knowledge (rules).

defuzzify(PD, FL, PDV): True if PDV is the crisp value obtained through defuzzifying the fuzzy variable PD using membership functions FL.

determine_load(A, LD): Determines the load LD which is to be transferred in carrying out the action A.

distance((A,B), D): Calculates the distance D between the workcells A and B.

evaluate_workstatus: Functional procedure of the KS "evaluating component status".

execute_KS(KS): Executes a knowledge source *KS*.

expand(N, V, CT): Fuzzy variable N which has value V is expanded into a set of propositions CT.

expand_and(A,B,Z): True if Z is the result of the fuzzy conjunction of A, and B.

expand_not(A, B): True if B is a set of negative elements of A.

expand_or(A,B,Z): True if Z is the result of the fuzzy union of A, and B.

fam(S, A, VD): Calculates the validity value VD of the action A under conditions of S which is a proposition set in which all propositions are treated as parallel.

feasibility(P, V): True if V is the feasibility index of load sharing of a pair of components P, which has the form $(C1, C2)$ where $C1$ and $C2$ are components of the same type. (A blackboard)

fps: Functional procedure of the KS "updating of system information".

free(C): True if component C is load free.

fuzzify(V, MF, F): F is a fuzzy value obtained through fuzzification of the crisp value V based on a fuzzy descriptor of which the membership functions are MF.

get_property(O, P, V): To report the value V of property P which is associated with object O. It may instruct a user to complete the information of an object if the property value is required but not given in the database.

group(G): True if G is a group of components.

group_components(S, T, CL): True if CL is a list of all the components of type T which have the status S, where S is a flag indicating "overloaded" or "undercapacity" conditions.

heuristic(A): True if A is an action deduced from heuristics.

holds(O, P, V): True if object O holds property P at value V.

ks(KS, P, V): True if V is a value of property P associated with a knowledge source KS.

load_distributing: Functional procedure for the KS "workload planning", which generates the capacity requirements for the overall FPS.

make_virtual(RC, VC): Makes a virtual component VC which is a copy of all properties of the component RC, but with zero capacity.

mem_function(N, L): True if L is the membership functions of a fuzzy descriptor N. L is in the form of $[(fuzzySetName, [A, B, C, D])|Othersets]$ where A, B, C, and D are points which define the shape of the membership function of a fuzzy set named $fuzzysetname$. In particular, A corresponds to the beginning point of this set, B to the first point where its membership grade value is 1, C to the last point where its membership grade value remains 1, and D to the last point of this set. (see Figure B.1)

modify_cell_property(W, P, V): Changes the value of property P to V, which is associated with the workcell W, and triggers the knowledge sources which have inputs from this property blackboard.

modify_comp_property(O, P, V): Changes the value of property P to V, which is associated with component O, and triggers the knowledge sources which have inputs

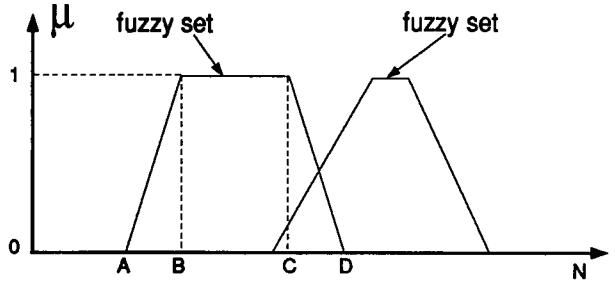

Figure B.1: The Representation of Membership Functions

from this property blackboard.

move(C, from(W), to(W1)): Moves the component C from workcell W to workcell $W1$.

move_component(C, W, W1): Moves component C from workcell W to workcell $W1$. (physical action)

not_overloaded(OC, L): True if component OC is not overloaded when the loads L (a list) are assigned to it.

op(P, R, N): Defines an operator N at precedence level P, and an associative relation R, where R has the form of xf, fx and xfx with f standing for operator and x for operand. Note that x may be replaced by y which can contain operators of the same level as P, while x can contain operators of lower precedence only.

optimal_action(A): True if A has the highest assessment value in the current database.

output: Functional procedure of the KS "output of restructuring plans".

overload(C1, F, C2): True if $C1$ and/or $C2$ are overloaded according to the flag F which takes the values "and"/"or".

plan(P): True if P is an action in the restructuring plan (A blackboard)

possible_actions(C, Flag): Finds and stores in a dynamic database all possible actions which can be applied to component C according to flag $Flag$, which has the values "within" and "between".

record(R): True if R is a recorded, sharing status reached due to a restructuring action.

releasing(C1, C2): Releases the component $C1$ by transferring its load to component $C2$.

released(VC, C2, Load): True if *Load* is the load transferred from a released component to $C2$, where VC is a virtual component, recording properties of the released component.

remove(R): Removes a record R.

remove_KS_trigger(KS): Sets the trigger status of the knowledge source KS to "off".

remove_zero_degree(S1, S2): True if $S2$ is the remaining part of the fuzzy value $S1$ after its all zero-grade elements are removed.

report_cost_function: Calculates the cost function of the restructuring system, and reports the value.

report_restructuring_failur(C): True if component C is overloaded.

report_restructuring_plan: Reports the designed restructuring plan.

restructuring: Functional procedure of the KS "restructuring".

select_highest_PR(KSs, KS): True if KS is the knowledge source with the highest priority among the knowledge sources KSs.

sharing(OC, UC): Shares load from component OC by component UC.

shared(OC, UC, Load): True if *Load* is the load shared by UC, which is originally assigned to OC.

situation(X, CT): True if CT is the expanded context of the fuzzy status X.

sum_up(L, LD): True if LD is the summation of the elements of the list L.

terminate(R): Terminates a recorded relation R which has been either a **"released/3"** or a **"shared/3"**.

transfer_load(C1, C2, L): Transfers the load L from component $C1$ to component $C2$. (a physical action)

trigger(KSs): Triggers all knowledge sources in the list *KSs*.

trigger_KS(KS): Sets the trigger status of the knowledge source KS to on.

trigger_property(P): Triggers the knowledge sources which have an input from the blackboard of property P.

update_feasibility: Functional procedure of the KS "updating of sharing feasibility".

user: Functional procedure of the KS "user interface".

workcell(W, P, V): True if V is the value of property P of workcell W (A blackboard).