A Rule-Based General AI Problem Solving Mechanism Supporting Real-Time and Cooperative Problem Solving

Mingshu Li[1]

Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, U.K
mingshu@dai.ed.ac.uk

Abstract

Rule-based systems, also called production systems, are one of the most well-known general AI problem solving mechanisms and have been widely used to build AI problem solving systems, but their adaptation to real-time and cooperative applications environments has been proven to be a formidable task. Based on analysing the major deficiencies in rule-based systems which make them unsuitable for real-time and cooperative applications, this paper presents a rule-based general AI problem solving mechanism with applications for real-time and cooperative problem solving, named as Function Module System (FM_S in short), and introduces some application examples.

1 Introduction

A working definition of artificial intelligence (AI) is that it studies and attempts to duplicate human problem solving methods [1]. Problem solving is the process of looking for a certain sequence of actions which will achieve some stated goal. Much AI research focuses on such a topic. However, most researchers conduct their research in terms of strategies, techniques and the realization of solving a particular problem, seldom thinking about what external functional characteristics should be possessed in a successful AI problem solving mechanism. In fact, the huge progress of AI in the last forty years has benefited greatly from cognitive psychology. Whereas cognitive psychology simply takes the human brain as a black box, exploring human cognitive methods through psychological phenomena and experiments, and studying regular patterns of human thinking through functional simulation. This suggests that the study of AI problem solving mechanisms should pay more attention to their external functional characteristics.

Function may be defined as what a component does in its respective environment [2]. Interest in function is currently being evinced in fields as varied as model based reasoning, vision, design, robotics, natural language processing and software debugging. It is distinct from component behaviour, the relation between the inputs and outputs of the component. Unlike behaviour, function is dependent on the environment of the component.

The structure of a device is a list of its components and their interconnections. The purpose of a component is the utility of the component to humans, and is meaningful only in a socio-cultural context. Function belongs between behaviour and purpose in the resolution with which it describes causality. Perhaps because of the amorphous nature of function, definitions abound on what it is, and how it should be treated in computation. It has been variously described as "an annotation of behaviour", "an abstraction of behaviour", "the relationship between the intended behaviour of

[1] On leave from Institute of Software, Chinese Academy of Sciences, P.O.Box 8718, Beijing 100080, P.R.China, *mingshu@ios.ac.cn*.

a system and the utilized behaviour of its component", "a statement about the intended state of a component, along with the conditions necessary to reach the state" and "the relation between the goal of a human user and the behaviour of the component". It should be noted that there are many similarities among these definitions, especially as indicators of what should necessarily be included in any successful function model.

When not meeting concrete AI problems, this paper intentionally confuses the conceptions of "*function*" and "*behaviour*" In another words, it takes function as a special behaviour. The term "*mechanism*" is used to mean a solving model and a programming language or tool based on the model. This is the reason for using the adjective "*general*" in the paper title.

As for the external functional characteristics needed in an AI problem solving mechanism: it should possess enough cognitive solving strategies so that it can take correct operational steps to complete the given tasks under the deterministic environment and produce a response to any real-time data sources after a (domain defined) fixed time has elapsed; at the same time, it should be able to organise one or more knowledge sources to solve a complex or huge problem cooperatively. Thus, an ideal general AI problem solving mechanism should display the external functional characteristics of *real-time* (abilities of real-time processing) and *cooperation* (abilities of cooperative solving), while possessing powerful abilities of problem solving and providing the distinct knowledge representation and effective inference mechanism.

This paper introduces and discusses such a general AI problem solving mechanism.

2 Rule-Based System and its Analysis for Real-Time and Cooperative Problem Solving

2.1 Real-time problem solving

A crucial aspect of the "real-ness" of many real-time systems is that they have a limited set of computational resources such as processors, memory, channels, etc., whose use must be scheduled appropriately for the real-time program to meet its deadline. "Time" has values from the real domain but there is no single system-wide clock. Instead, time is treated as a local property of each program unit: it can be observed only at the start or end of an action and it increases monotonically [3].

Viewed simply, a real-time system has two main parts: an external environment with a number of devices such as sensors and actuators; and a programmed system which registers events from the sensors and responds by producing actions to drive the actuators. It is usually called a "real-time" system because it must work in conjunction with some activity in the "real-world". The events produced by this activity are distributed in time and induce timing constraints, or deadlines, on the events it is to receive.

So the "real-time" problem can be seen in two ways: as one of representing time in the execution of a concurrent program; or as one of scheduling the tasks in a real-time system to meet particular timing constrains. Here, we may adopt the following definition of "real-time": a strictly limited time period in which the system must produce a response to environmental stimuli, no matter what kind of algorithms it employs [4].

Real-time AI problem solving systems have been defined as: "predictably fast enough for use by processes being serviced" [5]; "there is a strict time limit by which a system must have produced a response, regardless of the algorithm employed" [6]; "ability of the system to guarantee a response after a (domain defined) fixed time has elapsed" [7]; "a system designed to operate with a well-defined measure of reactivity" [8]. These definitions are general and open to interpretation depending on the problem at hand. An excellent analysis of real-time systems which lays out some of the shortcoming of current approaches has been presented [9].

Hence, the key feature of a real-time AI problem solving system is its ability to guarantee a response after a fixed time has elapsed, where that fixed time is provided as a part of the problem statement.

While speed is indeed fundamental to real-time performance, speed alone is not real-time. In fact, there are four aspects of real-time problem solving performance [10]: (1) speed; (2) responsiveness; (3) timeliness; and (4) graceful adaptation.

Speed is the rate of execution of tasks. Tasks could refer to problem-solving tasks (large or small) or event-processing tasks.

Responsiveness is the ability of the system to stay alert to incoming events. Since an interactive real-time system is primarily driven by external inputs, the system should recognize that such input is available. It may not necessarily process the new event right away; that may depend upon its criticality relative to other events the system is currently processing.

Timeliness is the ability of the system to react to and meet deadlines, which can be achieved by processing the more critical tasks first. Criticality of tasks may be defined by several factors, either domain dependent or domain independent or both. For instance, urgency of response to an event is related to the amount of slack time; that is, the difference between the earliest start and latest start which still allows the event to be completed before its deadline. The deadline for an event is specified by the domain. Yet another example is the importance of an event relative to the other events, strictly a domain issue. Importance of an event may depend upon the context, hence it would be dynamic.

Graceful adaptation is the ability of the system to reset task priorities according to changes in work load or resource availability. Graceful adaptation allows for a smooth transition across potential perturbations to the system. Without this ability, the system may waste processing resources as well as time by executing less relevant tasks or in waiting for scarce resources to become available.

It seems imperative that we utilize a knowledge processing approach in order to address all four aspects of real-time performance. Furthermore, "real-time" has another meaning which arises in most engineering applications. This requires that an AI problem solving system should work continuously with time. In another words, the real-time AI problem solving systems must be able to focus their attention on the relevant parts of the domain without ignoring on-going processes.

2.2 Cooperative problem solving

Cooperative problems occur frequently in many application domains. They exist whenever there is a contention over limited means, for instance, many transportation domains, and other complex application domains. One of the goals of AI is to develop systems that become part of our everyday life. To be sufficiently intelligent, a system may have to contain so much knowledge that it will be able to function efficiently only if it is partitioned into loosely coupled subsystems. This stems from the "bounded rationality" principle [11] and results in the cooperative problems.

Distributed artificial intelligence (DAI) refers to a subarea of AI that uses multiple processors. There is a collection of agents (logically distinct processing elements) which are attempting to solve a problem. Although there are identifiable global states and goals, each agent has only a partial and inexact view of these. Each agent tries to recognise, predict and influence the global state in such a way that its local view is satisfied. It computes solutions to a given subproblem and communicates results, tasks and goals to other agents. DAI is concerned with solving problems by applying both AI techniques and multiple problem solvers, and distributing both control and data. DAI has also been referred to as "distributed problem solving (DPS)" [12].

DPS means the work of solving a particular problem can be divided among a number of modules, or "nodes", that cooperate at the level of dividing and sharing knowledge about the problem and about the developing solution [13]. It is also known as a method of problem solving, that is, a decentralized and loosely coupled collection of knowledge sources (KS's) (procedure, sets of rules, etc.) solve the entire problem cooperatively [14]. The KS's cooperate in the sense that no one of them has sufficient information to solve the entire problem; mutual sharing of information is necessary to allow the group as a whole to produce an answer. Decentralized means that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. Loosely coupled means that individual KS's spend the great percentage of their time in computation rather than communication.

There are two forms of DPS: task-sharing and result-sharing. Task-sharing is a form in which individual KS's assist each other by sharing the computational load for the execution of subtasks of the overall problem. Result-sharing is a form in which individual KS's assist each other by sharing partial results, based on somewhat different perspectives on the overall problem.

They offer certain advantages for problem solving [15]: faster response, increased flexibility, robustness, resource sharing, graceful degradation, and better adaptability. Much of the power comes through cooperation and communication. It relates to how the elements of a loosely coupled network can work together to solve problems that are beyond their individual capacities. Each problem solving node in the network is capable of solving sophisticated problems and can work independently, but the problems faced by nodes cannot be solved without cooperation.

Cooperative distributed problem solving [16] may be viewed as a subfield of the larger field of DPS. It relates to how the elements of a loosely coupled network can work together to solve problems that are beyond their individual capacities. Each problem solving node in the network is capable of sophisticated problem solving and can work independently, but the problems faced by nodes cannot be solved without cooperation. This differs from approaches of DPS in which each node is an expert or specialist in one particular aspect of the problem solving task, and a node cannot function outside the context of other nodes and the communication and cooperation protocols that are specified by the designer. In contrast, each node in a cooperative distributed problem solving network has sufficient knowledge to at least formulate a partial solution without assistance from other nodes. Internode cooperation is often the only way to develop an acceptable overall solution. Each node also has significant expertise in control and communication strategies.

Cooperation among the nodes is necessary because no node has sufficient expertise, resources, or information to solve a problem. Some might be very fast at computation, others might have connections that speed communication, and still others might have more memory. The problem is decomposed into subproblems. As the nodes work together on the subproblems, their individual solutions should fit into a global solution. Thus, the nodes must coordinate their asynchronous and parallel problem solving activity to produce solutions compatible with their interdependent subproblems. Nodes have a limited communication capacity because of either the inherent bandwidth limitations of the communication media or the high computational overhead incurred in sending and receiving information. Nodes must rely on sophisticated local reasoning to decide on appropriate actions and interactions. In short, each node must be self-directed in planning its communication and cooperation strategies with other nodes.

In this paper, we shall call cooperative distributed problem solving, "cooperative problem solving" and discuss it in more detail, as real-time performance has to be examined in the context of a specific situation, rather than in general terms. Languages or tools supporting cooperative problem solving must have three characteristics: (1) a method that can express individual KS's conveniently; (2) a mechanism by which individual KS's can communicate with each other; (3) Each KS can be processed concurrently or distributively.

2.3 Rule-based system and its analysis for real-time and cooperative problem solving

There is a mismatch in the development of AI problem solving systems: most AI problem solving systems are rule-based; unfortunately, while several real-time and cooperative knowledge-based techniques are being rapidly developed and implemented, the adaptation of rule-based systems to real-time and cooperative environments has proven to be a formidable task.

A rule-based system consists of three modules [17]: production memory (PM, also called the rule base), working memory (WM, also called the data base) and inference engine (IE). The system records data in the WM; as it uses the knowledge in the PM to reason, under the strategies of the IE. A typical inference cycle of a rule-based system is composed of three steps of match, select (conflict resolution) and execute rules. OPS5 [18] is the most representative system.

Generally speaking, the standardized nature of rule-based systems, plus the fact that they are readily comprehensible to humans, make rule-based systems, whose nature is highly similar to that of human thinking, the most important knowledge representation technique. However, they have many deficiencies, such as that they are difficult to manage and maintain, limit the use of generic processing, are difficult to control, and to verify and validate. The most important one of these problems which needs to be improved is that conventional rule-based systems cannot solve a problem in a real-time and cooperative fashion, and so are not suitable for the complex problems. Rule-based systems have two major deficiencies which make them unsuitable for real-time and cooperative applications [19]:

1. The slow execution speed

A major obstacle in the processing of rule-based systems lies in the large search/match time. It is often the case that the PM and the WM needed to represent a particular rule-based system in a certain problem domain, is in the order of hundreds to thousands of production rules and assertions. Being computationally expensive, the inferencing speed of rule-based systems is too slow to match the workload in some dynamically-changing environments.

2. The insensitivity to asynchronous, external events

The only way that most conventional rule-based systems incorporate input data at run-time is by executing read statements as part of the right-hand side (RHS) actions of a selected rule. They can incorporate only highly-specific data at particular times during the run of the program, and cannot perceive the arbitrary and sudden changes which are characteristic of the real-time and cooperative data sources. The control-flow in the conventional rule-based systems is solely dependent on the state of the WM at any given time and the conflict resolution strategies built into the system. The WM can only be manipulated by the RHS actions, and the external data changes cannot affect the control of the rule-based system program directly.

We have to improve rule-based systems to make them suitable for real-time and cooperative applications. In fact, the two above-mentioned deficiencies will correspondingly result in research in real-time and cooperative rule-based systems being approached in two different ways.

1. Approaches to faster execution of rule-based systems

These approaches have concentrated on the development of efficient, dedicated algorithms and architectures to achieve an order of magnitude speedup in the execution time, especially in the matching time, of the rule-based systems: (1) developing new fast match algorithm; (2) improving the inference mechanism; (3) introducing other programming paradigms; (4) mapping rule-based systems into neural architectures. It has been proven that three layers of ring-structured feedback network is suitable for mapping rule-based systems into, and will obviously reduce the match time [20].

2. Approaches to building asynchronous environment

The second approach towards building real-time and cooperative rule-based systems environments has focused on the development of innovative data structures, execution strategies and shell architectures to facilitate the rapid intergration of asynchronous data into rule-based systems, and the subsequent provision of focused, real-time responses to such data.

It should be pointed out that due to great efforts devoted to developing real-time and cooperative rule-based systems, there has been considerable progress made towards solving this problem. However, the problem has not been settled once and for all, because most of the above-mentioned only improve the performance of one or two aspects of rule-based systems. Those successful techniques must be integrated in an unified model. A novel rule-based AI problem solving mechanism is necessary.

3 The Function Module System Model

3.1 The concept of function module in human thinking process

The principles of modular design and implementation tell us that by structuring a complex problem into self contained modules, the system is easier to build, debug and maintain and is less prone to errors. Many AI applications are inherently distributed: spatially, temporally, or functionally.

A thorough study must be made of human thinking processes to establish a general AI problem solving mechanism and to solve complex problems. There are at least three commonly acknowledged aspects to this: (1) The human brain is not a whole which cannot be separated. It can be divided into many basic units and independent neurons. (2) Generally speaking, a production rule is just the same as the action(s) of a neuron or a group of neurons: both of them have unidirectivity. But the size of human thinking may be big or small, and only the fine size of human thinking will correspond to a production rule. (3) People always organise units with certain external functional characteristics according to some connection, such as groups, families, factories, towns, cities, countries and so on, and deal with them in these terms. This can be referred to the coarser size of human thinking. From these, we can draw a conclusion that human thinking is a complex interactive process among those modules with multiple layers and multiple functions. Thus, a concept of function module in human thinking process has been abstracted, based on combining the study of cognitive psychology and AI problem solving, together with the study of neurophysiology. The term "*function*" stems from the idea of functional architecture [21] and "*module*" from the idea of the modularity of mind [22]. Other similar ideas can be found in the design of software or multi-agent systems [23].

The concept of function module in human thinking process can be described as follows: (1) The human thinking process is an interactive process between several function modules with the same or different external functions. (2) These function modules stem from learning and possess the ability to adjust and adapt by themselves. They exist within the human brain and may or may not share the unique structure. A neuron or some structural combination of neurons is the carrier of a function module. (3) Each function module has a certain size and one or more complete external functions itself. More functions can be realized by interaction among the function modules with the same or different external functions. A coarser function module can be organised by some interaction in functions or structures among function modules. (4) A function module may be activated by the information of an active neuron passed from the function module or out of the function module. (5) The transformation mechanism of human brains may solve the transformation problem from energy to information, or from one kind of representation to another kind of representation, in the architectures of function modules. (6) The selective attention mechanism of human brains may allow human thinking to operate by concentrating on one function module, or switching among several function modules, or concurrently processing multiple function modules according to some interaction in functions or structures.

The concept of function module concentrically embodies the idea of functional simulation. It can result in some useful corollaries to the study of general AI problem solving mechanism in this paper: (1) As soon as some information is available, no matter where it came from, one or several function modules must be activated and given some relative actions (including no reaction). (2) The function modules in a human brain can execute concurrently and have a high execution speed; in addition, they can give a response to the external stimulus in time, while executing internal processing. (3) The function modules in a human brain, even the human brains, can solve a complex problem or deal with a complicated matter together.

Based on the above-mentioned discussion, a conclusion may be reached that, for one thing, human brains possess the ability to solve complex problems or process complicated matters in a real-time and cooperative fashion; for another, that the concept of function module can be abstracted from the functional characteristics in human brains. Thus, according to the concept of function module, we may establish a model to support real-time and cooperative AI problem solving. Such a model presented in this paper is called Function Module System (FM_S in short). It is a rule-based general AI problem solving model.

3.2 The FM_S model

The function module with a certain size and which can complete some external function, may be called a function unit. It is a physical or conceptual entity in AI problem solving. It corresponds to a KS or an intelligent processing unit, and is a basic unit for parallel processing. It can be also contrasted with the node in distributed problem solving, the agent in multi-agent systems and the object in object-oriented programming.

Definition 1 (**Function Unit**)
A function unit is a quinary and can be written as:

$$FU = (NA, DB, RB, ST, IE)$$

in which, NA (NAme) is the function unit's name; DB (Data Base) is a generalized data base; RB (Rule Base) is a generalized rule base (so-called "generalized" means that not only the conventional rule-based systems, but also the systems with the characteristics of data-driven are referred to as rule-based systems); ST (STructural knowledge base) is a functional component part taking other techniques (referred to as structural knowledge representation) rather than production rules to represent knowledge; IE (Inference Engine) is the inference and control mechanism.

In above-mentioned definition, the elements in the generalized data base are called data, composed by some symbols in a certain structure.

The generalized rule base is composed of a group of generalized rules with OR relations, that is,

$$RB = \bigvee_{i \geq 1} (C_i \rightarrow A_i)$$

in which, C_i represents the condition part, and A_i represents the action part of generalized rule i, and

$$C_i = \bigwedge_{j \geq 1}^{m} (S_i^j ? \xi_i^j) \quad (m \geq 1)$$

$$A_i = \{(T_i^k ! \eta_i^k)\}_{i=1}^{n} = (T_i^1 ! \eta_i^1); (T_i^2 ! \eta_i^2); \ldots ; (T_i^n ! \eta_i^n) \quad (n \geq 1)$$

in which, $\bigwedge$ represents the condition of a generalized rule may be composed of 1 to m $(S_i^j ? \xi_i^j)$ with AND relations. $S_i^j ? \xi_i^j$ represents the message received from the source function unit (the function unit which sends messages) S_i^j that is satisfied the condition pattern words ξ_i^j; $T_i^k ! \eta_i^k$ represents the message sent to the target function unit (the function unit which receives messages) T_i^k that the action words are η_i^k.

$$\xi_i^j = \gamma_i^{ju_0}\mu_i^{ju_1}\gamma_i^{ju_1}\mu_i^{ju_2}\ldots\gamma_i^{ju_{j-1}}\mu_i^{ju_j}\gamma_i^{ju_j} \quad (j = 1,\ldots,m)$$

$$\eta_i^k = \delta_i^{kv_0}\mu_i^{kv_1}\delta_i^{kv_1}\mu_i^{kv_2}\ldots\delta_i^{kv_{k-1}}\mu_i^{kv_k}\delta_i^{kv_k} \quad (k = 1,\ldots,n)$$

in which, γ_i^{jh}, δ_i^{kl} $(h = u_0, u_1, \ldots, u_j;\ l = v_0, v_1, \ldots, v_k)$ are the word constants (fixed words), μ_i^{jh}, μ_i^{kl} are the word variables (arbitrary words). When the word variables are satisfied the prerequisites as follows,

$$\forall \mu_i^{kl} \in \eta_i^k, \exists \mu_i^{jh} \in \xi_i^j, \mu_i^{kl} = \mu_i^{jh}$$

then μ_i^{kl} and μ_i^{jh} are the same word variable.

The functional component part ST may be a framework or an object-oriented function, or an imperative-driven programming language, or any of other problem solving mechanisms. It may be in default. The purpose set up such a component part is improving the interface abilities (including the abilities of syntax and semantics representation) of a function unit. It will not be discussed in this paper.

The system which can represent the interaction relations among several function units with same or different external functions and can execute the actions of those function units, may be called a function module system.

Definition 2 (**Function Module System**)
A function module system is a tetrad and can be written as

$$FM_S = (FU_S, I_S, R_S, C_S)$$

in which, FU_S is a finite nonempty set of function units.

$$FU_S = \{FU_i | i \geq 1\}$$

FU_i is a function unit.

I_S is a set of messages transmitted among the function units.

$$I_S = \bigcup_{FU_i \in FU_S} I_{in}(FU_i) \text{ or } I_S = \bigcup_{FU_i \in FU_S} I_{out}(FU_i)$$

$I_{in}(FU_i)$ represents the messages set received by FU_i; $I_{out}(FU_i)$ represents the messages set sent by FU_i after its message processing. In general, $I_{in}(FU_i) \neq I_{out}(FU_i)$, but, $\bigcup_{FU_i \in FU_S} I_{in}(FU_i) = \bigcup_{FU_i \in FU_S} I_{out}(FU_i)$. Each message in $I_{in}(FU_i)$ and $I_{out}(FU_i)$ is a duality,

$$(S',\ I)$$

S' is the name of the source function unit sending messages; I is called message words,

$$I = \delta_0\xi_1\delta_1\xi_2\ldots\delta_{n-1}\xi_n\delta_n$$

both $\delta_i (i = 0, 1, \ldots, n)$ and $\xi_j (j = 0, 1, \ldots, n)$ are word constants.

R_S is a set of relations among the function units.

$$R_S = \{(FU_i, FU_j) | (FU_i, FU_j) \in FU_S \times FU_S \ and \ R(FU_i) = FU_j\}$$

C_S is a set of the characteristics of a function unit.

$$C_S = \{C(FU_i) | FU_i \in FU_S\}$$

The execution of a function module system represents the function units in this system parallel reduce the messages. According to the execution characteristics of the generalized rule base, function units and function module systems may be divided into two kinds of types: deterministic and nondeterministic. This paper only discusses the reduction of nondeterministic function units and function module systems.

3.3 Reduction of nondeterministic FU and FM_S

Firstly, some basic definitions should be given as follows:

Definition 3 (**Transform Operator**)
Suppose the messages received by a function unit FU,

$$I'_{in}(FU) = \{(S'_1, I'_1), (S'_2, I'_2), \ldots, (S'_m, I'_m)\}$$

in which, each message word $I'_i (i = 1, \ldots, m)$, may not be the messages FU will be able to process for the reason that the source function unit sending this message may not possess the same structural characteristics with FU. A formal transform is necessary in such situation, that is,

$$\ominus[I'_{in}(FU)] = \ominus\{(S'_1, I'_1), (S'_2, I'_2), \ldots, (S'_m, I'_m)\}$$
$$= \{(S'_1, I_1), (S'_2, I_2), \ldots, (S'_m, I_m)\} = I_{in}(FU)$$

so we call $I_{in}(FU)$ as the processable message of FU, $\ominus$ as the transform operator.

Definition 4 (**Selection Operator**)
For the processable message $I_{in}(FU)$ received by FU, if

$$(S', I) \in I_{in}(FU)$$

in which, $I = \delta_0 \xi_1 \delta_1 \xi_2 \ldots \delta_{n-1} \xi_n \delta_n$, and

$$\exists C \to A \in FU \ and \ \exists (S?\xi) \in C$$

in which, $\xi = \gamma_0 \mu_1 \gamma_1 \mu_2 \ldots \gamma_{n-1} \mu_n \gamma_n$, satisfying,

$$\oplus[(S', I), (S?\xi)] = \oplus[(S', S), (I, \xi)] = true$$

that is, $S' = SorS \in V(FU_S); \forall i (i = 0, 1, \ldots, n), \delta_i = \gamma_i$

thus we call message (S', I) satisfying the selection of condition element $(S?\xi)$, call $\oplus$ as selection operator.

After executing $\oplus$, if $S \in V(FU_S)$, then $S' = S$ and all the word variables in ξ, $\mu_i (i = 1, \ldots, n)$, will be constrained by the correspondent word constants in I, ξ_i, that is,

$$\forall i (i = 1, \ldots, n), \mu_i = \xi_i$$

Definition 5 **(Join Operator)**
For the processable message $I_{in}(FU)$ received by FU, if

$$(S'_i, I_i), (S'_j, I_j) \in I_{in}(FU)$$

in which, $I_k = \delta_{k0}\xi_{k1}\delta_{k1}\xi_{k2}\ldots\delta_{kn_{k-1}}\xi_{kn_k}\delta_{kn_k}$ $(k = i, j)$, *and*

$$\exists C \rightarrow A \in FU \text{ and } \exists (S_i?\xi_i) \bigwedge (S_j?\xi_j) \in C$$

in which, $\xi_k = \gamma_{k0}\mu_{k1}\gamma_{k1}\mu_{k2}\ldots\gamma_{kn_{k-1}}\mu_{kn_k}\gamma_{kn_k}$ $(k = i, j)$
if $\forall k(k = i, j)$

$$\oplus[(S'_k, I_k), (S_k?\xi_k)] = true$$

and

$$\otimes\{\oplus[(S'_i, I_i), (S_i?\xi_i)], \oplus[(S'_j, I_j), (S_j?\xi_j)]\} = true$$

that is, $\forall \mu_{ij} \in \xi_i and \mu_{ji} \in \xi_j$
after executing $\oplus$, $\exists \mu_{ij} = \xi_{ij}$ *and* $\mu_{ji} = \xi_{ji}$
if $\mu_{ij} = \mu_{ji}$, *then after executing* $\otimes$, $\xi_{ij} = \xi_{ji}$
thus we call $\oplus[(S'_i, I_i), (S_i?\xi_i)]$ *and* $\oplus[(S'_j, I_j), (S_j?\xi_j)]$ *satisfying the join relation, call* $\otimes$ *as join operator.*

Definition 6 **(Reducable Information)**
For a function unit FU, let the processable messages are

$$I_in(FU) = \{(S'_1, I_1), (S'_2, I_2), \ldots, (S'_m, I_m)\}$$

also let

$$\exists C \rightarrow A \in FU$$

in which, $C = (S_1?\xi_1) \bigwedge (S_2?\xi_2) \bigwedge \ldots \bigwedge (S_m?\xi_m)$
if $\forall i(i = 1, \ldots, m)$

$$\oplus[(S'_i, I_i), (S_i?\xi_i)] = true$$

and

$$\otimes\{\oplus[(S'_i, I_i), (S_i?\xi_i)]\} = true$$

simply written as

$$\otimes \oplus [I_{in}(FU), C] = true$$

thus we call $I_{in}(FU)$ as the rule reducible information of the function unit FU, simply called redin (REDucible INformation).

$$redin_{FU} = \{FU; \otimes \oplus [I_{in}(FU), C]\}$$

Definition 7 (**Constraint Operator**)
Suppose $redin_{FU} = \{FU; \otimes \oplus [I_{in}(FU), C]\}$

$$FU_i \in V(S_{FU}) \text{ and } FU_j \in V(T_{FU})$$

$$\mu_i \in V(\xi_{FU}) \text{ and } \mu_j \in V(\eta_{FU})$$

in which, $V(S_{FU})$ *and* $V(T_{FU})$ *represent the source function unit variable set and the target function unit variable set of* FU*, respectively;* $V(S_{FU}), V(T_{FU}) \subset V(FU_S)$. $V(\xi_{FU})$ *and* $V(\eta_{FU})$ *represent the word variable set of condition words and action words, respectively.*

Also suppose in $redin_{FU}$, $FU_{is} = FU_i, \mu_{is} = \xi_i$
if $FU_i = FU_j, \mu_i = \mu_j$
then the process as follows

$$\beta A = A[FU_{js} = (FU_{is} = FU_i); \mu_{js} = (\mu_{is} = \xi_i)]$$

$$= A[FU_{js} = FU_i); \mu_{js} = \xi_i]$$

is called the constraint to word variables in action word A. β *is called constraint operator.*

Definition 8 (**Nondeterministic Function Unit and Nondeterministic Function Module System**)
A function unit will be nondeterministic, if
for $I_{in}(FU)$, $\exists C_1 \to A_1 \in FU, \ldots, C_k \to A_k \in FU$
in which $(C_1 \to A_1) \bigvee \ldots \bigvee (C_k \to A_k) \subset FU$
$\exists \oplus_1, \ldots, \oplus_k$ *and* $\exists \otimes_1, \ldots, \otimes_k$
satisfy $\otimes_1 \oplus_1 [I_{in}(FU),\ C_1] \ = \ true$
$\vdots$
$\otimes_k \oplus_k [I_{in}(FU),\ C_k] \ = \ true$
thus $redin_{FU} \ = \ \{FU;\ \otimes_1 \oplus_1 [I_{in}(FU),\ C_1], \ldots, \otimes_k \oplus_k [I_{in}(FU),\ C_k]\}$
in which, $k \geq 2$.

A function module system will be nondeterministic if it contains more than one function unit are nondeterministic.

Definition 9 (**Characteristical Action Operator**)
For a nondeterministic function unit FU*, the process selecting a* $redin_{FU}$ *to execute according to some selective attention mechanism may be written as*

$$\begin{aligned} A(redin_{FU}) &= A\{FU;\ \otimes_1 \oplus_1 [I_{in}(FU),\ C_1], \ldots, \otimes_k \oplus_k [I_{in}(FU),\ C_k]\} \\ &= \{FU;\ \otimes_m \oplus_m [I_{in}(FU),\ C_m]\} \quad (1 \leq i \leq k) \end{aligned}$$

$A()$ *is called characteristical action operator. It makes a nondeterministic function unit or function module system becoming deterministic one.*

Theorem 1 *A nondeterministic function unit is reducible.*

Proof :
For a nondeterministic function unit FU, $I_{in}(FU) \Rightarrow I_{out}(FU)$:

1. $\exists C_1 \to A_1 \in FU, \ldots, C_k \to A_k \in FU$
in which $(C_1 \to A_1) \bigvee \ldots \bigvee (C_k \to A_k) \subset FU$

$\exists\oplus_1, \ldots, \oplus_k$ *and* $\exists\otimes_1, \ldots, \otimes_k$
satisfy $redin_{FU} \;=\; \{FU;\; \otimes_1 \oplus_1 [I_{in}(FU),\; C_1], \ldots, \otimes_k \oplus_k [I_{in}(FU),\; C_k]\}$
in which, $\quad k \geq 2$.

2. $\exists A(FU)$,
$A(redin_{FU}) \;=\; \{FU;\; \otimes_i \oplus_i [I_{in}(FU),\; C_i]\}$
in which, $1 \leq i \leq k$.

3. $\exists\beta$,
$A(redin_{FU}) \;=\; \beta A_i \;\rightarrow\; I_{out}(FU)$. $\quad\Box$

Theorem 2 *A nondeterministic function module system is reducible.*

Proof :
For a nondeterministic function module system FM_S,

$$\forall FU_i, FUj \in FU_S,\; I_{in}(FU_i) \Rightarrow I_{out}(FU_i) \| I_{in}(FU_j) \Rightarrow I_{out}(FU_j):$$

1. $\exists C_1^i \rightarrow A_1^i \in FU_i, \ldots, C_k^i \rightarrow A_k^i \in FU_i$
in which $(C_1^i \rightarrow A_1^i) \bigvee \ldots \bigvee (C_k^i \rightarrow A_k^i) \subset FU_i$
and

$$\exists C_1^j \rightarrow A_1^j \in FU_j, \ldots, C_l^j \rightarrow A_l^j \in FU_j$$

in which $(C_1^j \rightarrow A_1^j) \bigvee \ldots \bigvee (C_l^j \rightarrow A_l^j) \subset FU_j$

$$\exists\oplus_1^i, \ldots, \oplus_k^i \text{ and } \exists\oplus_1^j, \ldots, \oplus_l^j$$

and

$$\exists\otimes_1^i, \ldots, \otimes_k^i \text{ and } \exists\otimes_1^j, \ldots, \otimes_l^j$$

satisfy

$$\otimes_1^i \oplus_1^i [I_{in}(FU_i),\; C_1^i] \;=\; true$$
$$\vdots$$
$$\otimes_k^i \oplus_k^i [I_{in}(FU_i),\; C_k^i] \;=\; true$$

and

$$\otimes_1^j \oplus_1^j [I_{in}(FU_j),\; C_1^j] \;=\; true$$
$$\vdots$$
$$\otimes_l^j \oplus_l^j [I_{in}(FU_j),\; C_l^j] \;=\; true$$

thus

$$redin_{FU_i} \;=\; \{FU_i;\; \otimes_1^i \oplus_1^i [I_{in}(FU_i),\; C_1^i], \ldots, \otimes_k^i \oplus_k^i [I_{in}(FU_i),\; C_k^i]\}$$

and

$$redin_{FU_j} \;=\; \{FU_j;\; \otimes_1^j \oplus_1^j [I_{in}(FU_j),\; C_1^j], \ldots, \otimes_l^j \oplus_l^j [I_{in}(FU_j),\; C_l^j]\}$$

2. $\exists A(FU_i)$ *and* $\exists A(FU_j)$

$$A(redin_{FU_i}) \;=\; \{FU_i;\; \otimes_m^i \oplus_m^i [I_{in}(FU_i),\; C_m^i]\} \|$$
$$A(redin_{FU_j}) \;=\; \{FU_j;\; \otimes_n^j \oplus_n^j [I_{in}(FU_j),\; C_n^j]\}$$

3. $\exists\beta_i$ *and* $\exists\beta_j$

$$A(redin_{FU_i}) \;=\; \beta_i A_m^i \;\rightarrow\; I_{out}(FU_i) \;\|$$
$$A(redin_{FU_j}) \;=\; \beta_j A_n^j \;\rightarrow\; I_{out}(FU_j)$$

above $\quad k \geq 2,\; l \geq 2,\; 1 \leq m \leq k,\; 1 \leq n \leq l$. $\quad\Box$

It should be illustrated that only two function units has been cited for easily description. In fact, mutilple function units can be reduced in a parallel or concurrent fashion.

3.4 Evaluations of nondeterministic FM_S model

Now, we can find that a nondeterministic function module system is data-driven, i.e., as soon as it receives data information $I_{in}(FU)$, it will reduce the relevant action reactions, according to the knowledge (module) included in the rule base. Thus, the programming language designed on a nondeterministic function module system, will possess abilities of problem solving, just like other general AI programming language. It will provide distinct knowledge representation and effective inference mechanism. For one thing, the programming language based on FM_S may be executed in a parallel or concurrent fashion; for another, as the rule base and the data base have been divided into some relative, independent function units and managed by those function units respectively, the sizes of each rule base and data base have been reduced greatly and the match time has been decreased clearly, and communication time will take a negligible time compared to the match time. Thus, the programming language may run at a fast execution speed.

In addition, a nondeterministic function module system is composed of multiple nondeterministic function units possessed independent characteristics, according to some special interactions and relations. A complex problem can be solved in a coordination and cooperation fashion, by passing relative information among those function units and executing relevant actions in a function unit according to the received reducible information. So the FM_S model possess the abilities of real-time and cooperative problem solving.

4 A Function Module System Programming Language FM_S-1

Based on a conventional rule-based programming language OPS5 [18], a function module system programming language supporting real-time and cooperative AI problem solving, FM_S-1, has been developed by LISP.

4.1 The basic structure of a function unit

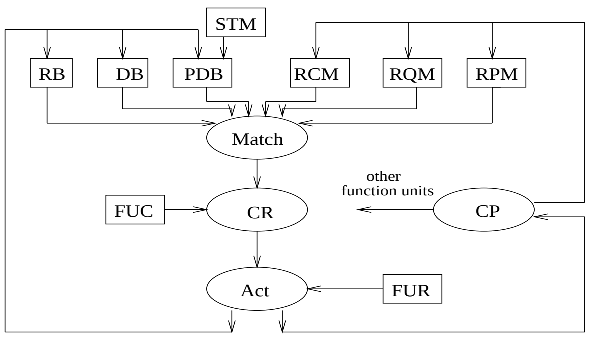

Figure 1: The structure of a function unit

Programs programmed in FM_S-1 consists of one or more function units. Each function unit is an individual knowledge source or a knowledge processing module composed of production rules, and performs the logical or physical functions for real-time and cooperative problem solving, and is a basic unit for concurrent processing. A complex problem can be solved in a coordinated and co-operative fashion, by passing relative information among those function units and executing relevant actions in a function unit according to the received information. In addition, it can tap the concurrency of rule-based systems as far as possible. So the FM_S-1 possesses the abilities of cooperative problem solving. Fig 1 shows the basic structure of a function unit, in which, RB represents Rule

```
<Function Module System>  ::= <FU>{<FU>}
<FU>                      ::= (F <FU-name>
                                 [{<declaration>}]
                                 [<cr-strategy>]
                                 <FU-body>)
<FU-body>                 ::= <FU-condition> --> <FU-action>
                          ::= <FU-presupposition> -->--> <FU-body>
                          ::= <FU-body>{<FU>}
                          ::= <rule>{rule}
                          ::= others
<FU-condition>            ::= <condition-element>{<condition-element>}
<FU-presupposition>       ::= <condition-element>{<condition-element>}
<condition-element>       ::= <pattern>|<-pattern>
<-pattern>                ::= -<pattern>
<pattern>                 ::= <term-list>{<term-list>}
                          ::= <comm-patn-primitive>{<comm-patn-primitive>}
<term-list>               ::= same as OPS5
<comm-patn-primitive>     ::= <syn-patn-primitive>
                          ::= <asyn-patn-primitive>
<syn-patn-primitive>      ::= (REQ-ACCEPT source message)
                          ::= (REP-ACCEPT source message)
<asyn-patn-primitive>     ::= (RECEIVE source message)
<FU-action>               ::= <action-element>{<action-element>}
<action-element>          ::= <action>{<action>}
                          ::= <comm-act-primitive>{<comm-act-primitive>}
<action>                  ::= same as OPS5
<comm-act-primitive>      ::= <syn-act-primitive>
                          ::= <asyn-act-primitive>
<syn-act-primitive>       ::= (REQUEST target message)
                          ::= (REPLY target message)
<asyn-act-primitive>      ::= (SEND target message)
<rule>                    ::= (P rule-name lhs --> rhs)
lhs                       ::= <condition-element>{<condition-element>}
rhs                       ::= <action-element>{<action-element>}
others                    ::= program modules in other languages
```

Figure 2: Thy syntax of a function module system

Base, DB represents local Data Base, PDB represents Public Data Base, RCM represent ReCeive message Memory, RQM represents ReQuest-accept message Memory, RPM represents RePly-accept message Memory, FUC represents Function Unit Characteristics, CR represents Conflict Resolution, FUR represents Function Unit Relationship, CP represents Communication Protocol, and STM is a STructure function Module.

The function unit records data in DB, PDB, RCM, RQM, and RPM; as it uses the knowledge in RB, FUC, and FUR to reason; under the strategies of IE (local inference engine). It can use CP to contact with other function units. A typical inference cycle, like a rule-based system, is also composed of three steps of match, conflict resolution and act.

4.2 The syntax of FM_S-1

In the following, we describe the fundamental syntax of FM_S-1 in terms of BNF as Fig 2, in which the details and parts of the syntax which are the same as OPS5, are omitted. The description also pays attention to the mechanism of cooperative problem solving in FM_S-1.

```
OPS5  ::= <declaration>{<declaration>}
          [cr-strategy]
          <rule>{<rule>}
```

Figure 3: The syntax of OPS5

4.3 The working principle of FM_S-1 interpreter

The working procedure of FM_S-1 interpreter is as follows: (1) loading the program programmed by FM_S-1, and doing preprocessing for it; (2) starting to run the system by using START command on the top level of LISP; (3) selecting some executable function unit according to the schedule algorithm and executing the function unit, automatically and repeatedly. It will not stop until receiving the FINISH command or having no executable function unit.

4.4 Compatibility with OPS5

The syntax of OPS5 can be described as Fig 3:

It is similar to the syntax rule of a function unit described as above, i.e., the syntax rule of OPS5 can be used in FM_S-1 for compatibility. So the programs coded in OPS5 can also run in the FM_S-1 environment, after adding the symbols of "(F FU-name" in the beginning and ")" at the end.

4.5 Communication and cooperation

The communication and cooperation mechanism of FM_S-1 is based on message passing, and designed to fit with the characteristics of production rules.

According to their position, communication primitives can be divided into communication pattern primitive and communication action primitive. The former means a pattern used to receive a message which is situated in the left-hand side (LHS) of a production rule; and the latter means an action used to send a message which is situated in right-hand side (RHS) of a production rule.

According to the their function, communication primitives can be also divided into synchronous communication primitives and asynchronous communication primitives. The former consists of action primitives REQUEST, REPLY, and pattern primitives REQ-ACCEPT, REP-ACCEPT; and the latter consists of action primitive SEND, and pattern primitive RECEIVE.

In synchronous communication, a function unit halts its current execution when it makes a request to other function unit(s). In asynchronous communication, a function unit continues its current execution after sending a message. The word "source" ("target") in the communication primitives is either a source (target) function unit name used to support static communication channel naming, or a source (target) function unit variable used to support dynamic communication channel naming.

4.6 Schedule and control

Schedule and execution is the section of FM_S-1 interpreter which manages the concurrent or parallel execution among the function units. It will coordinate the function running and communication processing of the function units, including: (1) deciding which function unit(s) can own processor(s), according to a certain strategy; (2) deciding which processor will be assigned to which function unit and for how long, according to a certain algorithm. From the reduction process of a nondeterministic function module system, we may find there actually exits two kinds of schedule algorithm, i.e., *Schedule Algorithm 1* based on receiving messages I_{in} and *Schedule Algorithm 2* based on reducible information *redin*. For FM_S-1, I_{in} are

$$DB(WM)\bigcup PDB\bigcup RQM\bigcup RMP\bigcup RCM$$

of a function unit, and *redin* is the conflict set of a function unit. *Schedule Algorithm 1* is suitable for parallel execution in a multiple processors environment; *Schedule Algorithm 2* is suitable for cooperatively work in a concurrent fashion in a single processor environment. The state transition diagrams of a function unit are shown as Fig 4 and Fig 5, respectively.

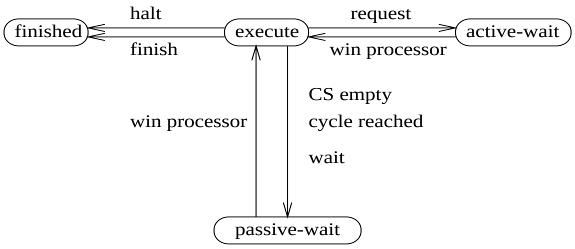

Figure 4: The state transition diagram in Algorithm 1

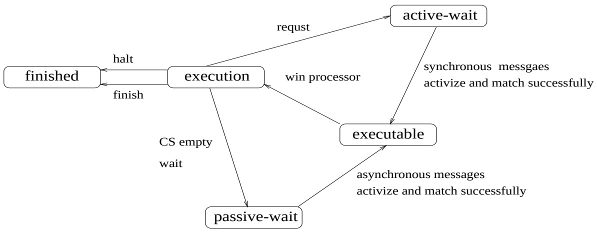

Figure 5: The state transition diagram in Algorithm 2

The schedule algorithms can be described as follows, respectively

Schedule Algorithm 1

def $schedule(num-processor)$
  if *ful* is empty
    then *ful* = function unit whose status is not "finished" in *current-FU-list*
  endif;
  current-FU = head(*ful*);
  term belongs to the union set of RQM, RPM, and RCM
    do constructing appropriate time-tages according to the "term" in every buffer;
      sending the "term" to Rete network of *current-FU* for matching
    enddo;
  work-cycle = working cycle of *current-FU*;
  repeat
    instance = results after conflict resolution;

```
        if instance is empty
            then *ful* = tail(*ful*);
                schedule
        endif;
        r-name = head(instance);
        executing RHS actions of "r-name" rule;
        work-cycle = work-cycle - 1
    until work-cycle = 0 or suspend = true;
    if finish-tag = true
        then ending the schedule
        else *ful* = tail(*ful*);
            schedule
    endif
enddef.
```

Schedule Algorithm 2

```
def schedule2 ≡
    while *executable-queue* is not empty
    do suspend = false;
        finish-tag = false;
        *current-FU* = one of *executable-queue*
        repeat
            instance = results after conflict resolution;
            if instance is not empty
                then r-name = head(instance);
                    executing RHS actions of "r-name" rule;
                else schedule2
            endif
        until suspend = true or finish-tag = true
        if finish-tag = true
            then ending the schedule
        endif
    enddo
enddef.
```

In *Schedule Algorithm 1*, after compiling and entering execution status in the multiple processors environment, the system is unifiedly managed by the the communication service modules. The processors cooperatively work in a parallel fashion.

Some characteristics of *Schedule Algorithm 2* are: (1) take the executable queue, which is dynamically established when running the system, as the schedule basis; (2) the currently executing function unit will be suspended when meeting one of the following situations: a) the conflict set of the function unit is empty, b) its RHS action is *wait*, c) its RHS action is *request*; (3) when executing the RHS actions of a function unit, match with the Rete network if modifying DB, revise the global variables if modifying PDB, and send to the target function unit to match if meeting communication primitives.

4.7 Execution and inference

Since concurrent execution of processes is required, the host system must possess the capability of providing several processors operating in MIMD mode. In such an environment, at least three distinct processors are required for a function unit. As the select phase takes a negligible time compared to the other processes and is fast enough for most applications, and the select process can be interrupted without significantly affecting the inference engine due to its predominantly sequential nature, all the three modules (match, select and execute) can concurrently and independently execute on two separate processors. The third processor performs the function of monitoring and pre-processing the external data changes asynchronously with respect to the execution of other three modules in the space production environment. Such a parallel execution mechanism is given in Fig 6.

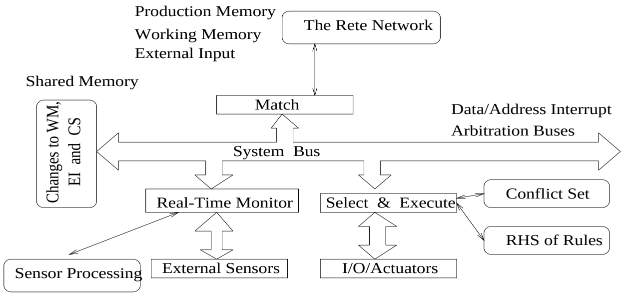

Figure 6: Execution mechanism of a function unit

The parallel inference mechanism (working procedure) of FM_S-1 interpreter is as follows: (1) loading the programs programmed by FM_S-1 in each processor, and doing preprocessing for them respectively; (2) starting to run the system by using START command on the top level of LISP in each processor; (3) selecting some executable function unit in some processor according to the schedule algorithm and executing the function unit, automatically and repeatedly. It will not stop until all the programs have received the FINISH command or have no executable function unit.

5 Application Examples

5.1 Real-time and cooperative rule-based diagnosis expert system

There is a great deal of technologies centring on the problem of diagnosis. Expert systems bank on looking for an economical, efficient diagnosis system. So diagnosis expert systems have become an area of growing interest in the application of expert system technology [24]. A diagnosis expert system is a knowledge-based system that emulates experts' knowledge to solve the problems of diagnosis.

Based on an analysis of the requirements of the automatic GaAs production diagnosis and maintenance on a space station, an overall design scheme of a rule-based diagnosis and maintenance expert system (RBDAMES) employed in the automatic GaAs production line on the space station has been presented [25]. Also, we have made great efforts to use the function module system to provide a powerful environment in which to develop a real-time and cooperative rule-based diagnosis and maintenance system for the automatic GaAs production line on a space station (contrast with RBDAMES, we call it RCDAMES).

Like RBDAMES system, the functions of RCDAMES are: the maintenance function of the normal production; the prevention and protection of the accidents function; the diagnosis and maintenance of the production line and production process function; the self-maintenance function of the RCDAMES system and its knowledge base; and the emergency processing function.

To achieve these functions, an advanced diagnosis and maintenance system is necessary. RCDAMES mainly consists of eight component parts: the section of production maintenance, the section of accidents prevention, the section of diagnosis and maintenance, the section of system maintenance, the knowledge base, the interpreting & learning module, the sensors(monitors) & executive installation, and the system controller (Fig 7).

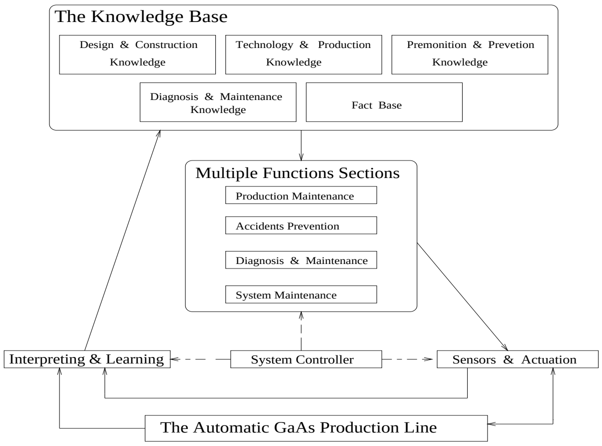

Figure 7: Architecture of RCDAMES

The knowledge in RCDAMES may be divided four categories of knowledge: knowledge about design and construction, knowledge about technology and production, knowledge about premonition and prevention, and knowledge about diagnosis and maintenance. According to our understanding of the requirements of the space production and the analysis of current technologies of expert systems, we still adopt the technique of rules as the main method of knowledge representation in RCDAMES. All the solving knowledge about the problems of normal production maintenance and general diagnosis and maintenance may be acquired from direct experience of domain experts and represented by the knowledge engineers as a series of experience rules in the form of "if condition, then conclusion/action". These can be stored and used on blocks and layers according to their function or frequency of use. These experience rules, might be referred to as "shallow knowledge", they model the process of expert solving, and moreover do so more directly. RCDAMES therefore is more capable and efficient. For the complexity and unpredictability of the space production, it is impossible to attend to each and every aspect of a matter. However, the autonomous production process requires the system can make a response to every case in time. So RCDAMES also consists of some prime inference tactics and will also consist of some prime inference tactics and will be fired only at the time when the experience rules lose efficacy. These tactics might be seen as "deep knowledge".

Under the control of the tactics of the inference engine, the functions unit of RCDAMES are driven by the data change of the production points in the fact base, using the diagnosis and maintenance knowledge in the experience rule base or in the prime knowledge base, to carry out the diagnosis reasoning and decision processing. The process of the RCDAMES system carrying out a task of diagnosis and maintenance, driven by the state information and production requirements on the GaAs production line, can be divided into: input analysis; diagnosis inference; diagnosis conclusion decision; system state determination; and appropriate measures adoption. After carring out the processing of these five sub-processes, the system accomplishes a complete task of diagnosis and maintenance. The system use the sensors to continue monitoring the working conditions of every production point or equipment, preparing to carrying out a new cycle processing.

A test system of RCDAMES has been implemented in a simulated environment. According to the user's input answer to the system query from the keyboard, based on the results of simulated monitoring, it can diagnose all the possible faults in the five sub-processes of the GaAs production process which the robot arms directly participate in, and prompt methods of maintenance. These five sub-processes are: the sub-process of electroepitaxial crystal growth preparatory, carring the grow-crystal to the temporary store area, carrying the grow-crystal to the cutting-station, carrying the seed-crystal to the polishing-station and carrying the seed-crystal to the try-station.

The system contains five distributed faults diagnosis modules and disposes them in five computers respectively, corresponding with the five above-mentioned sub-processes. In addition, a communication service module is included in each processor. The whole system contains fifteen function modules in all. The relations among them is illustrated as Fig 8.

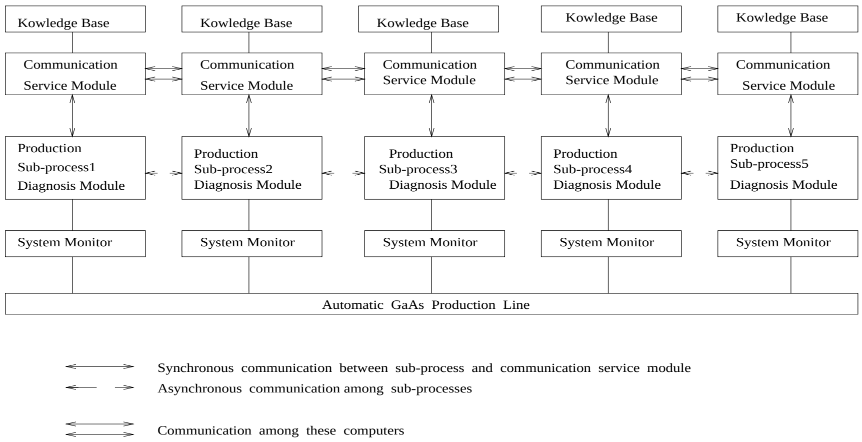

Figure 8: Module relations of RCDAMES

The parallel execution mechanism adopted in RCDAMES can be seen in Fig 6, in which the third processor performs the function of monitoring and pre-processing the external, real-time data changes, asynchronously with respect to the execution of other modules in the space production environment. The parallel inference mechanism of RCDAMES is shown in Fig 9.

5.2 Users-oriented requirements analysis tool

Management Information Systems (MIS) have been constructed with application to a wide range of problems. More and more MISs are expected to be generated automatically. Requirements analysis is the most important step among the construction of MISs.

The most important step in MISs development is to analyze, understand, and record the problem that the users are trying to solve. Requirements describe what the users want from the software. They should be helpful and understandable to end users, serve as a basis for design and testing, be suitable to the application, and encourage thinking in terms of external and not internal system behaviour [26]. Most faults found during testing and operation result from poor understanding or misinterpretation of requirements. Analysis faults are expensive because it costs 100 times more to correct faults found after delivery than during analysis.

In spite of progress in analysis techniques, CASE tool support, prototyping, and early verification and validation techniques, software development still suffers from poor requirements acquisition and analysis [27].

What we should do now? A basic conclusion may be reached that end-users can do and indeed

```
initialize
repeat

   while (no external input available) and
         (general maintenance requirements available) do
      begin
         carry out the general maintenance operation;
         RUN-PROCESSOR1;    {matching}
         RUN-PROCESSOR2;    {selecting & executing}
         RUN-PROCESSOR3     {monitoring}
      end

   repeat
      read and process the real-time external input;
      RUN-PROCESSOR3;       {reading real-time data}
      RUN-PROCESSOR1;       {diagnosing}
      RUN-PROCESSOR2        {maintaining}
   until (no more real-time data);

   wait to receive any internal instruction or external data;
   RUN-PROCESSOR1;          {waiting}
   RUN-PROCESSOR2;          {waiting}
   RUN-PROCESSOR3           {monitoring}

until (assigned space production tasks done) or
      (need giving an emergency alarm to the earth)
```

Figure 9: Inference mechanism of a function unit

will have to do many things for the requirements acquisition and analysis, but not all the things, based on the following accepted criteria: (1) all the computer-based information systems should be based on the current systems, and better than the current systems; (2) every end-user must clearly know what things he (she) should do every day and can give a clear explanation of them; (3) every end-user must clearly know their relations to other end-user(s); (4) almost all the end-users do not know or only know a little about what computers can do for them and what they should ask computers to do for them.

According to the basic conclusion, a new idea which I call "users-oriented requirement analysis" can be presented [28]. It means a distributed interactive multimedia environment tool will be provided for the end-users to describe their respective requirements. The function module system is just a suitable mechanism supporting us to realize such a software tool, called USERTOOL.

USERTOOL will integrate the system developers and the end-users in a distributed interactive multimedia environment. It provides a new technology for requirements analysis, in which the end-users may define their personal affairs and relations by themselves, without the participation of the developers. Each end-user or system developer can be taken as a function unit. USERTOOL can introduce such a problem solving mechanism: (1) "Near natural" specification language as the input technique, especially using technologies of tables and figures even voices: The benefits of formal languages for requirements specification have been well established and, increasingly, formal verification is being required for critical systems. Natural language therefore cannot be relied upon in the development of a system specification. At the same time we must recognise that some important requirements are difficult to quantify and may be impossible to express in current formal specification language. Certainly, natural language is difficult for computers to process automatically. So we put forward a "near natural" specification language. It can provide the end-users with a requirements description interface with tables, figures and some texts, and so on in a distributed interactive multimedia environment. (2) Rapid prototyping and the ability to be modified by the end-users themselves: The purpose of prototyping is to ensure that requirements specifications for the system adequately represent the "real requirements". If not, end-users may use USERTOOL to modify it, only spending a very little works. (3) Communication based on message passing: There is an increasing recognition that software development is not merely a mathematical or technological challenge, but a complex social process, in which the kind of communication and cooperative interaction between the participants determines the quality of the collaboratively developed product [29]. USERTOOL based on the function module system successfully presents a rule-based communication mechanism by introducing synchronous communication primitives and asynchronous communication primitives. (4) Knowledge-based coordination and cooperation: The requirements of a software system are expressed at different levels of abstraction and with different degrees of formality. The highest level requirements are usually informal and imprecise, but they are understood best by the end-users. The lower levels are more technical and more precise, are better suited for the needs of the system analysts and designers, but they are further removed from the end-users' experiences and less well understood by the end-users. Because of the differences in the kinds of descriptions needed by end-users and developers, it is not likely that any single representation for requirements can be the "best" one for supporting the entire prototyping process, and it is unavoidable that some conflicts and contradictions will occur. A knowledge-based cooperative solving mechanism is necessary. In fact, USERTOOL is a AI-based problem solving tool and can provide a knowledge-based coordination and cooperation techniques.

USERTOOL can be divided into four component parts (Fig 10).

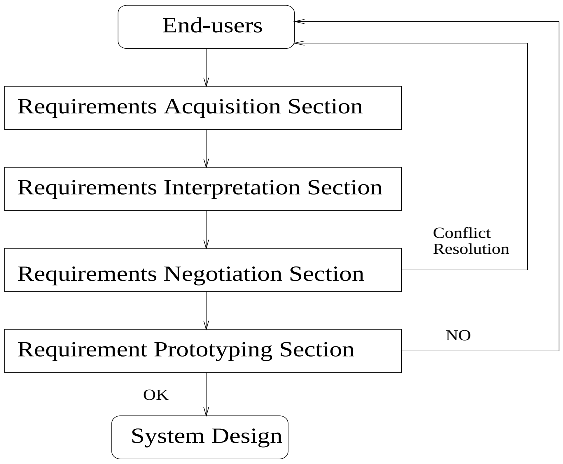

Figure 10: The architecture of USERTOOL

The requirement acquisition section includes requirements expression and definition, requirements capture, and requirements store. The requirement interpretation section includes format transmission, formal functions description, and formal relations description. The requirement negotiation section includes priorities assignment to function units (end-users), and conflict resolution. The requirement prototyping section will display the prototype system to the end-users.

A third example of cooperative problem solving in robotics has been presented [30].

6 Conclusion

This paper presents a novel and unique framework, a general AI problem solving mechanism, for developing real-time and cooperative rule-based AI problem solving systems, from the model and programming language, by analysing and overcoming the major deficiencies in rule-based systems make them unsuitable for real-time and cooperative applications. The relevant results and techniques can be a reference for real-time and cooperative applications. In addition, the ideas of function module systems can be extended as one of the multi-agent theories [23] [31].

7 Acknowledgements

The research described here is based on my past work in China and is still supported partly by the National Science Foundation of P.R.China (69583009). Currently, I am a visiting scholar in the Department of Artificial Intelligence at the University of Edinburgh, with the support of the Royal Society Exchange Fellowship. In addition, I gratefully acknowledge the computing facilities provided by the Department, and thank my host here, Dr John Hallam. Also, I appreciate all the help offered by everyone inside and outside the Department during the past twelve months of my stay.

8 References

[1] G.Uma, et al., "Distributed Intelligent Systems: Issues, Perspectives and Approaches", Knowledge-Based Systems, Vol.6, No.2, 1993, pp77-86.

[2] A.N.Kumar, "Function Based Reasoning", The Knowledge Engineering Review, Vol.9:3, 1995, 301-304.

[3] W.H.Simmonds, "Representation of Real Knowledge for Real-Time Use", Proc. of IFAC Symp. on AI in Real-Time Control, 1988, pp7-11.

[4] H.Dai, et al., "Issues of Real-Time Expert Systems", in AI and Cognitive Science'89, A.F.Smeaton et al.(eds), Springer-Verlag, 1990, pp179-197.

[5] J.Marsh and J.Greenwood, "Real-Time AI; Software Architecture Issues", National Aerospace and Electronics Conference, 1986, pp66-67.

[6] C.A.O'Reily and A.S.Cromarty, " 'Fast' is not 'Real-Time' in Designing Effective Real-Time AI Systems", Applications of AI *II*, Int. Soc. of Optical Engineering, 1985, pp249-257.

[7] T.J.Laffey, et al., "Real-Time Knowledge-Based Systems", AI Magazine Vol.9, No.1, 1988, pp27-45.

[8] M.P.Georgeff, "An Embedded Reasoning and Planning Systems", Advanced Proc. of the Rochester Planning Workshop, 1988, pp79-101.

[9] J.A.Stankovic, "A Serious Problem for Next-Generation Systems", IEEE Computer, Vol.10, No.21, 1988, pp10-19.

[10] R.Dodhiawala, et al., "Real-Time AI Systems: A Definition and An Architecture", Proc. of IJCAI-89, 256-261.

[11] H.A.Simon, The Sciences of the Artificial, MIT Press, 1985.

[12] K.S.Decker, "Distributed Problem Solving Techniques: A Survey", IEEE Trans. on System, Man and Cybernetics, Vol.SMC-17, No.5, 1987, pp729-740.

[13] V.R.Lesser and D.D.Corkill, "Distributed Problem Solving", in S.C.Shapiro (Ed.), Encyclopodia of Artificial Intelligence, John Wiley and Sons, 1987.

[14] R.G.Smith and R.Davis, "Frameworks for Cooperation in Distributed Problem Solving", IEEE Trans. on System, Man and Cybernetics, Vol.SMC-11, No.1, 1981, pp61-70.

[15] N.V.Findler and G.D.Elder, "Multiagent Coordination and Cooperation in a Distributed Dynamic Environment with Limited Resources", Artificial Intelligence in Engineering, 9(1995), pp229-238.

[16] E.H.Durfee, et al., "Trends in Cooperative Distributed Problem Solving", IEEE Trans. on Knowledge and Data Engineering, Vol.1, No.1, 1989, pp63-83.

[17] B.G.Buchanan and E.H.Shortliffe, Rule-Based Expert Systems, Addition-Wesley, 1984.

[18] L.Brownston, et al., Programming Expert System in OPS5: An Introduction to Rule-Based Programming, Addison-Wesley, 1985.

[19] A.Sabharwal, et al., "Asynchronous Production Systems", Knowledge-Based Systems, Vol.2, No.2, 1989, pp117-127.

[20] A.Sohn and J.-L.Gaudiot, "Representing and Processing Production Systems in Connectionist Architectures", Int. J. of Pattern Recognition and AI, Vol.4, No.2, 1990, pp199-214.

[21] Z.W.Pylyshyn, Computation and Cognition, MIT Press, 1984.

[22] J.A.Fodor, Modularity of Mind, Bradford, 1983.

[23] N.R.Jennings, "Commitments and Conventions: The Foundation of Coordination in Multiagent Systems", The Knowledge Engineering Review, Vol.8:3, 1993, pp223-250.

[24] G.Saucier, et al.(Ed.), Knowledge Based Systems for Test and Diagnosis, Elsevier Science Publishers, 1989.

[25] M.Li, et al., "A Real-Time Production Systems and its Analysis for the Application of Automatic Production on Space Station", Proc. of AMSE Int. Conf. on Information and Systems, 1991, pp761-765.

[26] R.Rada, Software Reuse, Intellect, 1995.

[27] P.Coad and E.Yourdon, Object-Oriented Analysis, Second Edition, Yourdon Press, 1991.

[28] M.Li, "Users-Oriented Requirements Analysis in Automated MIS Production", presented in the 12th Int. Conf. on CAD/CAM Robotics and Factories of the Future (CARS & FOF'96), 1996.

[29] L.Macaulay, "Requirements Capture as a Cooperative Activity", Proc. of IEEE Int. Symp. on Requirements Engineering, 1993, pp174-181.

[30] M.Li, "Cooperative Robot Problem Solving", presented in the 6th Int. Symp on Robotics and Manufacturing of 2nd World Automation Congress (WAC'96), 1996.

[31] M.Li, et al., "A Cooperative Intelligent System for Urban Traffic Problems", Research Paper No.822, Department of Artificial Intelligence, University of Edinburgh, July 1996.