ICOT Technical Report: TR-226

TR-226

—PROTON—
Expert System Tool on the PSI machine

by

Y. NAGAI, H. KUBONO and Y. IWASHITA

January, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

- PROTON -
Expert System Tool on the PSI machine

YASUO NAGAI, HIDEO KUBONO and YASUO IWASHITA

Institute for New Generation Computer Technology
4-28, Mita 1-Chome, Minatoku, Tokyo 108
Japan. Phone: Tokyo (03)456-3192. Telex: 32964ICOTJ

Keywords

Expert Systems, Expert System Tool, PSI machine, ESP, hybrid knowledge representation environment.

Abstract

An expert system tool called PROTON (PROtotype of expert system building TOol for the Next generation) is described. PROTON realizes a hybrid knowledge representation environment combining frames and rules and a metarule mechanism which controls multiple knowledge sources. Heuristic knowledge of the problem domain is represented in the rule system environment and static knowledge about components and relationships among them is represented in the frame system environment. PROTON's implementation is being carried out using the advantages of the Extended Self-contained Prolog (ESP) language, which has the features of logic and object-oriented programming and is running on the Personal Sequential Inference (PSI) machine of our institute. This paper gives a description of features of PROTON, it's frame and rule system environment, and it's user-interface environment.

1. Introduction

At present, expert systems constitute one of the most promising application fields of artificial intelligence. Their efficiency is recognized in some specific fields and extensions of the application area are in demand. First generation tools such as OPS5 and EMYCIN have many problems still to be solved, for example insufficient knowledge representation environment because of a single knowledge representation and a single inference mechanism and lack of the control mechanism for separating metaknowledge from the domain knowledge, etc. We are developing an expert system building tool called PROTON, which corresponds to the second generation tools such as ART, KEE and KC, on the personal sequential inference (PSI) machine, as the first step for solving these problems.

It is possible to carry out the research on the fundamental technologies necessary for knowledge information processing by implementing the application problems in various fields on PROTON, and thereby analyzing the application areas. Thus, PROTON as a tool is intended to be general purpose, to allow study of application problems in various fields.

PROTON is also utilized as an experimental system for evaluation of appropriate results of the Fifth Generation Computer System (FGCS) project in such areas as problem solving and inference mechanism, knowledge acquisition, knowledge-base management, intelligent user interface, and so on.

The design policy of PROTON is that it realizes a hybrid knowledge representation environment and implementation of this tool will be attempted by using the advantages of the Extended Self-contained Prolog (ESP) language, which has the features of logic and object-oriented programming. As the result of this policy, PROTON provides several good features, described in the next Section 1.1.

Overviews of PROTON are described in Section 1.2. , the frame system environment is described in Section 2, the rule system environment in Section 3, and a user-interface environment in Section 4.

1.1. Features of PROTON

Figure 1 shows a general configuration of PROTON. As a tool on the PSI machine, it has the following features:

(1) **Implemented using Extended Self-contained Prolog (ESP), it realizes rule and frame system environments based on object-oriented and logic programming.**

(2) **A comfortable user interface is provided by SIMPOS, PSI's Operating System.**

(3) **An open system is realized allowing users to add user-defined functional descriptions in ESP easily.**

(4) **A statistical data collection facility is incorporated.**

(1) means that PROTON realizes a hybrid knowledge representation environment based on the framework of ESP's object oriented mechanism and logic programming. Knowledge about components of problem domains is represented in the frame system environment, while knowledge about heuristics of the problem domain is represented in the rule system environment. Metaknowledge about the problem solving strategy can be described in the form of metarules.

(2) means that it provides two user interfaces, one for the knowledge engineer and one for the end user. The former provides a comfortable environment with the multiple windows and menus. The latter provides primitives for the explanation facility, as seen in an ordianry expert shell and user-ask facility.

(3) means that it can be easily customized by adopting new functions because of ESP's object oriented mechanism. In particular, the inference mechanism and user interface are designed to be easily customized by the user.

(4) means that it provides a data collection facility during execution for statistics such as the frequency of access to rules and frames, and system state transitions. The object of the data collection facility is to evaluate the tool and machine environment and also to study the characteristics of the application domain knowledge.

1.2. Overviews of PROTON

It should be noted that PROTON is a rule oriented system. In PROTON, the manipulations of WM elements and relations in a frame system environment are initiated by executing pattern matching for the WM element and their manipulation predicates in the rule system environment, as shown in Figure 2. Rule system environment consists of knowledge sources (KSs) and metaKS. KS contains the specification of the rules and inference control strategy , namely the control strategy for the rule application. MetaKS contains the specification of the metarules and inference control strategy of metarules. Working Memory (WM) consists of the WM for templates which corresponds to both templates of elements and relations (TEs and TRs), and the WM for instance objects. During the inference process, manipulations according to the requests from execution of rules are executed on instance objects instantiated from templates of facts (TF), composed of templates of elements and relations . The result of the manipulations are stored into WM as instance objects, via the working memory management system (WMMS) in a frame system environment. WMMS manages the above manipulations of WM and detail of it is shown in Figure 3. In this case, element *plate, block, and corn* are defined in the form of TEs and relationship among them , *on* is defined in the form of TR.

2. Frame system environment

The definition of elements in the problem domain and the relationships among elements can be represented in the form of TEs and TRs.

The TEs are equipped with an ordinary inheritance mechanism, and a demon mechanism to perform manipulations of attribute values. The system provides two built-in facilities, one to ask the user when attribute values are undefined and one to provide to explanation facilities (eg. WHY or HOW).

Definition of an user-defined relation among instances of elements can be given using the templates of user-defined relationships. A check of restriction conditions by using relations, can be represented by describing, to some degree semantically, an attached procedure. In the definition of relations, properties and restriction conditions can be inherited. The external descriptions of TE and TR are shown in Figure 4 . In this case, element *steel-block-with-bar* is represented in the form of TE and relationship *on* is represented in the form of TR. TE *steel-block-with-bar* consists of the *super* link name *block,* attributes composed of *material, grav, and weight,* and the *has-part* link name *bar.*

The frame system environment provides the description of inverse relations and the mechanism to search WM for the combination of the relations equivalent to the corresponding relation using the *substitution* function. The *substitution* function is described later. As the definition of a relation is represented in the form of TRs, relationships among more than two elements can be easily represented.

2.1. Inheritance mechanism

The inheritance mechanism is a process by which attributes of one instance object are assumed to be attributes of another instance object according to inheritance links in abstract hierarchies which consist of is-a hierarchy and has-part hierarchy. The mechanism is related to the instantiations of TE and TR. During the building of an expert system, the frequently used links, such as *super* and *has-part* links are predefined in the frame system environment. In the case of attribute value inheritance of TE's, the inheritance of attributes from templates specified in the *super* declaration is performed by checking all facets. In the case of the inheritance of TRs, the inheritance mechanisms check restrictions, and execute the ***substitution*** function. When the link information necessary for inheritance mechanism is the form of explicitly specified subordinate template objects, instance objects containing the inherited informations are made via the instantiation of the template objects. If the template object has multiple superordinate objects, a depth-first search for these superordinate objects is executed in a predefined order over the templates. Then, if the attribute values for the inheritance are specified explicitly in superordinate template objects, the search is terminated.

The inclusion relation that means the whole-part relation, can be realized according to instance links based on the *has-part* definition. When the instantiation information is specified in the *has-part* definition, the instances depending on this information are attached to the instance objects instantiated from the TE specified in the *has-part* definition.

2.2. Instantiation of template objects

The instance objects are instantiated from the definition templates of element and templates of relation and stored into WM. The identifier for instance objects, called *time-tag* , which shows the time when an object is instantiated in WM, is automatically added to each instantiated instance. The *time-tags* are also set to the latest value when they are modified. This identifier is used for conflict resolution of firable rules. On the instantiation of template objects, each attribute value is instantiated to a default value specified by the templates. If there exists no default value in the specification of the templates, the attribute value is set to *null* and variables for pattern matching are set to unbound when refering it.

During the inference process, the changes made to instance objects are executed in terms of manipulations of the attribute information of element instances and relation instances among them, such as addition, modification, and deletion.

2.3. Attached procedure

In template objects, a specific procedure called the attached procedure can be defined for any given attributes. The attached procedure is invoked at the time the access to the corresponding attribute value or instantiation of template objects is performed. It verifies, sets, or modifies the attribute values by executing a sequence of procedures or by making any possible queries to the user. These procedures are classified into the following three cases:

a. **Attribute values have some restrictions.**

b. **An attribute value is bound to some other attribute value.**

c. **Attribute values can not be predefined.**

Attribute values are checked at the same time as the modification of attribute values of instance objects. When errors occur due to this check, if an exception handling is described in the attached procedure form, this procedure is executed. Invocation priority of the attached procedure is higher than that of procedures used ordinarily in the system. When there exist inheritance relations, the attached procedure most adjacent to the current TE is executed. If the result of this procedure succeeds, the rest of the procedure is abandoned. The specification of attribute values of the TEs and TRs defined in the form of the attached procedures can be easily described by embedding the corresponding attribute names directly.

2.4. Definition of relations among instances

The manipulation of relations among instance objects are represented in terms of the creation, modification, and deletion of templates of relations. The number of objects involved as arguments of some relation instances can be more than two.

On the instantiation of relation instances by TRs, the element instance name is set to the argument specified in that TR, and, in the case of a rule matching pattern, the element instances of relation instances can be identified by the position of the argument in the argument list.

Moreover, a mechanism for the *substitution* is specified in the TR. When there are no relation instances requested for search among any of the element instances, the corresponding template of relation requests the search for combinations of relations, and equivalent relations based on the relations for the *substitution* represented in TR. When there exist no instance objects corresponding to the TR, the *substitution* function searches WM for the combinations of instance objects equivalent to the combinations of the relations for the *substitution* . Thus, the search mechanism for relation instances is performed by expanding relation instances until the equivalent relation instances can be found using the relations for the *substitution* .

3. Rule system environment

In the rule system environment, the rules can be grouped and moduralized in the form of multiple knowledge sources (KSs), as shown in Figure 2 . Thus, arrangement of knowledge is easy and inference can be performed efficiently by reducing the search space.

The access functions to element objects on working memory (WM) are initiated in the frame system environment by invocation from the rule system environments. They consist of manipulations such as creation, deletion, modification, search for an instance element having a certain pattern, and so on. ESP codes including methods and predicate defined in the user-defined class can be embedded into rules.

Inference type, inference control strategy, exit condition and goal to be verified for backward chaining reasoning can be specified in the KSs.

Metalevel knowledge is the knowledge about control of the KSs. It specifies the activation timing in the form of forward chaining rules called metarules. The left-hand side of a metarule gives the specifications of pattern matching with WM. The right-hand side of a metarule gives the specifications of WM modification and the KS to be executed. When KS inference type is backward chaining reasoning, the goal to be verified must also be specified on the right-hand side.

Rules are compiled into ESP code when executed.

3.1. Metaknowledge

Generally speaking, metaknowledge is knowledge in an expert system about how the system operates, or knowledge about the use and control of domain knowledge, namely knowledge about knowledge. Currently, it is introduced into the system with various architectures, such as frame system, rule system and an architecture integrating these. In many rule systems and frame systems, metaknowledge is represented in the form of hiearchical descriptions for metalayers, such as metarules and metaframes, and is separated from the domain knowledge. In the rule system environmemt of PROTON, when the application domain problem can be divided into subproblems and solved, the knowledge about control of subproblem solution is separated out as metaknowledge and provides a good prospect of the total system.

3.1.1. Description format

Metaknowledge consists of the metarules, their control strategy, and the termination condition of the recognize-act cycle of metarules. It's format is the same as that of forward chaining rule description. In addition to the metarule body, there are also the specifications of hitting strategies for rule and rule application priorities in metarule format. The hitting strategy for a rule specifies the activation of the rule, once only or many times. The left-hand side of metarules consists of the AND-combination of the WM element's pattern, user-defined predicates or methods, and ESP predicates, while its right-hand side consists of the AND-combination of the WM manipulation predicate, user-defined predicates or methods, ESP predicates, and the KS name. However, when the KS inference type is backward chaining, the goal pattern to be verified should be also given.

3.1.2. Control of knowledge source invocation

KS invocation control is performed in terms of metaknowledge and the inference mechanism of metaknowledge. Because various control mechanisms of KS invocation can be realized in the form of metarules, metaknowledge is represented in that form too. In this case, it is desired that the WM for KS control be separated from the WM for the problem domain, when the WM for a rule inference mechanism is designed.

The mechanism of control of KS invocation can be summarized as follows.

1. **Selection of one applicable metarule.**
2. **Execution of the KS specified in the right-hand side of this metarule.**
3. **If KS execution succeeds, then the actions specified in the right-hand side of metarule, in general WM manipulation predicates, are executed. If it fails, then end this metarule. If all applicable metarules have been exhausted, terminate.**
4. **Go to 1.**

If the user extends metaknowledge functions, then a blackboard model architecture can be easily designed. Details of this architecture are discussed later.

Items 1 and 2 above correspond to the recognize-act mechanism of metarules. There exist two strategies for this mechanism. One involves selecting the applicable rule and executing it immediately without making an agenda, and the other is the strategy of selecting one rule according to a certain procedure considering such parameters as rule application priorities and time tags, from the agenda made by evaluating the left-hand side of all rules

The KS control mechanism is shown in Figure 5. For example, when metarule 1 is selected and executed, knowledge source KS1 is activated and finally the rules in the KS1 are executed. If the KS1 succeeds, then the right-hand side of metarule1 is executed and the information for KS control in WM is updated or deleted. The user can indicate the exit option for checking the termination conditions in each rule's recognize-act cycle.

3.2. Knowledge Source (KS)

The advantage of introducing the knowledge source is that it makes it possible to perform the inference mechanism efficiently by grouping rules into multiple knowledge sources and by reducing the search spaces. In this system, there is no invocation path from one KS which is under execution to the other KS. In other words, the transfer between KSs must be through metaKS. If the problem domain cannot be divided into multiple subproblems, inference must be performed using only one KS. The inference type is determined in each KS as one of FC, BC, or MIXED.

3.2.1. Description format

Figure 6 gives an example of the KS description format. KS inference control involves the control strategy for the rule application, inference type and exit condition of the recognize-act cycle. When the KS inference type is backward chaining, goals can be specified statically to KS. There are two types of rule formats, for FC and BC. When the KS inference type is FC, KS rules are described in the FC rule format, when it is BC, KS rules are in the BC rule format, and when it is MIXED, KS rules are in the MIXED format using the combination of FC and BC rules.

Rule application priorities and rule hitting strategy can be specified as additional items to the rule description.

In the FC KS, namely rules from KS rules are described in only FC rule format: the right-hand side of the rule consists of user-defined predicates, ESP predicates and built-in predicates for WM manipulation, such as *make* , *modify* , and *remove* . The syntax of a BC rule is almost the same as that of an FC rule. But, it should be noticed that the right-hand side of a BC rule consists of the WM pattern or propose predicate, which is different from that of an FC rule. In the case of the WM pattern, the backward chaining is performed without modification of the WM, and in the case of the propose predicate, the addition to the WM is caused by executing right-hand side of rules containing the propose predicate. The user can describe a flexible rule specification using propose predicates.

Specifications about whether the user queries facts or not can be represented in the attached procedure form embedded in the template of fact (TF). WM element patterns can be specified by regarding the elements in the frame system as object-attribute-value triplets, and the relations among the elements as predicates. The *has-a* relation can be defined among elements in the frame system environment, and the inheritance mechanism of the instances among the templates of elements (TEs) is executed according to this relation. Thus, the inheritance link of the TEs can be represented using attribute names of WM element patterns in the rule description.

3.2.2. Inference type

The inference types in KS are:

* **Forward Chaining (FC)**
* **Backward Chaining (BC)**
* **MIXED reasoning (MIXED)**

Forward chaining (FC) is called data-driven or pattern-driven reasoning. FC performs the recognize-act cycle of applicable rules by entering the instance objects into the WM and updates the WM as a result until there exists no applicable rule in KS. A typical example is OPS5.

Backward chaining is called goal-driven reasoning. It verifies the goals by invoking BC rules and asking the user. A typical example is EMYCIN.

In general, mixed reasoning generates hypotheses by executing forward chaining rules and then verifies them by backward chaining rules. In PROTON, the mixed reasoning mechanism mainly uses the forward chaining rules, and BC rules are invoked and executed automatically to get the necessary data for invocation of FC rules. The inference mechanism of MIXED KS is shown in Figure 7.

3.2.3. Rule application mechanism

There are two control strategies for the rule application in a KS, namely the first-hit and conflict resolution strategies, which can be applied to the three inference types. Here, the inference mechanisms of FC KS and BC KS are omitted. The inference mechanism of MIXED KS is used in explanation of the following cases of first-hit and conflict resolution strategies.

1. **First-hit strategy**

   1) **Select one FC rule from MIXED KS.**

   2) **If the evaluation of the left-hand side of an FC rule selected succeeds, the execution of the right-hand side of this rule is performed.**

   3) **When step 1 fails, if the WM element patterns in the left-hand side of FC rule can't be matched against WM during recognition of this rule, then they are verified by invoking BC rules.**

   4) **If there exists no applicable rule, then terminate. Otherwise, go to step 1.**

2. **Conflict resolution strategy**

   1) **Make an agenda by selecting all applicable rules.**

2) **When step 1 fails, if the WM element patterns in the left-hand side of FC rule can't be matched against WM during recognition of all applicable rules, then they are verified by invoking BC rules.**

3) **Select one FC rule from agenda based on the given estimation and execute the right-hand side of this rule.**

4) **If agenda is empty, terminate. Otherwise, go to step 1.**

3.3. Association with blackboard model architecture

The concept of the blackboard model was adopted into the HEARSAY-II speech understanding system. In this system, the blackboard is a global data base which can be accessed by independent knowledge sources and used for means of communication with each other. Generally, it is said that the blackboard model architecture is an architecture with a global data base as the communication interface. The architecture consists of the three main components. They are the knowledge source, the blackboard data structure, and the control module. The knowledge source in the blackboard model architecture is almost the same as that of PROTON.

In PROTON, the blackboard data structure can be represented using the WM elements in a frame system environment. If desired, the blackboard data structure can be organized into an abstract hierarchy and be easily represented in the form of the WM elements using *has-part* and *super* link in a frame system environment. Then, if this data structure is desired to be described as an abstract hierarchy by uniquely using the whole-part relation, it can be easily represented as an abstract hierarchy by attaching a unique identifier to the has-part link. The general mechanism of the control module in the blackboard model architecture is as follows:

1. **Candidate generation of the action to be taken next (eg. KS invocation) by having the control manager continuously monitor blackboard changes.**
2. **Determination of focus of attention.**
3. **Selection (scheduling) of action to be done next by means of the focus of attention.**
4. **Invocation of KS according to changes of blackboard opportunistically.**
5. **Go to 1.**

Of course, some criteria are needed to determine when this control cycle is to be terminated.

In PROTON, the metaknowledge and its invocation mechanism almost correspond to the above control mechanism. For example, items 1 and 2 in the blackboard control mechanism corresponds to item 1 in the control mechanism of KS invocation of PROTON, item 3 to item 2 and item 4 to item 3. The activate condition in the blackboard model architecture corresponds to the left-hand side of a metarule. Thus, in PROTON, the various activate conditions can be represented using metarule, because the activate condition can be desribed in a metarule format. The corresponding KS is selected and executed by metarule invocation, and when KS execution is terminated, the right-hand side of a metarule, except for the KS invocation, is executed and elements in the WM are updated. This update process to the WM corresponds to the focus of attention, such as generation of event, goal, and so on.

In PROTON, the invocation mechanism of the metaKS monitors changes in the WM continuously and decides which metarule in the metaKS is executed next. From this iterative process, the sequences of KSs are performed dynamically as the result.

However, it is insufficient for PROTON's mechanism to represent the blackboard model architecture for the planning problem as in the case of B.Hayes-Roth [4], because PROTON does not provide the scheduling functions for effective utilization of the KSs. This results from the fact that PROTON is intended to be used, not for the special purpose of the design problem, but for general purposes.

3.3.1. KS invocation type

In the system with the blackboard model architecture, of course including PROTON, the KSs are invoked opportunistically. Invocation and execution of KSs correspond to solving the subproblems into which the problem can be divided. In most design problems, it seems that KS invocation is not performed very opportunistically and is performed almost procedurally. In other words, it appears that there exist many cases where only the subproblems into which a problem of the application domain can be divided, namely KSs, can be invoked procedurally and these are considered as the target for current design problems.

However, when considering a more complicated design problem as the target, which is more practical, it is thought that the KS invocation mechanism should not be performed according to the procedural approach, but rather according to the opportunistic approach. Naturally, considerable facilities for planning will also be needed, in order to deal effectively with these problems.

3.3.2. Necessary facilities

In order to achieve the above KS invocation mechanism, a sophisticated planning facility will be needed. Furthermore, an intelligent backtracking facility, especially for dependency-directed backtracking, will be needed to invoke KSs opportunistically. The reason why a dependency-directed backtracking is needed for opportunistic KS invocation is the fact that it stores the justifications for the KS invocation and the execution process. If the KS exectuion fails, it can then backtrack to the most promising alternative by using these justifications and thereby, facilitates efficient KS invocation and execution.

Of course, the justifications of KS rule execution must also be stored in order to use the dependency-directed backtracking approach. In this case, it is inefficient to apply it to all the rules over all the KSs as it would demand tremendously wastful computation. Therefore, only rules in the specified KSs should be applied in this approach.

In the future, PROTON will be extended to realize a framework including these planning (scheduling) and dependency-directed backtracking mechanisms.

4. User-interface facility

The user interface provides various facilities that assist in utilizing the system. In the case of the expert system building tool, the user can be classified into two types: the builder of the expert system, the Knowledge Engineer (KE), and the user of expert systems built by a KE, the End User (EU).

We think that the utilization stages of the tool are best represented by the following five stages:

(1) Knowledge representation

(2) System invocation

Here, KE and EU load the knowledge data constructed at the above stage and begin to execute inferences.

(3) Debugging

(4) System execution

This is the stage when EU executes and utilizes the expert system constructed by KE.

(5) System evaluation

This is the stage when the tool builder or KE evaluates the performance of the tool and knowledge base constructed by obtaining various data on system execution.

Items (1), (3) and (5) are discussed in the following sections.

4.1. Necessary facilities for knowledge representation

We consider necessary facilities for knowledge representation to be the editing and the browsing facilities. An editor is necessary to describe knowledge in a rule and frame description format.

The browser displays the global information about rules and frames:

* **Display of hiearchical DAG(Directed Acyclic Graph) expressions for relations among TEs and is-a hierarchy among TEs.**
* **Display of slot values inherited by certain TE's, their initial values, and demons**
* **Display of TE and TR referred to by rules**

In order to increase the efficiency of debugging, the information that the editing and browsing facilities provide should be accessable during debugging.

4.2. Facilities necessary for debugging

Debugging means the repetitive discovery and deletion of bugs. Functions are classified into three types: interruption, investigation, and revision.

- **Control of flexible actions = interruption**

This is the function that interrupts and resumes the action when possible, if the system is in execution. At an interrupt stage of the system, KE can investigate and change internal states of the system.

- **Probing the internal states of the system = investigation**

This is the function that comprehensively shows KE the internal states of the system, and the changes being made at the stage of the interrupt.

Trace

This displays the system's internal states at every execution of a rule. The contents of the display consist of an agenda, a firing rule name, and changes of WM as a result of the rule execution.

WM

The elements in WM consist of instance objects of TE and relations among TE instances. KE can have various viewpoints of WM. For example, there are cases when we wish to reference only certain TE instances, and relations corresponding to certain TE's. At the stage of probing WM, the search trees are prepared in order to deal with the various viewpoints of WM. KE can traverse these trees freely.

Inference history

The KSs and rules are displayed. The former shows the KS names executed sequentially up to the present. The latter shows the rule names executed in the forward chaining KS and the verification processes in the form of AND-trees in the case of backward chaining KS.

- **Revisions of internal states = changes**

This function helps to delete the bugs discovered at interrupts and investigations of the system, and to change the system internal states experimentally, in order to discover the bugs. Revisions are made on WM and rules.

Revisions on WM

Addition and deletion of WM elements

Rewriting the slot value of WM elements

Return to certain states in the past

The third item restores the current state to certain state in the past. It can return to the state just before the specified rules were executed and the KS's were activated.

Revision on rules

Addition and deletion of rules

Forced completion of inference actions

Selection of rules from the agenda to be executed next

4.3. Necessary facilities at the stage of system evaluation

The following data must be collected in order to evaluate the system.

Execution speed of rules

This shows the number of rule firings. The execution speed can be obtained according to differences of inference type and size of knowledge base. This is used as the parameter for comparation of performance with other tools.

History of changes within WM

This shows histories of the agenda and changes within WM. If the localities by refering to WM can be found using these data, it can be reflected in the architecture of WM.

Access frequencies to rules

In each rule of KS, they are obtained by collecting the number of registration of the rule to an agenda and firing of the rule in the case of forward chaining KS, and the number of references to the rule and successes of the rule in the case of backward chaining KS.

5. Conclusion

PROTON, the frame system environment, the rule system environment, and the user-interface environment were described.

PROTON provides several features, such as a hybrid knowledge representation environment, a comfortable user-interface, user-customization functions, and a statistical data collection facility.

Mainly static knowledge about components and relationships among them is represented, and attached procedure and inheritance mechanisms are provided in the frame system environment.

Because rules can be grouped and modularized in the form of multiple KSs in the rule system environment, the structure of rule representation is hierarchical and clear. Therefore, rule representation is easy and inference can be performed efficiently by reduction of search space. When the application domain problem can be divided into subproblems and solved, knowledge on control of subproblem solutions is separated out as metaknowledge (metaKS) which provides a good prospect of the total system.

User-interface environment provides various facilities that assist in utilizing the system. They consist of a browser, a debugger, an editor, and so on.

PROTON is currently under construction and is intended for experimentation. It will be evaluated by being applied to real application problems.

The results of evaluation will be used to improve the functions.

Acknowledgement

The authors wish to thank Jun Sawamoto and other members of the fifth research laboratory, ICOT, for helpful criticisms and comments.

Reference

[1] T. Chikayama, "Unique Features of ESP," Int'l Conference on Fifth Generation Computer Systems, November 1984.

[2] Y. Iwashita and J. Sawamoto, "DEVELOPMENT OF EXPERT SYSTEMS IN THE FIFTH GENERATOIN COMPUTER SYSTEMS PROJECT," 2nd Int'l EXPERT SYSTEMS Conference, London, September 1986.

[3] H. Kubono, J. Sawamoto, Y. Nagai and Y. Iwashita, "FACT/MODEL REPRESENTATION ENVIRONMENT IN AN EXPERT SYSTEM TOOL ON PSI," COMPCON '87, San Francisco, February 1987.

[4] H. Penny Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," THE AI MAGAZINE, Summer 1986.

[5] B. Hayes-Roth, "Blackboard architecture for control," Journal of Artificial Intelligence 26: 251-321.

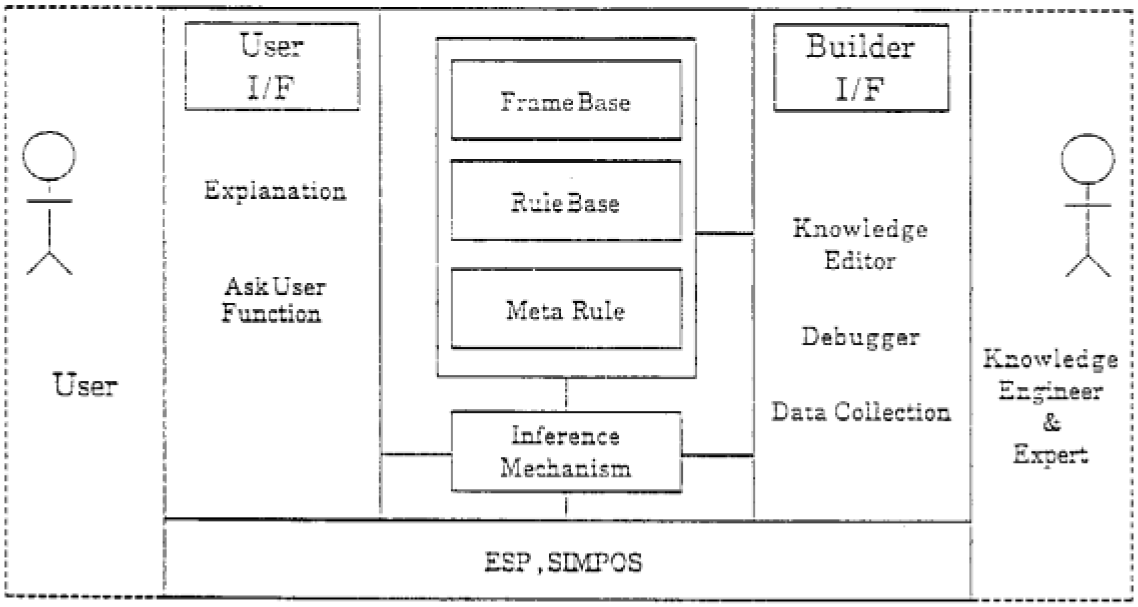

Fig.1 General Configuration

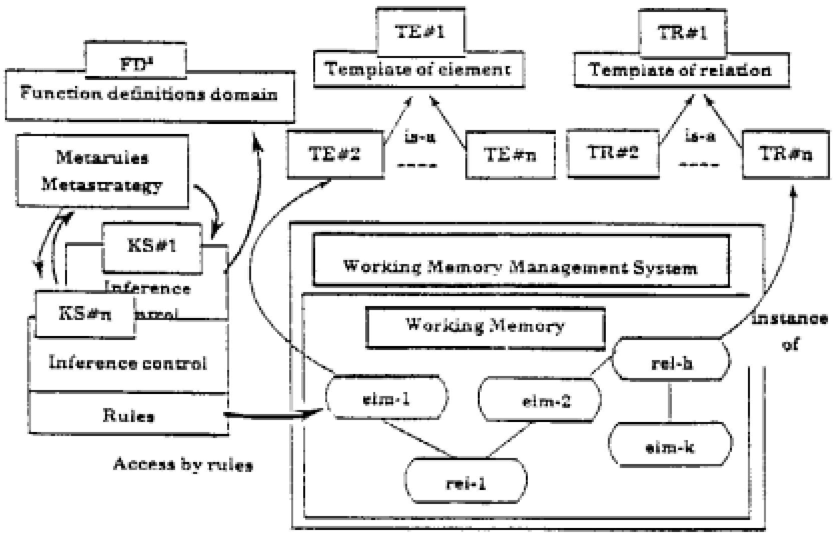

Fig.2. Structural overview of PROTON

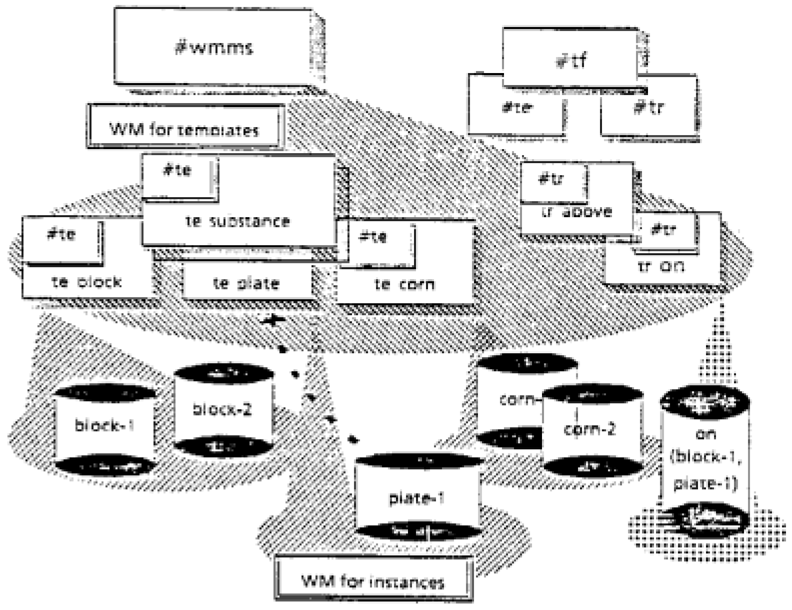

Fig.3. Working memory management system (WMMS)

```
te steel-block-with-bar
    super
        block
    attrib
        material default "steel" &
        grav default 5 &
        weight
            if-gotten , arbit , before
            weight
            <- grav * size-x * size-y
                    * size-z
    has-part
        bar { bar : (material , "steel") }
end.
```

(a)

```
tr on
    super
        above , contact
    restrict 2
        ( 1,substance ) ( 2,substance ) :
        ( 1,position-z ) > = ( 2,position-z )
    interpret       ( 1,position-z,new )
         <- ( 2,position-z,new )
        + ( ( 1,position-z,old )
        - ( 2,position-z,old ) )
    on ( [upper , lower ] )
        subst
    under ( [lower , upper ] )
    end.
```

(b)

Fig. 4. External representation (a) Example of TE (b) Example of TR

```
*-----------------------------------------------------------------
*          PROTON-PROGRAM OF Dilemma of Farmer
*-----------------------------------------------------------------

*** KNOWLEDGE SOURCE ***

ks:dilemma_of_farmer,
        strategy:crs,
        type    :fc.

preparation::
        hit:single,
        te:start:Start#(flg^on)
==>
        remove(te:start:Start),
        make(te:opposite-side:OS1#(this-bank^bank-1, another-bank^bank-2)),
        make(te:opposite-side:OS2#(this-bank^bank-2, another-bank^bank-1)),
        make(te:safe-check:Safe-check#(flg^ok)),
        make(te:repeat-check:Repeat-check#(flg^ok)),
        make(te:hypo:Hypo#(level^0,
                           parent^self
                           number^_
                           farmer^bank-1
                           fox^bank-1
                           goat^bank-1
                           cabbage^bank-1)).

farmer-to-another-bank::
        hit:multiple,
        te:opposite-side:OS#(this-bank^bank-2, another-bank^bank-1),
        te:hypo:Hypo#(level^No,
                      number^Number,
                      farmer^bank-2,
                      fox^Bank1,
                      goat^Bank2,
                      cabbage^Bank3)
==>
        {N is No + 1},
        make(te:hypo:NewHypo#(level^N,
                              parent^Number,
                              number^_
                              farmer^bank-1,
                              fox^Bank1,
                              goat^Bank2,
                              cabbage^Bank3)),
        {nl, write("Farmer only crosses to another bank")}.

farmer-and-fox-to-another-bank::
        hit:multiple,
        te:hypo:Hypo#(level^No,
                      number^Number,
                      farmer^Bank,
                      fox^Bank,
                      goat^Bank1,
                      cabbage^Bank2)
==>
        {N is No + 1},
        make(te:hypo:NewHypo#(level^N,
                              parent^Number,
                              number^_,
                              farmer^Opposide-bank
                              fox^Opposite-bank,
                              goat^Bank1,
                              cabbage^Bank2)),
        {nl, write("Farmer crosses to "), atom_to_string(Opposite-bank, OB),
         write(OB), write("with Fox")}.

end.
```

Fig.6.

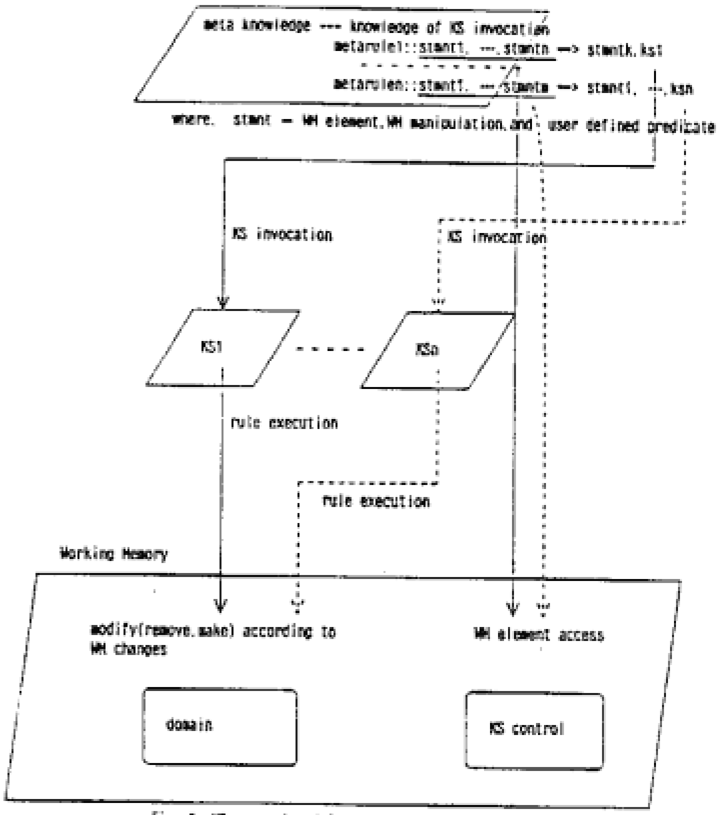
meta knowledge --- knowledge of KS invocation
metarule1::stmnt1, ---, stmntn ―> stmntk, ks1
metarulen::stmnt1, ---, stmntm ―> stmnti, ---, ksn
where, stmnt ― WM element, WM manipulation, and user defined predicate
KS invocation
KS invocation
KS1
KSn
rule execution
rule execution
Working Memory
modify(remove, make) according to WM changes
WM element access
domain
KS control

Fig. 5. KS control and invocation mechanism

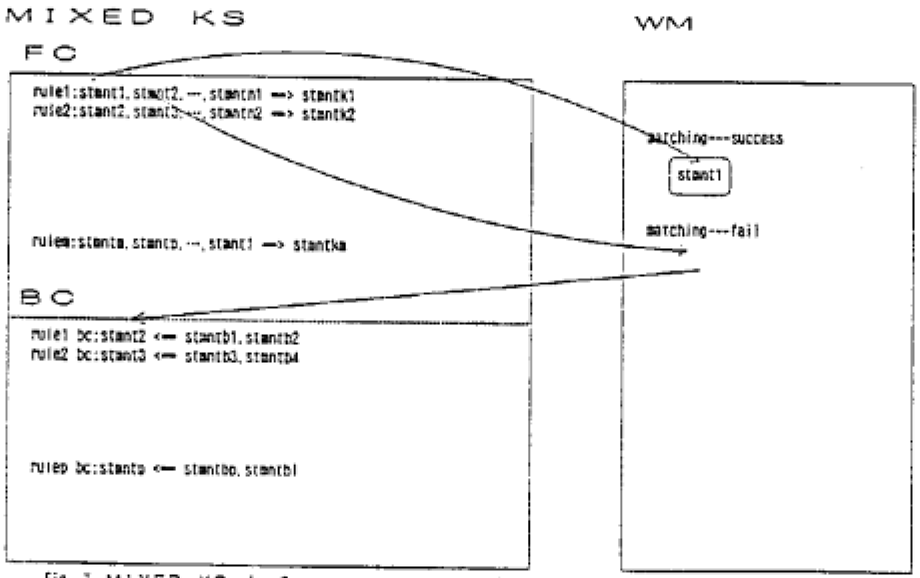
MIXED KS
WM
FC
rule1:stmnt1, stmnt2, ---, stmntn1 ==> stmntk1
rule2:stmnt2, stmnt3, ---, stmntn2 ==> stmntk2
rulem:stmnta, stmntb, ---, stmnt1 ==> stmntkm
BC
rule1 bc:stmnt2 <== stmntb1, stmntb2
rule2 bc:stmnt3 <== stmntb3, stmntb4
rulep bc:stmntp <== stmntbo, stmntbl
matching---success
stmnt1
matching---fail

Fig. 7. MIXED KS inference mechanism