TR-281

PROTON : An Expert System Tool on the PSI

by
S. Shindo, Y. Hosono (Mitsubishi), J. Sawamoto
H. Kubono, Y. Nagai and Y. Fujii

June, 1987

PROTON : An Expert System Tool on the PSI

J. Sawamoto, S. Shindo[1], Y. Hosono[1], H. Kubono, Y. Nagai, Y. Fujii

Institute for New Generation Computer Technology

4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan
phone: +81-03-456-3192 telex: ICOT J 32964

csnet:sawamoto%icot.jp@relay.cs.net
uucp:sawamoto%icot.uucp@eddie.mit.edu

ABSTRACT

This paper describes PROTON, an expert system tool (environment), on the PSI[2]. PROTON is a second generation expert system tool that provides three major knowledge representation constructs: production rules, frames and meta-rules. PROTON is implemented in ESP[3]. This paper discusses major features of PROTON architecture, i.e., the organization of each knowledge representation construct and the integration of the three knowledge representations. It also shows how PROTON is implemented by taking advantage of ESP's logic programming and object-oriented features. The usage of PROTON is demonstrated by an example of a design support expert system.

1. Introduction

The personal sequential inference machine (PSI) is one of the major results of the first three-year stage of the FGCS project. The PSI is currently used as one of the major software development environments in the FGCS project. Development of an expert system development environment on the PSI is necessary, because researchers of application knowledge systems need a powerful environment for the PSI, and because the development of an expert system tool is by itself a good research area for AI research activities such as problem solving methodologies, inference mechanisms, knowledge acquisition methods and intelligent user interfaces in the project. PROTON serves as a basic environment for those research activities [Iwashita 86].

1. Central Research Laboratory, Mitsubishi Electric Corporation, 1-1, Tsukaguchi-Honmachi 8-chome, Amagasaki, Hyogo, 661, Japan
2. The PSI is a Prolog based machine developed at ICOT in the Japanese Fifth Generation Computer Systems (FGCS) Project [Taki 84].
3. ESP (Extended Self-contained Prolog) is Prolog extended with an object-oriented mechanism. It is the end user language for the PSI as well as the system description language used for the PSI's operating system, SIMPOS, and its utilities [Chikayama 84].

PROTON is a second generation expert system tool, like ART [Clayton 85], KEE [KEE 86] [Fikes 85] and KC [KC 85], which provides hybrid knowledge representation schemata. ESP provides both logic and object-oriented programming. PROTON facilitates a frame-based and rule-based programming environment on the ESP programming environment.

Knowledge representation constructs most appropriate for the knowledge in the following three categories are provided in PROTON.

- *Static knowledge of the problem domain*
- *Knowledge for problem solving heuristics*
- *Knowledge for meta-level strategies*

Static knowledge is the knowledge required for modelling problem domains. A domain can be described in terms of its structural objects, attributes and values of objects, and relations among objects. The hierarchical and relational representation of frames is most suited to this type of knowledge. Knowledge for problem solving heuristics is the expert's domain knowledge used for problem solving. Production rules are most appropriate for this type of knowledge. Production rules search objects or relations of the problem domain for their premises and modify objects or relations in the action part. Knowledge for meta-level strategies is the knowledge for controlling the usage of the domain knowledge. Meta-rules are considered for providing this type of knowledge formalism.

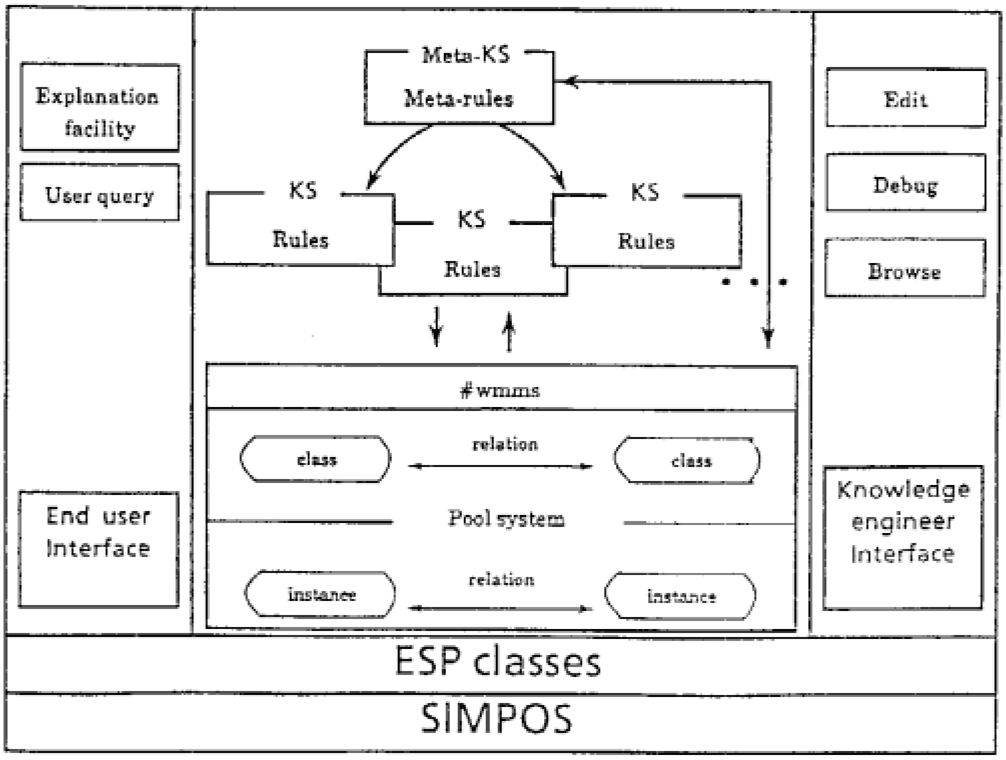


Figure 1 Outline of the PROTON configuration

Figure 1 shows an outline of the PROTON configuration. As an expert system tool on the PSI, PROTON has the following characteristics.

(1) Three knowledge representation constructs, production rules, frames, and meta-rules are integrated under ESP's object-oriented programming. Therefore, each construct can be used separately or in combination according to the application system's configuration. Thus, PROTON can provide a flexible and powerful knowledge representation environment on the PSI.

(2) Rules (including meta-rules) are modularized into multiple knowledge sources (KSs). KSs are implemented as class objects of ESP. KSs can be viewed as a special case of ESP coding which allows description of the rule format.

(3) Frames are also implemented as class objects of ESP. Class objects and their instances (instance objects) are stored in the *pool system* of the PSI (working memory: WM) and accessed by sending messages to the ESP class, working memory management system (#wmms), in PROTON.

(4) PROTON provides two kinds of user interfaces, one for the knowledge engineer (KE) and the other for the end user (EU). For KEs, PROTON provides an editor, a rule and frame debugger, and a browser. They are implemented using PSI's multi-window and menu system. For EUs, PROTON provides primitives for generating end user interfaces, such as a graphical explanation facility and a user query facility for the developed expert systems.

The next section first describes the major features of PROTON's knowledge representation. Section 3 discusses the implementation issues of PROTON on the PSI machine. Section 4 presents current applications on PROTON.

2. Features of PROTON

2.1. Frame-based Representation

Frames are basically very similar to classes of object-oriented mechanisms. In PROTON, frames are designed by incorporating necessary additional functions, such as relations, has-part, facet facility (constraint, default, etc.), and attached procedures, on the basis of ESP class mechanism [Kubono 86].

(1) Relations

There are two types of relations, inheritance relations (e.g., is-a, instance-of) and non-inheritance relations (e.g., set relations or any arbitrary relations).

Inheritance relations : *is-a* and *instance-of* relations are expressed by the token, *super*. They are not distinguished, because they play exactly the same role in PROTON from the functional point of view.

```
te mark_car                      te car                        te vehicle
  super car                        super vehicle                 attribute
  attribute                        attribute                       capacity
    id                               maker                       end.
    owner "Mark"                   has_part
    color if_modified after          engine(te:engine)
      esp:(:display(#window,       end.
           id,color))
end.
```

Here, *te* stands for template of element (TE). *mark_car* inherits the properties of *car* and *vehicle*. *mark_car* is an instance of the *car*. The *car* has an engine as one of its parts. The *color* attribute of *mark_car* has an attached procedure of the ESP method. These declarations correspond to the class definitions. At runtime, instances (in the sense of class-instance) of those classes are instantiated and accessed. Multiple instances of *mark_car*, for example, can be instantiated at the same time, supporting the situation that Mark has more than one car.

Non-inheritance relations : The following example shows a very simple declaration about academia. This example includes two set relations, *prototype-of* and *element-of*.

```
te professors                    te professor                  jones
  attribute                        attribute                     super professor
    duty                             name                        end.
    status                           major
end.                             end.

  tr prototype-of                              tr element-of
    substitution                                 restrict(professors,jones)
      prototype-of(X,Y)                          substitution
          -- has-prototype(Y,X)                    element-of(X,Y)
  end.                                                 -- has-element(Y,X)
                                               end.
```

Then, relation instances *prototype-of(*`an instance of` *professors,* `an instance of` *professor)* and *element-of(*`an instance of` *professors,* `an instance of` *jones)* should be created. (`an instance of` is omitted in the following description.)

Here, *tr* stands for template of relation (TR). PROTON provides a separate expression, *tr*, for non-inheritance relations. *restrict* specifies that the relation is valid only among the indicated classes. It is also true that *tr* declarations correspond to the class definitions of ESP. To improvise the specification mechanisms for inverse relations and the relation's transitivity, *substitution* is introduced. For example, the *has-prototype* relation is the inverse relation of the *prototype-of* relation.

It is not always sufficient to declare *element-of* as a relation only between *professors* and *jones*. We might want to use the *element-of* relation with a more general meaning (the easiest way is, of course, just to remove the *restrict*). In that case, the relation between *professors* and *jones* should be an instance of the *element-of* relation. Then, the following modified code is obtained.

```
tr element-of
  restrict(group,individual)
  substitution
    element-of(X,Y)
        -- has-element(Y,X)
end.
```

```
tr p-j-element-of
  super element-of
  restrict(professors,jones)
end.
```

For the request of *element-of(professors, jones)*, PROTON tries to establish the requested relation by searching the subordinate relations (e.g., *p-j-element-of* is subordinate to *element-of*), which gives the relation between the given objects. This mechanism is based on the idea that the subordinate relation is just a more specific instance of a superordinate relation.

In the following segment of substitution code, transitivity of relations is more fully demonstrated.

```
descendant-of(X,Y) -- son-of(X,Z),descendant-of(Z,Y)
descendant-of(X,Y) -- daughter-of(X,Z),descendant-of(Z,Y)
descendant-of(X,Y) -- son-of(X,Y)
descendant-of(X,Y) -- daughter-of(X,Y)
```

By this formalism for relations, PROTON enables users to describe the hierarchy of relations and the Prolog-like specification of the transitivity path. It is also possible to carry out searches just by specifying relation names, then obtaining objects which satisfy those relations, because the relation itself is treated as a frame object.

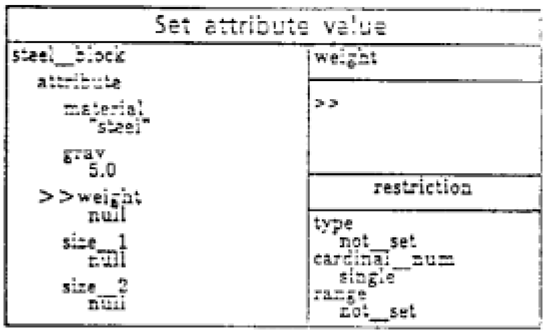
(2) Attached procedures

Table 1 summarizes the invocation timing of attached procedures. For instance, the combination of *if modified*, *null* and *after* indicates the usual "if-deleted" condition.

Items	*Options*
Event	*if_got/if_modified/if_error_occurred*
Slot value	*null/any value*
Timing	*before/after*

Table 1 Invocation timing of attached procedures

An ESP method call, a forward-type rule or the user query are allowed as procedures. When the user query is specified, a window (Figure 2) will appear and the end user will be asked about the value of the slot concerned.



```
Set attribute value
steel__block                 weight
  attribute
    material                 >>
      "steel"
    grav
      5.0
  >>weight                   restriction
      null
    size__1                  type
      null                     not__set
    size__2                  cardinal__num
      null                     single
                             range
                               not__set
```

Figure 2 User query window

2.2. Rule-based Representation

(1) Multiple knowledge sources

Rules can be organized into groups, and the selection of a certain group at a given time can be described using the knowledge of meta-level strategies. Usually, this organization is realized by utilizing the context mechanism [Brownston 85]. PROTON introduces the multiple knowledge source (KS) mechanism for this purpose. Using this mechanism, problem solving knowledge (rules) is well modularized, and from the implementation point of view, rules are executed efficiently because the KS mechanism limits the size of the conflict set at any one time. The following example shows a fragment of a KS coding.

```
%%% KS Header %%%%%%%%
ks : farmers_dilemma,
strategy : crs,
type : fc.
%%% Now rules follow %%%%%

init ::
        hit : single,
        te:start:Start#(attr^_)
    ==>
        remove(te:start:Start),
        make(te:hypo:_#(farmer^side_1,fox^side_1,
            goal^side_1,cabbage^side_1)),
        {:create(#standard_io_window_sq,[size(300,400),
            position(manipulator),title("FARMERS DILEMMA")], W),
        :show(W)}.
```

In the header part, the KS has the declaration for its name, conflict resolution strategy, inference type and exit condition. Table 2 shows the strategies and types currently supported.

Strategy	**do1**	*Execute the first hit rule*
	crs	*LEX type conflict resolution (recency, severity of LHS, priority)*
Type	fc	*Forward chaining*
	bc	*Backward chaining*

Table 2 Supported strategies and types

Each forward rule can have a specification on hit type and priority. For the hit type, two types, single and multiple, are allowed. Single means that the rule can be fired only once during an invocation of the KS, while multiple means that this restriction does not exist.

A forward rule has a left hand side (LHS) and a right hand side (RHS). On the LHS, WM elements, TEs or TRs, are specified for pattern matching. On the RHS, actions for the WM, make, modify or remove, are given to modify the WM. ESP predicates or methods, system defined or user defined, can be specified (inside the braces) on both the LHS and RHS, enhancing the expressive power of the rules.

Backward rules have different syntactic appearances. A rule corresponds to a Horn clause of Prolog. The LHS of the rule is the head of a Horn clause and the RHS is its body. When a propose predicate is specified to the LHS, the WM instance of the LHS will be asserted to the WM after successful chaining of backward rules. The major difference of the inference mechanism of the backward rules between PROTON and Prolog is that, in PROTON, predicates are WM elements and the WM must be searched to check whether they exist before other rules which can prove them backwards are searched for.

(2) Meta-rules

The knowledge about the invocation of the KSs belongs to meta-level knowledge. PROTON provides a KS (called a meta-KS) with forward rule format to express meta-level knowledge. The following code shows part of the sample coding of a meta-KS.

```
meta : meta_control,
strategy : do1,


first :
        te:context:C#(goal^get_paths)
    ==>
        call-ks(get_paths),
        modify(te:context:C#(goal^report_result)).


next :
        te:context:C#(goal^report_result)
    ==>
        call-ks(report_result), halt.
```

Invocations of KSs are specified on the RHS of the meta rules. Invocation of a KS directly from other KSs is not permitted. For the invocation of a KS with backward chaining rules, goals to be proved are attached. A backward KS may fail to prove the given goals. In that case, the execution of the RHS is curtailed at that point.

The advantage of rules over other programming systems is their modularity. However, rules interact with each other, and it is hard to guarantee the modularity of rules especially when controls are to be incorporated in rules. The grouping technique using context elements (control elements) is very common for implementing controls. Meta-rules can be viewed as the collection of the context element portion of domain rules separated from the main body of rules. By introducing the meta-KS mechanism, it is very easy to implement controls in rules, and domain rules in KSs can be organized with much more modularity when controls are not considered.

PROTON takes a rather conventional way in extending the rule-based representation, namely, the introduction of meta-KS and KSs. However, the meta-KS and KS mechanism in PROTON can be viewed as a simple implementation of the blackboard model [Nii 86]. The blackboard architecture generally consists of three major components, the knowledge sources, blackboard data structure and control. The knowledge source mechanism in PROTON is almost the same as that of the blackboard model. The blackboard data structure can be represented using objects and relations in the frame-based representation in PROTON. The control of the blackboard model performs opportunistic reasoning: data-driven and expectation-driven reasoning by responding to the changes of the blackboard data and determining the focus of attention. PROTON's meta-rules (forward rules) easily perform data-driven reasoning. However, an implementation of the sophisticated full control mechanism of the blackboard model demands additional coding by the user.

3. Implementation of PROTON

ESP's logic programming and object-oriented features are useful for implementing PROTON's knowledge representation environment. Unification and backtracking mechanisms are powerful for the implementing matching mechanism of the recognize cycle of rules. The frame system's configuration is fully dependent on the class mechanism of ESP, which is a very convenient and efficient environment for implementing the frame system.

3.1 Frame

(1) Inheritance

The TE or TR frame's inheritance relation, *super*, is mapped (translated) onto the ESP class-subclass inheritance mechanism, *nature*. Since all inherited attributes are copied once at compilation (by the ESP compiler), the inheritance has the meaning of the default setting at the initial setup rather than referring to attributes of the *super frames* at execution. This rather static inheritance mechanism was chosen for ESP because of its execution efficiency, although some flexibility was sacrificed.

In the case of multiple inheritances, *super frames* of a frame form a tree, called an inheritance tree. PROTON searches left-most and depth-first for the inherited attributes on the inheritance tree just like ESP does. For example, if a class X inherits from classes A and B in this order, and A from A1 and B from B1, the inheritance order for X is:

$$X \text{ -> } A \text{ -> } A1 \text{ -> } B \text{ -> } B1.$$

(2) Implementation of frame's features

This section shows how *substitution* and *attached procedure* are implemented. TEs and TRs inherit classes #te and #tr as defaults. #te and #tr inherit a class, #tf, for their common methods. They are all classes defined by the PROTON system. Classes #te and #tr give the basic methods for realizing the frame's functions, e.g., searching for certain instances in the given TE or TR classes, and accessing attributes of the given TE or TR instances. Class #tf gives common functions, such as clearing pools for instances of the given classes, creating and deleting instances.

Substitution : The *substitution* specification is translated into ESP codes. The following code shows a fragment of class, #tr and the translated ESP code of the *prototype-of* relation (in Section 2.1). First, a request for the search of an instance of *prototype-of* relation goes to class #wmms. If the request fails, then the message, *:request_subst*, is sent to the *prototype-of* class for the substitutional fulfilment of the request. In class #tr, the method, :request_subst, is also given for termination, if no *substitution* is specified in the TR concerned. The call of the method, *:refind_rel_first*, is the recursive request for the relation, *has-prototype*, to #wmms.

```
class tr has
  nature tf;
  :ask_find_rel_first(Class, Relname,
              Arg-list, Time-Tag) :- .....

  :request_subst(Class,X) :- !,fail;

end.

class prototype-of has
  nature tr;

  :request_subst(Class, [X,Y])
              :- :refind_rel_first(#wmms,has-prototype,_,[Y,X],_);
end.
```

Attached procedure : The body of the userspecified attached procedure is easily translated into the definition of the ESP method. The following code shows a fragment of class #te and the translated ESP code of the *mark_car* object (in Section 2.1). The access request for an attribute is expanded with calls of methods for attached procedures, e.g., *:modify_after_set*. The dummy definitions of clauses for those methods for attached procedures which always return successfully are coded in class #te, anticipating the case of no specified attached procedure.

```
class te has
   nature tf;

   :modify_after_set(Obj,_) :- true;

end.

class mark_car has
   nature car, te;
   instance
      attribute id, owner:="Mark", color;

      :modify_after_set(Obj, color)
               :- :display(#window, Obj!id, Obj!color);

end.
```

3.2 Production Rules

(1) KS and meta-KS

The meta-KS and KSs are translated into ESP classes. The KS invocation is implemented as sending a message to the KS. An invocation message to a KS from the meta-KS goes to the top level inference manager once for supervisory tasks, then an actual invocation message is sent to the KS. (See Figure 3.) Forward KSs always succeed, while backward KSs' failures are returned as the failures of the invocation messages.

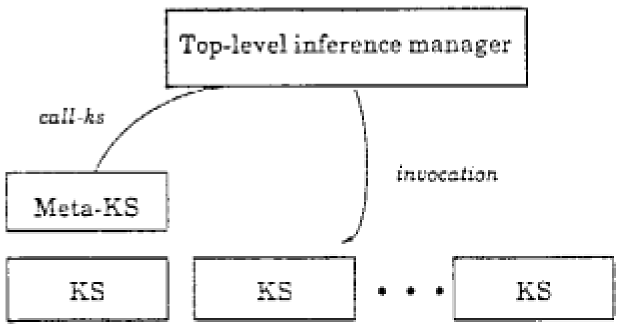


Figure 3 KS invocation

(2) Rules

Rules are translated into ESP code directly rather than preparing a rule interpreter or compiling into the Rete network. An easy yet efficient implementation of rules is sought taking advantage of the ESP's unification and backtracking mechanism.

For forward rules, the LHS and RHS are translated into ESP clauses separately. In the recognize cycle, LHS codes are checked one by one for executable rules utilizing backtracking, and the rule names and unification information are collected as an agenda (in the case of *crs* strategy), then followed by the conflict resolution. In the act cycle, the RHS code of the rule selected from the agenda is executed. Although a LEX-like conflict resolution strategy is given as a standard strategy in PROTON, it is easy to implement a user defined strategy by replacing the small *crs* class in PROTON with the user's definition.

Backward rules are more easily translated into ESP code. For example, the following backward rule means that if *sub_goal_1* is proved or exists in the WM then *goal* is proved and, as a result, *goal* is asserted into the WM.

```
r1 ::
propose(te:goal:_#(attr^normal))
<==
te:sub_goal_1:_#(attr^normal).
```

Then, the above rule is translated into the following ESP clause. *fetch_bc* of *#top_bc* tries to prove *sub_goal_1* first by searching the WM, then by issuing the *:rule_bc* method to this backward KS class recursively. Finally, the proved *goal* is asserted into the WM.

```
:rule_bc(Bc_ks_class, r1, [te,goal,_,[[attr,normal]]) :-
  :fetch_bc(#top_bc, [te,sub_goal_1,[[attr,normal]]),
  :generate_elm(#wmms,goal,[[attr,normal]]);
```

3.3. Integration with PSI Environment

The only underlying programming system of PROTON is ESP on the PSI. ESP functions, user defined or system defined, can be easily incorporated into the description of rules or attached procedures in frames. In rules, segments of ESP code are enclosed by braces and placed anywhere on both the LHS and RHS.

The PSI is supported by the *single language principle*, that is, the use of only one high-level language for describing everything from the operating system to application programs. The advantage of this principle is that the system can be quite open to users. The user's application programs can use any level of system facilities through the *inheritance* mechanism of ESP. PROTON users can easily utilize the PSI's window manipulation system, menu system, graphic system or file system, simply by inheriting provided classes.

4. Applications on PROTON

PROTON has been used so far to implement two demonstration-level expert systems, a support system for the design of the skeleton of a camera lens and a diagnosis system for the thyristor converter equipment of the cold rolling mill in steel factories, mainly for evaluating PROTON's features. More elaborate expert systems on PROTON are under way in the project, such as a planning system for the time and route schedule of material distribution by trucks in a local area and a CAD system for mechanical design.

One example clearly showing the usage of PROTON's knowledge representation is the camera lens design system. The support system for the design of the skeleton of a camera lens is a typical parametric design system for the zoom lens of a camera consisting of five lens groups. The task of the system is to determine the skeleton parameters, e.g., the focal length of each lens group, mutual displacement of lens groups and diameter of the diaphragm, under given constraints, e.g., the total length of the lens system, zoom ratio and diameter of the front end lens group.

The design procedure is generally depicted as in Figure 4. The first step of the procedure is to select an appropriate set of parameters (a hypothesis). The second step is to calculate a more precise skeleton configuration under the given set of parameters. The third step is evaluation where design constraints are evaluated by calculating the constraint parameters. The fourth step is to identify the reasons why the current design is not satisfactory, using the expert knowledge of the designer. The fifth step is to modify the current set of the design parameter following the knowledge of the design expert.

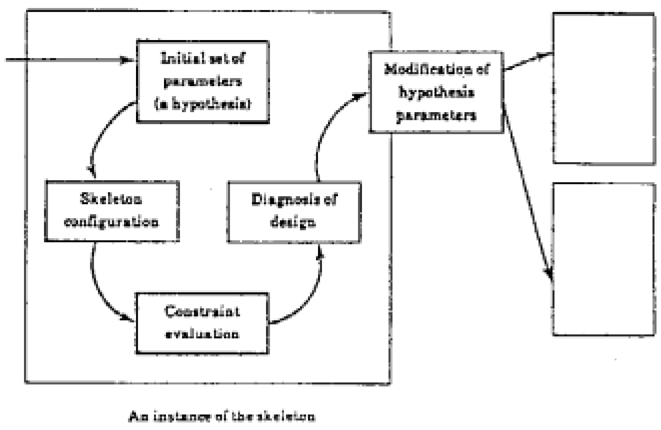

Initial set of parameters (a hypothesis)
Modification of hypothesis parameters
Skeleton configuration
Diagnosis of design
Constraint evaluation
An instance of the skeleton

Figure 4 Design procedure

Numerical simulation progams are used in the second and third steps. This expert system can be viewed as an intelligent front end system for conventional simulation programs. Domain knowledge of the design expert is separated and organized as sets of rules, i.e., knowledge sources.

The system configuration in PROTON is shown in Figure 5. Each knowledge source corresponds to one step of the design procedure. The meta-KS expresses knowledge about the overall design procedure. The knowledge shows how and when to use the domain knowledge expressed by each knowledge source. The system consists of roughly 50 rules in total and an additional 500 lines of ESP code for implementing simulation programs and graphical user interfaces. Figure 6 shows the display of the PSI running this system. The user of the system interacts with the system through a graphical window which displays a tree of hypotheses of the skeleton. One hypothesis usually makes several modifications possible. The choice is left to the user who is supposed to be an expert in this domain. The user can also confirm the current design visually in another window.

This lens design expert system is also implemented in ART on ICOT's Symbolics machine. It is found that PROTON's meta-KS and KS mechanism rather than the delicate viewpoint mechanism of ART is more suited to this type of general parametric design problem.

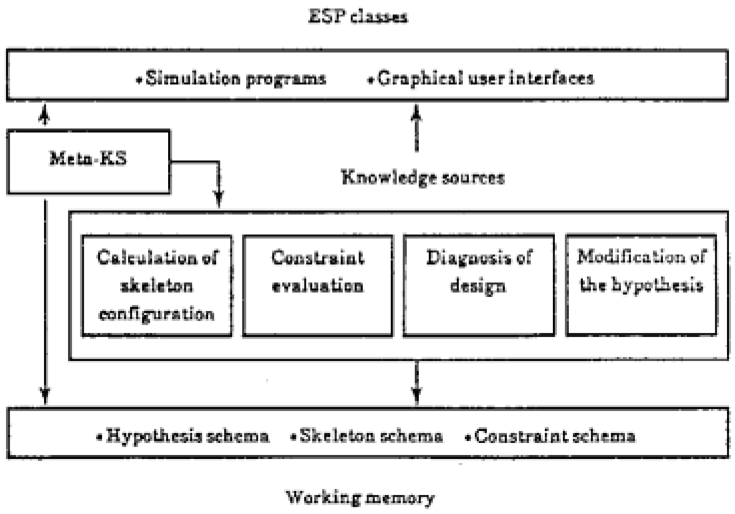


Figure 5 System configuration in PROTON

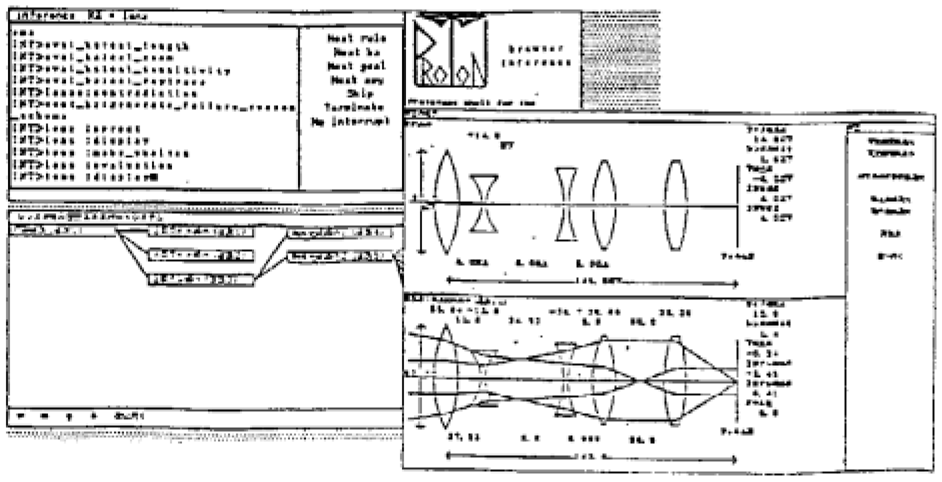


Figure 6 Display window of the lens design system on PROTON

5. Conclusion

This paper described PROTON, an expert system tool on the PSI. PROTON provides three major knowledge representation constructs, production rules, frames and meta-rules. These knowledge representation constructs are implemented and integrated utilizing ESP's logic programming and object-oriented features. The usage of PROTON is shown in the example of the design support expert system.

A more detailed evaluation of PROTON is currently being carried out by implementing expert systems in various problem domains. More efficient implementation of the production rules and frames is being sought.

For the expert system tool or expert system development environment, three levels of support hierarchy can be identified, the programming language level, general problem solving level and generic task level [Chandrasekaran 86]. PROTON, like other general expert system tools, provides the lowest level of support, that is, the programming language level. Facilitating higher levels of support is the focus of future research. At ICOT, we are working on the higher level of problem solving mechanisms, such as hypothetical reasoning by ATMS [de Kleer 86], distributed cooperative reasoning and qualitative reasoning on the PROTON environment.

ACKNOWLEDGEMENTS

We would like to thank members of the fifth research laboratory, ICOT, for their valuable discussions on the design and implementation of PROTON. Part of the implementation is due to K. Tsubaki of the fifth research laboratory and M. Hoshi, H. Ohsaki, and K. Amanuma of JIPDEC (Japan Information Processing Development Center). We also thank T. Asano and K. Kikuchi of Canon Research Center of Canon Inc. for providing us with the knowledge for the lens design system. (This work is partly supported by the ICOT-JIPDEC AI Center.) Our thanks must also go to Dr K. Fuchi, Director of ICOT Research Center, who gave us the opportunity to conduct this research in the Fifth Generation Computer Systems Project.

REFERENCES

[Brownston 85] L. Brownston et al., *Programming Expert Systems in OPS5 : An Introduction to Rule-based Programming*, Addison-Wesley Publishing, 1985

[Chandrasekaran 86] B. Chandrasekaran, *Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design*, IEEE Expert, 23/30, 1986

[Chikayama 84] T. Chikayama, *Unique Features of ESP*, in Proceedings of *FGCS'84*, ICOT, 1984

[Clayton 85] B. Clayton, *ART Programming Tutorial*, Inference Corporation, 1985

[Fikes 85] R. Fikes, T. Kehler, *The Role of Frame-based Representation in Reasoning*, Communication of the ACM, 28, 9, September 1985

[Hayes-Roth 83] F. Hayes-Roth, D.A. Waterman, D.B. Lenat, *Building Expert Systems*, Addison-Wesley Publishing, 1983

[Iwashita 86] Y. Iwashita, J. Sawamoto, *Development of Expert Systems in the Fifth Generation Computer Systems Project*, 2nd Int'l Expert Systems Conference, London, September 1986

[KC 85] *KC 3.0 Reference Manuual*, CGI, 1985

[KEE 86] *KEE Reference Manual*, IntelliCorp, 1986

[de Kleer 86] J. de Kleer, *An Assumption-based TMS*, Artificial Intelligence, 28, 2, 1986

[Kubono 87] H. Kubono, J. Sawamoto et al., *Fact/Model Representation Environment in an Expert System Tool on PSI*, Compcon'87, San Francisco, February 1987

[Nii 86] H. Penny Nii, *Blackboard System: The Blackboard Model of Problem Solving and the Evolution of Black board Architectures*, The AI Magazine, Summer 1986

[Taki 84] K. Taki, *Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)*, in Proceedings of *FGCS'84*, ICOT, 1984