ICOT Technical Report: TR-174

TR-174

An Integrated Knowledge Representation Scheme for Expert Systems

by
Hiroo Takenouchi and Yasuo Iwashita

May, 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

AN INTEGRATED KNOWLEDGE REPRESENTATION SCHEME FOR EXPERT SYSTEMS

Hiroo Takenouchi and Yasuo Iwashita,
Institute for New Generation Computer Technology, Japan

Keywords: *Expert Systems, Knowledge Representation, Integration, Predicate Logic, Function, Procedure, Production Rule, Unification.*
Abstract: The purpose of Knowledge representation for an expert system is to specify functions to be performed by the system. In this paper a knowledge representation scheme which mutually combines procedures, functions, production rules and Horn clauses, is outlined. Its knowledge representation model is an imaginary organization for performing functions of a target system, where a number of members try to solve given problems systematically. Knowledge is distributed to each of the members with considerable modularity. Functional specification of expert systems would be performed with less difficulty.

1 INTRODUCTION

The basic role of knowledge representation for an expert system is to specify functions to be performed by the system. Several knowledge representation models, such as the production system and frame system models, give us some guidelines on how to represent various types of knowledge (Ref. 1-5.). However, none of them is thought to be sufficient by itself, because an individual model may bring out both advantages and disadvantages depending on various characteristics of the system to be specified.

Thus, several schemes which allow a mixture of different knowledge representation models have been developed to make up for the weak points of one model or another (Ref. 6,7.). The integrated knowledge representation scheme we propose is similar to them with regard to the basic philosophy, however, we set out to construct a new model which might functionally cover most existing knowledge representation models. The new model we introduced is an imaginary organization for performing functions of a target system, where a number of members try to solve given problems in cooperation by offerring their individual knowledge to each other as in social organizations. Actually, in this model, it is supposed that every member of the organization is hierarchically arranged with its given role and all necessary knowledge for that role. That is, the model represents every piece of knowledge necessary for specifying functions of a system by constructing an imaginary organization and assigning it to an appropriate member of the organization.

To perform functions of an expert system, in general, a variety types of knowledge may be required. On the other hand, the most suitable

representation form may be thought to vary with the type of knowledge. Therefore, it might be effective to provide several representation forms out of which a user can select the most appropriate one for his purpose or characteristics of knowledge to be represented. As an attempt, we selected 4 types of representation forms, predicates, functions, procedures and production rules, and designed a knowledge representation language. We call it Modular Representation Language (MRL) temporarily.

This paper presents an outline of MRL, shows some examples of its use and discusses the features of MRL.

2 OUTLINE OF MRL

2.1 Knowledge Representation Model

As mentioned above, the knowledge representation model constructed on MRL corresponds to an imaginary organization which performs all the functions of a system. The organization consists of several members, each of which has a specific role and all necessary knowledge for achieving its role. We call each of the members a Knowledge Module (KM).

The behavior of the whole organization is considered to be controlled in a top-down manner, that is, we consider a model in which each KM receives a problem or a command from its superior KM and solves that problem or executes that command by itself or systematically mobilizes its subordinate KMs. The original problems to be solved by the whole organization -- which must be appropriately specified so that all functions of a target system are performed -- are supposed to be given from outside of the organization. Every such original problem must be given as a goal described in a predicate logic expression (Ref. 8.). Such goals are not necessarily required to be concrete, however, in such cases several concrete goals should be found by the organization itself.

Every goal given from the outside is accepted only by the KM located at the top of the organization, which we call the Main Module (MM). The process is started when the MM accepts a goal. It solves the goal in top-down style as shown in Fig.2.1.

When a goal is given to the MM, the MM tries to find its solution by use of its internal knowledge. If it detects a certain subproblem that it cannot solve by itself, it creates a subgoal corresponding to the subproblem, and requests an appropriate KM among its subordinate KMs to find its solution. If a KM can find a solution for its subgoal by applying its internal knowledge, it informs its superior KM of the result, then the superior KM restarts its problem solving and if necessary creates the next subgoal and repeats the same process. Thus, the existence of a solution for each subgoal is checked in bottom-up style, and finally, at the MM, the solution for the original goal is produced and its result is output.

Knowledge representation based on this model could be performed in a top-down and systematic style, where one might design several goals at

first so that by solving them the whole functions of a target system are accomplished. Each of these goals could be divided into an appropriate number of subgoals, and several KMs would be generated to solve them. The user would provide the knowledge necessary for the division into subbgoals. After that, in the same manner, one could design an imaginary organization and describe all necessary knowledge step by step.

2.2 Knowledge Representation Primitives

2.2.1 Knowledge Module

Each KM has its own role for performing functions of a system and it solves given problems using its internal knowledge and its subordinate KMs. The logical form of a KM is as follows:

```
< KM > ::= module  <module name>
             <knowledge body>
           end-module ;
```

The <kowledge body> in this form is detailed as follows:

```
<knowledge body> ::= [<module control information>]
                     [<variable definition>]
                     {<knowledge process> <internal module>}*
```

In the above description every <knowledge process> forms the nucleus specifying functions of the KM. The <internal module> is a KM utilized only by the knowledge processes of this KM, and it corresponds to a private subordinate of this KM. Global variables to be provided in this KM are defined in the <variable definition>. The <module control information> is used mainly to indicate several relations between this KM and other KMs.

2.2.2 Knowledge Process

A knowledge process (KP) is the basic unit of processing when solving given goals or subgoals, and it has the following logical form:

```
< KP > ::= <process name>[(<arguments>)]
           <process body> ;
```

Here, the part leading the <process body> is called the process head. The KPs are classified into 4 types corresponding to the representation form of the process body. They are shown in the following:

(1) predicate logic type process
This is a Horn clause (Ref. 9.). The head part of the Horn clause corresponds to the process head and the body part corresponds to the process body.
(2) function type process
This is a function which evaluates the truth-value of itself. The function name and its arguments correspond to the process head. The process body defines the details of the function.
(3) procedure type process
This is a procedure. The procedure name and its arguments corresponds to the process head. The process body defines the details of the procedure as a sequence of executable statements. By the way, the truth-value of the procedure always becomes true after its execution. Or it should be described as a function.

(4) production rule type process
This is a set of production rules. A process head, indicating an enabling condition of those rules, is attached at the head. The process body is a sequence of production rules, each of which is supposed to be a rule for forward-chaining (since it is supposed that any rule for backward-chaining is described as a predicate logic type process).

An invocation of a KP is performed based on unification between its process head and the subgoal to be solved by the KM including that KP in it (Ref. 10.). That is, only when the predicate name of the subgoal equals the process name of the KP and every argument of the subgoal can be unified with the corresponding argument in the process head, is that KP invoked.

A certain KM may contain several KPs in its knowledge body, each of which can utilize some predicates in its process body. A subgoal to be solved by a KM is first unified with each process head of its KPs, then if unified, one of the KPs is invoked and its process body is executed. If a new subgoal is generated corresponding to a certain predicate utilized in the process body, the next KP is invoked in the same manner. Thus, mutual invocations are accomplished between any pair of KPs, regardless whether their types are equal or not.

2.2.3 Global Variable and Attached Knowledge Process

Global variables can be provided in any KM. A global variable can be accessed from any position in the KM which includes the variable, and may be accessed from inside its subordinate KMs if they inherit all the knowledge of that KM. A global variable is generated by the following global variable definition in the same KM including this definition.

global <variable name> [<type definition>] [<initial value>]
[<attached KP>]*

Here, the <type definition> specifies a data type of this variable. When it is omitted any type of data must be storable. The <initial value> assigns an initial value. The <attached KP>, which is also a kind of KP, corresponds to an attached procedure (or demon) in the frame representation and it indicates an additional process to be executed when this global variable is accessed (Ref. 2.).

In addition, two specific global variables, module variable and process variable, are introduced. The module variable points to a certain KM or set of KMs and is used when the range of KMs must be limited. The process variable holds a process name and its arguments. It is mainly used for indirect invocation of KPs.

2.2.4 Atom

Atoms are the basic elements which construct the process body of each KP. The representation form of an atom is described as follows:

< atom > ::= [<module selector>.] <predicate name>
[(<arguments)>]

Here, the <module selector> specifies a range of KMs by which the subgoal generated corresponding to this predicate should be solved.

2.3 Description of Knowledge

2.3.1 The Structure of an Imaginary Organization

The structure of an imaginary organization is specified basically by superior/subordinate relations (SS relations) among KMs. A SS relation means that the superior KM forces the subordinate KM to solve its subgoal in order to solve a goal given to itself. There are two types of subordinate KMs. One is the above-mentioned internal module, corresponding to a private subordinate of its superior KM, whose whole body is embedded in the knowledge body of the superior KM. The other is called an external module. An external module can be utilized in common by several KMs and its SS relations is described by including its module name in the preceding module control information of its superior KM (in reality, by an external statement). The body of such an external module must be described elsewhere.

By the way, SS relations among KMs can be settled without any restriction, therefore, a recursive structure such that a subordinate KM of a certain KM has itself as one of its subordinate KMs is also allowed.

In addition to these descriptions for a static structure, several built-in predicates are provided to represent a dynamic transformation of the structure. They are create, copy and delete predicates. The create predicate creates a vacant KM. The copy predicate creates a KM by duplicating a designated KM. Those created KMs become the internal modules of the KM which invoked the corresponding predicate. The delete predicate is used for removal of a designated KM.

2.3.2 Internal Structure of a KM

When constructing an imaginary organization, it must be clearly stipulated exactly what subgoals are given from a superior KM to its subordinate KM for every pair of KMs with a SS relation. Accordingly, here, under the premise that all subgoals which a certain KM should try to solve are known, how we can design an internal structure for the KM is shown.

A subgoal given to a KM is processed by a certain KP whose process head is unifiable with the subgoal. Therefore, it is necessary to provide such a process head in the KM for every subgoal. Individual process heads may be prepared for each different subgoal, and common process heads may be prepared for several subgoals. Moreover, it is possible to arrange several process heads for a single subgoal. Thus, an internal structure of a KM can be designed with considerable flexibility.

Now, for a certain subgoal when several KPs are unifiable, one of them is first invoked, then if it cannot solve the subgoal, the next KP is selected and invoked, and so on. In such cases, a backtracking mechanism works.

Besides, between a pair of KMs which has a SS relation, the subordinate KM can inherit the whole knowledge of the superior KM, provided that such inheritance is made effective by the module control information (in reality, by an inherit statement) of the subordinate KM. At that time, as all KPs which are included in the superior KM are also inherited, the subordinate KM is able to utilize them of course.

2.3.3 Process Body of a KP

The process body of a KP can be described in predicate logic, function, procedure or production rule style. In any style, the basic elements constructing a process body are atoms. While execution of a KP is controlled by each mechanism which corresponds to the type of the KP, atoms are processed in the same way.

The corresponding subgoal is generated for each atom, and it is solved by other KPs. Such KPs must be included either in the same KM as the atom is described or in the subordinate KMs of that KM. In other words, every subgoal generated by a KM must be solved either by its own KPs (including all KPs inherited from its superior KMs) or by its subordinate KMs.

3 AN EXAMPLE OF KNOWLEDGE REPRESENTATION

As an example of knowledge representation in MRL, a medical diagnosis system is considered, which identifies the disease name of a patient by use of causality knowledge between diseases and symptoms.

An imaginary organization for that system could be constructed as shown in Fig.3.1. Each member of the organization, that is each KM, has the following functions:

(a) diagnostician:
identifies the disease name of a patient and outputs it.
(b) basic_symptoms_collector:
inquires the patient about his basic symptoms.
(c) symptoms_inquirer:
inquires the patient about presence of a disignated symptom.
(d) causality_specialist:
answers questions about causality between diseases and symptoms.
(e) disease_specialist:
answers questions about the causality for i-th disease.
(f) patient:
records various information about the patient.

The major contents of these KMs are shown below:

```
module diagnostician ;
```

```
global  D, C  chrstring ;

diagnose : procedure ;
  create( patient ) ;
  basic_symptoms_collect ;
  for_every  coincide( @D )
    assert( patient, condidate( ?D ) ;
  if not patient.candidate( @D )
    then write( 'conflict !'), stop ;
  retract( patient, candidate( ?D ) ;
  while patient.candidate( @C )
    do
    retract( patient, candidate( @C ) ;
    select_candidate( ?D, ?C, @D ) ;
    end ;
  write( 'The disease may be', ?D ) ;
end diagnose ;

coincide ( D ) :- not ( causality( D, S, A ), patient.( symptom( S, B ) )
                        , A <> B ) ;

select_candidate( D1, D2, D ) : rule ;
  if causality( D1, S, A1 ), causality( D2, S, A2 ), A1 <> A2
    then ask( S, A ), assert( patient, symptom( S, A ) ) ;
  if patient.symptom( S, A ), causality( D, S, B ), A <> B
    then retract( patient, candidate( D ) ) ;
  if patient.symptom( S, A ), not ( causality( D, S, B ), A <> B )
    then retract( patient,symptom( S, A ) ) ;
  if candidate( D1 )
    then D := D1, stop ;
    else D := D2, stop ;
end select_candidate ;

module causality_specialist ;

  global  M  module ;

  causality( D, S, A ) :- @M.( disease( C ), causes( S, A ) ) .

  module disease_specialist_#1 ;

    disease( disease_name_#1 ) ;
    causes( symptom_a, yes ) ;
    causes( symptom_b, no ) ;
        . . . . . .
        . . . . . .
  end_module disease_specialist_#1 ;

        . . . . . .
        . . . . . .

  module disease_specialist_#N ;

        . . . . . .
```

```
    end_module disease_specialist_#N ;

  end_module causality_specialist ;

          . . . . . .
          . . . . . .
          . . . . . .

end_module diagnostician ;
```

4 DISCUSSION

The major features of MRL are the hierarchical organization model to specify functions of a system and the flexible description form to represent a variety of knowledge.

The model should be very familiar to all of us, since we all have some experience of belonging to actual organizations. Moreover, the top-down style design process of the organization is quite similar to conservative programming schemes which we used to use (Ref. 11.). Thus, it is expected that anyone who can design software -- even if he or she is not a so-called knowledge engineer -- would be able to utilize MRL quite easily.

Besides, the model would supply considerable modularity of knowledge distributed to each KM, since the basic interaction between KMs is fairly simple. This would contribute to easier description, understanding, debugging and modification of knowledge and, in addition, would enable us to conceive parallel processing between the KMs.

The second feature, the flexible description, would enable a user to choose an optimal representation form considered the easiest to describe or understand. This would also free up the time spent trying to represent a variety of knowledge to coincide with a limited frame. There already exist several languages which allow mixed utilization of different representation forms. However, we believe the knowledge representation capability of MRL may be superior to them since it enables mutual access between different representation forms.

H.Takenouchi and Y.Iwashita
5th Laboratory, Institute for New Generation Computer Technology
Mita Kokusai Building 21F., 4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan. Tel: Tokyo(03)456-3192. Telex: 329641COTJ.

REFERENCES

[1] Shortliffe, E.A.: Computer-Based Medical Consultations MYCIN, American Elsevier, 1976.
[2] Minsky, M.: A Framework for Representing Knowledge, The Psychology of Computer Vision, Winston, P., Ed., McGraw-Hill, pp.211-277, 1975.
[3] Quillian, M.R.: The Teachable Language Comprehender, Simulation Program and Theory of Language, Communications of the ACM, Vol.12, No.8, pp.45-476, 1969.
[4] Schank, R.C., et al.: Scripts, Plans, Goals, and Understanding, Lawrence Erlbaum, 1977.
[5] Brachman, R.J., et al.: Krypton: A Functional Approach to Knowledge Representation, Computer, pp.67-73, Oct. 1983.
[6] Stefik, M., et al.: Knowledge Programming in LOOPS: Report on an Experimental Course, Artificial Intelligence, Vol.4, No.3, pp.3-14, 1983.

[7] Aikins, J.S.: A Representation Scheme Using Both Frames and Rules, Rule-Based Expert Systems, Buchanan, B.G. et al., Eds., Addison-Wesley, pp. 424-440, 1984.
[8] Bowen, D.L.: DEC System-10 PROLOG USER'S MANUAL, University of Edinburgh, Dept. of Artificial Intelligence, 1981.
[9] Colmerauer, A.: Prolog and Infinite Trees, Logic Programmimg, Clark, K. L., et al., Eds., Academic Press, 1982.
[10] Kowalski,R.A.: Predicate Logic as Programming Language, Proc. IFIP-74 Congr., North-Holland, Amsterdam, 1974.
[11] Dahl, O.J., et al.: Structured Programming, Academic Press, 1972.

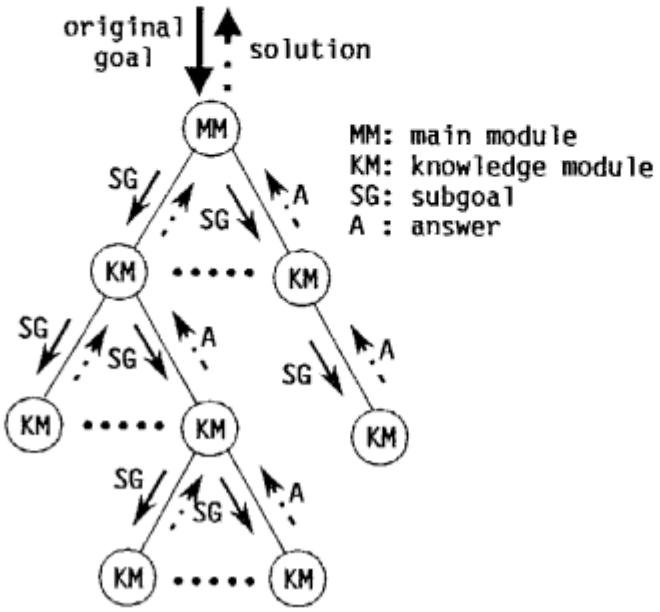

Fig.1. Problem Solving Model

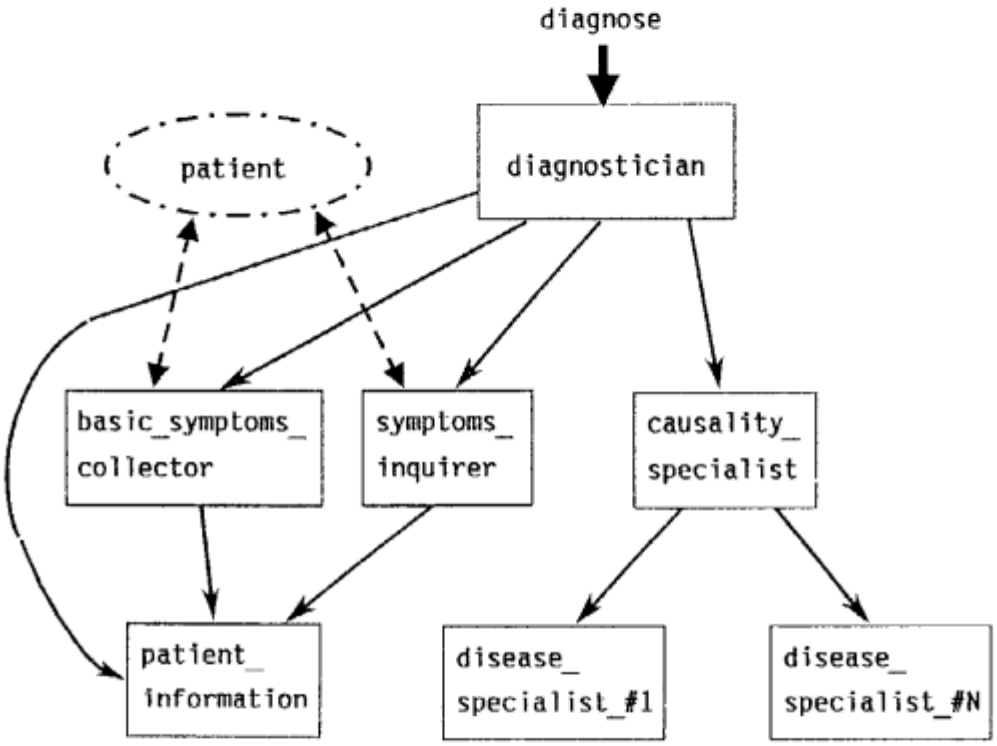

Fig.2. An Imaginary Organization for Medical Diagnosis