

AN ACTIVE FRAME FOR THE KNOWLEDGE REPRESENTAION

Hitoshi Ogawa,^{*} Hideaki Nanba^{**} and Kokichi Tanaka^{*}

^{*} Faculty of Engineering Science
Osaka University
Toyonaka, Osaka 560
JAPAN

^{**} NIHON-DENSO
Kariya, Aichi 448
JAPAN

This paper describes micro-actors which construct the knowledge representation in which the modularity was established without sacrificing the interaction. The behavior of the micro-actor is defined in terms of only one kind of action: sending message to another micro-actor. The micro-actor is composed of a script and an acquaintance. The script consists of the pairs of the message pattern and the action corresponding to the pattern. The micro-actor has the acquaintance as a memory space to hold the data and the names of the other micro-actors it directly knows about. As a result of using the micro-actors, we can deal with knowledge about the actions as well as knowledge about the world states. The information about an action sent from a micro-actor is effective only in the micro-actor which received it. This guarantees the modularity of micro-actors.

An example shows that the continuation in a message make both the subroutine control and the coroutine control possible. The other indicates a dialogue in the question-answering system using the micro-actors.

1. INTRODUCTION

Artificial Intelligence is a branch of computer science which is mainly concerned with the problem solving, the cognition, and the comprehension of natural languages. Many approaches to the problem solving have been developed; for example, the algorithm A* (Hart) [4] for searching trees, the resolution principle (Robinson) [13], GPS [2] and its application: STRIPS (Fiks, Nilsson) and so on. The resent approach is the development of the programming language for artificial intelligence. The cognition is the intelligence process to acquire knowledge in the image understanding and the speech understanding. The comprehension of natural languages needs the integration of many approaches in artificial intelligence.

The common issues of the above mentioned themes are the knowledge representation and the mechanism of understanding and inference. In this discussion, "Knowledge" does not mean only the facts about a world state but also the techniques dealing with the facts. The artificial intelligence system obtains information from the scene and/or the statements in natural language. The system, then, acquires new knowledge by comparing the obtained information with the knowledge the system has. Therefore, the knowledge representation is necessary, in which it is easy

to specify knowledge, modify it and refer to it. There have been two types for the representation of knowledge: the declarative one and the procedural one [17]. The modularity is emphasized in the former; e.g., the logical formulas. On the other hand, the interaction is stressed in the latter; e.g., PLANNER [5], QA4 [16], CONNEVER [9] and so on. The declarative representation has several benefits: The separation of knowledge into data and procedures brings the flexibility and economy of the declarative representation. The independence of the axioms or facts leads to the understandability and learnability of them. On the contrary, in the procedural representation, a frame axiom need not be specified. Furthermore, the heuristic knowledge can be treated in it. To utilize both benefits of the declarative representation and the procedural one, it is necessary to develop the method in which the modularity is established without sacrificing the possibilities for interaction.

KRL (Bobrow, D.G. & Winograd, T.) [1] and FRL (Roberts, R.B. & Goldstein, I.P.) [14, 15] are developed as the approaches to the problem mentioned above. Both KRL and FRL are based on frames, and attempt to integrate procedural knowledge with a broad base of declarative forms. By the way, there is some knowledge about the action; for example, the control of the reasoning mechanism, the data base processing and so on.

Such knowledge is preferable to be expressed by the procedures rather than the declarative forms. If declarative knowledge is integrated with the procedures which deal with it, and the procedures have the modularity, then we can freely use both declarative knowledge and procedural one.

This paper describes a micro-actor* as a method for knowledge representation. The script of a micro-actor contains procedural knowledge to deal with only the acquaintance of the micro-actor. The acquaintance has declarative knowledge in the form of frame.

2. REPRESENTATION OF WORLD-STATE

2.1 Frame Representation In Acquaintance

A micro-actor is composed of a script and an acquaintance. The script describes the behavior which the micro-actor should take when it receives a message. The acquaintance is the memory space for the micro-actor. The acquaintance stores the data that the micro-actor knows; for example, its native values, its attributes, the names of the other micro-actors it directly knows about, and the relation with them. The followings are the reason why the concept of Frame Theory [8, 10] is used for construction of an acquaintance:

- (1) A frame is a data-structure representing a stereotyped situation.
- (2) Several kinds of information can be attached to each frame, and some of this information is about how to use the frame.
- (3) Frames enable us to represent knowledge in a hierarchical structure.

Fig. 1 shows a slot array representation to indicate a frame intuitively. A frame is, however, represented by the nested association lists as shown in Fig. 2 in its implementation. The respective substructures of a frame are named; Slot, Facet, Datum and Comment. A slot shows which property the data in it are connected with. A facet shows kinds of the data; e.g., values, constraints, procedures and so on. A datum is elementary information associated with the facet, and a comment is additional information

* The micro-actor was proposed as the implementation of some of abilities of actor (Hewitt, C.) [6, 7] in 1977 [11]. A new version of a micro-actor described in this paper has similar abilities of actor. Parallel process and coroutine control are possible in a micro-actor system. There is a slight difference between the micro-actor and the actor in point of notation.

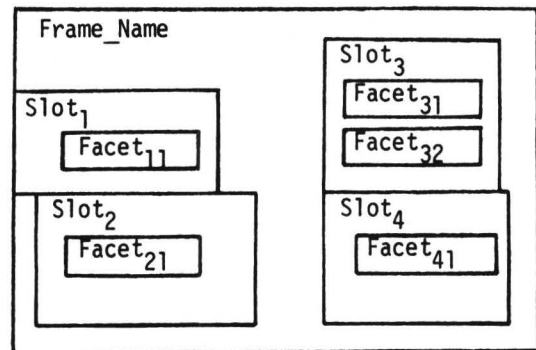


Fig. 1

Slot array representation of a frame.

```
(Frame Name
(S1 (F11 (D111 C111) ... (D11h C11h)))
(S2 (F21 (D211 C211) ... (D21i C21i)))
(S3 (F31 (D311 C311) ... (D31j C31j)))
      (F32 (D321 C321) ... (D32k C32k)))
.....
.....
(Ss (Fs1 (Ds11 Cs11) ... (Ds1m Cs1m)))
)
```

Fig. 2

Nested list representation of a frame.
S: slot, F: facet,
D: datum, C: comment.

for the datum.

2.2 Effect On Other Frames

An addition of procedures to a slot of a frame enables the frame to acquire information from other frames and to change the contents of the other frames. This may make it possible to combine the declarative knowledge and the procedural knowledge; that is, the modularity and the interaction. However, a question still remains to be solved.

A frame is freely dealt with using three basic functions; obtaining something from the frame, putting something to it, and removing from it. The use of only the three functions is, however, inconvenient and uneconomical when a frame is dealt with in several ways. New functions should be made either in advance or afterward. We will try to put box A on box B in a block world.

Assume that there are frames which correspond to box A and box B. To put box A on box B is

executed by the following process; If both box A and box B are a kind of Block and Cleartop*, and box B is Supportable**, then let box A be supported on box B and let box B support box A. The knowledge concerning the process mentioned above is represented in three methods: (1) Function ON is made in order to put an object on another, and is added to the basic functions. (2) The process is distributed to the frames corresponding to box A and box B. (3) A new frame is created corresponding to the function ON.

In (1), there exists the chunk of procedures dealing with frames. A slight change of a procedure often affects the other procedures. As a result, (1) is not good to represent the knowledge about actions in A.I. system. General functions can be used in both (1) and (3). On the contrary, a procedure depends on a particular frame in (2). (2) is convenient to describe

the procedure closely depending on the frame, and to establish the modularity among procedures. It is, however, wasteful when several frames have the similar function. (3) is equal to (1) in essence. So there remains the possibility of the unlimited and unrestricted interactions. However, the application of the idea of actor to (3) leads the module for representing knowledge with proper interaction method.

The actor model is suitable to control and describe the interaction between the modules. Namely, the action of an actor is defined in terms of only one kind of action; sending messages to the other actors, and any other side effect is not permitted. When the actor receives a message, it becomes active and tries to play its role. The authors think that the actor model is useful for parallel processing.

3. REPRESENTATION OF ACTION

The chapter 2 said that a frame is good for representing the knowledge about a world state, but not so good for representing the knowledge about an action. On the other hand, a micro-actor is very suitable for representing the action. The micro-actor, which is a procedure, is a basic module in order to construct the knowledge base. This chapter describes the form of script which represents the behavior, and illustrates how to deal with the knowledge about the world state using message passing.

 * Cleartop means the state in which the box supports nothing.

** The box which is supportable can support another box.

3.1 Micro-Actor

The script of the micro-actor precisely represents the message passing, the parallel computation and the coroutine control. The form of the script is similar to Yonezawa's notation[19]. But the micro-actor is based on LISP whereas his notation is based upon PLASMA.

(i) Script

The script is specified with a set of pairs of message patterns and rules as follows:

```
((message_pattern1 rule11 ... rule1m)
 (message_pattern2 rule21 ... rule2n)
 :
 (message_patternp rulep1 ... rulepq)).
```

(ii) Message Pattern

There are two types for pattern:

(1) (MES: ?%N content TO: ?CONT).

(2) (RE: ?%N content).

?%N is only matched with the message number %n, where n is a positive integer. A message number is assigned to an input sentence or a message for distinguishing it from the others as a tag. The message number is also used to distinguish a datum from the others stored in the same micro-actor. The atoms prefixed by "?" (e.g., ?CONT) are the variables for pattern matching. The prefix "!" is used instead of the prefix "?" to refer to the pattern matching variables for their values; for example, !CONT. "MES:" means a message and "RE:" means a reply. "TO:" means that the continuation is !CONT. The continuation is the micro-actor which is sent a reply. Assign "NO-ONE" to !CONT in the case that the answer is not required. Assign "ME" to !CONT when the reply is demanded. This is equal to the subroutine control.

(iii) Rule

A rule is a set of programs, which carries out a job. The rule consists of three parts as follows:

```
((P-C: programp1 ... programpq)
 (N-C: programn1 ... programnr)
 (C-E: programc1 ... programcs)).
```

"P-C:" is an abbreviation for pre-condition. "N-C:" means next-condition which is an option. "C-E:" is an abbreviation for caused-events in which the messages are sent to the other micro-actors when both the pre-condition and the next-condition are satisfied. The form of the rule is given by Dr. Yonezawa, and it is sug-

gested that the form of the rule is useful to deal with the parallel computation. The programs should be carried out in order of the appearance of them in the pre-condition and the next-condition. On the other hand, the programs in the caused-events can be executed concurrently.

(iv) Program

A program is written in the restricted LISP notation. The user can freely use the LISP functions which don't have influence upon the pointers to lists and atoms; e.g., MEMBER, ASSOC and so on. The variables are restricted to the form !m or !!n (Both m and n are positive integers.) for the LISP functions which have influence upon the pointers; e.g., SETQ and so on. This restriction prevents the behavior of a micro-actor from giving unintentional effect to other micro-actors. The variable !m is used as well as PROG variable, while !!n can be used in a special way as mentioned in the next section.

A specification for message passing is as follows:

(=> *the name of a micro-actor a message*).
The user uses the basic function (FGET, FPUT and REMOVE) to deal with the acquaintance. These functions differ from those of FRL in point of the necessity of the frame name. The functions in this paper do not need the frame name because the micro-actor deals with only its acquaintance. Therefore, the execution of the basic function does not affect the other micro-actors.

(v) Behavior of Micro-actor

A micro-actor tries to find a message pattern which is matched with a message M_1 when it receives M_1 . If the message pattern P_1 is found, a rule R_1 is searched for, which is one of rules making a pair with the pattern P_1 , and whose pre-condition should be satisfied. When there is the next-condition in R_1 , all programs in the next-condition should be evaluated. If the next-condition is satisfied, the caused-events are executed. If there is no rule whose pre-condition is satisfied concerning the pattern P_1 , the micro-actor tries to find another message pattern which is matched with M_1 .

Since the system holds the COMMON-SCRIPT which is necessary for all micro-actor, the users may specify the micro-actor by the peculiar script. The COMMON-SCRIPT has the message pattern (MES: ?%N ?X TO: ?CONT) and acts as follows:

(a) If the variable !X (corresponding to ?X) indicates one of the basic functions (FGET, FPUT

and REMOVE), then it is put into practice.

(b) If !X shows one of the extension functions (i.e., there exists a micro-actor corresponding to it), the rules are obtained from the micro-actor corresponding to the extension function, and they are performed. The variables !!n ($n = 1, \dots, 5$) are corresponded to the arguments; for example, !!1 means the first argument.

(c) If !X indicates a set of rules, then !X is accomplished.

3.2 Example

A block world gives us good examples to illustrate the representation of the script and the acquaintance of a micro-actor, and message passing. The following discussion concerns with the micro-actor PROPERTY. It is used in the implementation of putting an object on another. The script of the micro-actor BOX1 is NIL, because it needs just only the COMMON-SCRIPT. The acquaintance is expressed by one slot as follows: ((AKO (:VALUE (BOX))). This means that box1 is a kind of box. The AKO slot has a higher level concept of box1 as the value marked by the :VALUE facet. Concerning the micro-actor BOX, the script is NIL as well as box1, and the acquaintance is as follows:

```
((AKO (:VALUE (BLOCK)))
 (INSTANCE (:VALUE (BOX1 BOX2 BOX3)))
 (PROPERTY (:VALUE (SUPPORTABLE))))
```

This shows the following facts: (1) Box is a kind of block. (2) The instances of box are box1, box2 and box3. (3) The native property is supportable which means that the box can support another box.

Fig. 3 gives the script and the acquaintance of the micro-actor PROPERTY which deals with the rules about the properties. In the rest of this paper, a variable prefixed by "!" indicates its value until it is quoted. The script has a rule to ask the micro-actor !B if it has the property !A. According to the rule, PROPERTY sends !B the rules in which !A is substituted for "!!1". The AKO slot means that PROPERTY is a kind of E-FUNCTION (which is an abbreviation of extension function). The CONTENT slot shows the rules which contain the following algorithm. If the micro-actor which executes the rules has the value SUPPORTABLE in its PROPERTY slot, then it sends the reply OK to !CONT. Otherwise, if it has the value A in the AKO slot, then A sends the message to ask if A has the property SUPPORTABLE. If A has it, A sends the reply OK to !CONT; otherwise, A sends NIL to !CONT. Note that A need not answer the micro-actor which sent the message to A. This means the possibility of a coroutine control.

```

((MES: ?%N ?B PROPERTY ?A TO: ?CONT)
 ((P-C: (SETQ !1 (FGET CONTENT :VALUE)))
  (N-C: (SETQ !1 (*SUBS !1 (LIST (CONS
    (QUOTE !!1) !A))) )))
 (C-E: (=> !B (MES: !1 TO: !CONT)))
))

(a)

(AKO (:VALUE (E-FUNCTION)))
(CONTENT :VALUE
 ((P-C: (SETQ !5 (FGET PROPERTY :VALUE))
  (ASSOC !!1 !5))
  (C-E: (=> !CONT (RE: !%N OK)))
 )
 ((P-C: (NOT (ASSOC !!1 (FGET PROPERTY :VALUE)))
  (SETQ !4 (FGET AKO :VALUE))
  (SETQ !4 (CAAR !4)))
  (C-E: (=> !4 (MES: !%N PROPERTY !!1
    TO: !CONT)))
 )
 ((P-C: (NULL (FGET AKO :VALUE)))
  (C-E: (=> !CONT (RE: !%N NIL)))
 )))

(b)

```

Fig. 3

- (a) The script of the micro-actor PROPERTY.
 (b) The acquaintance of PROPERTY.

Fig. 4 shows the diagram of the message passing for the process to see if box1 is supportable. The user can communicate with the micro-actor system using the special micro-actor * as shown in Fig. 4. For example, when the user wants to know whether box1 is supportable or not, he may write as follows:

(* PROPERTY BOX1 PROPERTY SUPPORTABLE).
 The first argument of * is the target of the message whose content follows the first argument. The micro-actor * composes the message and sends it to the micro-actor PROPERTY (See line1 and line 2 in Fig. 5).

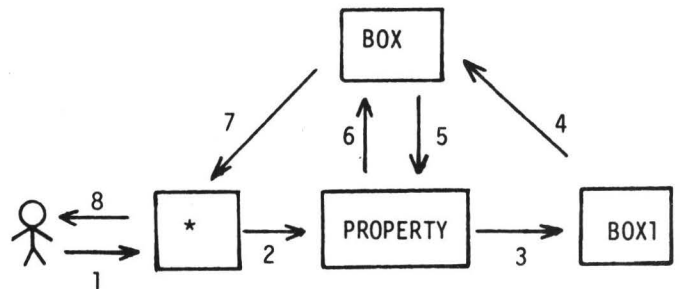


Fig. 4

The diagram of the message passing.

```

1 (* PROPERTY BOX1 PROPERTY SUPPORTABLE)
2 (* PROPERTY <= (MES: %2 BOX1 PROPERTY SUPPORTABLE TO: ME))
3 (*+ BOX1 <= (MES: %2 ((P-C: (SETQ !5 (FGET PROPERTY :VALUE))
  (ASSOC SUPPORTABLE !5))
  (C-E: (=> !CONT (RE: !%N OK))) )
  ((P-C: (NOT (ASSOC SUPPORTABLE (FGET PROPERTY :VALUE)))
  (SETQ !4 (FGET AKO :VALUE))
  (SETQ !4 (CAAR !4)))
  (C-E: (=> !4 (MES: !%N PROPERTY SUPPORTABLE TO: !CONT))) )
  ((P-C: (NULL (FGET AKO :VALUE)))
  (C-E: (=> !CONT (RE: !%N NIL))) ) )
  TO: ME))
4 (*** BOX <= (MES: %2 PROPERTY SUPPORTABLE TO: ME))
5 (**** PROPERTY <= (MES: %2 FGET CONTENT :VALUE TO: ME))
6 (**** REPLY: (RE: %2 ((P-C: (SETQ !5 (FGET PROPERTY :VALUE))
  (ASSOC !!1 !5))
  (C-E: (=> !CONT (RE: !%N OK))) )
  ((P-C: (NOT (ASSOC !!1 (FGET PROPERTY :VALUE)))
  (SETQ !4 (FGET AKO :VALUE))
  (SETQ !4 (CAAR !4)))
  (C-E: (=> !4 (MES: !%N PROPERTY !!1 TO: !CONT))) ) )
  ((P-C: (NULL (FGET AKO :VALUE)))
  (C-E: (=> !CONT (RE: !%N NIL))) )
  )))
7 (* REPLY: (RE: %2 OK))
8 (* RE: %2 OK)

```

Fig. 5

Messages and Targets shown in Fig. 4.

Fig. 5 gives us the messages and the targets illustrated in Fig. 4. Each line in Fig. 5 shows the message passing which is represented by the arrow attached the same number that the line has. The front atom in each list (i.e., *, **, etc.) illustrates the level of message passing. The symbol "*" means the subroutine control, and "+" means the coroutine control. Each line will be explained.

Line1: The user communicates to the system the message (* PROPERTY BOX1 PROPERTY SUPPORTABLE).

Line2: The micro-actor * composes and sends it to the micro-actor PROPERTY.

Line3: The micro-actor PROPERTY sends the BOX1 the message including the rules to see if the target has a datum SUPPORTABLE in its property slot. The continuation of the message on the line3 is the same one on the line2. In other words, "ME" means the micro-actor *.

Line4: BOX1 puts the rules into practice according to COMMON-SCRIPT. As BOX1 does not have a property slot, BOX1 sends the message to the BOX which is the higher level concept of box1. The continuation "ME" is BOX in this case, and this means the subroutine control.

Line6: PROPERTY returns the rules to BOX.

Line7: BOX carries out the rules. As BOX has the value SUPPORTABLE in the property slot, BOX sends OK to the micro-actor *.

Line8: * replies OK to the user.

This example shows about the method of the coroutine control and the subroutine control, and explains how the extension function defined as the micro-actor is used and is executed. Furthermore, it gives the preferred inference about the knowledge represented with frames.

4. QUESTION ANSWERING SYSTEM

In the question answering between a machine and a human, one of most important things is a topic of conversation and a context to understand what the partner said. We assume that the speaker and the listener have the common knowledge about the discourse. Consequently, the inference plays a great role to understand the partner's intention. The micro-actor can get a good policy from the acquaintance in which the facts of the world state are expressed by a frame.

4.1 Mapping From World

The authors consider that the knowledge about a world is a "snapshot (Bobrow, D.G.) of world state. The followings show the difficulty of mapping data from a world state to a knowledge state. What part of a house one directs his attention to when he sees it? Someone might direct his attention to the door. A gardener

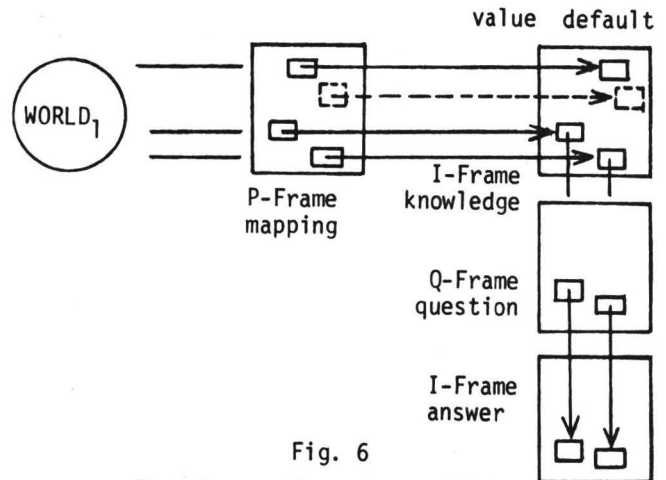


Fig. 6

The observation of a world.

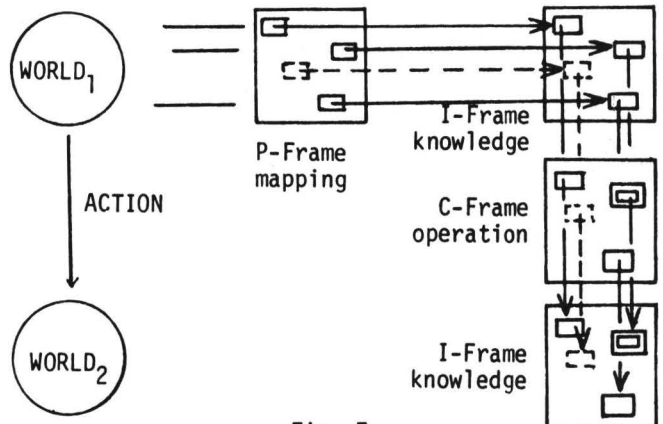


Fig. 7

The change of a world.

might be interested in the trees and flowers in the garden. An architect might look the style of the house. Consider another example. When we hear "John is like a frog", what individuality do we assume about John? The answer depends on the context and the topic of the discourse. If one heard that Mike was a handsome boy just before, then he considers that John is ugly. If one heard that Mary was gentle then thinks that John is noisy.

The frame representation is suitable for extracting information from the world. The mapping method using the frame is to move a part of the world state to a slot of the frame. The data acquired by the mapping must satisfy the condition of the slots which depend on the topics and the context. So, we will use the frame called P-Frame (Prototype Frame), which has the slots to fill out. The P-Frame is used to compose an I-Frame (Instance Frame) which

represents the knowledge state. The P-Frame has the default values which are communicated to the I-Frame if necessary. In the same way, using a Q-Frame (Query Frame) which corresponds to "question", the I-Frame for the answer obtains the data from the I-Frame for the knowledge. A Q-Frame means such an action as human observes a world and gets the data (See Fig. 6). In Fig. 6, the knowledge is indicated by a frame for brevity. The knowledge is, however, represented by several frames in practice.

Fig. 7 gives two I-Frames which correspond to the world1 and the world2 changed from the world1. A C-Frame (Change Frame) represents the action which affects the world, or which changes the observer's situation (In the latter case, the world is not changed.). A C-Frame transfers the data from an I-Frame to another frame, and changes the data if necessary.

4.2 Conversation Examples

The question answering system about the geometrical figures is constructed with the micro-actors according to the consideration in 4.1. The user communicates with the system in the limited Japanese for the present. The authors are developing a Japanese analyzer. After it is accomplished, the user will be able to converse with the system in Japanese. Fig. 8 presents the examples in the conversation. The sentences in the figure are written in English for the readers to understand. The user's sentences are marked by "*", and the replies are marked by "!".

The system has the geometrical world as the context in advance. The context, of course, is changed during the conversation. The system also has the knowledge about several kinds of triangles, the rectangle and square. The system tries to associate the input sentence with the knowledge. If it fails, the system asks the user about the sentence as shown in line2.

When the micro-actor build receives the sentence (line1), it tries to make the micro-actor house. As "house" is still undefined and the other parts of line1 are not associated with the knowledge, the system asks the user about a house (line2). The user defines the house as shown in line3, then build can create the micro-actor house (line4). The context plays a great role to decide kinds of slot in the acquaintance of house and to fill the slot with the data. In this case, build is regarded as P-Frame. As the world "roof" on line5 is not defined yet, the micro-actor prefer will associate an isosceles triangle with the knowledge as follows. Since

1. * Build a house.
2. ! What is a house.
3. * It is the composition in which a triangle is on a rectangle.
4. ! OK.
5. * I prefer an isosceles triangle for the roof.
6. ! Is the roof the triangle of the house?
7. * Yes, it is.
8. ! OK.
9. * The house is like a tower.
10. ! OK.
11. * What a kind of house is it?
12. ! It is a tall house.

Fig. 8

Examples of the conversation.
*: user, !: computer.

the topic is about the house, prefer investigates the relation between the house and the isosceles triangle. The triangle is in the AKO slot of the micro-actor isosceles triangle. The house is in the PART-OF slot of triangle. Then prefer infers that the roof is a part of the house, and asks the user if the result of the inference is correct (line6). If the user answers yes (line7), prefer creates roof and adds the PREFER facet into the PART slot of house (line8). In this case, prefer acts as a C-Frame. The above process is smoothly performed using message passing between the micro-actors as mentioned in the chapter 3.

There have been several studies of metaphors using transfer frames [18]. Our system makes use of these studies (lines9,10,11 and 12). The like behaves like a P-Frame. That is, like receives the slot which has an extremal value from the tower, and communicates it to house.

5. CONCLUSION

This paper presented the micro-actors which construct the knowledge representation in which the modularity is established without sacrificing the interaction. The micro-actor is also suitable to represent the knowledge about the actions. The micro-actor mainly consists of the message patterns and the actions corresponding to them in the script. The micro-actor acts according to the script, refers to and modifies the acquaintance, and sends the messages to the other micro-actors if necessary. Since only message passing is the interactive method between the micro-actors, the modularity is established. The message can communicate enough information to the micro-actor. Furthermore, the continuation in a message makes both the sub-

routine control and the coroutine control possible. This is very useful for obtaining the data from another micro-actor, in particular, acquiring information from the higher level concept.

The acquaintance is a memory space which stores the data and the names of the other micro-actors. Since the frame is employed to represent the acquaintance, the data in the acquaintance is used smoothly. The acquaintance is treated by only the micro-actor which has it. This prevents the acquaintance from being affected by another micro-actor.

The chapter 4 presented examples of the question answering system using the micro-actors. The dialogues show the following abilities: the creation of new micro-actors, the association using the higher level concept, the comprehension of metaphors, and the recognition of the constructed figures. These abilities greatly depend upon the frame representation. On the other hand, the form of the micro-actor is very useful to implement the system.

ACKNOWLEDGEMENTS

The authors appreciate many helpful conversations with several graduate students and members of Prof. Tanaka's Laboratory of Osaka University.

REFERENCES

- [1] Bobrow, D.C. & Winograd, T. "An Overview of KRL, a Knowledge Representation Language", Cognitive Science, 1:1 (1977) 3-46.
- [2] Ernst, G.W. & Newell, A. "GPS: A Case Study in Generality and Problem Solving", Academic Press, 1969.
- [3] Fikes, R.E. & Nilsson, N.J. "STRIPS: A New Approach to The Application of Theorem Proving to Problem Solving", Artif. Intell. 2 (1971) 189-208.
- [4] Hart, P.E., Nilsson, N.J. and Raphael, B. "A Formal Basis for The Heuristic Determination of Minimum Cost Paths", IEEE, Trans. on Sys. Sci. Cyb. SSC-4:2 (1968).
- [5] Hewitt, C. "PLANNER: A Language for Proving Theorem in Robots", Proc. IJCAI-69 (1969) 295-301.
- [6] Hewitt, C., Bishop, P. and Steiger, T. "A Universal Modular Actor Formalism for Artificial Intelligence", Proc. IJCAI-73 (1973) 248-249.
- [7] Hewitt, C. & Baker, H. "Towards a Programming Apprentice", IEEE Trans. on Sof. Eng. (1975).
- [8] Kuiper, B. "A Frame for Frames", Representation and Understanding, Academic Press (1975).
- [9] McDermott, D.V. & Sussman, G.J., "The CONNIVER Reference Manual", MIT AI Lab, AI-MEMO No. 259 (1972).
- [10] Minsky, M. "A Framework For Representing Knowledge", The Psychology of Computer Vision, McGraw-Hill (1975).
- [11] Ogawa, H. & Tanaka, K. "A Structure For the Representation of Knowledge --A Proposal for a Micro-Actor--", Proc. IJCAI-77 (1977) 248-249.
- [12] Ogawa, H., Kitahashi, T. and Tanaka, K. "The Theorem Prover Using A Parallel Processing System", Proc. IJCAI-79 (1979).
- [13] Robinson, J.A. "A Machine-Oriented Logic Based on the Resolution Principle", Jour. ACM 12 (1965) 23-41.
- [14] Roberts, R. & Goldstein, I. "The FRL Manual", AI-MEMO 409 (1977).
- [15] Roberts, R. & Goldstein, I. "The FRL Primer", AI-MEMO 408 (1977).
- [16] Rulifson, J.F., Derksen, J.A. & Waldinger, R.J. "QA4: A Procedural Calculus for Intuitive Reasoning", SRI Tech. Note No. 73 (1972).
- [17] Winograd, T. "Frame Representations and The Declarative/Procedural Controversy", Representation and Understanding, Academic Press 1975.
- [18] Winston, P.H. "Learning by Creatifying Transfer Frames", Artif. Intell. (1978) 147-172.
- [19] Yonezawa, A. "A Specification Technique For Abstract Data Types With Parallelism", Res. Rep. C-17, Dept. of Information Science, Tokyo Institute of Technology (1978).
- [20] Bobrow, D.G. "Dimension Of Representation", Representation and Understanding, Academic Press (1975).