ICOT Technical Memorandum: TM-0144

TM-0144

MENDEL: Prolog Based Concurrent Object Oriented Language

by
S. Honiden, N. Uchihira and T. Kasuya
(Toshiba Corp.)

November, 1985

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

MENDEL:PROLOG BASED CONCURRENT OBJECT ORIENTED LANGUAGE

Shinichi Honiden, Naoshi Uchihira, and Toshiaki Kasuya

Systems and Software Engineering Division,Toshiba Corporation

1-1-1,Shibaura,Minato-ku,Tokyo 105,JAPAN

Abstract

The concurrent object oriented language MENDEL(MEta iNferential system DEscription Language) which is based on Prolog is described. The characteristics of MENDEL are as follows, (1)meta inference mechanism, (2)object concurrency, (3)propositional temporal logic. This paper also describes the way of MENDEL object generation.

INTRODUCTION

When describing many systems in this world, the object modeling is to naturally pursue the development of models which modify the internal states in response to external messages, as well as issuing messages. At this time, the internal object description and the description of the relation between the objects is completely a separate dimension for discussion. The concurrent object oriented language can be applied to extensive domain such as expert system, simulator, and software specification. However, for the real time system, few language has been proposed.

Under the circumstance, the authors considered language for real time systems satisfying requirements such as,

1) object-based software development,
2) concurrent execution of the object,
3) time concerned model,
4) software reusability, and
5) logic programming language based specification,

and developed Prolog based concurrent object oriented language MENDEL (MEta iNferential system DEscription Language). The characteristics of MENDEL are as follows,

1) meta inference mechanism,
2) object concurrency,
3) linkage to C language,
4) propositional temporal logic, and
5) Prolog predicate such as SEIZE, RELEASE for performance prediction simulators.

In this paper, Section 2 outlines the MENDEL and Section 3 explains the way of MENDEL object generation.

MENDEL

The internal and external objects of MENDEL can only transmit information through pipe caps. Each pipe can be used for either input or output but only on a one-way basis. An attribute name is assigned to each pipe cap and is used by the internal object to refer to the input-output messages. This attribute name is not simply the pipe cap ID, but also regulates the input-output message attributes (specifications). The relation between the objects is created according to the binding between pipe caps by the transmission pipes, as shown in Fig. 1. This transmission pipe is a one-to-one asynchronous one-way path.

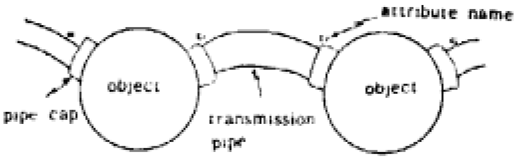

Fig. 1 Object and communication pipe

A. META-INFERENCE

The binding between pipe caps is performed automatically in MENDEL. For object groups, it selects the necessary objects by inference to accomplish a certain goal. If a certain goal (input-output specification) is given for a group of objects, the binding agent selects the necessary objects by inference in order to accomplish that goal, and binds the transmission pipes (Fig. 2). This inference is called a meta-inference, which means that it infers a strategy to derive a solution. A meta-inference is performed according to the knowledge concerning the object interface specifications (meta-knowledge), and the inference rules of the binding agent. To accomplish a given goal, the necessary object is found by trial and error. The meta-inference in MENDEL can be considered to be broadcasting using a method search of one realized configuration. For many object-oriented languages the method search order is depth-first, left-to-right. However, if this method can be selected, even in multiple cases, the first method which is arrived at is selected and executed. The meta-inference for this selects the most suitable object method by broadcasting from the input-output attributes declared by each object. At this time, there are multiple paths which exist, but in order to deal with them it is necessary to establish some kind of evaluation standards. A meta-inference has many strategies for that purpose. For example, if every method is appended at processing time, by considering each method at the defined processing time from any number of candidate paths, the path with the shortest processing method is selected. When the processing time is not fixed, the method using the least number of pipes is used. Attributes which have different name caps but have the same meaning, can be defined for that purpose.

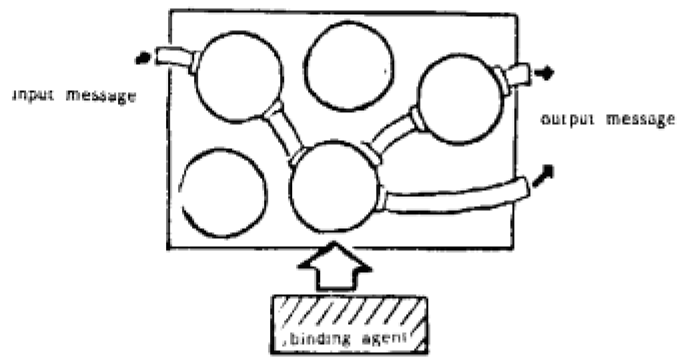

Fig. 2 Binding agent and meta-inference

B. MENDEL SYNTAX SUMMARY

The object is configured from the meta-knowledge and object-level knowledge.

Meta-knowledge is the knowledge concerning the concept "which way can be used, and which way has to be used?", as well as the external object specifications and especially the interface relationships. For object-level knowledge, the processing contents are described from the externally input messages. The actual external independence of the object-level knowledge can be increased by dividing the meta-knowledge and the object-level knowledge. In the MENDEL objects there are classes and instances. Initially, only the class of sets exists in the system. The binding agent then selects and binds the necessary classes. When the binding is completed the object is called an instance. It is also possible to create multiple differing instances from the identical class.

Object-level knowledge is configured from any number of methods and common knowledge. The input values (attribute values) are obtained from each pipe and the attribute name of each pipe cap are called the port table. The object is to start the appropriate method if messages are received in this port table. The method is declared as follows.

```
Method (attribute ? logical variable
        name) ...(attribute ! logical
        variable name ...)
         -- Prolog clause body
        Prolog clause
```

If the logical variable after each attribute? has just been input from the port table, that method is started. Then, when the method is completed, if the logical variable after attribute! is unified, and the unified value is output to the transmission pipe through the port table. The MENDEL description example is shown in Fig. 3.

```
EmployeeAccess
{
 spec:{
      supers( Employees ).
      method( name? sex? idnumber! ).
      method( name? age? idnumber! ).
      method( sex? age? idnumber! ).
    }
 body:{
      method( name?X sex?Y idnumber!Z
            <-database( Z, X, Y, _ ).
           }
      method( name?X age?Y idnumber!Z
            <-database( Z, X, _, Y ).
           }
      method( sex?X age?Y idnumber!Z
            <-database( Z, _, X, Y ).
           }
      }
}
```

Fig. 3 Example of MENDEL

Meta-knowledge includes the following three items.

1) input-output description of the method
2) super class definition
3) definition of internal state variables

The input-output specifications of all object methods are expressed using the system predicate "method".

    method (attribute? ... destination ID:
        attribute!   ...)

The destination ID specifies the destination of messages and includes these types.

    object name - specifies the object
                  name directly
    METAPHOR - decided by meta-inference
    BROADCAST - is broadcasting
                  (transmission (!) only)

The default value is METAPHOR. The binding agent performs the binding using this meta-knowledge. The most basic binding rule is to connect pipe caps which have the same attribute name by pipes.
The definition of a super class uses the system predicate "supers" (statement).

    supers (upper position object name)

When this is specified from the upper level object, its object inherits the undefined predicate.
This is the definition of internal state variables.

    variables (variable name! initial
                  value)

The system predicate "communicate" is called the expanded module call. If the given attribute and the desired attribute are specified, the appropriate object is selected and binding is performed. The desired attribute value is obtained by operating each object concurrently. Since the object selection and binding are left to meta-inference, the program description is unnecessary. When binding is performed for a meta-inference, the following is expressed.

    Communicate METAPHOR (attribute! logical
        variable name ... attribute? logical
        variable name ...)

This communicate sentence is not only written for a Prolog clause of a object body part, but is also used for the goal calls to open a MENDEL program.
The description of concurrency and hierarchy is also possible by using the communicate statement. The portions surrounded by the communicate statement become the AND-parallel description. In the communicate statement, the Prolog predicate can be considered to be the object. On one hand, concerning the hierarchy, if the goal clause in the communicate statement is given, an inference configuration which is bound by the meta-inference is formed. If a communicate statement appears in the instance body part, the lower level configuration is formed by the meta-inference. At this time objects of the same class exist for differing instances in each of the successive configurations.

For the processing units (tasks) of real-time systems to be executed concurrently, the problems of synchronicity and exclusivity are produced. In MENDEL, message passing from object concurrency is not the procedure call format adapted in Smalltalk-80[3]. When messages are transmitted in Smalltalk-80, the program control is transferred to the object receiving the messages, and the method which corresponds to those messages is executed. The return value is then sent to the transmission side object, and the rules are adapted. Since each concurrent object execution is difficult using rules like this, the concurrent object-oriented language requests the no-return message passing for transmission only. For the concurrent object, the messages are sent asynchronously from the concurrently operating object, and the method is selected according to the combination of those messages. In the example

    method (a?, b?, c?, d!)

when a?, b?, c? are transmitted from different objects, this method is not selected until a?, b?, c? are all completed. Consequently, when multiple messages are transmitted from different objects, the initial method is selected when the messages are totally completed. In MENDEL, the synchronous structure for this purpose exists internally. The selection method for MENDEL is considered to be attached to a Dijkstra's guarded command [2]. For the declared message, when a certain compound "and" condition becomes true (i.e., when all messages are completed, or when the conditions of all message contents are satisfied), the execution of the message is performed since this guard is permitted. When the object method is selected, and another message is sent to the object during execution, the most recent message will be processed.

The following system predicate is provided in MENDEL for simulation description use.

    $decresource (resource name) :
        resource declaration
    $seize (resource name) : resource
        secured
    $release (resource name) : resource
        release
    $dectime (time) : simulation time unit
        declaration
    $hold (time) : period of time of hold

C language is called from MENDEL by using the system predicate, and is performed by the $usercall command. The first argument of the $ usercall command is the C language filename. The second and succeeding parameters are the arguments of the object function. The symbols ? and ! are attached before each argument to distinguish between each input or output parameter.

For process control systems which use external production control devices, in order for the time and conditions to be modified together, it is difficult to express the system adequately by only using the predicate calculus. Therefore, to apply the expert system in the real time process control system, it is necessary to overcome this problem point. MENDEL adapts temporal logic which solves the problem mentioned above. Deadlock may occur, especially when the execution order of the concurrently operating object is not regulated. Definitions are produced which are necessary to define the sequential relationship between the objects. In MENDEL it is possible to regulate a message exchange which does not produce deadlock by meta-inference. On one hand for object-level knowledge, the temporal relationship operation in a object-level system can be expressed by temporal logic, but there is also the main purpose of also utilizing a simulation which evaluates performance factors.

MENDEL object generation

For support systems for real time systems, the objective is to create a MENDEL object by program transformation from concurrency, and temporal dependence requirement specifications of certain real-time systems. In this case program transformation is realized by either algorithms or expert systems.

The meta-knowledge of MENDEL are first generated from the requirement specifications by program transformation. In MENDEL the requirement specifications are configured from the system transitional logic and the required proposition. The system transitional logic is the expression of the system logic by the Petri net. The "fact" in each transition unit is described here. This is the description format.

transition (transition name, input place name, output place name)

The required proposition is the description by temporal logic of temporal relationship between each transition, and the determination of the system operations described by system transitional logic. The exchange of messages between objects to avoid deadlock is realized by these two descriptions. This means that the MENDEL meta-knowledge is generated. At this time the generation is performed by the expanded method of the tableau method in the temporal logic[4]. The procedure is as follows.

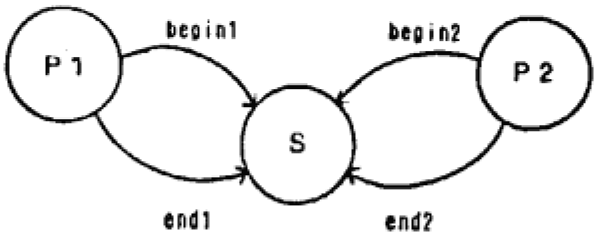

Fig. 4 Example of mutual exclusion

```
transition(begin1,[place1],[place2]).

transition(end1,[place2,place3],[place1,place4]).

transition(begin2,[place5],[place6]).

transition(end2,[place4,place6],[place3,place5]).
```

Fig. 5 System transitional logic

$\Box(S!begin\ i \supset \bigcirc S!end\ i)$

$\Box(S!end\ i \supset \bigcirc S!begin\ i)$

$\Box(P1?begin\ 1 \supset ((\neg P2?begin2) \cup (P1?end\ 1)))$

$\Box(P2?begin\ 2 \supset ((\neg P1?begin1) \cup (P2?end\ 2)))$

$\Box(P1 \supset \bigcirc P2)$

Fig. 6 Required proposition

```
S
(spec(
      method(begin1?, N-out?'1', N-in!'2')
      method(begin2?, N-out?'1', N-in!'3')
      method(end1?, N-out?'2', N-in!'1')
      method(end2?, N-out?'3', N-in!'1')
      )
)
P1
(spec(
      method(N-out?'1', begin1!)
      method(N-out?'2', end1!)
      )
)
P2
(spec(
      method(N-out?'1', begin2!)
      method(N-out?'3', end2!)
      )
)
N
(spec(
      method(N-in?, N-out!)
      )
)
```

Fig. 7 Part of meta knowledge

1) A tableau graph is generated by the tableau method.
2) The system is checked to make sure that no deadlock condition exists.
3) The meta-knowledge concerning message exchange is created.

A simple example is shown in Fig. 4. Process P1 and P2 are exchanging S which is the critical section at this time. In these relationships, P1 is sending the begin-1 and end-1 message to S, and P2 is sending the begin-2 and end-2 message to S. The corresponding system transitional logic for these relationships is shown in Fig. 5, and the required proposition is shown in Fig. 6. The MENDEL meta-knowledge is created from these definitions as shown in Fig. 7. In this case, P1, P2, S and the object N which shows the tableau position are created. From a given user requirement, the components to satisfy that requirement are selected and combined by meta-inference, so it is possible to configure a prototype rapidly. By using these input-output specifications which are generated by tableau method mentioned above, it is indicated if a check would be made to determine whether a series of components exist or not. Component detection is also possible using the specification level, since information concerning object-level knowledge can be described by meta-knowledge, and because the meta-knowledge level can be detected.

CONCLUSION

The summary of MENDEL has been briefly described. For the intelligent programming environment, MENDELS is under development. The interpretation of intelligent programming environment for this MENDELS is not that suitable programs can be automatically created by inputting ambiguous specifications, but that if rather stringent specifications are persistently assumed, lightening the burden of the intellectual operations of programming. The key point of MENDELS are functions which effectively implement certain meta-inferences provided by the distinguishing characteristics of MENDEL. In other words, rigidly classified types are incorporated in attributes, and the interface between types is syntactically and semantically strict. However since only simple strictness becomes difficult to use, the ACM (Attribute Control Module) is provided. MENDELS is configured from the MENDEL compiler and the object manager which includes ACM, DWIM, intelligent editor etc.

ACKNOWLEDGEMENTS

Parts of this work are based on the results of the R & D activities of the Fifth Generation Computer Systems Project. The authors would like to thank Dr. T. Yokoi of ICOT (Institute for New Generation Computer Technology) for his encouragement and support.

The authors wish to express deepest gratitude to Akira Ito, Masahiko Arai, and Shinsuke Tamura of Systems and Software Engineering Division in Toshiba Corporation for providing essential support.

REFERENCES

[1] D.G.Bobrow and M.Stefik,"The LOOPS Manual",Memo KB-VLSI-81-13,Xerox Palo Alto Research Center,1982.
[2] E.W.Dijkstra,"Guarded Commans,Nondeterminacy and Formal Derivation of Programs",CACM,Vol.18,No.8,pp.453-457,1975.
[3] A.Goldberg et al.,"Smalltalk-80 Language and its implementation", Addison-Weslay,1983.
[4] Z.Manna and P.Wolper,"Synthesis of Commmunication Processes from Temporal Logic Specification",Lecture Notes in Computer Science 131,pp.253-281, Springer-Verlag,1982.
[5] D.May,"occam",SIGPLAN Notices,Vol.18,No.4,pp.69-79,1983.
[6] L.M.Perira and R.Nasr, "Delta-Prolog:Distributed Logic Programming Language", Proc.FGCS'84,pp.283-291,1984.
[7] K.Furukawa et al,"Mandala:A Logic Based knowledge Programming System", Proc.FGCS'84,pp.613-622,1984.