# Suitable Daemon Function for Building Knowledge Base in Mechanical Design Field*

Yasumi NAGASAKA**, Happy WIBISONO**, Hideyuki OHTAKI** and Yoshio ISHIKAWA**

To build an expert system, especially in the mechanical design field, a knowledge base which contains a wealth of knowledge is an important issue for improving the efficiency of the design process. However, it is very difficult to build a knowledge base in mechanical design because there is a great deal of complicated knowledge in this field. In this work we derive an approach to build large knowledge bases in mechanical design by using an extended daemon function. Using this method, we believe it becomes possible to realize the characteristics of data base systems (DBS) in knowledge base systems (KBS). As a result, KBS will be suitable for managing and representing knowledge, and it will become easier to build a knowledge base in the mechanical design field.

***Key Words***: Design, Artificial Intelligence, Knowledge Base, CAD, Expert System, Frame Representation, Daemon Function

## 1. Introduction

Recently, the application of expert systems to engineering has drawn much attention. Introduction of knowledge engineering to CAD systems may expand computer application to wider domains in engineering design beyond the current state of numerical computation and drawing.

To build an expert system, especially in mechanical design, a knowledge base which contains a wealth of knowledge and experience of experts, is an important issue for improving the efficiency of the design process. However, there is a vast amount of complicated knowledge in mechanical design, so it is very difficult to build a knowledge base in this field. Takizawa et al.(1) have attempted to build a large knowledge base effectively by connecting data base systems (DBS) and knowledge base systems (KBS) systematically. However in their work most DBS were only used as a memory because DBS are suitable for storing much data, not for representating knowledge. If the characteristics of DBS can be realized in KBS, we believe a great deal of knowledge (data and formulae) of mechanical design may be efficiently knowledge-based.

In our investigation, we have built a management system(2) of a knowledge base for engineering design based on frame representation(3). The system has a data structure (basic building blocks to build up the knowledge base) which is similar to that in KEE(4), i.e., "UNITS-SLOTS-FACETS-VALUES". Here, we have introduced the "ATTRIBUTION" concept(2) in place of "FACET". The principle of ATTRIBUTION is the incorporated frame of knowledge representation based on attributes of the objects. In other words, in the current data structure, the frame of FACETS (attributes) inside the SLOTS is fixed; however, in ATTRIBUTIONS, attributes in the SLOTS can be defined and changed freely, similar to the relation of "SLOT-FACET" in STROBE(5).

Initially, we defined the basic building blocks of knowledge base in SLOTS as (slot-name slot-type slot-value). Then, the default values and daemon

* Received 8th December, 1992. Paper No. 91-1385
** Department of Mechanical Engineering, Saitama University, 255 Shimo Okubo, Urawa, Saitama 338, Japan

functions can be defined freely if needed. Daemon functions are an attached-procedure which can work automatically when we are referring and editing slot-values. Consequently, because the daemon function to manage slot-values can be defined and executed before the slot-values are evaluated, the variables described in the slot-value can be defined automatically. As a result, the complex formulae used in mechanical design, and also logical expressions, can be described in slot-value. For that reason, it is possible to realize the characteristics of DBS in KBS.

We derive the extension of the daemon function for building a knowledge base in the mechanical design field, and generation of the functions to realize the characteristics of DBS in KBS, based on the above concept.

## 2. Suitable Daemon Function for Building Knowledge Base in Mechanical Design Field

### 2.1 Daemon Function and extended function

Currently, typical daemon functions are "if-needed", "if-added" and "if-removed", for example. These daemon functions do not provide ability to directly manage slot-value. Therefore, we have tried to make an extension of the following functions so that they will be able to manage slot-value directly.

**2.1.1 Handling numerical and empirical formula as a slot-value** Regarding mechanical design, there are many numerical and empirical formulae, so if these formulae can be described in term of slot-value, it will be very useful when building a knowledge base. In connection with the above matter, we have realized a function to evaluate the variables which are used in such formulae and append it to the daemon function.

**2.1.2 Realizing characteristics of DBS in KBS** There are some problems in connection with the capacity of the memory if data in the recording units of DBS are stored directly in terms of frame-based representation units. For that reason, it is important to create a function in a slot of frame representation,

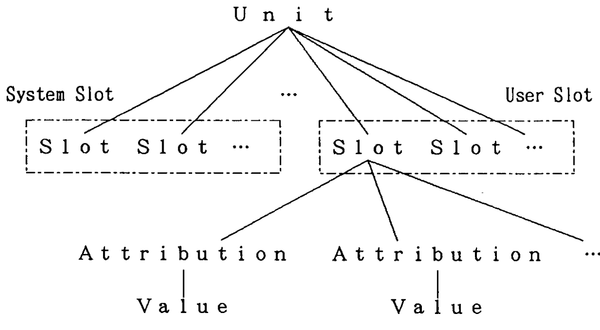


Fig. 1 Data structure of knowledge base

which plays the role of a record in DBS. To realize such a function, first the relationship between the record and field in slot-value is described. Then, by using this relationship we simply incorporate the evaluation function when we are referring to the data.

### 2.2 Extension of daemon function

Figure 1 shows the data structure of the knowledge base. Slots consist of system slots and user slots. System slots, as described above, have "SLOT-NAME, SLOT-TYPE, SLOT-VALUE" as their attributes which manage independence between units. In user slots, in addition to these attributes, the user can define any other attributes if needed.

Figure 2 shows the concept of the daemon "if-varied" function which is adopted to realize the function described in subsection 2.1. This daemon function is described as

( (variable-1 attached-procedure-1)
(variable-2 attached-procedure-2) ....).

Here, variable-1 is the name of a variable which is described in slot-value. The result of evaluation of attached-procedure-1 is substituted to this variable-1. Then, if attached-procedure-1 is omitted, variable-1 signifies a slot-name in class frames in frame representation.

If the slot-value is referred by message-passing (Fig. 2 (1)), the system determines whether this daemon function "if-varied" is already defined or not (Fig. 2.(2)). In the present case, it has already been defined, and hence the daemon function "if-varied" will first evaluate attached-procedure-1 (Fig. 2(3)), then the result of evaluation is substituted to variable-1 (Fig. 2(4)). After all the variables in the list are evaluated (Fig. 2(5)), they, in turn, will evaluate slot-value. Then the result is returned to function which is defined by the user (Fig. 2(6)).

In the current daemon function, the formula is defined in the attached-procedure function, so it is very difficult to understand the details of the formula and to modify the slot-value used in the formula. In comparison, by using the daemon function "if-varied" suggested here, because of the reasoning route shown in Fig. 2, formula can be treated as slot-value because the variable used in the formulae is described in the form of ((variable-1 attached-procedure-1) ..). Hence, if the slot-name is described in variable-1, it is easy to understand and to modify the slot-value in that formula. With this extension function, as will be described in subsection 2.4, formula can be described in a slot directly, because variables in this formula will be evaluated by the daemon function "if-varied". For that reason, it will be easier to build and to modify the knowledge base.

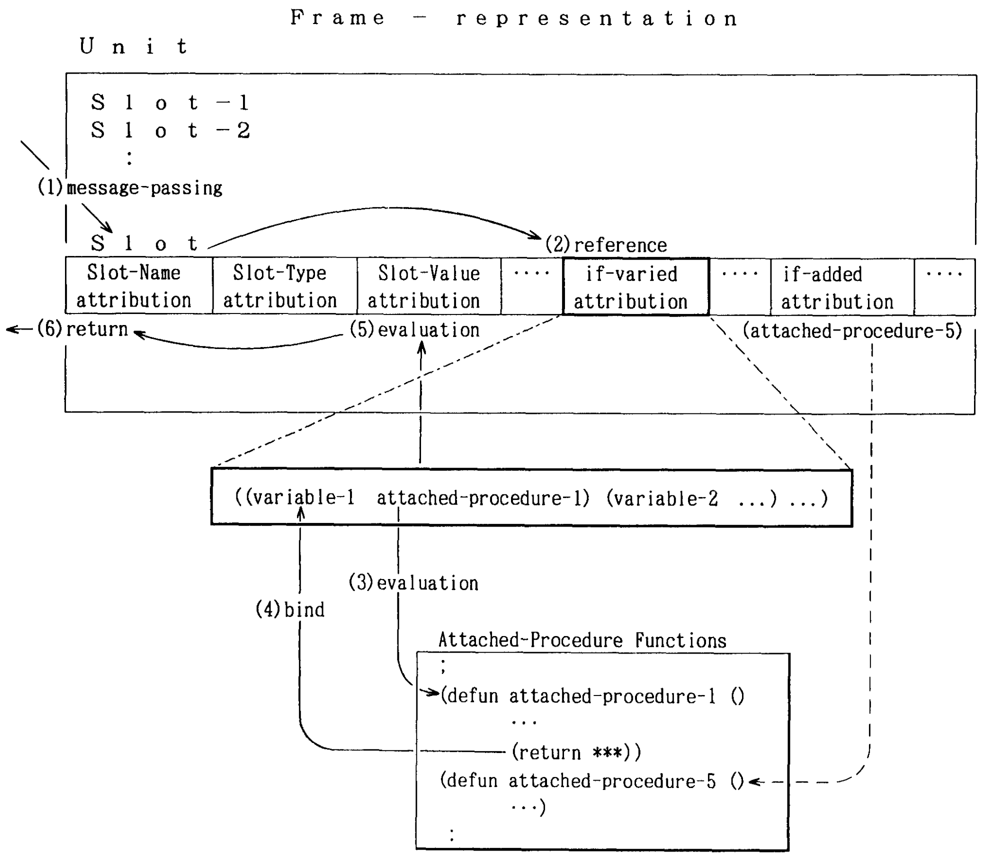


Fig. 2 Concept of If-varied daemon function

### 2. 3 Extension of system slot

Figure 3 shows the concept of the system slot "Cluster-Units" which is extended for realizing a function in KBS as well as DBS. Cluster-Units manage the relationship between fields in a record in DBS. If the constructions of slots in a unit are identical, all similar units will be represented as a single unit. The figure shows an example for choosing the "basic dimension of metric coarse screw threads". Figure 3(a) is a sample of the record representation of the data base, Fig. 3(b) shows a sample of frame representation and Fig. 3(c) shows a sample of frame representation of the knowledge base, using extended system slot "Cluster-Units". The thick full line in this figure shows the representation of the "M 8 Screw".

Figures 3(a) and 3(b) are current representations, whereas Fig. 3(c) is the new representation suggested here. This approach is a combination of the characteristics which are provided in the current representations (Figs. 3(a), (b)).

For example, a slot in frame representation (thick full line in Fig. 3(c)) expresses a record in DBS; the conditions for obtaining the field in this record are defined in slot-value of the same unit, as shown within the dotted line in Fig. 3(c). In the system slot "Cluster-Units", the unit name shown in Fig. 3(b) is stored in a list. This list of unit names is automatically recorded in a global variable (*Cunits*) in the system. Now, the search route (message-passing) with reference to a slot-value, by using this knowledge representation, is described as follows. First, to obtain the "pitch of M 8 metric-coarse-screw-threads", Message-1 is sent as (MGR-GET-VALUE 'M 8' Pitch). More explanation about this function MGR-GET-VALUE will be described later in subsection 3. 1.

During this message-passing, first, unit "M 8" is searched. If the object unit is absent, the search will be done in the global variable (*Cunits*) to check whether the unit is recorded in the form of Cluster-Units or not. If it is recorded in the form of Cluster-Units, it will be treated as a function of system slot, and the following search will be done.

As the first step, the function will select a present unit from the global variable (*Record*), and then Message-2 is done, so that the information which is stored in a record of DBS will be substituted to the global variable (*Record*). Subsequently, Message-3 is done, and finally, the value 1.25 is returned as the result.

Message-2 : (MGR-GET-VALUE 'M-COARSE-SCREW-THREADS 'M 8)

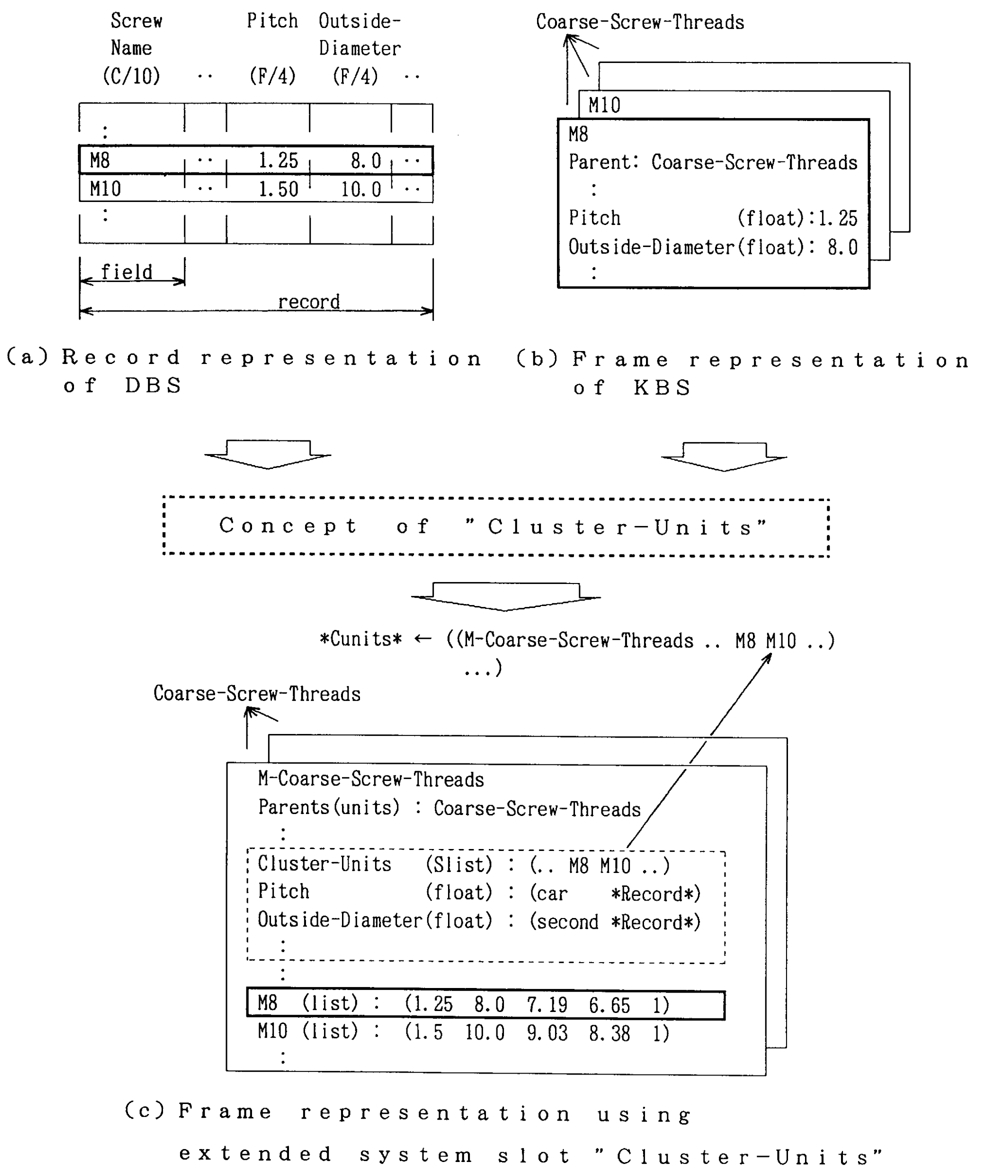

Screw Name (C/10)
Pitch (F/4)
Outside-Diameter (F/4)
M8 .. 1.25 8.0 ..
M10 .. 1.50 10.0 ..
field
record
(a) Record representation of DBS
Coarse-Screw-Threads
M10
M8
Parent: Coarse-Screw-Threads
Pitch (float):1.25
Outside-Diameter(float): 8.0
(b) Frame representation of KBS
Concept of "Cluster-Units"
*Cunits* ← ((M-Coarse-Screw-Threads .. M8 M10 ..) ...)
Coarse-Screw-Threads
M-Coarse-Screw-Threads
Parents(units) : Coarse-Screw-Threads
Cluster-Units (Slist) : (.. M8 M10 ..)
Pitch (float) : (car *Record*)
Outside-Diameter(float) : (second *Record*)
M8 (list) : (1.25 8.0 7.19 6.65 1)
M10 (list) : (1.5 10.0 9.03 8.38 1)
(c) Frame representation using extended system slot "Cluster-Units"

Fig. 3 Concept of system slot “Cluster-Units”

Message -3 : (MGR - GET - VALUE * 'M - COARSE-SCREW-THREADS 'Pitch)

Hence, by applying the extension of the daemon function described in subsection 2. 2 and the extension of system slot, it is possible to create a KBS which has functions to express a great deal of knowledge, as well as the DBS. Furthermore, by using the current message-passing, it is also possible to determine the slot-value.

## 2. 4 Description of formula

Figure 4 shows the difference in the description of formulae between current frame representation and ones which use the extended daemon function.

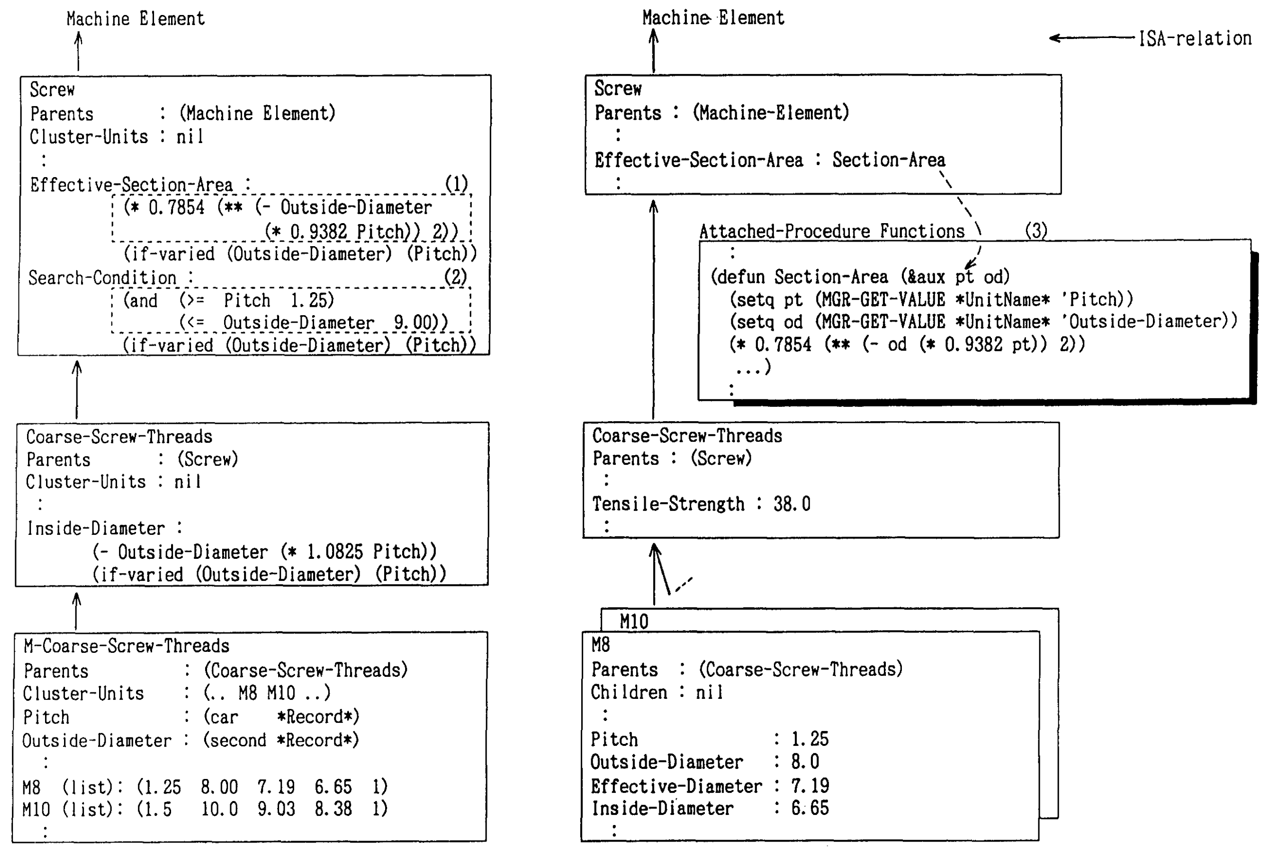
Here, a formula related to “the effective sectional area of a screw” is used as an example. Referring to ISO 898/1, the formula is obtained as

$$As = 0.785\,4 * (d - 0.938\,2 * P)^2$$

$As$ (mm$^2$) : Effective sectional area of screw
$d$ (mm) : Outside diameter of male screw
$P$ (mm) : Pitch of screw.

Figure 4(a) shows that the above formula is described in a slot-value by using the extended daemon function (dotted line (1)). In addition, in this formula not only arithmetic operations, but also multinominal differential and integral calculus, can be described. Even if there are operators which are not in this system, since the operators can be recorded in the library, it will be able to treat such formulae which are used in the mechanical design field. In particular, the formulae which are developed in attached-procedures languages such as C-language or FORTRAN, can also be treated in slot-value.

In comparison, Fig. 4(b) shows an example of a formula which is described by current frame represen-



```
Screw
Parents       : (Machine Element)
Cluster-Units : nil
  :
Effective-Section-Area :                          (1)
        (* 0.7854 (** (- Outside-Diameter
                         (* 0.9382 Pitch)) 2))
        (if-varied (Outside-Diameter) (Pitch))
Search-Condition :                                (2)
        (and  (>=  Pitch  1.25)
              (<=  Outside-Diameter  9.00))
        (if-varied (Outside-Diameter) (Pitch))

Coarse-Screw-Threads
Parents       : (Screw)
Cluster-Units : nil
  :
Inside-Diameter :
      (- Outside-Diameter (* 1.0825 Pitch))
      (if-varied (Outside-Diameter) (Pitch))

M-Coarse-Screw-Threads
Parents          : (Coarse-Screw-Threads)
Cluster-Units    : (.. M8 M10 ..)
Pitch            : (car    *Record*)
Outside-Diameter : (second *Record*)
  :
M8  (list): (1.25  8.00  7.19  6.65  1)
M10 (list): (1.5   10.0  9.03  8.38  1)
  :
```

```
Screw
Parents : (Machine-Element)
  :
Effective-Section-Area : Section-Area
  :

Attached-Procedure Functions                      (3)
  :
 (defun Section-Area (&aux pt od)
   (setq pt (MGR-GET-VALUE *UnitName* 'Pitch))
   (setq od (MGR-GET-VALUE *UnitName* 'Outside-Diameter))
   (* 0.7854 (** (- od (* 0.9382 pt)) 2))
   ...)
  :

Coarse-Screw-Threads
Parents : (Screw)
  :
Tensile-Strength : 38.0
  :

M10
M8
Parents  : (Coarse-Screw-Threads)
Children : nil
  :
Pitch              : 1.25
Outside-Diameter   : 8.0
Effective-Diameter : 7.19
Inside-Diameter    : 6.65
  :
```

(a) Example using extended daemon function "if-varied"     (b) Example using current Frame-representation

Fig. 4 The differnce in description of formulae

tation. In this representation, the formula is described by the function's name in the slot-value as an attached-procedure, or described directly by using the lambda-format. To do so, one must create a complicated program, and consequently, it becomes more difficult to build a knowledge base using this representation.

Another example of the description of the logical operators is shown within the dotted line ( 2 ) in Fig. 4. This is an example of the search screw which suits the following conditions : "pitch $>=$ 1.25 and outside-diameter $<=$9.0". As we know from this example, functions similar to DBS's search functions can easily be described in slot-value.

## 3. Realizing Extended Daemon Function and Its Evaluation

### 3.1 Method for realizing daemon function

Figure 5 shows the listing of the MGR-GET-VALUE function which refers to slot-value. These functions are encoded in KYOTO COMMON LISP (KCL), and because KCL is developed based on C-Language, the functions can easily be connected to C-Language. They can also be connected to FORTRAN and other programming languages easily. As a result, formulae defined in slot-value described in subsection 2.2 can be used.

The MGR-function group (functions which start with "MGR-") in the figure, can be modified freely by the user, while COR-function group (functions which start with "COR-") is managed by the system. From the list within the dotted line, the MGR-function group is described by using the COR-function group and the defined MGR-function. This means, since the user can define the MGR-function group freely, that if necessary, the user can modify the data structure in the slot and its functions. For example, the dotted line ( 1 ) in the figure shows the extended part of the system slot. This part (by using the MGR-CHK-CUNITS function) is to determine whether the object unit is registered in Cluster-Units (*Cunits*) or not ; subsequently the function we described in subsection 2.3 will be realized. Furthermore, the dotted line ( 2 ) in the figure shows the extended part of the daemon function.

Figure 6 shows a simple flowchart of the main part of the MGR-GET-VALUE* function which refers to slot-value in Fig. 5. This function consists of two parts. The first part obtains a slot and the second part evaluates it when the slot is present. The former is executed by the COR-GET-SLOT function, while the latter is executed in the steps below.

First, if there is an attached-procedure which expresses the "if-needed" daemon function in the slot, COR-EVL-FUNCTION is executed to evaluate the attached-procedure. Then, if the list formation which expresses an extended "if-varied" daemon function is present, the function described in subsection 2.2 will be executed (see dotted line (2) in Fig. 5). At this point, all of the attached-procedures in this format are evaluated, and the evaluation results will be connected with the variables. If only a variable (without attached-procedure) is present, the variable is interpreted as a slot-name, and function MGR-GET-VALUE is executed recursively; then, the result is substituted to the variable.

Finally, in the case when the slot-value is a formula, the above variables will be used to evaluate it, and then the above variables will be returned as an evaluation result of the function MGR-GET-VALUE.

In this method, the extension of the daemon func tion and system slot is realized by modifying the MGR-function. The interdependence of the MGR-function group is managed by the system, which keeps the data structure in the knowledge base and the function of system slot consistent.

## 3.2 Evaluation of building knowledge base using extended daemon function

Figure 7 shows the comparison of the memory that is used when building the knowledge base "ISO metric screw threads - Basic dimensions (ISO-724)"

```
; Message function
(defun MGR-GET-VALUE (UnitName SlotName)
  (cond ((COR-GET-UNIT UnitName)
           (setq *UnitName* UnitName)
           (setq *SlotName* SlotName)
           (setq *Record*   nil)
           (MGR-GET-VALUE* UnitName SlotName))
       ; Extended Cluster-Units                              (1)
       ((MGR-CHK-CUNITS UnitName)
           (setq *UnitName* (MGR-CHK-CUNITS UnitName))
           (setq *SlotName*  SlotName)
           (setq *Record*   (MGR-GET-VALUE* *UnitName* UnitName))
           (MGR-GET-VALUE*   *UnitName* SlotName))))

; Message function
(defun MGR-GET-VALUE* (UnitName SlotName
                        &aux Slist Vlist Elist Edata (Rdata nil))
  (setq Slist (copy-tree (COR-GET-SLOT (UnitName SlotName))))
  (when Slist
      (setq Rdata (nth 2 Slist))
      ; if-needed daemon function
      (when (assoc 'if-needed (nthcdr 3 Slist))
           (COR-EVL-FUNCTION (cdr (assoc 'if-needed
                                        (nthcdr 3 Slist)))))
      ; if-varied daemon function                           (2)
      (when (assoc 'if-varied (nthcdr 3 Slist))
           (do ((Vlist (cdr (assoc 'if-varied (nthcdr 3 Slist)))
                 (cdr Vlist))
                ((null Vlist) nil)
                (setq Elist (copy-tree (car Vlist)))
                (if (second Elist)
                    (setq Edata (copy-tree
                           (COR-EVL-FUNCTION (nthcdr 1 Elist)))
                    (setq Edata (copy-tree
                           (eval (list 'MGR-GET-VALUE
                                        *UnitName* (car Elist))))))
                (eval (list 'setq (car Elist) Edata)))))
      ; evaluation
      (unless (and (member (nth 1 Slist) '(list Ulist Slist))
                    (listp Rdata))
              (setq Rdata (copy-tree (eval Rdata))))
   Rdata)

; Check function for extended Cluster-Units
(defun MGR-CHK-CUNITS (UnitName &aux Ulist (Uname nil))
   (do ((Ulist *Cunits* (cdr Ulist))
        ((null Ulist) nil)
        (when (member UnitName Ulist)
              (setq Uname (copy-tree (car Ulist)))))
    Uname)
```

Fig. 5 List of MGR-GET-VALUE function

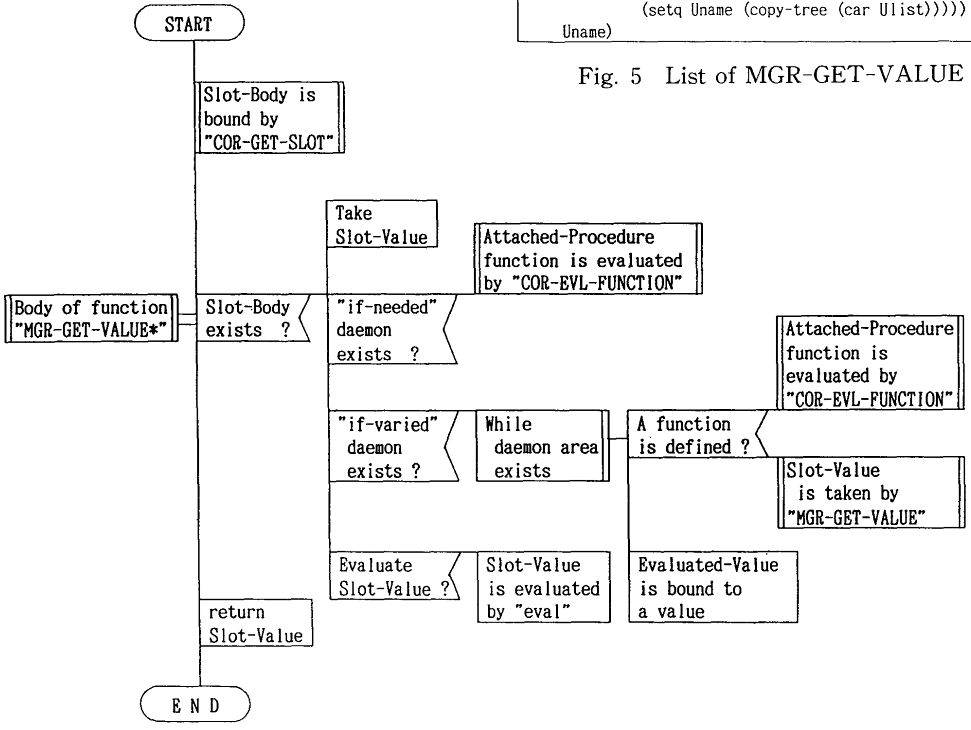

START
Slot-Body is bound by "COR-GET-SLOT"
Take Slot-Value
Attached-Procedure function is evaluated by "COR-EVL-FUNCTION"
Body of function "MGR-GET-VALUE*"
Slot-Body exists ?
"if-needed" daemon exists ?
Attached-Procedure function is evaluated by "COR-EVL-FUNCTION"
"if-varied" daemon exists ?
While daemon area exists
A function is defined ?
Slot-Value is taken by "MGR-GET-VALUE"
Evaluate Slot-Value ?
Slot-Value is evaluated by "eval"
Evaluated-Value is bound to a value
return Slot-Value
E N D

Fig. 6 Flowchart of MGR-GET-VALUE* function in Fig. 5

between two systems: (a) by using the extended daemon function represented in Fig. 5(a), and (b) one which uses the current frame representation. In the case of Fig. 7(b), the formula cannot be described in slot, so it must be described as the function name of the attached-procedure.

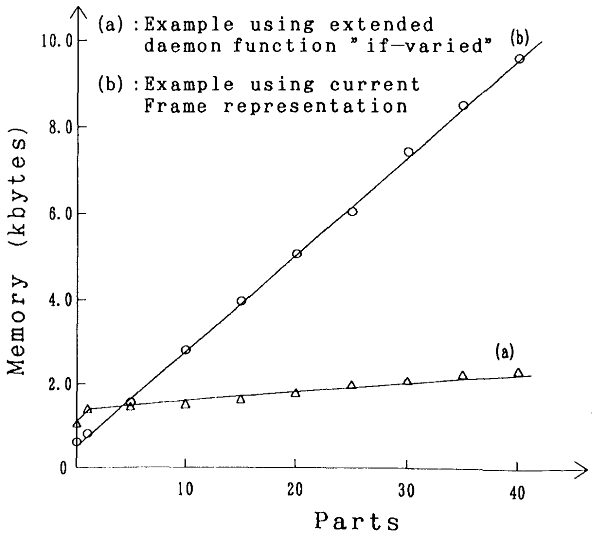


Fig. 7 Comparison of memory

The abscissa in the figure shows the number of specifics (facts) of "abstract-specific" relations in the frame theorem. In case of ISO-724, the number of specifics corresponds to the number of parts in the standard. If there are none, the number will be zero. In other words, the memory under this condition shows Class-frames of "COARSE-SCREW-THREADS" in Fig. 4, and memory in (a) becomes bigger because the formulae of the effective sectional area and inside diameter of the screw are stored.

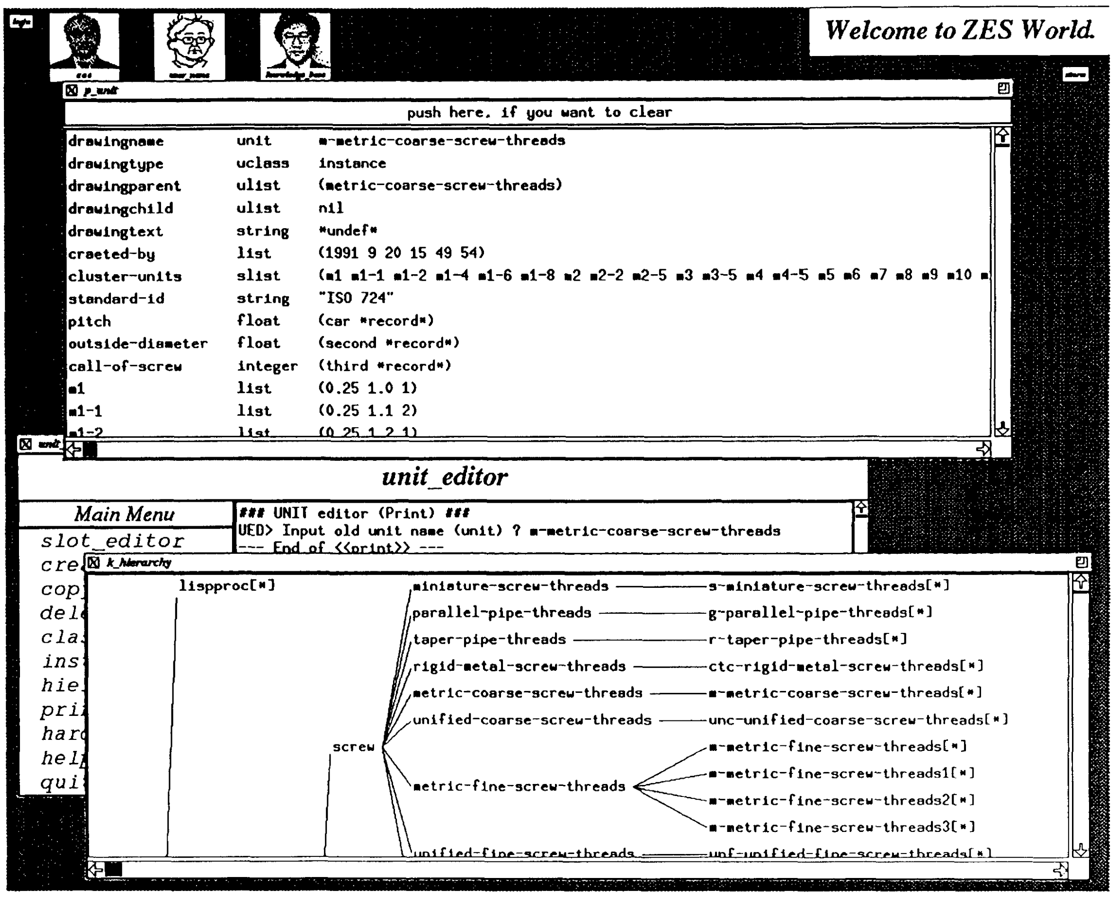
Moreover, the memory increment factor depends (a) on the amount of slot and (b) on the amount of unit. In a unit there is information to manage frame representation, e.g., inheritance-relationship. The difference in these formulae (see below) is expressed as the difference in the straight angle (slope).

memory (a) = A 1 ∗ parts + A 2

memory (b) = B 1 ∗ parts + B 2

Here, A 1 (bytes) is the memory of a slot. A 2 (bytes) is the memory which contains formulae using extended daemon functions. B 1 (bytes) is the memory of information using frame representation. B 2 (bytes) is the memory of the current representation.

For example, these variables can be quantified by counting backwards in Fig. 7. It is considered that

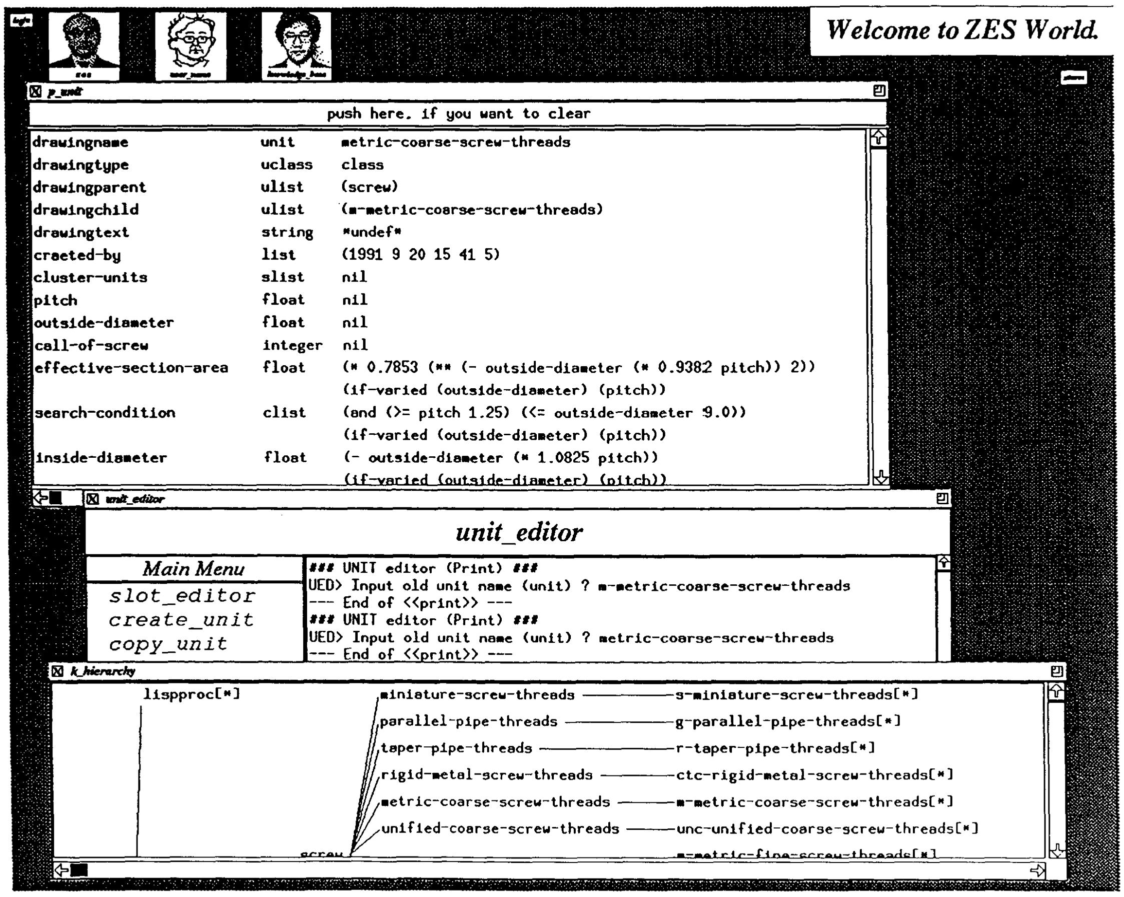


```
drawingname         unit      m-metric-coarse-screw-threads
drawingtype         uclass    instance
drawingparent       ulist     (metric-coarse-screw-threads)
drawingchild        ulist     nil
drawingtext         string    *undef*
craeted-by          list      (1991 9 20 15 49 54)
cluster-units       slist     (m1 m1-1 m1-2 m1-4 m1-6 m1-8 m2 m2-2 m2-5 m3 m3-5 m4 m4-5 m5 m6 m7 m8 m9 m10 m
standard-id         string    "ISO 724"
pitch               float     (car *record*)
outside-diameter    float     (second *record*)
call-of-screw       integer   (third *record*)
m1                  list      (0.25 1.0 1)
m1-1                list      (0.25 1.1 2)
m1-2                list      (0.25 1.2 1)
```

```
### UNIT editor (Print) ###
UED> Input old unit name (unit) ? m-metric-coarse-screw-threads
--- End of <<print>> ---
```

```
lispproc[*]
screw
  miniature-screw-threads ---------- s-miniature-screw-threads[*]
  parallel-pipe-threads ------------ g-parallel-pipe-threads[*]
  taper-pipe-threads --------------- r-taper-pipe-threads[*]
  rigid-metal-screw-threads -------- ctc-rigid-metal-screw-threads[*]
  metric-coarse-screw-threads ------ m-metric-coarse-screw-threads[*]
  unified-coarse-screw-threads ----- unc-unified-coarse-screw-threads[*]
  metric-fine-screw-threads -------- m-metric-fine-screw-threads[*]
                                     m-metric-fine-screw-threads1[*]
                                     m-metric-fine-screw-threads2[*]
                                     m-metric-fine-screw-threads3[*]
  unified-fine-screw-threads ------- unf-unified-fine-screw-threads[*]
```

Fig. 8 A sample of a part of knowledge base discussed in subsection 3. 2



```
drawingname              unit      metric-coarse-screw-threads
drawingtype              uclass    class
drawingparent            ulist     (screw)
drawingchild             ulist     (m-metric-coarse-screw-threads)
drawingtext              string    *undef*
craeted-by               list      (1991 9 20 15 41 5)
cluster-units            slist     nil
pitch                    float     nil
outside-diameter         float     nil
call-of-screw            integer   nil
effective-section-area   float     (* 0.7853 (** (- outside-diameter (* 0.9382 pitch)) 2))
                                   (if-varied (outside-diameter) (pitch))
search-condition         clist     (and (>= pitch 1.25) ((<= outside-diameter 9.0))
                                   (if-varied (outside-diameter) (pitch))
inside-diameter          float     (- outside-diameter (* 1.0825 pitch))
                                   (if-varied (outside-diameter) (pitch))
```

```
### UNIT editor (Print) ###
UED> Input old unit name (unit) ? m-metric-coarse-screw-threads
--- End of <<print>> ---
### UNIT editor (Print) ###
UED> Input old unit name (unit) ? metric-coarse-screw-threads
--- End of <<print>> ---
```

```
          lispproc[*]        miniature-screw-threads ---------- s-miniature-screw-threads[*]
                             parallel-pipe-threads ------------ g-parallel-pipe-threads[*]
                             taper-pipe-threads --------------- r-taper-pipe-threads[*]
                             rigid-metal-screw-threads -------- ctc-rigid-metal-screw-threads[*]
                             metric-coarse-screw-threads ------ m-metric-coarse-screw-threads[*]
                             unified-coarse-screw-threads ----- unc-unified-coarse-screw-threads[*]
                       screw                                    m-metric-fine-screw-threads[*]
```

Fig. 9 A sample of a part of knowledge base discussed in subsection 2. 4

there are many kinds of screws for an element in machine parts. Assume that to store 10 000 kinds of screws in a knowledge base, using the formula above, about 2.3 (Mbytes) in the current frame representation are needed. However, it takes only 0.3 (Mbytes) in the representation using the extended daemon function. In this comparison, we see that it is possible to build a knowledge base by using the extended daemon function.

As described above, this extended daemon function may be an effective function for building a knowledge base because it uses less memory and consequently, the searching time can also be reduced.

Furthermore, Fig. 8 shows a sample of a part of a knowledge base (ISO 724) discussed in section 3. 2. The upper window shows the internal part of the frame representation, and the lower window shows the structure of hierarchy in the knowledge base. Figure 9 shows a sample of the formulae in frame representation, as discussed in section 2. 4.

## 4. Conclusions

In this paper we described an extension for the daemon function "if-varied" and an extension for the system slot "Cluster-Units" which involves it, to build a knowledge base containing a great deal of complicated mechanical design knowledge including numerical formulae. With the extension function suggested here, we expect a standard, such as ISO or any other fixed standard, to be built as a wealthy knowledge base and to become more effective. We also explained how numerical equations can be described easily and effectively.

The following conclusions are drawn from the present study.

(1) By extending the daemon function, a formula can be described directly in slot-value. Because of this, we believe that it will be an effective function for building and describing knowledge bases.

(2) By extending the system slot, it is possible to reduce the memory of the knowledge base and to express the reduced knowledge base concisely.

(3) Through the extension of the two points mentioned above, it is possible to create in KBS, a system which has functions comparable to DBS.

## References

(1) Takizawa, M., Itoh, H. and Moriya, K., Logic Interface System on CODASYL Database System, Proc. of The Logic Programming Conf. (1986),

p. 111.
( 2 ) Nagasaka, Y., Ohtaki, H. and Ishikawa, Y., Management System of Knowledge Base for Mechanical Eng. based on Frame Representation, Jpn. Soc. Mech. Eng., (in Japanese), No. 58-547, C (91-1151 A), (1992), p. 301.
( 2 ) Minsky, M., A Framework for Representating Knowledge, J. Mind. Design, MIT, (1981), p. 95.
( 4 ) Kehler, T.P and Clemenson, G.D., KEE, Knowledge Engineering Environment for Industry, Artificial Intelligence, (1986), p. 573.
( 5 ) Smith, R.G., STROBE, Support for Structured Object Knowledge Representation, IJCAI, (1983), p. 855.