日本機械学会論文集(C編)
58巻549号(1992-5)

論文 No. 91-1385

機械工学分野の知識ベース化に適合したデーモン機能*

長坂保美*1, 大滝英征*2, 石川義雄*2

Suitable Daemon Function for Building Knowledge Base in Mechanical Engineering Field

Yasumi NAGASAKA, Hideyuki OTAKI and Yoshio ISHIKAWA

It is very difficult to build a knowledge base in the mechanical engineering field. Because there is a great deal of complicated knowledge in mechanical engineering in comparison with other engineering fields. To build a large knowledge base, much research to connect Data Base System (DBS) and Knowledge Base System (KBS) systematically has been conducted. DBS is suitable for storing much knowledge but is not suitable for the representation of knowledge. Therefore, we believe it is possible in the mechanical engineering field to realize DBS characters in KBS by using an extended deamon function and extended system slot. If this system is realized, KBS will be suitable for managing and representing knowledge, and it will be easy to build a knowledge base in the mechanical engineering field. This paper describes the method of building a large knowledge base in mechanical engineering using an extended daemon function and an extended system slot, and "the realization of such a knowledge base."

Key Words: Expert System, Artificial Intelligence, Production System, CIM, Knowledge Base, Frame Representation, Daemon Function

1. 緒　　言

設計者や専門家の豊富な知識や経験を知識ベース化することは，設計製造業務全体を効率化する上で重要な課題である．

そこで，データベースシステム(DBS)と知識ベースシステム（KBS）を有機的に結合する研究[1]が行われてきた．この研究では，DBSを単に記憶領域として使用している場合が多い．これは，DBSがKBSに比べて容量性に優れているものの記述性に難があるためである．機械工学分野では，多くの数式などを用いるため記述性に優れていることが肝要である．そのため，上記の方法では有効的なシステムとして構築し難いきらいがある．

これに対し，KBS上でDBSの特徴を生かすことができるなら，機械工学分野の数式を含む膨大な知識を有効的で，記述性のある知識ベースとして構築できるものと期待される．

そこで，著者らは機械工学分野の知識ベース化のためのツール構築と知識ベース構築を行っている[2]．ツール構築では，フレーム形表現KEE[3]のデータ構造「ユニット-スロット-ファセット-バリュー」を準用し，「ファセット」に代わる「アトリビューション」という考え方を導入した．アトリビューションの基本は，対象の知識に基づいた知識表現の枠組みを取入れることである．つまり，従来のデータ構造は，スロット内のファセット（属性情報）の枠組みが固定であるのに対し，STROBE[4]における「スロット-ファセット」関係のように，スロット内の属性情報を自由に定義できる枠組みとすることである．

そこで，まずスロットのデータ構造の基本を「スロット名-スロットタイプ-スロット値」とし，初期値やデーモン機能などを必要に応じて自由に定義できるようにした．さらに，スロット値のみを管理するデーモン機能が定義できれば，スロット値の評価前にこのデーモン機能を起動し，スロット値内で記述されている変数が決定できる．これにより，スロット値に論理演算式なども記述できるようになり，KBS上でDBSの機能を生成することが可能であると考えられる．

本研究は，上記の考え方を基に，機械工学分野の知識ベース化に適合したデーモン機能の拡張，それに付随するシステムスロット（システムが管理するスロット）の拡張，およびその具現化のための関数群などに

* 原稿受付　平成3年10月3日.
*1 正員，埼玉大学理工学研究科(〒338 浦和市下大久保255).
*2 正員，埼玉大学工学部.

ついて述べる．

2. デーモン機能とシステムスロットの拡張

2・1 機械工学分野の知識ベース化に適合したデーモン機能　デーモン機能は，スロット値を参照，書込み，あるいは削除時に自動的に起動する付加手続きをいい，知識そのもののモジュール性や可読性を高める特徴を持っている．従来の代表的なデーモン機能には，if-needed, if-added, if-removed などがある．これらの機能は，スロット値内を直接管理するための機能を持ち合わせていない．そこで，以下のような機能拡張を行った上で，スロット値内を直接管理できるよう試みた．

スロット値として数式が扱える：機械工学分野には，多くの実験式や公式が混在している．スロット値にこれらの数式が記述できれば，知識の整理に役立つ．そのためには，数式内で使用されている変数が，あらかじめ評価されるような機能をデーモン機能に付加する．

DBS と同等な機能が扱える：DBS のレコード単位の情報をフレーム表現単位で格納すると容量的な問題がある．そこで，大容量の知識を格納するためには，フレーム表現内のスロットに DBS のレコードに相当するような機能を持たせることが必要になる．そのためには，スロット値内のレコードとフィールドの関係をあらかじめ記述しておく．そして，データの参照が行われた際，その関係を用いて評価を行う機能を付加すれば良い．

2・2 デーモン機能の拡張　図1は，前節の機能を実現するために新しく付加したデーモン if-varied 機能の概念図である．このデーモン機能は以下の形式で記述する．

((変数1　付加手続き1)　(変数2‥)　‥)

ここで，リスト内の変数1はスロット値内で記述されている変数名であり，付加手続き1の評価結果がこの変数1に結合される．もし，この付加手続き1が省略されている場合は，この変数1は ISA 継承関係上のスロット名を示している．

このデーモン if-varied 機能は，スロット値の参照(メッセージ交信) がある［図1中(1)］と，まず付加手続き1での評価が行われ［図1中(2)］，その評価結果が変数1に結合される［図1中(3)］．リスト内すべての変数が評価された後，これらの変数を用いてスロット値の評価が行われ［図1中(4)］，その結果を返す［図1中(5)］という一連の動作を行っている．

従来のデーモン機能を用いた表現では，スロット値に記述すべき数式が，付加手続きの関数内(Attached-Procedure Functions) で記述されているため，そこで使用されている数式や呼び出されているスロットが理解できず，スロット値の追加変更が容易に行えない．これに対し，デーモン if-varied 機能では，上記一連の動作により，スロット値として数式が扱え，かつ数式内で使用されている変数が形式((変数1付加手続き1)‥)内に記述されているため，変数1などにスロッ

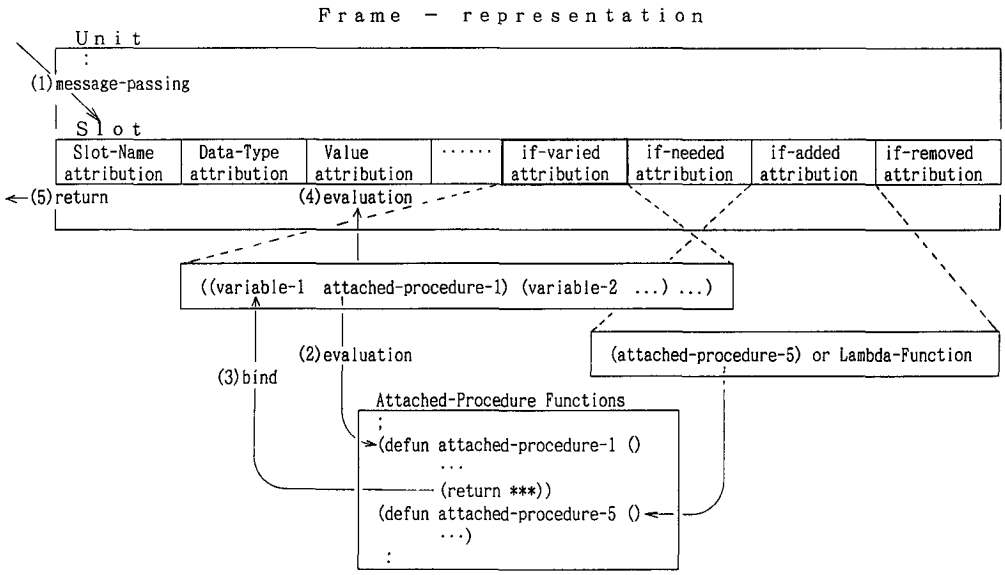

図1　デーモン (if-varied) 機能の概念図

ト名を記述すれば，使用されているスロットが理解できスロット値の追加変更が容易になる．

このように，この機能拡張により，2･4節で後述するように，スロット値に数式をそのまま記述でき，かつそこで記述されている変数をデーモン機能内で評価されることとなる．これにより，知識ベースの構築性や追加修正が容易になる．

2･3　システムスロットの拡張　図2は，DBSと同等な機能をKBS上に実現するために拡張したシステムスロットCluster-Unitsの概念を示したものである．Cluster-Unitsは，DBSの「レコード-フィールド」関係を管理するもので，フレーム表現(ユニット)内のスロット構成が全く同じ場合，複数ユニットを一つのユニットで表現し，無だなユニットを削減する．

図は，「メートル並目ねじの基準寸法」の選定に適用した例である．図中について，

（a）　はデータベースのレコード表現による例

（b）　は知識ベースのフレーム表現による例

（c）　はCluster-Unitsを用いた知識ベースのフレーム表現による例

である．(a)と(b)は従来の表現法であるが，著者が提唱する(c)の表現は(a)と(b)のおのおのの特徴を生かした表現となっている．

例えば，フレーム表現内のスロットは，データベースのレコードに相当し，フィールドを取出すためのヘッダが同一フレーム内のスロット値に記述されている．また，システムスロットCluster-Unitsには，図中(b)のユニット名がリスト形式で格納されている．このユニット名リストは，システム内の大域変数（＊Cunits＊）に登録される．

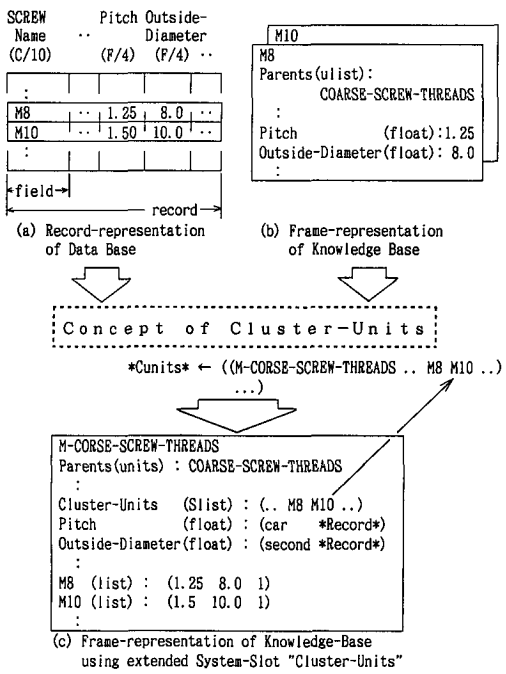

図2　システムスロット（Cluster-Units）の概念

このような知識表現下で，スロット値を参照（メッセージ交信)する場合の探索経路について追ってみる．まず，メートル並目ねじ「M8のピッチ」を参照するには以下のメッセージ-1を送る．なお，関数MGR-GET-VALUEについては2･4節で後述する．

メッセージ-1：（MGR-GET-VALUE 'M8 'Pitch) このメッセージ交信では，まずユニット「M8」を探索する．しかし，該当するユニットが存在しないと，Cluster-Unitsの形式で登録されているかについて，大域変数（＊Cunits＊）内を探索する．登録されている場合は，システムスロット機能として，以下の探索が続行される．

最初に，大域変数（＊Record＊）より実在するユニットを認識しメッセージ-2を送り，DBSのレコード単位の情報（スロット値）を大域変数（＊Record＊）に結合する．次に，メッセージ-3を送り，評価値として最終的に1.25という値を返している．

メッセージ-2：

(MGR-GET-VALUE 'M-CORSE-SCREW-THREADS 'M8)

メッセージ-3：

(MGR-GET-VALUE ＊ 'M-CORSE-SCREW-THREADS 'Pitch)

このように，前節のデーモン機能の拡張とこのシステムスロットとの拡張により，KBS上にDBSと同等な機能が実現でき，KBS上で大容量の知識を可能にしている．また，従来の知識ベースと同様なメッセージ交信でスロット値を参照することができることも特徴の一つである．

2･4　数式の記述　図3は，数式の記述例について，拡張したデーモン機能を用いた場合と従来のフレーム表現による場合とを示したものである．

記述例として，「ねじの有効断面積」に関する数式を取り上げる．これは，JIS　B　1082により以下の式で規定されている．

$$A_s = 0.7854 * (d - 0.9382 * P)^2$$

A_s：メートルねじの有効断面積　mm^2

d：おねじの外径　mm

P：ねじのピッチ　mm

図3(a)は，拡張したデーモン機能を用いて，上記

の数式をそのままスロット値に記述している［点線枠(1)］．付言しておくと，使用される数式は四則演算のほかに，多項式や微積分なども記述することができ，不足する演算式があるならライブラリに登録することにより，機械工学分野で使用されている数式を扱うことが可能である．特に，C言語やFortran言語など，手続き形言語により開発された数式も扱える．

これに対し，図3(b)では，従来のフレーム表現で数式を扱った例である［関数群の枠(1)］．従来の表現では，機械工学分野で多く用いられる数式を付加手続きとしてスロット値に関数名を記述するか，ラムダ式で直接記述する．そのため，知識ベースを構築するには面倒なプログラム作成が要求される．これが，機械工学分野の知識ベース化が遅れている大きな理由の一つであろう．

他の記述例として，「検索条件」の論理演算子を用いた式［点線枠(2)］がある．これは，「ピッチが1.25以上で，外径が9.00以下」という条件下に該当する「ねじ」を検索する例である．この例でもわかるように，スロット値内でDBSの検索機能と同等な機能を簡単に記述できる．

このように，デーモン機能の拡張により，スロット値に数式やDBSのための論理演算式の記述が可能になり，フレーム表現内ですべてが記述され，知識ベースの記述性や構築性に有効な機能であることがわかる．

3. デーモン機能の具現化のための関数群と評価

3・1 デーモン機能の実現方法　図4は，スロット値を参照する関数MGR-GET-VALUEの具体的内容を示したものである．これらの関数はKyoto Common Lisp (KCL) で構築されている．KCLはC言語で開発された言語で，C言語との結合性に優れているため，C言語を介してFortran言語などとも結合されている．そのため，2・2節で述べたスロット値内の数式において，豊富な演算式が使用できる．

図中のMGR関数群（[MGR-」で始まる関数）はユーザが自由に変更できる関数を示し，COR関数群（「COR-」で始まる関数）はシステム側が管理している関数を示している．このMGR関数群は，点線枠の例でもわかるように，COR関数群と定義済みのMGR関数群を用いて記述される．つまり，ユーザはMGR関数群を自由に定義することにより，必要に応じてスロット内のデータ構造やそれに伴う機能の変更を容易に行える．例えば，図中の点線枠(1)はシステムスロットの拡張部分である．この拡張部分は，関数MGR-CHK-CUNITSにより，図2のCluster-Unitsとして登録されているか否かを判断し，2・3節で述べたような機能が実現される．また，図中の点線枠(2)は，デーモン機能の拡張部分である．

図5は，わかりやすくするため，図4のスロット値

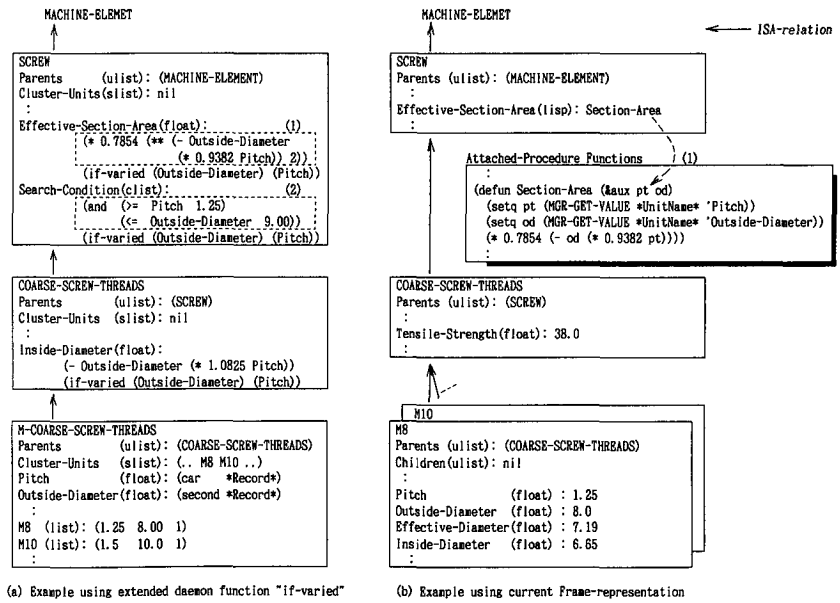

図3　拡張デーモン機能を用いた場合と従来のフレーム表現の場合（数式の記述例）

を参照する関数の本体部分の関数 MGR-GET-VALUE＊の概略フローを示したものである．この関数は，スロットの参照部と，スロットが存在した場合の評価部から構成されている．スロットの参照部は，関数 COR-GET-SLOT によって実行される．また，スロットの評価部は，以下の順で実行される．

まず，デーモン if-needed 機能を示す付加手続きが存在すれば，付加手続きの評価を行う関数 COR-EVL-FUNCTION を実行する．次に，デーモン if-varied 機能を示すリスト形式が存在する場合は，2･2 節で述べた機能が実行される[図 4 の点線枠(2)]．ここでは，この形式内のすべての付加手続きが評価され，対応する変数に結合される．しかし，もし変数のみで付加手続きが存在しない場合，この変数をスロット名と解釈し関数 MGR-GET-VALUE を再帰的に実行し，変数に結合している．最後に，スロット値が数式である場合は，上記の変数を用いて評価が行われる．最終的に，関数 MGR-GET-VALUE の評価値として返される．

```
; 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)
```

図 4　関数（MGR-GET-VALUE）の具体的内容

このように，デーモン機能の拡張とシステムスロットの拡張は，MGR 関数内を変更することにより実現されている．また，この MGR 関数群は，システムによってその依存関係が管理されており，知識ベース内のデータ構造やシステムスロットの呼び出し機能が，矛盾を生じないように関数間の関係が維持されている．

3･2　デーモン機能の構築性の評価　図 6 は，例として「メートル並目ねじの基準寸法（JIS B 0205）」の選定を構築した場合のメモリ容量について，図 4 に示した拡張デーモン機能を用いた場合(a)と，従来のフレーム表現による場合(b)とを比較したものである．(b)では，数式の記述ができないために付加手続きの関数名で記述されている．図の横軸は，フレーム理論における「抽象-具体」関係の具体（実体）に相当する数で，JIS B 0205 の場合は，そこに記載されている部品数が該当する．部品数が零の状態は，具体的な（実体）部品が零であることを意味する．つ

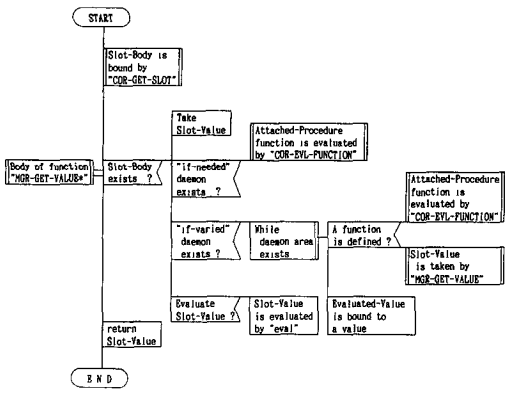

図 5　関数（MGR-GET-VALUE＊）本体の概略フロー

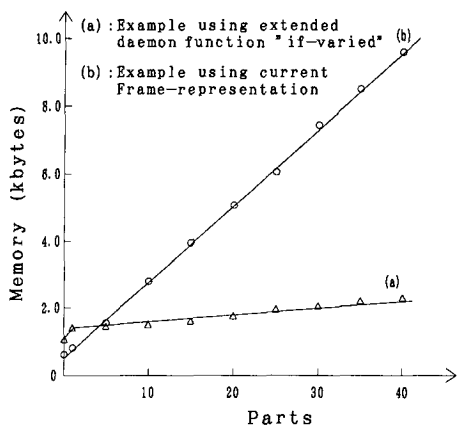

図 6　知識ベースを構築した場合のメモリ容量

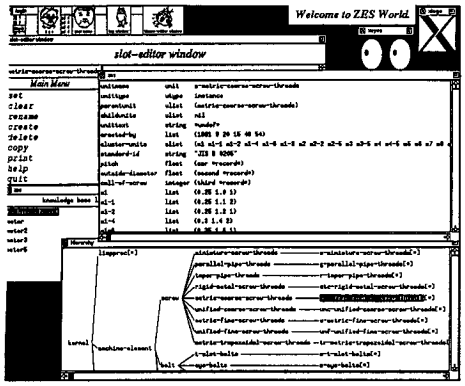

図 7　拡張デーモン機能を用いたウィンドウの例

まり，図3に示した「並目ねじ（COARSE-SCREW-THREADS)」フレーム表現以上のメモリ容量である．また，(a)のメモリ容量が大きいのは，ねじの有効断面積や内径などの数式が記述されているためである．

さて，メモリ増加の要因は，(a)ではスロット数，(b)ではフレーム数である．フレーム表現内には，継承関係などの情報も含まれており，それらの情報量の差が直線部のこう配の差となって現れている．つまり，

(a)のメモリ容量＝$A1$＊品目数＋$A2$

(b)のメモリ容量＝$B1$＊品目数＋$B2$

$A1$：スロット情報の容量　bytes

$A2$：拡張デーモン機能を用いた数式を含んだ容量　bytes

$B1$：フレーム付属情報の容量　bytes

$B2$：従来表現の容量　bytes

と表される．図6により諸数値を逆算してみると，

$A1=26, A2=1\,300$

$B1=230, B2=570$

となる．ここで，機械要素の一つであるボルトは，JIS規格以外を含めると数万種存在するといわれる．そこで，1万種のボルトを知識ベースに格納する場合を想定すると，上式に従えば，従来のフレーム表現では約2.3 Mbytes必要するのに対し，拡張デーモン機能を用いた表現では約0.3 Mbytesのメモリ容量で済むことがわかる．つまり，従来の表現では現在のエンジニアリング・ワークステーション（EWS）でも容量的に不可能な状態である．これに対し，拡張デーモン機能を用いれば知識ベース化が十分可能であることがわかる．

このように，拡張したデーモン機能は，知識ベースの容量性にかなり有効な機能であることがわかる．また，探索時間の短縮などにも大きな効果が期待できる．

4. 結　　言

本研究は，デーモン機能 if-varied の拡張と，これに伴うシステムスロット Cluster-Units の拡張を行い，機械工学分野の数式をも含めた豊富な知識を知識ベース化する一助とした．ここで提案した拡張機能は，JIS 規格のように既に固定化された知識をいかに容量的に効率よく知識ベース化していくか，また，数式をいかに有効的にかつ容易に記述していくかについて重要な知見を与えるものと期待される．ちなみに，図7はこれらの拡張機能を利用し，3・2節で検討した知識ベース（JIS B 0205）部分[6]の例を示したものである．図中の右上のウィンドウ部はフレーム表現内部を示しており，下のウィンドウ部は知識ベースの階層構造を表している．ここで，本研究をまとめてみると以下のようになる．

（1）　デーモン機能の拡張により，スロット値に数式をそのまま記述できる．そのため，フレーム表現内部の記述性や構築性を高めることができる．

（2）　システムスロットの拡張により，大容量の知識を小容量に，かつ簡潔に表現できることを可能にする．

（3）　上記の二つの拡張により，KBS 上に DBS と同等な機能を実現することを可能とした．これにより，構築性や記述性の優れている KBS が実現でき，3・2節で検討したようにその効果は十分に期待できる．

文　　献

（1）　滝沢，人工知能学会誌，2 (1987)，184.
（2）　長坂・大滝・石川，機論，58-547，C(1992)，975.
（3）　KEE Software Development System User's Manual, (1986), IntelliCorp.
（4）　Smith, Reid G., *IJCAI*, (1983), 855.
（5）　上野，知識工学入門，(1985)，84-187，オーム社.
（6）　JIS ハンドブック「ねじ」，(1991)，117，日本規格協会.
（7）　馬場，機械工学必携（第7版），(1986)，325，三省堂.