知識工学と人工知能　46－4
（１９８６．５．９）

フレームモデルによる構造の記述と対象の理解

中間　正人　　上野　晴樹

カシオ計算機（株）　東京電機大学　理工学部

フレーム型知識表現言語ＺＥＲＯの応用のひとつとして設計型問題における設計対象のモデル表現と対象の理解があげられる。本稿はモデルを機能と構造で明確に記述できる分野、特に部品の組み合わせによって構築されうるモデル（たとえば、デジタル回路や自動車など）に対する拡張フレームモデルによる構造の記述方法とこれを用いた対象理解の基本思想について報告する。

一般に構造は構成要素の列挙（ＰＡＲＴ－ＯＦ関係）と構成要素間の関係（ＮＥＴＷＯＲＫ関係）によって記述される。本研究ではＮＥＴＷＯＲＫ関係をフレーム内のＣＬＡＵＳＥで表現する事で構造の記述をＩＳ－Ａ階層と融合させることを示すとともに、このＣＬＡＵＳＥを用いた対象理解の基本思想ついて述べる。また、これに伴うＺＥＲＯ言語の拡張についても述べる。

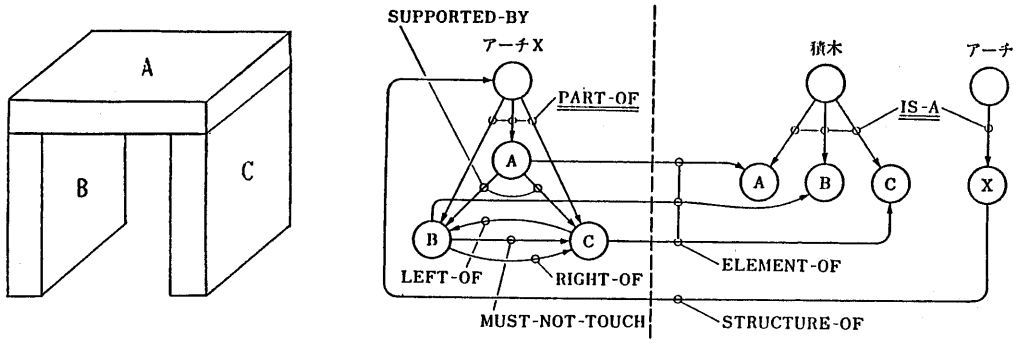
Description of Structure and Understanding of Objects by Means of Frame Based Model

Masato Nakama (Casio Computer Co. Sakae-chou 3-2-1, Hamura-machi, Tokyo, Japan, 190-11),
Ueno Haruki (Tokyo Denki Univ.)

In this paper, We describe a new method on how to describe a structure of a structure object in ZERO, a frame-based knowledge representation language, and to understand the object. This method intends to manipulate the knowledge about a structure and a behavior of the object according to the concept of an object model. The structure is described by means of a combination of PART-OF and network relations which is represented in a set of PROLOG clauses within the frame formalism, and the understanding is done by a bottom-up approach by interpriting the clauses.

1．構造の記述について

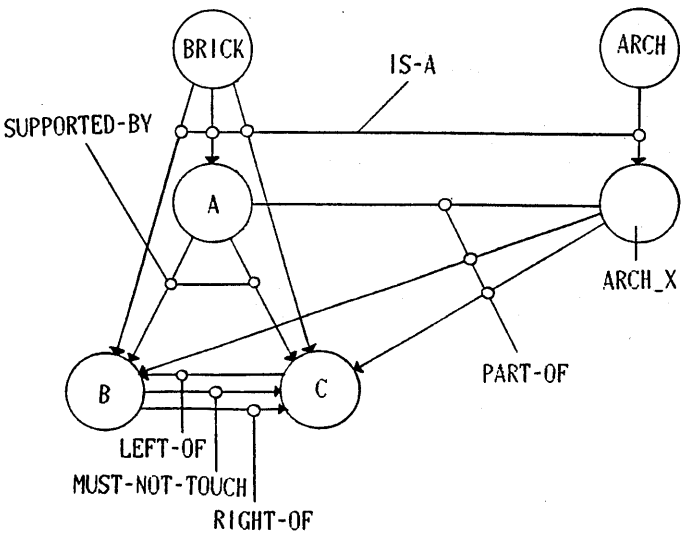
フレームモデルにおける構造の記述は一般にｐａｒｔ－ｏｆ関係とｎｅｔｗｏｒｋ関係によって表現される。［5］これは構造の記述が一般に構成要素の列挙とその構成要素間の関係によってなされるためで［4］、ｐａｒｔ－ｏｆ関係が構成要素の列挙に対応し、ｎｅｔｗｏｒｋ関係が構成要素間の関係に対応する。ｆｉｇ．1に従来のＺＥＲＯ言語［1，2，3］による構造物の表現例（積木の橋）［5］を示す。



ｆｉｇ．1　従来のＺＥＲＯ言語による構造物の表現例（フレーム図）

ここでｐａｒｔ－ｏｆ関係とｎｅｔｗｏｒｋ関係の性質の違いについて考えてみる。従来のＺＥＲＯ言語においては、共にスロットタイプがｆｒａｍｅであるスロットを用いて表現され、そのスロット名はその関係リンクの意味をあらわす。従って、２種類の関係の間にはその表現形式による区別は存在しない。しかしながら、ｐａｒｔ－ｏｆ関係はその性質上、全体と部分に識別されｈｉｅａｒａｃｈｙを形成する。又、全体から見ると部分は自分自身の一部分を表現している。一方、ｎｅｔｗｏｒｋ関係では各フレームは対等であり、そのフレーム自身からみると自分自身と外界との関係を表現したものである。ｆｉｇ．1でもわかるとおり、従来のＺＥＲＯ言語ではこのｐａｒｔ－ｏｆ関係とｎｅｔｗｏｒｋ関係を表現するのにプロトタイプフレームを定義し、構造を表現させていた。そのため、目的に応じて多数のプロトタイプフレームを用意する必要がある。しかし、インヘリタンスはｉｓ－ａ階層における属性遺伝の機能であり、フレーム自身と外界との状態を表現する動的な関係であるｎｅｔｗｏｒｋ関係には静的な機能であるインヘリタンスはなじまない。つまり、ｎｅｔｗｏｒｋ関係に関するスロットは各インスタンスフレームにおいて定義されるものであり、そのためにプロトタイプフレームを用意する必要はないと思われる。又、ｎｅｔｗｏｒｋ関係の設定はそのフレームがどの構造物の要素であるかによって決まるものだから、その管理はｐａｒｔ－ｏｆ階層によって行なわれるべきものである。この考え方にそって、図1の例を表現したものをｆｉｇ．2に示す。

さらに、従来のＺＥＲＯ言語では構成要素間の関係を図1のようにｐａｒｔ－ｏｆ階層の下位フレーム間のネットワークリンクで記述することによって表現している。従って、あるフレームが表現している概念の構造を説明する機能を持つ解釈の知識を記述しようとすると注視点フレームだけはなくｐａｒｔ－ｏｆ階層の下位フレームからｎｅｔｗｏｒｋ



ｆｉｇ．2　拡張後のＺＥＲＯによる表現（フレーム図）

関係を抽出する機能が必要になる。従来のＺＥＲＯ言語では前述したとおりにｐａｒｔ－ｏｆ関係とｎｅｔｗｏｒｋ関係を区別できないので、各部分専用にｎｅｔｗｏｒｋ関係を抽出する手続きを用意する必要がある。このことはｎｅｔｗｏｒｋ関係についての管理をユーザーに押しつけるだけでなく、フレーム自身に関する情報であるにもかかわらず、その情報は他の複数のフレームにまたがって存在することになり、ユーザーとって把握しずらいだけでなく、そのフレーム構造を知らなければ手続きを記述することはできない。又、ｎｅｔｗｏｒｋ関係自身は構造物についての制約条件として考えられるので、その概念フレームにおいて内包的に表現されるべきである。

そこで、この問題を解決するためにｐａｒｔ－ｏｆ階層の上位フレームにそのフレームの構造を記述するスロットｓｔｒｕｃｔｕｒｅを設ける。スロットｓｔｒｕｃｔｕｒｅは、ｎｅｔｗｏｒｋ関係が注視点フレームのｐａｒｔ－ｏｆ関係を表わすスロット間でも成り立つことに着目して、このｎｅｔｗｏｒｋをｃｌａｕｓｅの形で記述したものである。このとき、ｃｌａｕｓｅ中の述語はｎｅｔｗｏｒｋ関係を表わすスロット名であり、引数はｐａｒｔ－ｏｆ関係を表わすスロットのｖａｌｕｅで且つｐａｒｔ－ｏｆ階層の下位フレームを表わす。そして、従来のｎｅｔｗｏｒｋ関係をあらわすスロットは設計過程におけるそのフレームに対する操作の履歴を表わすのでそのまま残す。但し、ｐａｒｔ－ｏｆ関係が一意に定まっている時にはスロットを隠匿する。つまり、スロットｓｔｒｕｃｔｕｒｅは今までｎｅｔｗｏｒｋ関係を表わすスロットが持っていたふたつの意味を分離し、構成要素間の関係について同一フレーム内で取り扱えるようにしたものである。又、ｓｕｂｃｌａｓｓレベルでスロットｓｔｒｕｃｔｕｒｅを導入しようとすると、ｐａｒｔ－ｏｆ関係を表わすスロットのｖａｌｕｅが未定のときが多いので、ｃｌａｕｓｅが記述できなくなる。そのため、フレーム内を有効範囲とする大域変数を導入し、未定なｖａｌｕｅの部分を変数で仮定することによってｃｌａｕｓｅを記述する。（ｆｉｇ．３）

```
FRAME  :  ARCH     TYPE CLASS
          a-kind-of  U  frame   ZERO
          ddescendent U  filst   (ARCH_X)
          part-of    U  frame   not-defined
          girder     SP flist   (*W *Y)
          board      SP frame   *Z
          location   U  string  not-defined
          length     U  number  not-defined
          structure  SP prolog (? (structure (*W *Y *Z)))
                               ( (structure (*W *Y *Z))
                                 (supported-by *Z (*W *Y))
                                 (left-of *W *Y)  (not-touch *W *Y) )
          behaviour  S  lisp    (ARCH-behaviour)

FRAME  :  ARCH_X   TYPE INSTANCE
          a-kind-of  U  frame   ARCH
          ddescendent U  filst   nil
          part-of    U  frame   not-defined
          girder     SP flist   (B C)
          board      SP frame   A
          location   U  string  TOKYO-YOKOTA
          length     U  number  100
          structure  SP prolog  (? (structure ( B  C  A)))
                                ( (structure ( B  C  A))
                                  (supported-by  A ( B  C) )
                                  (left-of  B  C) (not-touch  B  C))
          behaviour  S  lisp    (ARCH-behaviour)
```

ｆｉｇ．３　拡張後のＺＥＲＯ言語によるクラスＡＲＣＨとインスタンスＡＲＣＨ＿Ｘの表現

```
FRAME : BRICK      TYPE CLASS
 a-kind-of  U  frame    ZERO
ddescendent U  filst    (A B C)
 part-of    U  frame    not-defined
 hight      U  number   not-defined
 width      U  number   not-defined
 length     U  number   not-defined
 behaviour  S  lisp     (BRICK-behaviour)
```

```
FRAME : B          TYPE INSTANCE
 a-kind-of  U  frame    BRICK
ddescendent U  filst    nil
 part-of    U  frame    ARCH_X
 hight      U  number   50
 width      U  number   20
 length     U  number   100
 behaviour  S  lisp     (BRICK-behaviour)
 left-of    I  relation C
 not-touch  I  relation C
 support    I  relation A
```

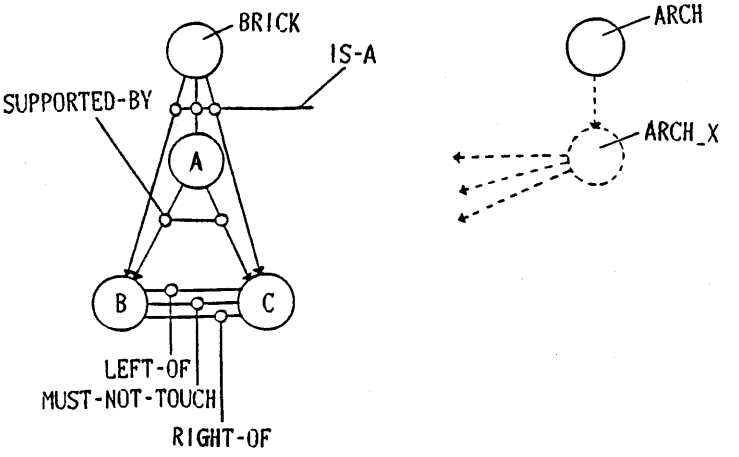
```
FRAME : A          TYPE INSTANCE
 a-kind-of  U  frame    BRICK
ddescendent U  filst    nil
 part-of    U  frame    ARCH_X
 hight      U  number   50
 width      U  number   20
 length     U  number   100
 behaviour  S  lisp     (BRICK-behaviour)
 supported-by I  relation (B C)
```

```
FRAME : C          TYPE INSTANCE
 a-kind-of  U  frame    BRICK
ddescendent U  filst    nil
 part-of    U  frame    ARCH_X
 hight      U  number   50
 width      U  number   20
 length     U  number   100
 behaviour  S  lisp     (BRICK-behaviour)
 right-of   I  relation B
 not-touch  I  relation B
 support    I  relation A
```

ｆｉｇ．４　拡張後のＺＥＲＯによるクラスＢＲＩＣＫとインスタンスＡ，Ｂ，Ｃ

２．対象の理解手法

ここで言う対象の理解はボトムアップな設計プロセスにおいて、ユーザーが部品を組み合わせる形でｎｅｔｗｏｒｋ関係を定義している時に、ｐａｒｔ－ｏｆ階層が上位であるインスタンスフレームを自動生成することと等価である。そこで、ｐａｒｔ－ｏｆ階層の上位のインスタンスフレームを生成すべき状態かを考えてみると、ｆｉｇ．５のようになる。



ｆｉｇ．５　PART-OF階層のフレームが生成されるべき状態

このとき、フレームA，B，Cで定義されるｎｅｔｗｏｒｋ関係を述語の形で表現するとｆｉｇ．６のようになる。

```
ｓｕｐｐｏｒｔｅｄ－ｂｙ（A，［B，C］）．
ｓｕｐｐｏｒｔ（B，A）．
ｓｕｐｐｏｒｔ（C，A）．
ｌｅｆｔ－ｏｆ（B，C）．
ｒｉｇｈｔ－ｏｆ（C，B）．
ｎｏｔ－ｔｏｕｔｃｈ（B，C）．
ｎｏｔ－ｔｏｕｔｃｈ（C，B）．
```

ｆｉｇ．６　ｆｉｇ．５におけるｎｅｔｗｏｒｋ関係のａｓｓｅｒｔｉｏｎ

ここで、このａｓｓｅｒｔｉｏｎをデータベースとする空間でｆｉｇ．７のようなゴール節を起動するとフレームＡＲＣＨのスロットｓｔｒｕｃｔｕｒｅのｃｌａｕｓｅがｍｅｓｓａｇｅ　ｐａｓｓｉｎｇによって起動し、この空間上でパターンマッチを行なう。この例では、ｃｌａｕｓｅは成立し、変数＊Ａには（Ｂ　Ｃ　Ａ）というリストがバインドされてくる。つまり、これは変数＊Ｗ，＊Ｙ，＊Ｚの値であり、ｐａｒｔ－ｏｆ階層を表わすスロットｇｉｒｄｅｒ，ｂｏａｒｄの値である。従って、次の述語であるＭＡＫＥＦＲＡＭＥでインスタンスフレームＡＲＣＨ＿Ｘが生成される。又、スロットｓｔｒｕｃｔｕｒｅのｃｌａｕｓｅが成功しなければ、当然、インスタンスフレームは生成されない。

```
?－ｍａｓｓａｇｅｐ（ｓｔｒｕｃｔｕｒｅ，ＡＲＣＨ，＊Ａ），！，
        ｍａｋｅｆｒａｍｅ（ＡＲＣＨ，ＡＲＣＨ＿Ｘ，＊Ａ）．
```

ｆｉｇ．７　ＡＲＣＨ＿Ｘを理解するｇｏａｌ節

対象理解のシナリオ

ユーザーはシステムに用意された部品を選択して、部品の細部の仕様を与える。そして、部品同士の組み合わせをｎｅｔｗｏｒｋ関係を定義することによって与える。このとき、システムはｎｅｔｗｏｒｋ関係が定義される度に、前述したゴール節を複数起動し、ｐａｒｔ－ｏｆ階層の上位フレームを生成するかどうかを判断していく。

このときのシステムの動きを表現したフレームをｆｉｇ．８に示す。このとき、ｐｒｏｌｏｇのＤＢの空間は前述した通り、ｎｅｔｗｏｒｋ関係のａｓｓｅｒｔｉｏｎの集合だけでよい。しかし、今までのＺＥＲＯ言語ではｎｅｔｗｏｒｋ関係を識別する方法はないので、フレームシステムすべてのａｓｓｅｒｔｉｏｎをＤＢとしてパターンマッチしてしまう。そこで、ｎｅｔｗｏｒｋ関係を識別する為の新しいタイプｒｅｌａｔｉｏｎを導入する。タイプｒｅｌａｔｉｏｎの定義は次の通りである。

＊ｎｅｔｗｏｒｋ関係を表現するスロットタイプでタイプｆｒａｍｅおよびｆｌｉｓｔの特殊形。

又、このフレームは１つの構造物について理解する汎用手法を記述してあり、（deductiveスロット）これを利用しながら制御するかたちで全体の理解を進める手続きを記述することによって（u-moniotorがこれに当たる）理解を進める。従って、個々の構造物を理解するためのフレームをフレームｕｎｄｅｓｔａｎｄｉｎｇのインスタンスとして実現する。また、このことにより、この手法の理解能力はこのインスタンスフレームの数とＵ－ｍｏｎｉｔｏｒの制御能力に左右される。尚、ここに示したＵ－ｍｏｎｉｔｏｒは多層構造になっているモデルには対応していないが、これを基本に下の層から上の層に適応して行くことにより多層構造にも対応できると思われる。

```
        FRAME : UNDERSTANDING          TYPE  :  CLASS
 a-kind-of          U       frame      ZERO
ddescendent         U       flist      (U-ARCH U-DISK U-CHAIR .....)
 target             SP      frame      *Z
 deductive          SP      prolog     (? (undersanding *Z))
                                       ( (understanding *Z)
                                            (messagep 'structure *Z *Y) (cut)
                                            (hashname *W) (makeframe *Z *W *Y) )
 understanding      I       lisp       (U-monitor)

 (defun  U-monitor ()
      (prog (*A *B)
                 (setq *A (getval# *ddescendent 'understanding))
            loop (cond ((null *A) (return nil))
                       ((messagep# 'deductive (car *A) *B) (return t))
                       (setq *A (cdr *A))  (go loop) ) )

        FRAME : U-ARCH                   TYPE  : INSTANCE
 a-kind-of          U       frame      UNDERSTANDING
ddescendent         U       flist      nil
 target             SP      frame      ARCH
 deductive          SP      prolog     (? (undersanding  ARCH))
                                       ((understanding  ARCH)
                                            (messagep 'structure  ARCH *Y) (cut)
                                            (hashname *W) (makeframe  ARCH *W *Y))
```

ｆｉｇ．８　理解の知識を表現したフレーム

３．知識表現言語ＺＥＲＯの拡張*

１、２で述べたことを実現させるために次の点を拡張する必要がある。

（１）スロットタイプｆｒａｍｅ及びｆｌｉｓｔの未定義ｖａｌｕｅを表わすためのフレーム内を有効範囲とする大域変数の導入

（２）ｎｅｔｗｏｒｋ関係を表わすスロットタイプｒｅｌａｔｉｏｎの導入

（３）ｎｅｔｗｏｒｋ関係の双対をあらわすｒｅｌａｔｉｏｎ　ｔａｂｌｅの導入

（４）（１）で述べた大域変数を含む場合のインヘリタンスロールＳＰの追加

（１）（２）については１，２で既に述べているので、ここでは（３）と（４）について説明する。

３．１ｒｅｌａｔｉｏｎ　ｔａｂｌｅの導入について

ここではスロットタイプｒｅｌａｔｉｏｎの具体的な性質からくる問題点について述べ、それを解決するためにｒｅｌａｔｉｏｎ　ｔａｂｌｅを導入する必要性を述べる。

スロットタイプｒｅｌａｔｉｏｎはｎｅｔｗｏｒｋ関係を記述する事は前に述べた。あるフレームにおいてｎｅｔｗｏｒｋ関係が設定されたとする。このとき、ｎｅｔｗｏｒｋ関係の相手方においても、双対のｎｅｔｗｏｒｋ関係が設定されたことになる。（ｆｉｇ．６）従って、一つのｎｅｔｗｏｒｋ関係を設定したとき、双対のｎｅｔｗｏｒｋ関係も必ず設定する必要がある。これは、一つのｎｅｔｗｏｒｋ関係を記述した時に双対になるスロット名をシステムが要求することで解決するであろう。しかし、状態が変化し、この双対のｎｅｔｗｏｒｋ関係が消滅したときはふたつのスロットを削除する必要がある。この種

---

*現在、ＺＥＲＯ言語自身は［１１］に基づく形で拡張がすすんでいるので注意されたい。

の管理は当然、デーモン機構で行なうべきである。しかし、現状のデーモン機構は全てユーザーの管理に任されており、その記述はｌｉｓｐ関数によって行なわれている。従って、スロットタイプｒｅｌａｔｉｏｎにすべてにその記述が必要になり、ユーザーに重い負担がかかることになる。そこで、この記述について考えてみる。その手続きはｉｆ－ｒｅｍｏｖｅｄデーモンで行なわれ、内容はｖａｌｕｅによって示されているフレームにある双対するスロットを削除するという汎化が可能なものであるから、ｌｉｓｐ関数をひとつ用意すれば済む。しかし、この時に問題になるのが双対するスロットの名前である。現状ではスロットの名前がリンクの意味を表わすが、名前その物が意味を表わすのではなく、単なるｉｄｅｎｔｆｉｅｒにすぎないので、ｌｅｆｔ－ｏｆとｒｉｇｈｔ－ｏｆのような双対関係はどこかに双対であることを記憶しておかなければシステムはわからない。そのため、ｒｅｌａｔｉｏｎ　ｔａｂｌｅに双対関係を記憶する必要がある。さらにこうすることにより、ユーザーは１回、双対関係をｒｅｌａｔｉｏｎ　ｔａｂｌｅに定義すれば次回からはシステムにいちいち要求されなくても済むなど、双対関係のｃｏｎｓｉｓｔｅｎｃｙの管理が行ない易くなる。

```
supported-by($a,[$x|$y]):-support($x,$a),surport($y,$a).
supported-by($a,$b):-support($b,$a).
support($a,$b):-supported-by($b,$y),member($a,$y).
support($a,$b):-supported-by($b,$a).
left-of($a,$b):-right-of($b,$a).
right-of($a,$b):-left-of($b,$a).
not-touch($a,$b):-not-touch($b,$a).
```

ｆｉｇ．９　ｒｅｌａｔｉｏｎ　ｔａｂｌｅの構造

ｒｅｌａｔｉｏｎ　ｔａｂｌｅの構造はｆｉｇ．９に示す通りであり、双対なｎｅｔｗｏｒｋ関係をｃｌａｕｓｅの形で記憶している。尚、この管理により双対なｎｅｔｗｏｒｋ関係に対する保証が得られるので、スロットｓｔｒｕｃｔｕｒｅにおいて双対なｎｅｔｗｏｒｋ関係はどちらか一方だけ書けばよい。

３．２　変数を含むインヘリタンスロールＳＰについて

ｓｕｂｃｌａｓｓレベルのフレームのスロットｓｔｒｕｃｔｕｒｅはｎｅｔｗｏｒｋ関係を一般化した定理を表現したものとして解釈できる。このようなフレームのインスタンスフレームを生成すると、定理を具体化したｎｅｔｗｏｒｋ関係が形成され、インスタンスにおけるスロットｓｔｒｕｃｔｕｒｅも具体化されなければならない。これは、大域変数に具体的な値を設定するだけであり、一種のマクロ展開である。尚、ＳＰの意味はｓｐｅｃｉａｌｉｚｅの略である。

４．　まとめ

この論文におけるモデルの領域は部品の組み合わせによって構成されているものである。従って、明確に分割できない構造を持つものに対しての考察は今後の課題である。しかし、現在のＣＡＤの適用領域の多くの製品は部品の組み立てによって製造されているので、対象を理解することで設計に対する支援が行ない易くなる利点は大きいと思われる。

尚、まだ机上の理論の段階なので、細かいところで問題点があると思われる。問題点に気づいた人は遠慮なく指摘して頂たい。

又、スロットｓｔｒｕｃｔｕｒｅのｃｌａｕｓｅはその中の述語を設計操作を抽象化したものとして解釈できるので、様々な事に活用できるとおもわれる。ここに２、３の例をあげておく。

利用例１

ユーザーは部品を組み合わせ、ある目的物（例えば、机など）を設計しているとする。一通り、設計が終わったとユーザーが判断したとき、目的物の抽象概念を表わすフレームを指定し、スロットｓｔｒｕｃｔｕｒｅを起動するとｃｌａｕｓｅが成立したときは検証されたことになるし、成立しなかったときは失敗した述語を示すことにより、行なっていない操作をアドバイスすることができる。

利用例２　　システム主導型設計支援

ｐａｒｔ－ｏｆ階層の最上位のフレームを選択し、スロットに変数を含まない時は値を要求し、変数を含む場合は下位フレームを生成し、そのフレームに対しても同様の動作を行なう。このとき、下位フレームを生成するにあたって、スロットｓｔｒｕｃｔｕｒｅが成立するようにタイプｒｅｌａｔｉｏｎのスロットを追加する。

さらに、現在は変数を限定した型のスロットのみに用いているが単に仮定の記述として捉えるとその適用領域は広がり、デーモンなどの記述が行ない易くなると思われる。この件については今後検討を加えて行きたいと思う。

謝辞

この研究は東京電機大学大学院在学中の経験［１２］に基づくものであり、この研究にあたり、議論をさせて頂いた日本情報処理開発協会の伊藤秀昭氏及び、論文執筆の機会を与えて下さった弊社福村正明次長に深く感謝いたします。

参考文献

［１］伊藤、上野：フレーム型知識表現言語ＦＭＳの構造について、情報処理学会
知識工学と人工知能研究会　３０－４　１９８３
［２］伊藤、上野：フレームモデルと述語論理の結合、情報処理学会
知識工学と人工知能研究会　３９－９　１９８５
［３］伊藤、上野：ＺＥＲＯ＝ＦＲＡＭＥ＋ＰＲＯＬＯＧ
ＬＯＧＩＣ　ＰＲＯＧＲＡＭＩＮＧ　ＣＯＮＦＥＲＥＮＣＥ’８５　４．２
［４］D.G.Bobrow: Qualitative Reasoning about Physical System;An Introduction
ＡＲＴＩＦＩＣＡＬ　ＩＮＴＥＬＬＩＧＥＮＣＥ　１９８４．１２
［５］上野　　：知識工学入門　オーム社　１９８５
［６］R.Kowalski ：Logic for Data Description
Logic and Database PP77-103 ＰＬＥＮＵＭ　ＰＲＥＳＳ
［７］R.G.Smith,P.Friedland,M.Stfik ：UNIT PACKAGE USER'S GUIDE
ＨＰＰ－８０－２０　ＳＴＡＮＦＯＲＤ　ＵＮＩＶ．
［８］国分、古谷、樋口、半田：意味記憶システムＩＸ　電総研
第２９回情報処理全国大会　３Ｎ－６，７，８，９
［９］久野、中島、他：設計支援システムにおける設計対象の階層的表現法
第２９回情報処理全国大会　７Ｎ－７
［１０］大須賀：コンピュータによるモデリング
精密機械　第５１巻　第６号　PP1149-1155
［１１］上野、他：対象モデルの概念に基づく階層的知識表現と推論制御の研究
文部省科学研究費　特定研究　多元知識情報報告書
［１２］中間：知的プログラミング支援システムの研究
－対象モデルの概念にもとづくプログラム表現の試み－
東京電機大学大学院修士論文　１９８５．３