Prologプログラミング環境の作成

沼尾　雅之　　藤崎　哲之助
日本アイ・ビー・エム株式会社　サイエンス・インスティチュート

１．はじめに

Smalltalk-80[1]に代表される対象指向型言語は、表現の素直さ、モジュラリティ及び、再利用性の高さ等から知識表現用言語及び、プロトタイプ作成用言語として注目されている。また、Flavor[2],ESP[3]のように、マルチ・ウィンドウやマルチ・プロセスなどのシステム記述言語としても重要である。

一方、この対象指向型言語をプログラミング環境作成用言語として用いることを考えると、ビットマップディスプレイの特徴を最大に生かした、マルチ・ウィンドウ、マウス、アイコン、メニューなどから構成される豊富なユーザ・インタフェース機能を、対象としているプログラミング環境として用いることが可能である。これは、Smalltalkにおけるクラスが、再利用性をよく考えて構成されているため、既存のクラスを利用し、そのサブクラスを定義することによって、自分向きの環境が容易に実現できるからである。また、オブジェクトとその間のメッセージパッシングという計算機構は、対象とするプログラミング言語のモデル化を容易にするとともに、実行過程の視覚化や制御に非常に適しているとおもわれる。

本稿では、Xerox 1100SIP上のSmalltalk-80を用いてPrologのプログラミング環境を作成したので、その概要を報告する。

PrologはAIシステム構築用言語として広く使われているが、現在、Prologのプログラム環境としてはまだ十分なものがあるとは言いがたい。ここでは、Prologの複雑な実行過程の視覚化の方法について説明するとともに、統合的環境を実現するための、マルチ・ウィンドウによる、エディタ、ブラウザ、実行モニタ等の構成方法及び、メッセージパッシングの特徴を生かしたモードレスオペレーションの手法について述べる。

２．　Prolog実行の特徴

Prologのプログラミング環境を設計するまえに、まずPrologの実行の特徴とプログラミングの手順を知らなければならない。プログラミングは従来の言語と同様に、プログラムの入力、実行、デバッグというサイクルで行なわれるわけであるが、特に問題なのは実行時のエラーであり、以下に挙げるようなPrologの特徴が、その実行をわかりにくいものにしている。

(a)　プログラムの実行順序がリニアでない。
プログラムの書かれた順番と実行順序は異なる。

(b)　バックトラックメカニズム。
どこでバックトラックが起こり、どこまで戻るかがわかりにくい。

(c)　変数のユニフィケーション。
特に変数どうしのユニフィケーションのために、ある変数に値がユニファイされるとそれが波及的に他の変数にも変化を与えるので、述語呼出しの際に変数にどんな値が入っていたのかわかりにくい。

このような、従来の言語とまったく異なる実行形態が、Prologのデバッグを困難なものにしている。現在あるようなbox model[4]に基ずいたデバッガのトレース出力では、ユニフィケーションについての十分な情報が得られないし、またあまりに大量の出力のなかで、必要な情報が探しきれないことが多い。特に端末を利用しての対話的デバッグをするためには、従来のデバッガでは不十分である。わかりやすい実行の視覚化の方法が望まれるわけである。文献[5]ではイオニックモデルというもので導出木の生成の過程を視覚化している。本システムの方法は以下に述べるように、プログラムの対応が明確になるような視覚化をしている。

３．Prologプログラミング環境の構成

３．１　画面構成

プログラムの入力、実行、修正というプログラム開発のサイクルは、全てのプログラム言語に共通のことであるが、これを通常のディスプレイ端末で行なうことを考えると、それぞれのフェーズで、エディタ、Prologインタプリタ、その中のトレーサというぐあいに、新しい環境に入りなおして、画面を切り換えたりしなければならなかった。このため画面を切り換える前にそのデータを紙に控えるなどの不便が往々にしてあった。こうした不都合は、マルチ・ウィンドウ・システムを用いることによって、大きく改善される。さらに、ビットマップ・ディスプレイでは任意の形状のアイコンを作れるので、ユーザの理解しやすいインタフェースを設計できることが期待される。

本システムはこのビットマップ・ディスプレイの利点を最大限にいかして、以下に示すような機能別のウィンドウと、実行過程表示用アイコンから構成されている。実行過程表示用アイコンは、Prologの実行をディスプレイ上でシミュレートするためのもので、これは本システムの最大の特徴の１つである。

これらは図１に示すように、すべてスクリーン上に配置されているので、ユーザはそれらを参照しながら仕事を進めることができる。

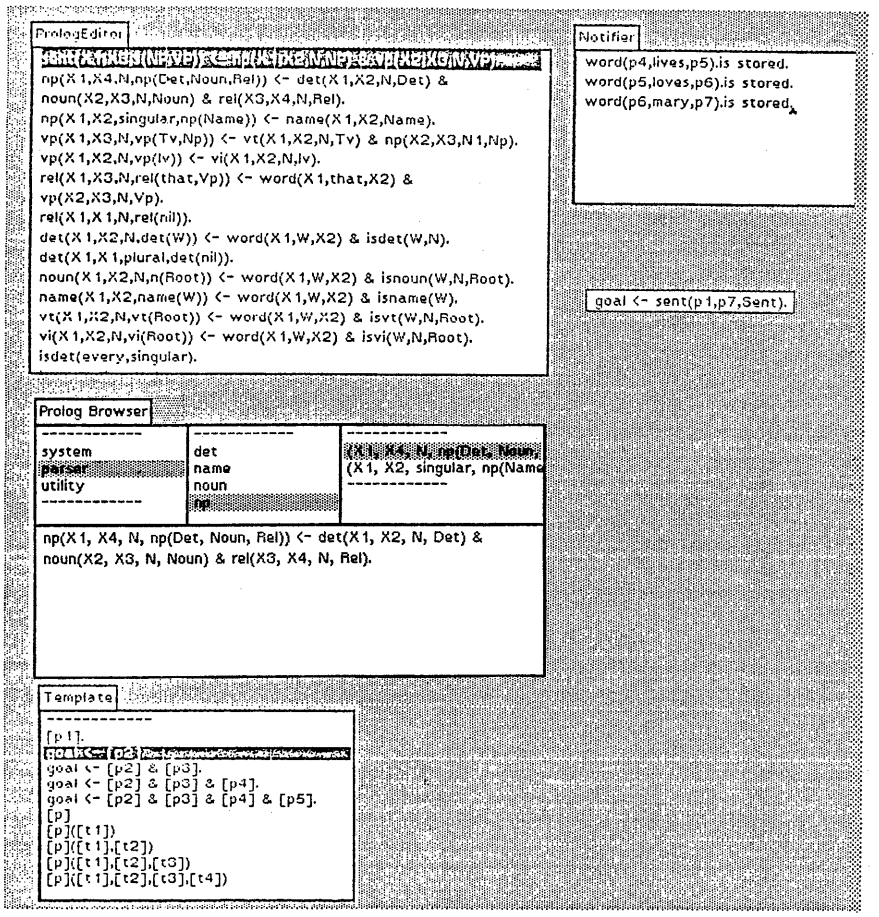

図１．スクリーン上のウィンドウの配置

ウィンドウ

(a) editor window

テキスト・エディタで、キーボードからのプログラム入力及び、外部ファイルとの入出力、通常のゴール実行をおこなう。SmalltalkのWorkspaceに対応している。

(b) browser window

システムまたは、ユーザによって定義された述語の管理、検索、修正を行なうためのデータベース。各述語はそのカテゴリー名、述語名、引数によって構造化されて登録される。SmalltalkのSystem Browserに対応している。

(c) template window

実行過程表示用アイコンのゴール節を生成するためのテンプレートを保持する。

(d) notifier window

システムからのメッセージや、述語writeの実行結果などの表示をする。SmalltalkのSystem Transcriptに対応する。

アイコン

(e) clause image

実行過程表示用のアイコンで、Prolog実行における内部表現のディスプレイ上の姿である。外見上はプログラムの字ずらの表現と同じである。これによって、実行にともなう導出木の展開、各サブ・ゴールの成否、変数の値の変化などを表示することができる。

このアイコンは、また、そのクローズの構造を正確に保持して、自分自身を構造化している。つまり、クローズが述語、述語が述語名と引数から構成されているように、アイコンは、クローズ、述語、関数、変数、定数を認識して、マウスのクリックにたいして反応する。例えば、あるアイコン中の述語をマウスでクリックしてコピーして、それを別のアイコンに代入することができる。これは、このアイコンが、構造エディタにもなっていることを意味している。

これらのウィンドウおよび、アイコンは相互に密接に関連している。例えば、template windowとbrowser windowから、ホーン節に対応したclause imageを作り出して、実行してみたり、修正したのち再びデータベース (browser window) に登録したりすることができる。また、 editor window 中のプログラムは、ロードすることによって、内部表現に変換されてデータベースに登録されるし、その逆であるデータベースから、 editor へのプログラム出力も可能である。こうしたオペレーションは全て、マウスによるメニューの選択ですることができ、ウィンドウ間のデータの移動もマウスによる cut&paste または copy&paste で行なわれる。

３．２　アイコンによる実行の視覚化

本システムでは、clause imageアイコンが重要な役割を持っている。つまり、このアイコンによって、Prologの実行がディスプレイ上に視覚化される。ここでは、以下のようにProlog実行の視覚化を実現している。

(a)　リゾリューションの表示

実行時のクローズのイメージを画面上にアイコンで表現し、実行にともなってボディのサブゴールがリゾリューションされていく過程を表示する。つまり、サブゴールと同じ述語名をヘッドにもつような候補クローズのアイコンを生成し、そのうちのどのクローズ が選択されたかなどを表示する。

(b)　サブゴールの状態表示

各サブゴールが実行中であるか、あるいは、成功したか失敗したかを表示する。ヘッドの述語にたいしては、ユニフィケーションが、成功したか失敗したかを表示する。

(c)　変数の値の表示

変数に付けられたモニターウィンドウ（変数デモン）によって、その変数の値を常に監視することができる。

画面上に生成されたアイコンは、ユーザが特に消さないかぎりはそのまま残っているので、実行が終わった段階で、そのゴールが成功または失敗に至るまでのリゾリューション木を見ることができる。また、アイコンは他のアイコンの上にも上書きできるので、画面をクリアする必要はない。

３．３　実行の制御機能

従来のデバッガ、トレーサの欠点の１つとして、コマンドの貧困なことが挙げられる。そのため、状況に応じたきめの細かい制御が不可能であったし、トレーサは表示するだけなので、バグを見つけてもまたエディタに入ってからでなくては、それを直せないなどの問題があった。本システムでは、以下に示すような豊富な制御が可能である。

(a)　実行開始点の決定
アイコンがクローズか述語であれば、任意の時点で、それを実行してみることができる。生成されたアイコンは画面上に残っているので、実行が終わったあとで適当なクローズを再実行してみることも可能である。

(b)　実行モードの設定
ステップ実行か、連続実行かの設定および、連続実行の場合の速度指定。

(c)　ブレークポイントの設定
述語名指定によるブレークポイントの設定および、変数デモンによるブレークポイントの指定が可能。

(d)　候補クローズの選択
リゾリューションの過程で現われた候補クローズの中から、どのクローズを選択するかを決定できる。

(e)　実行時のエディット
実行時に画面上に現われるクローズのアイコンは、構造エディタになっているので、述語、タームレベルでマウスによるcut&paste的なエディットが可能である。

３．４　実行例

図２(a-d)に、簡単な英文の構文解析プログラムの実行例を示す。

(a)　初期設定。
ゴール節を作り、変数Sentにデモンを付けたところ。ゴール節は、Templateウィンドウからゴールテンプレートをディスプレイ上にコピーして、その述語部分にBrauserウィンドウからsent述語をコピーして作る。変数デモンは、変数上でマウスをクリックして、メニューを選ぶことによって付けることができる。

(b)　実行過程１。
ゴール節のボディの述語sentと、それをヘッドにもつホーン節がユニファイして、さらにそのボディの１番目の述語であるnpを導出するための候補節が現われたところ。実行中の述語のアイコンは点滅で、またユニフィケーションの成功は、アウトラインで表示される。

(c)　実行過程２。
npの候補節の中から最初のものが、ユニフィケーションの成功によって選ばれ、そのボディの１番目の述語detは成功して、２番目の述語nounが実行されようとしているところ。述語の成功もアウトラインで示される。

(d)　実行過程３。
sentをヘッドに持つ述語の１番目の述語npが成功し、２番目の述語vpを実行しているところ。失敗した述語は黒白反転で、また失敗した節は灰色で表示される。

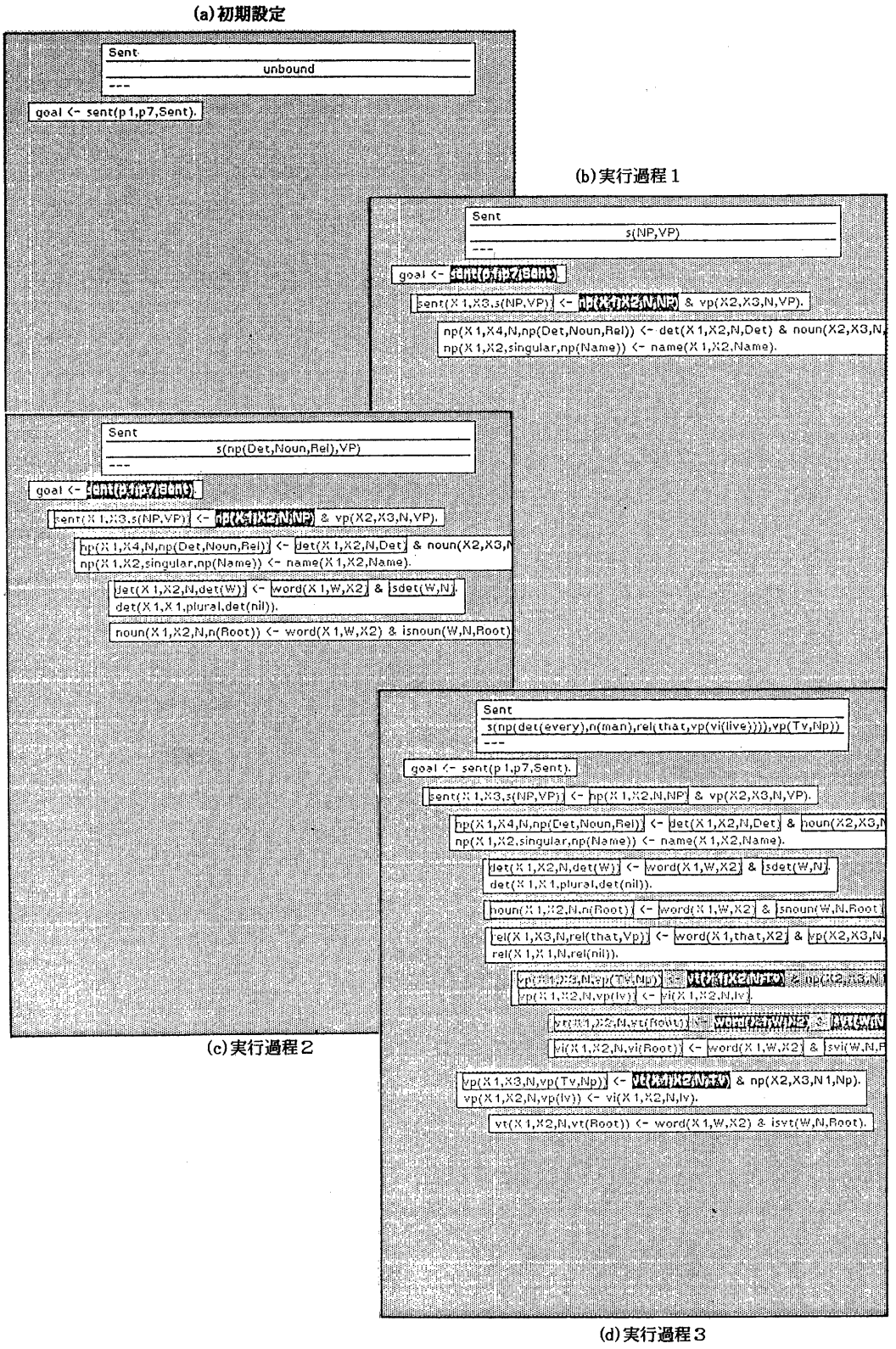
(a)初期設定
Sent
unbound
goal <- sent(p1,p7,Sent).
(b)実行過程1
Sent
s(NP,VP)
(c)実行過程2
Sent
s(np(Det,Noun,Rel),VP)
(d)実行過程3
Sent
s(np(det(every),n(man),rel(that,vp(vi(live)))),vp(Tv,Np))

図2．実行例

４．対象指向型言語による実現

４．１　アイコンとマウスの制御

３章で述べたような、実行の視覚化およびその制御を可能にするためには、画面上のアイコンは、実際の実行における内部表現（モデル）に密接にかかわっていなければならない。つまり、視覚化においてはモデルの状態をアイコンに正確に反映させなければならないし、実行の制御に関しては、ユーザからアイコンに与えられたコマンドをモデルに伝えなければならない。こうしたメカニズムは、対象指向型言語を用いることによって比較的簡単に実現ができる。即ち、モデルの計算過程をオブジェクトとその間のメッセージパッシングによって記述し、各オブジェクトに自分の状態を示すアイコンを持たせればよい。

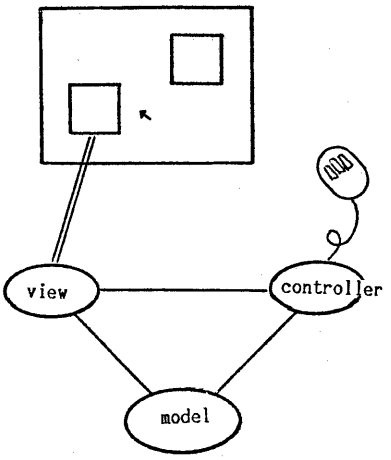

図３．Smalltalk-80におけるウィンドウの構成

Smalltalk-80では、ディスプレイ上のウィンドウは実際には、図３に示すように controller, view, modelの３つのオブジェクトからできている。modelはウィンドウで表示したいものの実体であり、内部表現である。viewはディスプレイに表示されているウィンドウ自体であり、controllerはマウスの管理、メニューの表示などをおこなう。これら３種類のオブジェクトを用いることでユーザインタフェースに関係する部分と、その実体とをそれぞれ独立に設計することが出来る。

本システムのウィンドウの外部表現、つまりviewはこの独立性を利用して、Smalltalkシステムのviewを流用している。例えば定義されたProlog節を保持するデータベース用のウィンドウを実現するために、Smalltalkのクラスを管理するSystem Browserのviewのサブクラスを利用している。

また、実行状態の表示及び制御をするためのアイコンの実現にもこの方法を取り入れている。つまり、Prologインタプリタを構成する各オブジェクトを modelとして、それに view（アイコン）と controller をつけて、視覚化および、マウスセンシティブ化を実現している。

４．２　Prolog インタプリタのオブジェクトによる実現

前節で述べたようにユーザ・インタフェースを設計するとすると、モデルとなる Prologインタプリタをいかに対象指向型に実現するかによって、それが視覚化したときにわかりやすいか否かが決定される。つまり、環境として使いやすいものになるか否かがきまる。Prologのプログラムであるクローズを構成する、述語、関数などを、オブジェクトとして定義し、その間のメッセージパッシングによって、インタプリタが作れれば、視覚化にもふさわしいし、ユーザがみても理解しやすいものになるとおもわれる。

Prolog コンポーネントのオブジェクトによる定義

クローズ、述語、関数、変数、定数を、それぞれクラスとして定義した。各オブジェクト（インスタンス）間の関係は、図４のようになる。modelとviewの関連から、この階層性は、そのディスプレイ上での表現であるアイコンにもそのまま反映される。これがアイコンが構造エディタにもなっている理由である。

クローズ、およびそれを構成するに述語などは、リゾリューションの過程でそれらのインスタンスがつぎつぎと生成される。つまり、実行方式としてはストラクチャーコピー方式である。各インスタンスは自分の内部状態を保持できるので、グローバルスタックのようなものは必要ない。

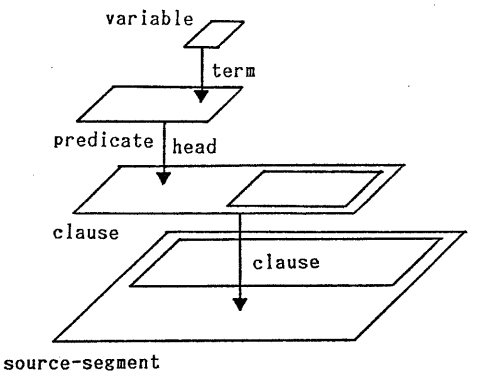

図４．Prologオブジェクトの階層構成

メッセージパッシングによるPrologの実行

クローズ、述語等の各インスタンスは、図5に示すようにメッセージパッシングによって実行を進めていく。

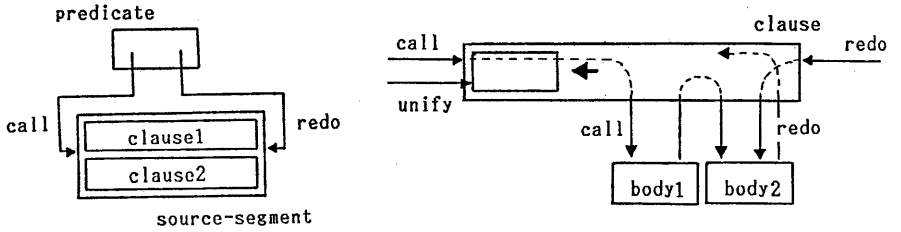

図5．Prolog実行におけるメッセージの流れ

例えば、リゾリューションはつぎのように行なわれる。

サブゴールとなった述語は、自分にマッチするヘッドを持つ候補クローズを生成して、その先頭のものに『call』というメッセージを送る。そのクローズが成功すると、trueという値が返り、失敗すると falseが返ってくる。trueであれば、自分自身が成功となり、falseであれば、次のクローズに『call』というメッセージを送る。最後のクローズが失敗すると、自分自身も失敗となる。

次に『call』というメッセージを受けたクローズは、ボディを構成する述語に『call』というメッセージを送る。成功した述語は trueという値を返し、失敗したら falseを返す。trueであれば、次の述語に『call』というメッセージを送るし、falseであれば、前の述語に『redo』というメッセージを送る。最後の述語が成功したら、自分自身が成功となるし、最初の述語が失敗したら、自分自身も失敗となる。この部分のSmalltalkによるプログラムを、図6に示す。

```
next

    index ← index+1.
    index > (body size)
          ifTrue: [ ↑true ]
          ifFalse: [ ((body at: index) call)
                      ifTrue: [ ↑self next] ifFalse: [ ↑self redo]].!

redo

    index ← index-1.
    index <= 0
          ifTrue: [ ↑false ]
          ifFalse: [ ((body at: index) redo)
                      ifTrue: [ ↑self next] ifFalse: [ ↑self redo]].!
```

図6．プログラム例

また、ユニフィケーションもメッセージパッシングで行なう。Prologの項をあらわすオブジェクトに『unifyTo: aTerm』というメッセージが送られてくると、受け取ったオブジェクトは自分を aTermにユニファイしようとする。ユニファイできれば trueを返し、できなければ falseを返す。

このように全てがメッセージパッシングで記述されているので、Prologプログラムの実行をシステムに任せずに、ユーザが『マニュアル操作』をすることができる。たとえば、ユーザがトップレベルのゴールを実行する時にゴールクローズに送る『call』メッセージと、インタプリタがリゾリューション時にサブゴールに送るメッセージは全く同じものである。これが、ユーザがマウスを通して『call』、『redo』などのメッセージを任意のクローズに送って実行を制御できる理由である。

4．3　モードレス・オペレーションの実現

以上のように、Prologインタプリタを注意深く対象指向型に実現することによって、それがそのままプログラミング環境としての実行過程の視覚化に適応したモデルとなることは、注目すべき特徴である。実行の視覚化としてもっとも自然なものは、実行の過程をそのまま画面上に表示することであるから、インタプリタの動作が視覚化されるのが一番である。

実行の制御もここでは簡単に実現される。つまり、Prologインタプリタを構成する各オブジェクトは、メッセージパッシングによってPrologの実行をする。そのオブジェクトは、ディスプレイ上のアイコンとつながっているわけであるから、実行時に交わされるメッセージと全く同じものを、ユーザからも出すことができる。これが即ち実行の制御に当たる。ユーザからどんなメッセージが出せるかは、マウスがクリックされた時に、そのオブジェクトに定義されているメソッドをメニューとして表示してそれをユーザに選ばせればよい。

このように、ユーザ・インタフェースを設計することによって、システムの実行とユーザからのコマンド入力による実行の区別のない、つまり、実行モードとか、デバッグモードとかの区別のないモードレスオペレーションが可能になる。

５．おわりに

対象指向型言語による、Prologのプログラミング環境の実現についてのべた。対象指向型言語が、こうした環境の作成用言語としても非常に優れたものであることがわかった。特に、対象言語をうまくオブジェクトとその間のメッセージパッシングによってモデル化することによって、簡単に視覚化およびモードレスオペレーションが実現できるということは、大きな特長であるとおもわれる。

現在、本システムを用いて実際にいくつかのPrologプログラムのdebugをすることによって視覚的環境の有効性を検証中である。現システムの問題点としては、実行のモニター中に生成されるアイコンの配置を、現在はユーザが指定しているが、これをいかに自動的に行なわせるかということと、上記のPrologインタプリタの実現方法は、たしかに視覚化には向いているが、メモリー及び速度は大きく犠牲にされていることである。

今後の課題としては、このPrologの環境を整えることと、その他に、Prolog以外の言語の実行過程、あるいはエキスパートシステムの推論過程を、対象指向型言語で実現することを考えている。

謝辞

学生研究員として、システムのインプリメンテーションおよび討議に加わって頂いた東京大学の中川幹夫氏に感謝します。また、富士ゼロックスの田中義男氏をはじめとするＳＥの方々には、画面のハードコピー作成などに際して大変お世話になりました。ここに感謝します。

参考文献

[1] Goldberg,"SMALLTALK-80 The Language and its Implementation",Addison-Wesley,1983.
[2] Weinrep,"LISP Machine Manual",Symbolics inc,1981.
[3] Chikayama,"Unuique feature of ESP",Proc. FGCS'84,1984.
[4] Clocksin "Programming in Prolog",Springer-Verlag,1981.
[5] 後藤茂樹『Prologの図形的な動作表示法』 記号処理27-6,情報処理学会,1984.