EC85−71

視覚的プログラミング・ツールObject Peeper

阿部雅彦・神田陽治
田中英彦
（東大）

1986年3月17日

社団法人 電子通信学会

EC85-71

視覚的プログラミング・ツールObjectPeeper

VISUAL PROGRAMMING TOOL ObjectPeeper

阿部　雅彦　　神田　陽治　　田中英彦
Masahiko ABE　Yohji KOHDA　Hidehiko TANAKA

東京大学工学部
Tokyo University

1　視覚的プログラミング・ツールObjectPeeper

近年のハードウェアの進歩により、高解像度のCRTディスプレイが低価格で利用できるようになり、これを用いた新しいプログラミング環境が考えられつつある。

ObjectPeeperはビットマップディスプレイを用い、並列オブジェクト指向言語DinnerBellを対象としたプログラミング・ツールである。このシステムは特に、

・オブジェクト指向に適している。
・表示上の工夫によりプログラムの実行の様子を理解しやすくする。
・並列プログラムのデバッグを容易化する。

等を目標とする。

2　ObjectPeeperの構成

ObjectPeeper内部の構成は図 1のようになっており大きく分けて、

・プログラム入力のためのエディットシステム
・オブジェクト個々を観察するためのオブジェクト表示システム
・オブジェクト間の関係を調べるメッセージ調査システム

の3つの部分から成っている。ObjectPeeperはこれら3つが協調して動作することによりユーザフレンドリなプログラミング環境を形成することを目指している。

現在エディタの部分については、計画のみではっきりしたことは決まっていない。本論文では実験的に実装されたオブジェクト表示システムとメッセージ問合せシステムについて述べる。これらはプログラムのデバッグに関する部分である。

3　オブジェクト表示システム

3．1　基本方針

次の方針に基づいて表示を行う。

・オブジェクト単位で表示する
・表示に際しての処理により、適切な表示結果を得る

方針の第一は表示をオブジェクト単位で行うということである。ソフトウェアの生産性を向上させるにはモジュール化が有効であるという主張がされている。Dinner Bellもオブジェクトという形でモジュール化の考えを取入れている。このようにプログラミングの際にはオブジェクトを単位として考えているわけだから、デバッグの際もオブジェクト単位でプログラムを見るようにし、ユ

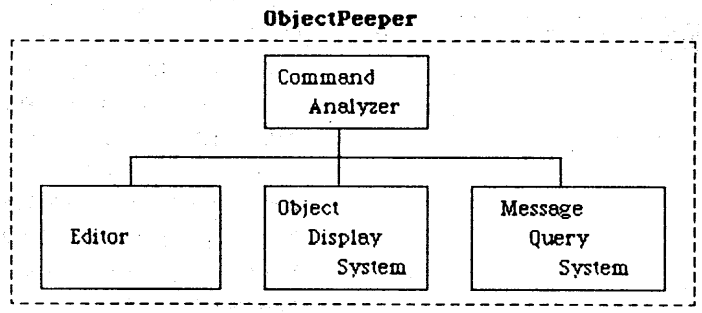


図 1　ObjectPeeperの構成

ーザがプログラムに対してもつイメージを統一することが適切と考えられる。以上のような理由で本システムでは、１オブジェクトに対して１ウィンドウを用い、オブジェクト単位での観察を可能とする。

本システムでは、マルチウィンドウを用いて表示を行う。マルチウィンドウを用いるとき一般に問題になるのは表示面積の不足である。限られたサイズのＣＲＴ画面上に複数のウィンドウを開こうとすれば、個々のウィンドウのサイズも限られてくる。ウィンドウをオーバーラップさせる方法もあるが、マルチウィンドウの目的がいくつかの画面を同時に見られることにあると考えれば、他のウィンドウをおおい隠してしまうオーバーラップでは解決にならない。またスクロールによる方法では、見たい部分が全体にちらばっている場合に何度も上下（あるいは左右）スクロールをしなければならなくなる。

方針の第二は上記のような表示面積不足の問題の解決を目ざしたもので、ウィンドウの表示サイズが十分とれなかった場合にも、ユーザの意図を反映した、オブジェクト全体の様子が読み取れるような表示を作りだすということである。

方法としては、パレットと圧縮という考え方を取入れる。パレットとはユーザの意図をシステムに伝えるための道具であり、圧縮はウィンドウ内に情報を凝縮して表示する処理である。これらによりオブジェクトを適切に表示することを可能とする。

３．１．１　パレット

パレットはクラスに対して定義され、各インスタンスには、それが属するクラスのパレットが指示する表示形式がとられる。ここで注意することは表示は単にパレットの情報をインタープリトするわけではない点である。オブジェクト表示システムはパレットに記述された内容と、ウィンドウのサイズを見て適切な表示を作り出す。具体的にはパレットは図 2のようなもので、表示オブジェクト全体に対応する長方形の内部を、長方形に切りわけたような形をしている。各長方形（ロットと呼ぶ）は

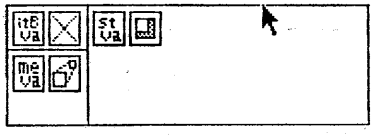


図 2　パレットとアイコン

オブジェクトの、 itBlock変数、method変数、 statement変数実行状態等の表示領域に割当てられる。ユーザは区切りの方法、各領域の対応のさせかた等を自由に設定することができる。パレットの作製は、マウスとアイコン（命令や属性などを図形的に表現したもの）を用いて行う。

３．１．２　圧縮

圧縮は、ウィンドウサイズが小さかった場合に適切な表示を得るための処理の総称である。例えば文字のサイズを小さくする、クリッピングを行う等の方法がある。本システムではNameMasterとの連携により、略語表現を行うことも計画している。圧縮方法は表示属性の一つとして、パレットのロット内に定義する。３．１．２で述べたように、ここで定義した圧縮方法がそのまま表示の際に利用されるわけではない。圧縮方法の選択は、表示用エリアと表示したいものの量の関係を見てシステムが適宜おこなう。

３．１．３　ズームアップ

ズームアップはオブジェクトの中の一部のみを見たい場合に使用される機能である。これは表示されたオブジェクトのうちのロットを指定して、ウィンドウ内にそのロットのみを表示するものである。指定されるロットは一つに限らず複数可能である。この機能は画面上にどうしても充分なサイズのウィンドウをとれず、かつシステムの作り出した表示では見たい部分が見られなかった場合に、一時的に希望する部分だけを拡大することを目的としたものである。

３．２　実験システム

３．２．１　実装環境

オブジェクト表示システムはMacintosh 上に実装されている。このシステムはSUMacC（SUMacCはStanford University Medical Centerで開発されたMacintosh 用のプログラム開発ツールである。このツールはＶＡＸのＵＮＩＸ上で動作するものでＣ言語のクロスコンパイラ、プログラム中でMacintosh のウィンドウ等のリソースを利用できるようにリンクするリソースコンパイラ、およびダウンロード用のプログラムから成る。）を用いて作製されたもので、Macintosh の画面制御機能、マウス制御機能等を利用している。

Macintosh はＲＯＭ内ルーチンとして、様々な画面制御機能を有する。ウィンドウ制御機能としては、ウィンドウの生成、消去、移動、サイズの変更、ウィンドウ同志がオーバーラップした時の優先順位に基づいた表示、等の機能を持つ。また、Quick Drawと呼ばれる作画ライブラリがあり、線を引く、テキストや図形を表示する等の作業を高速に実行できる。

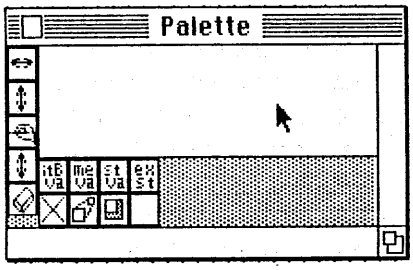
実験システムの開発はＶＡＸ１１／７３０上でＣ言語を用いて行い、これをコンパイルした後、Macintosh にダウンロードするという形で行った。現在プログラムは約３０００行である。

３．２．２　実験システムによる表示

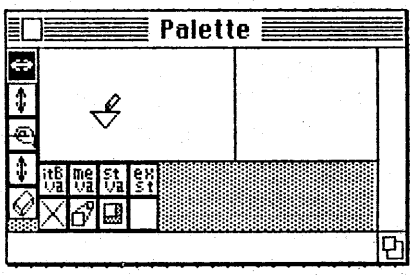
実装済みの表示システムについて実行例をもちいて説明する。

(1) パレットの作成

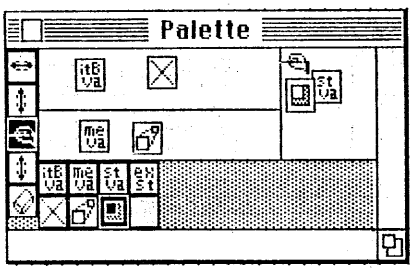
パレットの作成はパレット作成用のウィンドウを開きその中で行う。最初の表示は図 3a)のようになる。空白のエリアがパレット作成用のエリアである。左側にメニューとして並んでいるアイコンはロットの区切りをいれるためのもので、下側のアイコンは表示属性定義用のものである。左側のメニューは上から、横に線を引く、縦に線を引く、属性定義を行う、セレクト、消去、を意味するアイコンである。下側のメニューの上段は左から、itBlock 変数、method変数、 statement変数、実行状態を意味するアイコンであり下段は、無圧縮、縮小、クリッピング、略語表示、を指定するアイコンである。これらの意味は後で説明する。



a)



b)

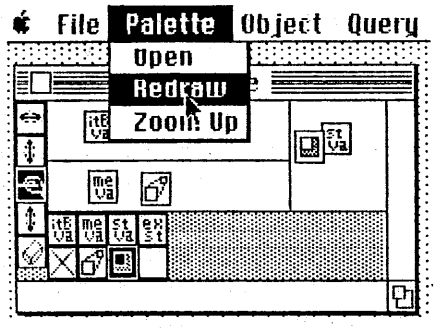


c)

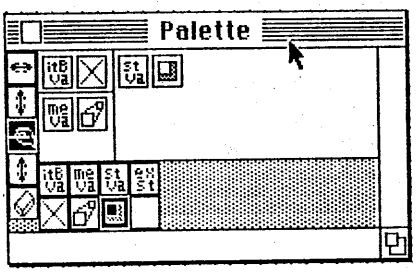
まず例としてパレットを作ってみる。最初にウィンドウ左側のメニューから区切りを選択する。（図 3b)）カーソルが三角定規の形に変化し（カーソルの形が変わるのはエディットエリア内だけで、これ以外の場所では矢印の形に戻る。）パレットが区切れるようになる。ここでは itBlock変数、method変数、 statement変数に対応した３つの区画をつくる。

続いて表示属性を定義する。（図 3c)）それぞれのロットに対し、表示対象と圧縮法を指定する。このときには、左側のメニューで上から３番目のアイコンを選択してカーソルを指の形に変えておく。この例では３つのロットのうち左上を itBlock変数、左下をmethod変数、右を statement変数の領域に指定している。また圧縮方法としては itBlock変数はできる限り圧縮しない、method変数は縮小、 statement変数はクリッピングを指定している。

今作ったパレットを使うには、これを登録することが必要である。システムメニューのpalette でRedrawを選択すると、パレットが登録され、同時に表示の再配置が行われる。（図 3d)、図 3e)）本来Redrawは、パレット上での変更を、そのパレットを利用するオブジェクトの表示に反映させるためのコマンドだが、現在はパレットの登録のみに使われている。

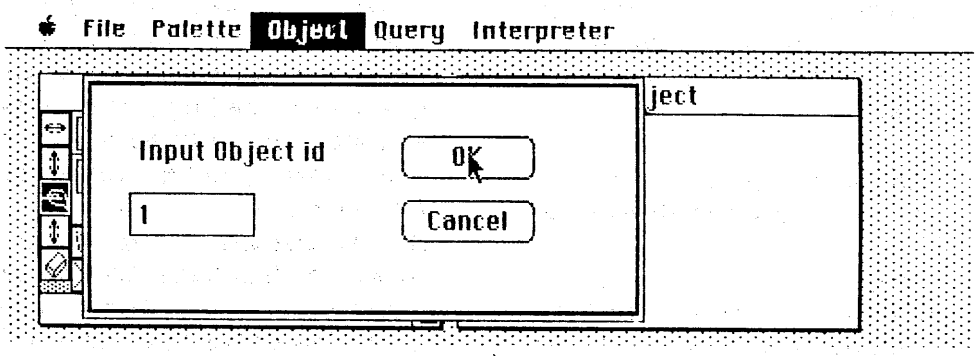


d)

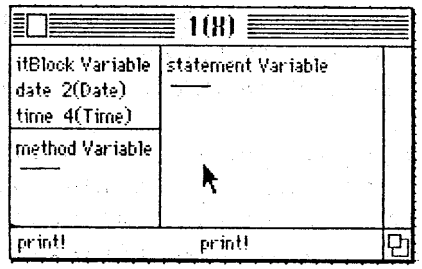


e)

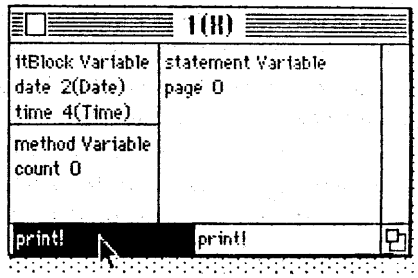
図 3　パレットの作成



a)



b)



c)

図 4　オブジェクトの表示

(2) オブジェクト表示

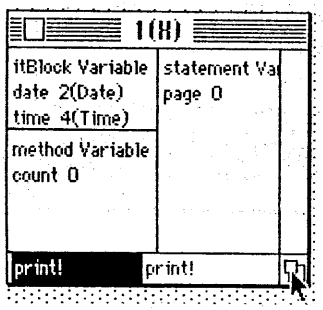
(1) で作成したパレットに基づいて、オブジェクトを表示してみる。システムメニューのobjectでOpenを選択すると、表示すべきオブジェクトのidをきいてくる。(図 4a))

idを入力すると、図 4b)のようなウィンドウが開きオブジェクトが表示される。このウィンドウのタイトルは表示されたオブジェクトがidが１でクラスXに属するものであることを示している。またオブジェクト全体で共有される変数である itBlock変数が表示される。この例では itBlock変数としてdateとtimeが表示されている。変数名の隣の整数値はその変数に代入されているオブジェクトのidであり、かっこ内に表示されているのはそのオブジェクトが属するクラス名である。

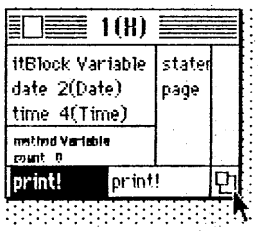
次にウィンドウの下部を見るとprint!が２つ表示されている。これはこのオブジェクトでメソッドprint!が２つ発火していることを示すものである。この表示は同時にメニューになっており、カーソルをもっていってボタンを押すと選択できる。選択を行うと図 4c)のようになり、method変数、 statement変数が表示される。表示方法は itBlock変数の表示と同様である。

さて、以上はウィンドウの大きさが充分ある場合だった。これに対し図 5はウィンドウの大きさが充分になかった場合である。図の番号の順にウィンドウの大きさが小さくなり、圧縮方法が変わっていくのがわかる。

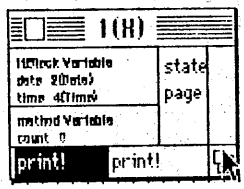
図 5a)は横の長さが足りない場合で statement変数の部分がクリッピングされる。図 5b)ではさらに縦の長さも足りなくなる method 変数の部分が文字を縮小して表示される。どちらの場合もpalette の指定通り itBlock変数の領域だけはもとのまま残される。図 5c)はさらに縦の長さを減らした場合で、この場合には itBlock変数の部分も圧縮せざるをえなくなっている。



a)



b)



c)

図 5　圧縮

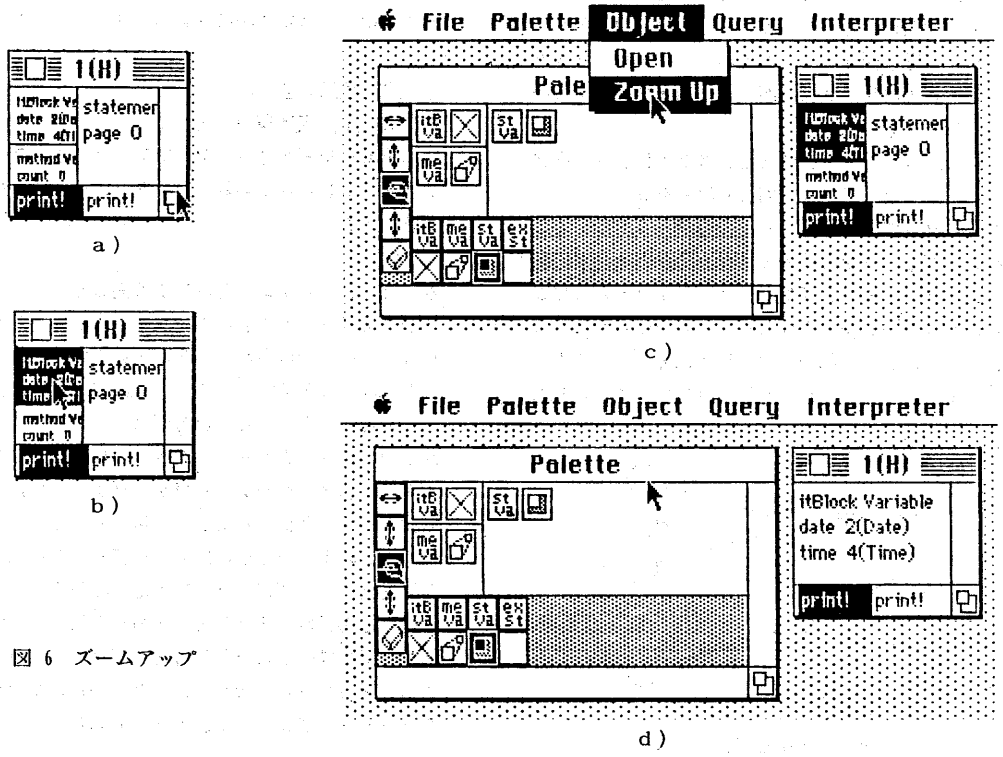


図 6 ズームアップ

(3) ズームアップ

ズームアップは、ロット単位で選択を行い、選択されたロットのみをウィンドウ内に表示する機能である。図6がズームアップの例である。

図 6a)はウィンドウがかなり小さい場合で、圧縮が itBlock変数の部分にまで及んでいる。そこで itBlock変数の部分をズームアップする。まずカーソルを itBlock変数の部分にもっていき、ボタンを押して選択する。選択が行われると表示が反転してそのことを示す。(図 6 b)) その上でシステムメニューのobjectから Zoom Up を選ぶと(図 6c))、ウィンドウ内には itBlock変数の部分だけが残り、圧縮されない表示が行われる。(図 6 d)) ズームアップによる表示は一時的なものなので、このときパレットの方はもとのままである。

4 メッセージ問合せシステム

4.1 概略

4.1.1 目的

どのオブジェクトからどのオブジェクトへ、どんなメッセージがいつ送信され、いつ受信されたかを、効率よく調べるのがこの問合せシステムの目的である。オブジェクト指向プログラムでは

・オブジェクト間の情報伝達手段はメッセージ送受のみである

・プログラムの実行は、メッセージの受信によって始まる。すなわち、メッセージの受信順序により実行の順序が決定される。

等の特徴があるからメッセージのやりとりを調べることで、プログラムの実行を調べることができる。このためには、プログラム実行時にオブジェクト間でのメッセージを記録しておき後で調べられるようにすればよい。しかしながらプログラム実行に伴うメッセージ量はプログラムサイズとともに増大すると考えられるから、これを人手だけで調べるのは困難である。そこでこの仕事を計算機で支援する。

4.1.2 方法

方法としては、メッセージの記録に対し、リレーショナルデータベースの問合せ言語のような形で検索することとする。問合せにはＱＢＥ(Query By Example)のようなものを考えている。ＱＢＥは表の中の必要な部分を穴うめ式にうめていくことによって問合せをするものであり、どのようにしてデータを取り出すかではなく、どんな種類のデータが欲しいかを記述する問合せ言語なので、他の問合せ言語に比べわかりやすいという特徴をもっている。

ObjectPeeperではメッセージの記録に対し、ＱＢＥのような形式で問合せを行う。蓄えるデータの項目としてはメッセージ送信側オブジェクトのクラスとid、メッセージのキー送信時間、受信時間、受信側オブジェクトのクラスとid、を考えている。

本システムのメッセージ問合せシステムは、特別な機能として、メッセージパスの問合せ機能をもつ。

オブジェクト間でのメッセージのやりとりを考えると１つのオブジェクトから他のオブジェクトへメッセージが送られ、メッセージを受信したオブジェクトからはまた新たにメッセージが送られということを繰返していくように考えられる。（図 7）この様子はオブジェクトの間にメッセージによってパスがはられていくように見える。

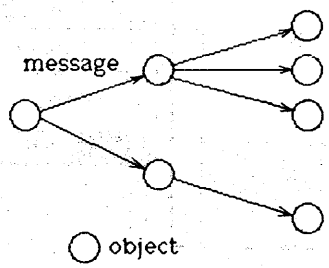


図 7　メッセージのパス

メッセージ問合せシステムは、このようにしてできるパスの中から特定の特徴をもつパスをさがしだす。

４．２　実験システム

４．２．１　実装環境

メッセージ問合せシステムはＶＡＸ１１／７３０上にＣ言語で実装されている。プログラムの大きさは約６００行である。

４．２．２　実験システムによる問合せ

現在実験システムではスクリーンエディットができない等、かなり制限されているが一応の問合せはできる。

今、図 8に示すように４つのオブジェクトが通信を行っていた場合を考える。この図は例えば、クラスＸのidが１のオブジェクトから、クラスDateのidが２のオブジェクトへ、メッセージprint!が時刻１００に送信され、時刻２００に受信されたことを示している。

以下このような場合に問合せを行った例を示す。

(1) 送信側のオブジェクトを指定した場合

図 9は送信側のオブジェクトのidを指定した問合せである。図の横線から上が問合せ、下がそれに対する答である。図 8でidが１のオブジェクトから送信された２つのメッセージが答として出力されている。

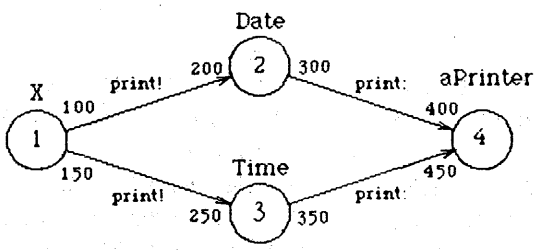


図 8　メッセージ送受の例

```
Query
      source class    id              message tran rec      destinate class    id
                       1
-------------------------------------------------------------------------------
      source class    id              message tran rec      destinate class    id
                   X    1               print!  100  200                Date    2
                   X    1               print!  150  250                Time    4
```

図 9　送信側オブジェクトを指定した問合せ

```
Query
      source class    id              message tran rec      destinate class    id
                                                                               3
-------------------------------------------------------------------------------
      source class    id              message tran rec      destinate class    id
                Date    2               print:  300  400           aPrinter    3
                Time    4               print:  350  450           aPrinter    3
```

図10　受信側オブジェクトを指定した問合せ

(2) 受信側のオブジェクトを指定した場合

(1) 同様にして受信側のオブジェクトを指定した場合である。オブジェクトid3に対して送られた2つのメッセージが答として出力されている。(図10)

(3) クラス名による問合せ

問合せをクラス名によって行った場合である。この場合にはそのクラスに属する全オブジェクトが探索の対象となる。図11は送信側のクラスを指定した場合である。

(4) メッセージパスの問合せ

メッセージパスの問合せはこのシステムにおいて最も特徴的な問合せである。(図12−14) 問合せの中のメッセージ指定の項のコマンド.1がパスの問合せであることを示すものである。図12は送信側のオブジェクトを指定し、そこから2つめのオブジェクトまでのパスを問合せる例である。一方図13は受信側のオブジェクトを指定した場合である。図14は両端のオブジェクトを指定し、また_2とすることによってその間の任意の長さのパスを問合せる例である。答としてはいずれの場合も図 8の上側のパスと下側のパスが出力されている。

```
Query
      source class    id            message tran rec     destinate class    id
                   X
-------------------------------------------------------------------------------
      source class    id            message tran rec     destinate class    id
                   X    1            print!  100  200              Date     2
                   X    1            print!  150  250              Time     4
```

図11 クラスを指定した問合せ

```
Query
      source class    id            message tran rec     destinate class    id
                      1                .12
-------------------------------------------------------------------------------
Message pass 1
      source class    id            message tran rec     destinate class    id
                   X    1            print!  100  200              Date     2
                Date    2            print:  300  400          aPrinter     3
Message pass 2
      source class    id            message tran rec     destinate class    id
                   X    1            print!  150  250              Time     4
                Time    4            print:  350  450          aPrinter     3
```

図12 送信側オブジェクトを指定したパスの問合せ

```
Query
      source class    id            message tran rec     destinate class    id
                                       .12                                  3
-------------------------------------------------------------------------------
Message pass 1
      source class    id            message tran rec     destinate class    id
                   X    1            print!  100  200              Date     2
                Date    2            print:  300  400          aPrinter     3
Message pass 2
      source class    id            message tran rec     destinate class    id
                   X    1            print!  150  250              Time     4
                Time    4            print:  350  450          aPrinter     3
```

図13 受信側オブジェクトを指定したパスの問合せ

```
Query
      source class    id            message tran rec     destinate class    id
                      1                .1_2                                3
-------------------------------------------------------------------------------
Message pass 1
      source class    id            message tran rec     destinate class    id
                   X    1            print!  100  200              Date     2
                Date    2            print:  300  400          aPrinter     3
Message pass 2
      source class    id            message tran rec     destinate class    id
                   X    1            print!  150  250              Time     4
                Time    4            print:  350  450          aPrinter     3
```

図14 両端のオブジェクトを指定したパスの問合せ

5 検討

5．1 デバッギング例

簡単なプログラムを用いてObjectPeeperを使ったデバッギングの様子を示す。

まずプログラム例を挙げる。（図15）これはクイックソートのプログラムで、クラスqsに数のリストをアーギュメントとするメッセージを送って起動する。クラスNodeのインスタンスはリストを構成する。またクラスsplit のインスタンスは数値とリストを受取り、リストを与えられた数値より大きい数のリストと小さい数のリストに分割する。ただしこのプログラムでは図の下線部に作為的にバグをいれてある。（cdr!とすべきところがcar!になっている）

```
$class TRUE
$class FALSE

$class NULL [
    _isNull! || sender#1 ret: TRUE
]

$class Node [
    _isNull! || ^ ret: FALSE;
    _cons: ~Car ;
    _and:  ~Cdr ;
    _car!  || ^ ret: ~Car ;
    _cdr!  || ^ ret: ~Cdr ;
    _append: Tail  ||
        (~Cdr isNull!)
            yes: (|| ^ ret: Tail)
            no:  (|| ^ ret: (Node cons: ~Car and:
                (~Cdr append: Tail )))
]

$class split [
    _split: ~S of: List ||
        ^ high: (high: List).
        ^ low:  (low: List) ;
    _high: List ||
        (List isNull!) yes: (|| ^ ret: NULL) no:  (|| ^ ret: (
                ((List car!) >: ~S )
                    yes: (|| ^ ret:
                        (Node cons: (List car!) and:
                            (high: (List cdr!))))
                    no:  (|| ^ ret:
                        (high: (List cdr!))))); % recursion point
    _low: List ||
        (List isNull!) yes: (|| ^ ret: NULL) no:  (|| ^ ret: (
                ((List car!) <=: ~S )
                    yes: (|| ^ ret:
                        (Node cons: (List car!) and:
                            (low: (List cdr!))))
                    no: (|| ^ ret:
                        (high: (List cdr!))))  % recursion point
]

$class qs [
    _sort: List ||
        (List isNull!) yes:
            (|| ^ ret: NULL)
        no: (||
            _high: H low: L <- split split: (List car!) of: (List car!).
            ^ ret: ((qs sort: H) append:
                    (Node cons: (List car!) and: (qs sort: L))))
]
```

図15 プログラム例

このプログラムをクラスqsにメッセージ

```
sort: [2,5,4,1,3]
```

を送って実行すると、クラスinteger にはメッセージisNull! に対応するメソッドが存在しないというエラーが発生する。

そこでまずこのメッセージを送ったオブジェクトがどのオブジェクトかを調べる。メッセージ問合せシステムを用いて次の問合せを行う。（図16）

この結果原因はクラスsplit にあることがわかる。そこでid10のオブジェクトを表示すると、メソッドsplit: of: 、high: 、low:が発火している。ここではsplit:of

```
Query
      source class    id            message tran rec      destinate class    id
                                    isNull!                       integer     2
-------------------------------------------------------------------------------
      source class    id            message tran rec      destinate class    id
             split    10            isNull!   120   130           integer     2
```

図16　バグ検出のための問合せ例（１）

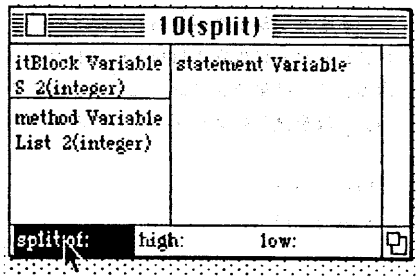


図17　バグ検出のための表示例

```
Query
      source class    id            message tran rec      destinate class    id
                                  split:of:                       split    10
-------------------------------------------------------------------------------
      source class    id            message tran rec      destinate class    id
                qs     5          split:of:   100   110             split    10
```

図18　バグ検出のための問合せ例（２）

：を選んでみる。メソッド変数Listに整数２が代入されている。（図17）

変数Listにはクラスnodeのインスタンスが代入されるはずであるので整数２が入るのはおかしい。splitの記述を見るとListはメッセージ受信の際アーギュメントとして代入される。そこでもう一度メッセージ問合せシステムを使う。（図18）問合せの結果原因は送信側のクラスqsにありそうなことがわかる。qsでメッセージsplit:of:を送っている場所を調べると、アーギュメントの設定が両方ともcar!になっていることがわかり、バグの原因を見つけだすことができる。

５．２　オブジェクト指向と並列性

５．２．１　オブジェクトの表示

オブジェクト指向言語では変数の値はオブジェクトへのポインタのみである。オブジェクトの状態を表示する際には変数によってさされるこのオブジェクトをどのように表示するかが問題となる。

オブジェクト指向では、acquaintanceについては、その入力と出力（外部仕様）だけが関心事であり、インプリメンテーションはacquaintance内部の問題である。従ってオブジェクトを表示する場合にも、変数にバインドされたオブジェクトについては、その内部を見せるのではなく、性質を示すような表示が必要である。例ばあるacquaintanceをスタックとして利用しているとき、それがアレイで実現されているか、リスト構造で実現されているか等は表示されるべきではなく、そのacquaintanceがスタックの性質をもつことだけがわかればよい。

ObjectPeeperでは相手のオブジェクトの性質を示すものとしてクラス名を採用している。そしてacquaintanceについて見たい場合には新たにウィンドウを開いて表示することにより、オブジェクト指向の考えかたに添った表示を実現している。

５．２．２　並列時の問題とその検出

オブジェクトは、実行環境を持つアクティブプロセスなので、個別に保存して後で再び活動を続けることができる。また、複数のオブジェクトを実際に並列に動かすことができる。このため、オブジェクトを単位としてプロセッサに割当てると、高い並列性が得られると言われる。並列性はオブジェクト内の手続きのメッセージを一斉に起動することで得られる。一つの手続きが複数のメ

ッセージを放つと連鎖反応のように活動が起きる。

しかし、オブジェクトは内部状態を持ち、かつ共有されるから、副作用が存在する。従って制限なしに並列実行を行うと、実行結果は非決定的になる。以下ではオブジェクトを並列動作させた場合の問題点について、いくつか例を挙げて説明する。

(1) エレベータ昇降問題

舞台は、ビルの一角にあるエレベータである。エレベータを一つのオブジェクトとして扱う。最初、エレベータは5階にある。8階にいる人を乗せるために3階分昇る命令を与え、その後4階にいる人を乗せるために4階降りる命令を与える。試しにプログラムしてみると、

```
anElevater up:3.
anElevater down:4.
```

この問題では、二つのことが問題になっている。

第一に、エレベータオブジェクトは現在いる階数を状態として維持し、その一貫性を保つため、続けて来る二つのメッセージによって起動される二つのメソッドをserialize する必要がある。

第二に、二つのメッセージ up:3とdown:4がこの順でエレベータオブジェクトに到着することを保証する必要がある。もし、出したメッセージが逆の順でつくと、エレベータは1階と4階にいき、8階にいる人を乗せることができなくなってしまう。

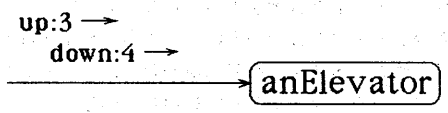


図19 エレベータ昇降問題

このような問題をObjectPeeperは次のようにして検出する。

まずひとつめのメッセージup:, down: の受信順序が問題だが、受信順序が正しいかどうかは、メッセージ問合せシステムによって、エレベータオブジェクトに到着したメッセージを表示させ、受信時刻を調べればよい。受信順序が狂っていた場合には送信側オブジェクトのクラスをエディットして順序化の機構をいれる。

もう一つの問題はエレベータの状態に対する排他制御である。まず一貫性が保たれているかどうかは、オブジェクトを表示し、その状態を見ることによって調べられる。一貫性が保たれていなかった場合その原因の検出は次のように行う。

一般に排他制御の必要なオブジェクトへのアクセスはクリティカルスポットの中で行われる。どこかで一貫性の保たれない状況が発生するとすれば、それはクリティカルスポットを通らずにオブジェクトにアクセスできる別のパスが形成されていることを意味する。このようなパスの検出は、メッセージ問合せ機能を用いて、排他制御すべきオブジェクトから逆向きにパスを探索すればよい。

(2) プリンタ共有問題

現在の日付を管理するオブジェクトと、現在の時刻を管理するオブジェクトがあるとする。いま、この二つのオブジェクトに現在の日付と時刻をリポートするように頼み、その結果をプリンタに日付、時刻の順で印刷したい。これを試しにプログラムしてみると、

```
Date print!.
Time print!.
```

Date Time のオブジェクトはそれぞれプリンタオブジェクトにメッセージを送って印字させる。印字される順序はメッセージが着く順になるから、仕事を頼んだ二つのオブジェクトからプリンタに転送されるメッセージの到着順を固定できなくてはならない。

この問題がエレベータ問題と違うのは、プリンタオブジェクトへのメッセージの発信元が、順序を制御したいオブジェクトとは別のオブジェクトである点である。

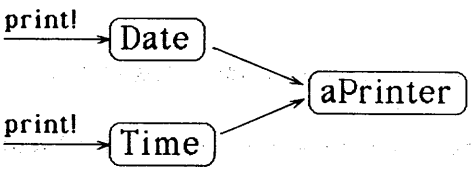


図20 プリンタ共有問題

これはオブジェクトが直接見えない場所で共有される問題である。この場合には

・実行のもととなるオブジェクトから、副作用をもつオブジェクトへのパスが複数ないかどうか
・パスが複数あるとすればどこに原因があるか

の2つを発見することが必要である。

この問題はメッセージ問合せシステムを用いてパスを見つけることにより解決できる。パス上の各オブジェクトをオブジェクト表示システムを用いて表示することにより、パスが複数できる原因もわかるはずである。

6 今後の課題

今後の課題を以下に列挙する。

表示属性の多様化

圧縮方法、表示対象の指定の他に文字のサイズやフォ

ント、スタイルの指定等も行えるようにする。またアミカケ等も用い、これらの拡張によりさらに見やすい表示を目ざす。

・実行状態の表示

メソッドの実行状態はプール中に残っているコンテクストをデコンパイルすることによって表示できると考えられる。問題は１コンテクストが必ずしも１メッセージに対応していない点である。（複数のアーギュメントをもつメッセージがあった場合、コンパイラはこれをアーギュメント１つの複数のメッセージに分解する。）このためソースプログラムとの対応づけに注意しなければならない。

・オブジェクトの履歴

現在のObjectPeeperはメッセージについては履歴をとっているが、オブジェクトについては履歴を考えていない。あるメッセージが送られた時点でのオブジェクトが見たい場合その時点でのオブジェクトの状態を知るためには、なんらかの方法で履歴をとる必要がある。DinnerBellでは変数は単一代入則に従っている。このため、基本的には変数に値が代入された時刻とその値を記録すればよい。

・メッセージ問合せ用コマンドの充実

現在のメッセージ問合せシステムは、.l以外は特別なコマンドはもっていない。考えられるコマンドとしては

ａ）メッセージの受信時刻によりソートする。この機能によりメソッドの発火順序がわかりやすくなる
ｂ）受信時刻が最大のものを選ぶ。この機能により一番最近受信したメッセージがわかる
ｃ）特定の時間範囲に送信または受信したメッセージを探索する。

等が挙げられる。

・メッセージ問合せシステムとオブジェクト表示システムの結合

オブジェクトを表示する場合、現在の実験システムはidを入力して対象となるオブジェクトを指定している。これを画面内でマウスを使って選択することにより指定できるようにする。特に、メッセージ問合せによって発見されたオブジェクトを表の中からマウスを用いて選択することにより表示できるようにする。

・エディタの実装

DinnerBellは、アンダーラインや特殊記号、その他のグラフィック機能を用いることができるビットマップディスプレイを用いたエディタを仮定している。このためテキストのみでプログラミングした場合、プログラム例を見るとわかるように、やや理解しにくい。プログラミングの能率を上げるためには、専用のエディタが必要である。

・インタプリタとの結合

DinnerBellとObjectPeeperとの間でのやりとりをファイル渡しではなく、ObjectPeeperが直接インタプリタのデータにアクセスできるようにする。問題は、Macintosh の記憶容量が限られていること（５１２KB）、SUMacCのＣ言語とＵＮＩＸのＣ言語では、仕様が多少違っていること、等である。ビットマップディスプレイとマウスを装備した新しい実装環境を得ることが望ましい。

・マルチプロセッサ上への実装

ＯＲＡＧＡシステムはマルチプロセッサ上に実装されることを目的としている。従ってObjectPeeperもマルチプロセッサ実装することを考慮しなければならない。マルチプロセッサシステムではコンテクストが複数のプロセッサに適当にばらまかれて実行される。

マルチプロセッサで問題なのは、時間の管理である。インタプリタ上での実装では時計を１つにすることができたが、マルチプロセッサ上ではプロセッサ間での時刻合わせをしなければならない。オブジェクト指向の場合には、基本的にはオブジェクト内で一貫性のある時間をもてればよいが、コンテクストが複数のプロセッサにばらまかれてしまうため１つのオブジェクトに限っても時間管理は難しい。

・DinnerBell第二版への対応

本研究で対象としたDinnerBellは第一版のものだったが、現在は第二版が開発されつつある。第二版での大きな変更点はストリームが取りいれられたことだが、この他にも若干のセマンティックスの相違がある。第二版の仕様が固まったところで対応を検討する必要がある。

７　おわりに

本論文では、並列オブジェクト指向言語を対象としたデバッギングシステムについて

・オブジェクト指向に適している
・並列下でのデバッグを容易にする
・視覚的効果にすぐれている

等を目標としたObjectPeeperについて述べた。

ObjectPeeperは表示の際の圧縮、メッセージの記録に対する問合せ等他のシステムにない特徴をもっている。

今後は６で挙げたような課題についての検討を行いつつ、システムの構築を進めていく計画である。

参考文献

[1] B.A.Myers : Displaying Data Structures for Interactive Debugging
CSL-80-7 June 1980 Xerox PARC

[2] : Inside Macintosh
Apple Computer Inc.

[3] 阿部他：オブジェクト指向言語に向いたディスプレイ・エディタ
情報処理学会第29回全国大会 4R-6(1984)

[4] 阿部他："DinnerBell+NameMaster+ObjectPeeper"による統合プログラミング環境 －ユーザインタフェース
情報処理学会第30回全国大会 5T-5(1985)

[5] 阿部他：統合プログラミング環境ＯＲＡＧＡ（Ⅲ）ユーザインタフェースObjectPeeper
情報処理学会第31回全国大会 1E-8(1985)

[6] 阿部他：並列オブジェクト指向システム ＯＲＡＧＡ －デバッギングシステム ObjectPeeper
情報処理学会第32回全国大会 6F-2(1986)

[7] 河野他：並列オブジェクト指向言語DinnerBellの実装
電子通信学会 電子計算機研究会 1986.3