# 柔軟なソフトウェアの構築原理とその実現

原田 康徳

## ■ 研究のねらい

機能の拡張が容易なシステムをどのように作ればよいのかを探る。従来のソフトウェアの作り方では、プログラムの開発者とユーザとの境界がはっきりと分かれていた。ユーザはアプリケーションへの要求を開発者に上げ、開発者はそれらを元にアプリケーションを更新する。しかし、様々なユーザが多種多様な用途でコンピュータを使用している現状では、すべてのユーザの要求を満足するようなアプリケーションを実現することは難しく、逆に不必要な機能が追加されてしまうことで、全体が分かりにくくなる。この問題の解決には、ユーザが自ら、もしくはよりユーザに近い開発者が、アプリケーションの拡張を行えるようにする必要がある。

これまでプログラムを拡張するという課題は、主にプログラムの開発者の立場で研究が進められて来た。それは、たとえばプログラムのソースコードが存在し、それを最少の手間で拡張を行う、といったものである。しかし、ユーザの立場でみた場合、必ずしもプログラムのソースコードが手に入るわけではないし、手に入ったとしても、それを部分的にでも理解しなければ拡張はできない。また、ユーザが必要とする拡張が、プログラムの開発者があらかじめ想定したような拡張からはずれた場合、ソースコードは大幅に手を入れなければならない。

本研究のねらいは、ユーザの立場から、プログラムを柔軟に拡張できるようにするための、ソフトウェアの構築法を探り、それを実証することである。具体的には、従来のプログラムはデータとそれを操作する小さなプログラムの組を一まとまりのオブジェクトとして扱い、それをたくさん集めて一つのアプリケーションとしていた。これはデータごとの拡張には優れているが、あるデータを操作する新しい機能を拡張することは難しい。それを本研究では、すべてのデータをユーザとコンピュータとで共通にアクセスできるようにし、データを各機能に対応した複数のプログラムから操作することでアプリケーションを構成する、という新しいソフトウェアの枠組みを提案する。新しい機能は新しいプログラムを作成して追加する。この枠組みに必要なプログラミングの仕組みはなんであるかを探り、プログラムを柔軟に拡張できることを示す。

## ■ 研究成果

本研究の成果は、新しいソフトウェアの構造を提案し、そこに新しいプログラミング技法（ダイナミックデータ抽象）を導入して、実装したことである。以下、それに至るまでの研究の経緯に沿って示す。

### 1. Visibility Programming

柔軟にシステムを拡張できるようなソフトウェアの枠組みを考える。従来のシステムは主にオブジェクト指向プログラミング（OOP）という枠組みで作られていた。OOP はデータとそれを操作するプログラムとを一つの塊（オブジェクト）としてプログラムを作るもので、複雑なシステムを表現する場

合に極めて有効なプログラミング技術である。たとえば、プログラムの開発者のための技術であり、拡張に際してオブジェクト間の詳細な約束を理解する必要があるため、プログラムの動作を理解しなければ、拡張は難しい。また、あらかじめ拡張を想定してシステムを設計しなければならない。

本研究ではプログラムの詳細を理解する必要なく、システムを拡張することができるプログラムの枠組みを提案する。ここで、提案する新しいソフトウェアの構造を Visibility Programming(VP)と呼ぶ。これは、システムのすべての状態を表現する共有データ空間と、それをアクセスする複数のプロセスからなる構造である。アプリケーションは複数のプロセスで構成され、データ空間の一部を書き換えることで計算が進む。従来から協調計算モデルの一つとして知られている黒板モデルを、すべてのデータを黒板で表現するように極端にしたものと考えても良い。

すでに、VP とみなすことができるシステムが存在する。例えば、Emacs という文字エディタの上では、様々なアプリケーションがe-lispという言語によって作られている。これらのアプリケーションは文字バッファにすべての状態を文字で表現し、それを読み書きする複数のe-lisp のプログラムによって構成されている。各プログラムは小さな機能（文字バッファを書き換える）を実装しており、それらの組み合わせで複雑なアプリケーションが動作している。この特徴は既存のアプリケーションに対して，その内部の詳細を知らなくても、簡単に機能を拡張できるという点である。その理由として、データは画面上で常に見えており、何をどのように書き換えればよいかが、簡単に理解できる点があげられる。

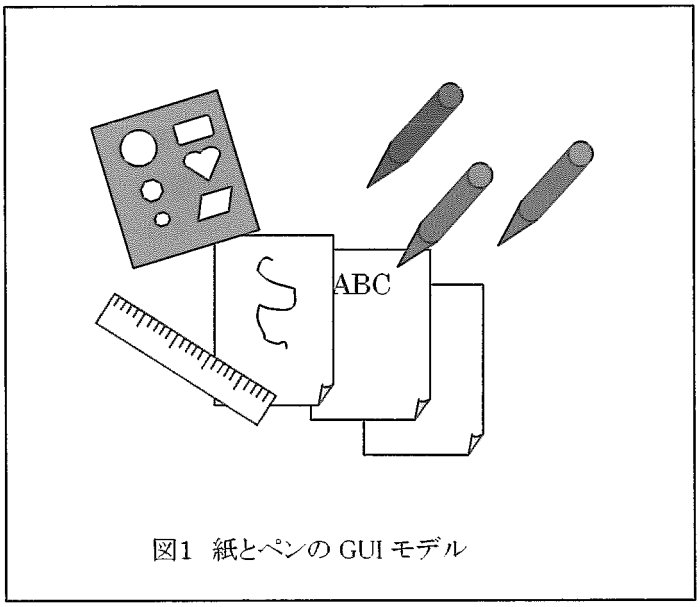

図1 紙とペンの GUI モデル

ここでは、図形エディタのような GUI をプログラミングの対象として、どのようにして Emacs のような拡張が容易なシステムが実現できるかを探ることとした。GUI を VP で実現すると、図１のような紙とペンによるモデルになる。紙は自由に情報を格納表示することができ、様々なペンや文房具を用いて紙上の情報を編集してゆく。ペンには読み取り、解釈し、計算し、書き込むという機能がある。実際の紙と異なり、情報を書き換えることは容易である。対する従来のオブジェクト指向による GUI は、賢い紙（Window）とそれをなぞる指によるモデルである。賢い紙はどこを触られたかによって、どのような動作をすればよいか知っている。新しい機能を追加したい場合に、紙とペンのモデルでは、新しくペンを追加すればよいのに対して、賢い紙のモデルでは、紙の内部構造に新しい機能を追加するか、その機能を持った紙をもってきて、データを Copy & Paste によって移動させる必要がある。このように紙とペンのモデルでは機能の拡張が容易であることがわかる。

まず GUI を表現するために必要なデータ構造を考える。典型的な GUI プログラムのデータ構造を調べるために SmallTalk の実行状態を調べた。それによると、オブジェクトの約４％が直接や間接の相互参照をしていることがわかった。例えば、子オブジェクトは一つの親オブジェクトを参照し、逆に親

オブジェクトはコレクションオブジェクトを経由して複数の子オブジェクトを参照している。これは典型的なGUIのデータ構造である。

従来はアプリケーションが決まり、必要な機能が決まり、そのためのアルゴリズムが決まってから、それに必要なデータ構造が決まっていた。しかし、本研究が目指しているのは、どのような用途にでもデータが使われるようにあらかじめ設計しておく必要がある。そこで、共有構造として全ての参照が逆参照を持つデータ構造を採用した。実際には、ほとんどのアルゴリズムでは逆参照を必要としないかもしれないが、どんなときでも逆参照をたどることができるということが重要なのである。

## 2．CCC： ユーザ定義可能なオブジェクト構造

VPの実装は共有メモリとそれに接続するプロセスというソフトウェアの構造で作られる。共有メモリにデータを格納する際に、その基本的な実装にはオブジェクト指向技術を使用したい。しかし、代表的オブジェクト指向言語C++ではオブジェクトの構造が固定されているため、共有メモリ上にオブジェクトを格納できない。指向言語が言語の組み込み機能として提供してきたメソッドディスパッチをCCCではプログラマが自由に定義できるようにする。

CCCのメソッドディスパッチは、プログラマがクラス毎に定義された条件式で行われる。クラス階層の上から順に条件を調べ、条件が成立するサブクラスが無くなったクラスが、そのデータのクラスである。この機能をもちいて共有メモリ上にオブジェクトを定義することができるようになった。

図2はCCCのプログラム例である。この例では外部から与えられたデータselfがどのような値を持っているかで、クラス分けしている。ここではクラスの構造の定義しか示されていないが、これにメソッドやマクロの定義などを追加することができる。

```
typedef unsigned int ptr;
@class object (ptr self) {
  @class data switch (self & 0x6) {
    @class nil_or_cons case (0x0) {
      @class nil if (self == 0) {}
      @class cons elsif (1) {}
    }
    @class number case (0x2) {}
    @class string case (0x4) {}
    @class other  case (0x6) {}
  }
}
```

図2　CCCのプログラム例

CCCの重要な特徴は同じデータに対して、クラス階層の木を取り替えることで、自由にそのデータの解釈を変更することができるという点である。この性質はVPの考え方である、データが最初にあり、後からそのデータを解釈するプログラムを与える、そのものである。CCCは最初VPのシステムを構築するために使われたが、結果的にVPのモデルをシステムレベルに適用したものになった。このことから、CCCでのプログラミングの経験が、VPの上位レベルの言語をデザインするのに大きく貢献した。

## 3．パターンマッチによるメソッドディスパッチ

VPシステムの共有メモリの基本データ構造は逆参照を必ず持つものであった。そこで表現されているデータは複雑なデータ表現に使われる。CCCではクラス階層を条件式で定義していたが、ここでは、より高度にパターンマッチを用いてクラス階層を定義する言語PDL（Pattern Dispatch Language）を提案する。

まず、パターンを定義する（２行目)。これはデータの接続関係を定義するもので、これによりオブジェクトの形を決定する。次に、そのパターンにメソッドを定義する（６行目)。あるデータに対してメッセージを送ると、そのデータにマッチするパターンを探し出し、見つかったパターンに定義されているメソッドを呼び出す。

パターンには変数が含まれている（大文字で始まる名前)。変数はパターンマッチの際に任意のデータを対象から探し出して格納する。パターンをクラスとみなした場合、これらの変数はインスタンス変数のように振舞う。したがって、メソッド内での変数への代入によって、対象の書き換えがおこる。

ここでは、固定された形状のパターンのみを扱っているが、よりダイナミックな形状や繰り返しを含む形状にマッチするパターンを定義できるように、拡張した。これにより、複雑な構造を一つのオブジェクトのように扱うことができるようになった。たとえば、従来のオブジェクト指向では木のような複雑な構造は、個々の部品はオブジェクトとして定義できても、全体を一つのオブジェクトのようには扱えなかった。しかし、この言語では自由にデータをオブジェクトとして取り扱う粒度を変更できるため、複合データもオブジェクトとして扱える。

```
cLine:{A
  pLine = [pattern [L=[line A B]
                   A=[location Xa Ya]
                   B=[location Xb Yb] ]
                  [L A B] [Xa Ya Xb Yb] ]
  [method pLine draw ()
    (progn
      (edPoint A Xa Ya)
      (edPoint B Xb Yb)
      (winline Xa Ya Xb Yb)
      (edLine L Xa Ya Xb Yb))]

  [method pLine dmove (DX DY)
     (progn
       (setnth A 2 (add Xa DX))
       (setnth A 3 (add Ya DY))
       (setnth B 2 (add Xb DX))
       (setnth B 3 (add Yb DY)) )]
A}
```

図３　パターンによるクラスとメソッド定義

ダイナミックな形状のパターンマッチを行うと、メソッド本体の処理とパターンマッチでの制御との対応をとる必要がある。たとえば、パターンマッチである部分構造をＮ回繰り返した構造とマッチした場合、メソッド本体でも同じようにある部分処理をＮ回繰り返さなければならない。また、パターンである分岐のどちらを選択したかという情報も、メソッド本体に伝えなければならない。これらの情報をメソッド本体での条件式などによる動的な処理を行うと、パターンマッチで行ったことと同じことをメソッド本体でなぞることになり、無駄である。そこで、メソッド本体をメソッドコンビネーションによって記述する。部分パターンを、条件分岐を含まない、静的な要素だけで定義する。ただし、他の部分パターン（自分自身も含む）を呼び出すことを可能とする。部分メソッドを部分パターンで定義する。パターンマッチに成功すると、部分パターンの呼び出し関係の木構造が出来上がる。その木に従って、部分メソッドを結合して一つの大きなメソッドを作る。

作られたメソッドは条件分岐や繰り返しなどの制御文を含まない、静的な構造をしている。また、メソッドはパターンマッチの度に合成される。これは対象の構造が変化しない場合に、それ上のプログラムの実行を静的で効率よく行うことができるということである。たとえば、２進木のパターンを再帰的に定義する。木のノードには値が格納されており、sum というメソッドは木のノード全体の値を

合計するとする。パターンは２進のノードと葉の２つに分かれる。sum の部分メソッドは，葉の場合はそれ自身の値を返す、ノードの場合は両方の枝の値と自分自身の値の合計を返す、というものである。ある木とパターンマッチすることで、その木と同じ形状をしたデータの合計を求めるプログラムが生成される。このプログラムは対象がこの形状であると仮定して、エラーチェックなしで、高速に動作する。

以上の動作は、パターンマッチごとの対象の粒度に応じて、オブジェクトの大きさが変化する、ということである。

## ４．ダイナミックデータ抽象

VP はオブジェクト指向プログラミングと対極をなすプログラミングモデルである。オブジェクト指向プログラミングは、データとそれを操作するプログラムとを一つにまとめてオブジェクトとし、それによってデータ抽象、継承、多相というプログラミング技術を実現した。一方、VP ではデータとプログラムを分離し、一つのデータに複数のプログラムを起動できるようにすることで、プログラムの拡張性を高めた。ところが CCC や PDL にみられるように、VP でも継承と多相を実装したプログラミング言語は実現可能であった。このことから、オブジェクト指向プログラミングと VP との本質的な違いはデータ抽象の有無にかかってくる。

データ抽象はデータの実装上の違いを吸収してプログラムの違いに現れないように、データアクセスを手続きにより行い、その手続きをデータの実装と組にするものである。同様な実装を隠蔽する抽象化は CCC においてもマクロを定義することで行うことができる。しかしこれまでのデータ抽象とは異なり動的にこれらの抽象化を行う。そこで、この新しい抽象化にダイナミックデータ抽象と呼ぶことにする（図４）。

ダイナミックデータ抽象は、すでに存在しているデータに対して、それを解釈する方法とプログラムを組にして与えるものである。データ抽象がデータ構造の定義を与えていたのに対して、解釈を与える点が異なる。解釈は実行時に行われ、解釈が成功したときだけ、そのプログラムを実行できる。解釈には条件式、パターンマッチ、構文解析などが含まれる。

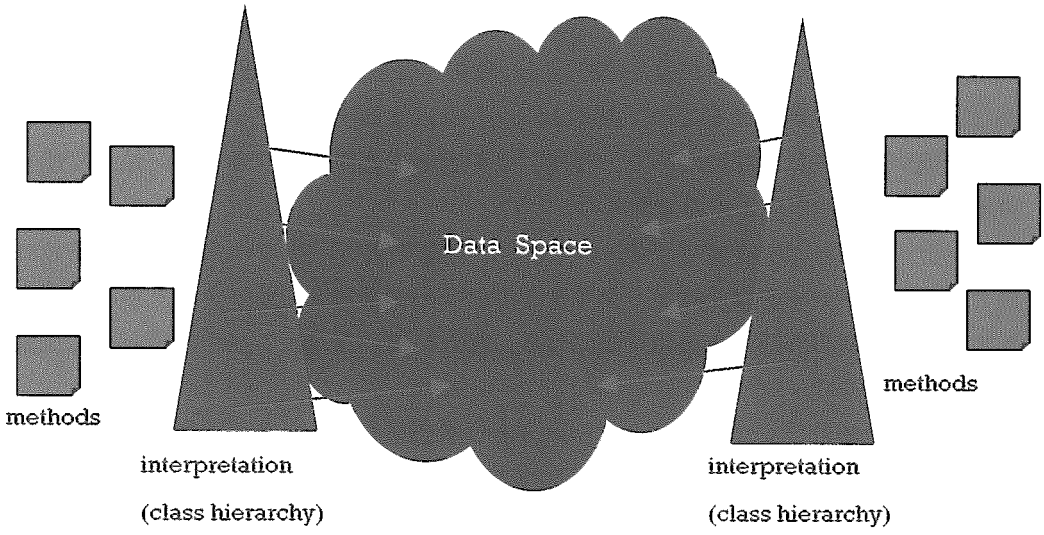


図４　ダイナミックデータ抽象

## 5．アプリケーションの試作

CCC や PDL を用いて、GUI アプリケーションを試作した。本アプリケーションは、図１の紙とペンのモデルを忠実に実装している。紙はデータの可視化とユーザとのインタラクションを受け持つ。データを画面に表示する際に、そのデータの占める領域を登録する。その情報を用いて、画面上でマウス操作などを行った際に、座標値から適当なデータを探し出す。操作の対象のデータを引数として、ペンが起動される。ペンは自分の持つクラス階層を用いて、対象のデータを解釈する。解釈に応じて、それぞれのメソッドを呼び出し、その結果対象を書き換える。

ここでペンの実装には言語を選ばないという点に注意して欲しい。複雑なパターンを必要とせずに、逆に高速できめ細かなインタラクションを望む場合には低レベルの言語 CCC が用いられる。一方、複雑なパターンマッチを行うが、その分速度を犠牲にする場合には、PDL を用いる。

全ての機能はペンによって実装される。たとえば、クリップボードにコピーするペンは、紙上に見えているどのようなデータでもコピーすることができる。ここでコピーされるデータはそのデータを特長つけている全ての情報であるから、後でペーストされたときに、そのデータを完全に復元できる。一方、オブジェクト指向で作成されたシステムでは、コピー&ペーストは一つのプロトコルとして各オブジェクトに実装されるものであるから、１）そもそもこのプロトコルが実装されていないデータはコピーできない、２）受け渡されるデータの型を合わされる必要がある、３）受け渡すデータの型ごとに操作を用意する必要がある（しかもそれが網羅しているわけでは)、などの問題がある。３）の例としては、ある文書をフォント情報込みで受け渡す場合と、それを捨てる場合などである。これらの問題はペンと紙のモデルでは原理的に存在しない。紙に格納できるデータは全ての型を許すし、受け渡すペンは必要なものを必要な数だけ作り、追加すればよいからである。

## ■ 今後の展開

今後の展開としては、基本的なツール群とライブラリを元に、実用的なアプリケーションの作成に取り掛かる。しかし、これまでもそうであったように、VP を実用的に使うためには、まだ多くの問題が残されていると予想される。それらの問題は新しいプログラミングの研究テーマとなるであろう。

言語の拡張としては、多くの課題が残されている。CCC では型などの情報を有効に利用したり、C++とより親密な言語仕様に発展させることである。パターンによるディスパッチ言語は、まだ言語の核とその周辺の機能が実装されただけである。今後、この言語で応用プログラムを記述してゆくにしたがって、言語としての機能が向上するであろう。

ダイナミックデータ抽象は本研究で考案された新しい概念である。VP の共有メモリとそれにアクセスするプロセスというアーキテクチャで、ダイナミックデータ抽象を必要とされたが、概念として取り出すことで、それは別の問題にも適用可能になる。新しいプログラミングの技法として洗練させ、提案してゆきたい。

逆参照のメモリ構造をもつシステムは、新しいメモリ構造をもつシステムである。特徴はあらかじめ想定されていなかったデータへのアクセスにも対応できる点である。しかし、その特徴を活かした利用がなされているとはいえない。個々の用途のレベルでみると、決定的な使用法があるとは考えにくく（そういう用途があるなら、そのためにデータ構造を設計すればよいからである)、それよりも、様々な局面で少しずつ利益を得るという性質のものであろう。これに関しても、今後の発展が楽しみ

である。

VP はソースコードのいらない、もしくはプログラムの詳細を理解しなくてもよい、プログラムの拡張を目指している。これは、オープンソースなどに代表される、インターネットを用いた多人数によるオープンなソフトウェア開発のモデルとして有効かもしれない。現状のオープンソースはコアとなる開発者と、その周辺の協力者という関係が主であるが、VP ではこのコアの部分を小さくすることが可能となる。

## ■ 成果リスト


1) Yasunori Harada, Kenichi Yamazaki, Richard Potter: CCC: User-Defined Object Structure in C. ECOOP 2001: 118-129

2) 原田康徳、山崎憲一 : CCC --データの内部表現に依存しないオブジェクト指向、情報処理学会論文誌 : プログラミング、Vol.42, No.SIG2(PRO 9), pp. 48-60 (2001).

3) Kenji Miyamoto, Yasunori Harada, Richard Potter: KVispatch : A Visual Language for Animation that Rewrites Kinematic Objects, Advanced Visual Interfaces 2000, 255-260.

4) 宮本健司、原田康徳 : 分散書換えビジュアル言語によるネットワークアプリケーション構築法、**コンピュータソフトウェア、Vol. 16, No. 5 (1999).**

5) 原田康徳 : 逆ポインタを持つ CONS、プログラミングシンポジウム(2000)。

6) 原田康徳 : Visibility Programming - 宣言的状態記述によるシステム記述、SPA2001。

7) 原田康徳 : データ抽象のないプログラミング、夏のプログラミングシンポジウム(2001) 。