マイクロコンピュータと
ワークステーション 55－4
（1989. 5. 16）

図的記述機能を持つオブジェクト指向言語ｖｏｃ

田代　秀一　，　岡田　義邦
電子技術総合研究所

ｖｏｃは，テキストによる手続の記述と，アイコンを結び合せることによる図的記述の２つの記述スタイルを　混在してプログラムを記述できるオブジェクト指向言語である．手続的な部分はテキストで，モジュール間の関係等は図的に記述できる為，プログラムの了解性を向上させることができる．又，教示的プログラミングにより，図形の描画等を効率的にプログラムすることができる．本稿では，c++に上位互換性を持たせて開発しつつあるｖｏｃの概要を紹介する．

VOC : an Object oriented language with visual programming facility

Shuichi TASHIRO and Yoshikuni OKADA
Electrotechnical Laboratory
1-1-4, Umezono, Tsukuba, Ibaraki, JAPAN 305

VOC is a new style object oriented programming language, which allows programming both with text expressions (one dimensional stream of characters) and visual expressions (two dimensional icon networks). By mixing these two style of expressions, understandability of program can be increased. VOC also provides for programming-by-example facilities for developping visual man-machine interface applications. This paper discusses the design of VOC which we are developping based on C++.

1．はじめに

ソフトウエアの生産性向上のためには，プログラムの記述性，了解性を高めることが重要である．

命令をアイコンという形で視覚化し，それを画面上に二次元的に配置し，結合することでプログラミングを行なおうという試みがある[1],[2]．このような言語はアイコニック言語，visual language，視覚言語，２次元言語等と呼ばれているが，プログラムの了解性が良くなる点，非熟練者に対する親和性が良い点等で従来の１次元的なテキストによるプログラミング言語に比べ有利となる可能性がある．しかし，プログラムの全てを図的に記述することは必ずしも得策ではない．論理的，あるいは手続き的な概念の記述にはテキスト的記述が適している場合が多いからである．一方，モジュール間の関係等の記述には図的記述が適している．又，アイコンとして視覚化する対象はある程度具体性を持った概念である方が適しており，あまりに抽象的なものをむりやりアイコン化しても了解性の向上につながるとは限らない．

プログラミング作業の過程における様々な局面で図的記述とテキスト的記述を使い分けられるようなプログラミング言語を使用すれば，ソフトウエア開発をより効率的に行ない得ると考えられる．

又，プログラミング過程の一部に図的記述を用いることのできる言語システムは，アイコン化したモジュールを利用者に提供し，利用者がそれを画面上で組合せることにより，最終的に目的とするプログラムを自己の手で容易に完成させられるという，今後需要が大きく伸びるであろうイージープログラミング可能なアプリケーションの開発にも適している．

我々は，図的プログラミング環境を実現する為の基本システムとしてDialog.iの研究を行なってきたが[3]，そこに載せる言語として，上記の様なプログラミングスタイルに答えるための，テキスト記述と図的記述を混在させたプログラム記述に重点を置いた新しい言語を検討している．

そのプロトタイプとして，ｃ＋＋[4]を拡張し，図的記述機能を持たせたオブジェクト指向言語であるｖｏｃ(visual object-oriented C)の開発を行なっている．

ｖｏｃでは，アイコンの仕様をテキストによって記述し，そのアイコンを画面上で結合する事により，更にプログラミングを進めて行くことができる．

又，一部programming by exampleのための機能を備えており，現在急速に需要の高まってきている図的マン－マシンインタフェースの記述にも適している．

2．システムの基本方針

ｖｏｃは以下の様な方針で設計している．

①図的表現形を持ったアイコンを，オブジェクトの一種として持つ，オブジェクト指向言語である．

②クラスの定義は，画面上にアイコンを配置し，それらを相互に接続すること（アイコンネットワーク）による図的記述と，手続を定義するテキスト的記述を混在させて行なえる（図１参照）．

③オブジェクト間の相互参照において，名前による参照先指定と，図形による参照先指定との双方が可能である．

図形による指定は，２つのアイコンを画面上で直接接触させるか，又は，線で結ぶことによって行なう．

④programming by exampleスタイルのプログラムを可能にする．

図を描く際に，プログラムで座標を指定して描くのではなく，マウスポインタで操作し，座標，大きさ等の決定を対話的に行なえると大変有利である．

アイコンの配置は，図の直接操作により座標を決定するという意味で，それ自体教示的プログラミングである．又，3.5で述

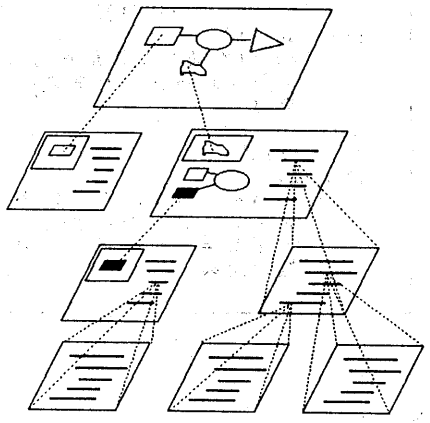

図1. 図的記述とﾃｷｽﾄ記述を混在させた
ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ

べるpermanent変数と未完成ライブラリの導入により，教示的プログラミングを可能にしている．

3．ｖｏｃの概要

3．1 クラスの定義

ｖｏｃではオブジェクトに対し，‘名前’に加えて‘形’を与えることができる．‘形’の与えられたオブジェクトを，特にアイコンと呼ぶ．

オブジェクトを定義するテンプレートであるクラスの定義は以下の物からなる．

①テキストプログラム(c++に上位互換性を持つ文法)

②表現形（このクラスをオブジェクトとして実体化した場合に与えられる図形）

③図形プログラム（定義済のオブジェクトを利用し，各オブジェクトの間を線で結んだ図形）

これらはクラス定義ウインド（c-window, 3.1.1参照）の中に記述する．

上記①②③は，全てが同時に含まれる必要はない．少なくとも一つがあれば，クラスの定義として十分となり得る．

上記②の含まれる場合，そのクラスはアイコンを定義している．そこから生れるオブジェクトであるアイコンは，他のオブジェクトから，名前によるだけでなく画面上で‘差し示す’ことにより参照することが出来る．

上記②が含まれない場合，このクラスによって定義されたオブジェクトは，名前で参照することしか出来なくなる．

3.1.1 c-window

クラスはc-windowと呼ぶウインド内部に定義する．

一つのc-window中には一つのクラス定義を行なうことができる．

c-windowの中にはv-window及びt-windowと呼ぶサブウインドをそれぞれ最大１個作ることができる．v-window内部にはアイコンの表現形を，t-window内部にはc++に上位互換性を持った言語による手続きを記述する．

c-window内部で，かつ，t-windowの外部の任意の位置に，他のc-windowで定義され，実体化（3.4参照）されたアイコンを張込むことができる．

c-windowにアイコンを張込むことは，テキスト記述部における‘オブジェクトの宣言’に相当する．

3.1.2 v-window

v-windowはこのクラスが実体化されてアイコンとなった場合，ディスプレイにどの様な形を表出するか，即ちアイコンの形を定義するためのサブウインドである．

この図形の定義のためには以下のような３通りの方法が用意される．

(1)ビットマップエディタを呼出し，直接図形を描く．

(2)t-window中のプログラムによって描く．vocには，t-window内のローカル座標に対して図形を描画する為の命令が用意されている．

(3)既に定義されたアイコンを張込む．

c-window中にアイコンを張込む場合，そ

こがv-window内部であるか外部であるかは，意味的には影響を与えない．アイコンを置いた場所がv-windowの中である場合，その形がそのまま外部に表出される．v-windowに張込んだアイコンの上には，ビットマップエディタ又はプログラムによって，更に図形を上塗りすることができる，これにより，定義済のアイコンの図形を利用し，その一部を変更することで自己の図形を定義することができる．

3.1.3 t-window

t-window中にはこのクラスの手続きを記述する．言語仕様はc++を拡張したものである．

c++との大きな違いは，外部参照をアイコンの接続という図的な記述により，クラス定義の外部で指定できる点，メンバ関数の呼出口を，ポートに対応して複数持つ点である．

v-window中にポートと呼ぶ点あるいは領域を定義して名前を与えることができ，テキスト記述によるプログラムからはその名前によりポートを参照できる（3.2参照）．

外部参照の実際の参照先を図的記述により決定する場合，テキストプログラム内部では，このポート名を仮参照しておく．

ポートに対する参照が，実際にどのオブジェクトに対して行なわれるかは，このクラスから実体化されたアイコンのポートが，画面上で他のアイコンのポートに接続されたときにはじめて決定される．

又，以下の節で述べるように，複数のポートからのメッセージを待ち合せる機能，permanent記憶クラスなどが拡張されている．

t-window内部のプログラムは，基本的に次のような形をとる．

```
class class_name{
      <ﾛｰｶﾙｵﾌﾞｼﾞｪｸﾄの宣言      >
  port(port1):
      <port1を通して外部から
        参照されるメンバ関数の
        定義                  >
  port(port2):
      <port2     〃           >
  public:
      <portを介さず，外部のオ
        オブジェクトから名前で
        直接参照されるメンバ関
        数の定義              >
};
```

3．2 ポート

ポートは，v-window内部の座標点あるいは領域に対して名前を与えることで定義する．ポートには入力ポート，出力ポート，入出力ポートの3種類がある．

例えばt-window内の宣言部で，

```
port(port1):
```

なるラベルの後に宣言したメンバ関数は，v-window内に定義されport1という名前を与えられた入力ポートあるいは入出力ポートを通して外部に公開される．

又，例えばport2という名前の出力ポートの先に接続されたアイコンのメンバ関数を呼出す場合，

```
port2.func(arg);
```

の様に記述する．

port3が入出力ポートである場合，

```
port(port3):f(){.....};   …①
```

でメンバ関数の宣言をし，

```
port3.f();                ……②
```

として外部参照を記述する．ここで，①と②で関数名が一致していたとしても，②での参照はあくまで外部参照となり，①が参照されることはない．

ポートを介してのアクセスは，相手オブジェクトのメンバ関数の名前を意識せずに行なえる方が便利である．そのため通常はoperator overloadingの機能を用い，

```
port2 << arg;
```

のように，全てのオブジェクトについて統一的にアクセスするよう，プログラムを記述することが便利であろう．

3．3 複数ポートの待合わせ機能

vocのオブジェクトは，複数のポートから同時にメッセージの受信（メンバ関数の起動）を行ない得る．

そこで，複数ポートにメッセージが揃うことの待合わせを行なう為に，semaphore型変数と，疑似関数waitを用意した．

semaphore型の変数は，０又は１の２つの値を持つ．

疑似関数waitは，

```
wait(s1, s2, ..., sn){
 ...
}
```

の様に任意個のsemaphore型変数を引数として持ち，引数に指定された変数の値が全て‘1’になると起動される．

wait疑似関数が起動されると，引数に指定された全てのsemaphore型変数の値は‘0’にリセットされる．

２つのポートにデータが揃ってから，ある処理を開始したい場合，

```
class xx{
    semaphore s1,s2;
    int a,b;
      ....
    port(port1): f(int);
    port(port2): f(int);
}

xx::port1::f(x){
    a=x;
    s1=1;
}
xx::port2::f(x){
    b=x;
    s2=1;
}
xx::wait(s1,s2);
    <aとbの値を使う処理>
}
```

の様に記述する．

なお，wait疑似関数が起動されるのは，最後のsemaphore変数の値に１をセットしたメンバ関数の実行が終了し，リタンする直前となる．

3．4 オブジェクトの実体化と利用

c++では，クラスの定義に基づいてオブジェクトが生成されるのは，オブジェクトの宣言されたスコープ内部に実行が移ったときとなっている．

ｖｏｃでは，一般のオブジェクトの生成はc++と同様であるが，アイコンに関しては異なる方式を採用している．

c-window内部でクラスを定義し，c-windowに用意されている「実体化」ボタンをマウスでクリックすることにより，アイコンを生成できる．

こうして生成したアイコンは，別のc-window中に移動して貼り込み，利用することができる．

あるc-window中にアイコンを貼り込むという事は，そのクラスのプライベートなオブジェクトとして，そのアイコンを宣言したことに相当する．

従って，t-window中のプログラムからは，

貼り込まれたアイコンに定義されたpublicなメンバ関数は名前によって参照することが可能である．

3．5 permanent記憶クラス

ｖｏｃにはpermanent記憶クラスを用意した．permanent記憶クラスの変数は，プログラム全体の実行が終了するとファイルとして保存される．

再びそのプログラムが実行されると，permanent変数はファイルからロードされ，前回実行終了時の値が復活される．

プログラムが初めて作られて実行されたとき，即ちpermanent変数の値を保管したファイルが未だ存在していない場合，その初期値はUNDEFという値をとる．

permanent変数は，教示的プログラミングの為に応用することができる．

即ち，例えば，長方形を描くプログラムにおいて，出力する長方形の大きさを教示的にプログラミングしようとする場合，

```
permanent  height, width;
...
if(height==UNDEF){
  create_rectangle(&height,&width);
}else{
  write_rectangle(height,width);
}
```

のようなスタイルでプログラムを作成することができる．

このプログラムを作成した直後の最初の実行では，create_rectangleなる関数が呼ばれ，動的図形表示とマウスによる対話により，長方形の大きさを決定する．次回の実行以降からは，ここで決定した大きさの長方形が表示される．

ｖｏｃのライブラリとして，上記のようなスタイルにより，初回実行時に人間との対話によって各種パラメータを決定して行くような，教示的プログラミングを行なう為のライブラリを各種揃えておくことが便利である．このようなライブラリを未完成ライブラリと呼ぶ．

4．ｖｏｃプログラミング環境

ｖｏｃの処理系は，プログラミング環境と実行環境を一体化した環境を提供する(図2参照)．

ｖｏｃにおけるプログラムとは，オブジェクトの集合である．あるプログラムを構成する全てのオブジェクトを含むウインドがworkspaceウインドである．

workspaceウインド内部には，system-library，user-library，local-libraryの3種類のライブラリウインドを開くことができる．各ライブラリウインド内部には定義済みのクラス名がアルファベット順に表示される．

system-libraryは，システムにあらかじめ用意してある基本的ライブラリ，user-libraryは，利用者が定義して保存したもの，local-libraryはこのプログラムに固有のクラス定義を保持したライブラリである．

プログラミングはworkspaceウインドのc-windowボタンを選択してクラス定義ウインド(c-window)を生成し，そこへテキストや図形を定義することで進めて行く．

ライブラリウインド内部の項目をマウスポインタで選択することによりそれを定義したc-windowを呼出すことができる．

プログラムの実行は，c-window単位で部分的に行なうことができる．c-windowのrealizeボタンをマウスで選択することでそのクラスのconstructorの実行が行なわれる．

メインプログラムという特別に分類されたc-windowは持たないが，メインプログラムに相当する最上位のc-windowのconstructorには，プログラム全体の実行の引金となるようなプログラムが記述される必要がある．

5．終りに

図的記述とテキストによる記述を混在させてプログラムを記述することのできるオブジェクト指向言語ｖｏｃについて述べた．

ｖｏｃを用いると，手続的部分はテキストにより記述し，モジュール間の関係などは，アイコンの結合により視覚的，２次元的にプログラミングするというように，それぞれ，人間にとって理解しやすい記述法を選んで記述することができる．

また，図的記述によりプログラムを記述できる点は，図的マン－マシンインタフェース，あるいは図的なイージープログラミング機能を持つアプリケーションを記述する場合に都合が良い．図的な表現を，図を直接に扱うことで，教示的に，直接に記述することができるからである．

ｖｏｃは，現在Sun3/160，Ｘwindow V.11 R3，g++の環境下で試作を進めている．

ｖｏｃは，オブジェクトが並行に実行される並行動作システムである．又，図形処理と記号処理の混在したシステムである．このようなシステムを効率的に実行するために，マルチプロセッサ，ライトウエイトプロセスによる分散システムへの適用を計画している．

謝辞

本研究の機会を与えていただいた棟上情報ｱｰｷﾃｸﾁｬ部長，有益な助言をしていただいた分散ｼｽﾃﾑ研究室の諸氏に感謝する．

文献

[1]Proceedings of IEEE Workshop on Visual Languages(1988).

[2]Shu, Nan C. :"Visual Programming", Van Nostrand Reinhold Co. New York (1988).

[3]田代，岡田:"ｱｲｺﾆｯｸ言語Dialog. iによるCAD用ﾏﾝ-ﾏｼﾝｲﾝﾀﾌｪｰｽ"，情報処理学会ﾏｲｸﾛｺﾝﾋﾟｭｰﾀとﾜｰｸｽﾃｰｼｮﾝ研究会，1988-5

[4]Stroustrup, B. :"The C++ Programming Language", Addison-Wesley(1987).

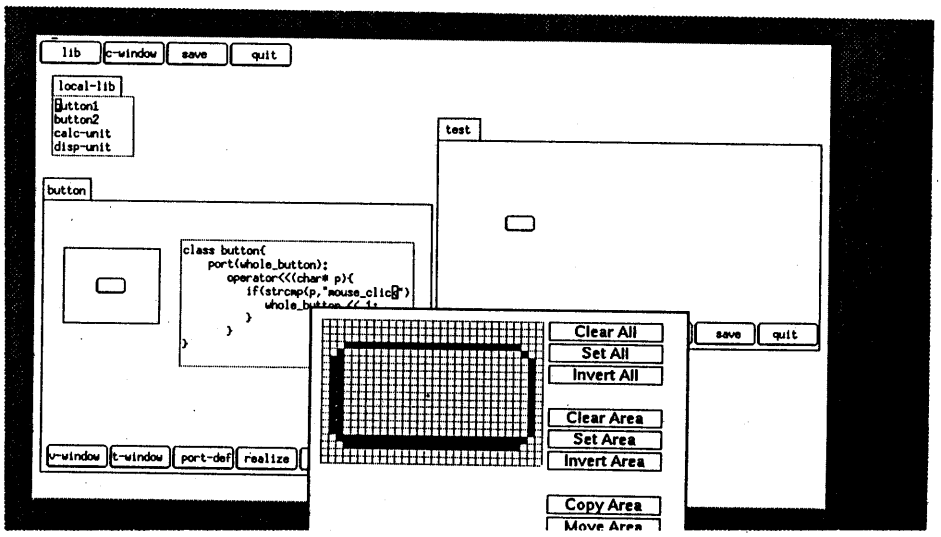

図2．ｖｏｃに於けるプログラミングの様子