ソフトウェア工学 97－4
（1994．3．1）

オブジェクト指向パラダイムに基づく プログラミング言語の視覚化

秋山憲一，上田賀一

茨城大学

〒316 茨城県日立市中成沢町 4-12-1

近年，エンドユーザコンピューティングが盛んになり，これまでコンピュータやプログラミング言語にあまり接することのなかった初心者がプログラムを作る機会が多くなっている．しかし，現存のプログラミング言語は多様化，複雑化し，初心者にとって扱いにくいものになっている．

そこで，本研究では初心者でも容易にプログラムを作成できるようなプログラミング言語の開発を目的とする．開発にあたり，視覚的プログラミングとオブジェクト指向パラダイムの考え方に着目し，これを基盤としたプログラミング言語の設計とその開発支援環境の試作を試みる．また，ユーザの能力に応じたプログラム作成レベルを設定し，この方針に沿った環境とすることを考える．

A Visual Programming Language based on Object-oriented Paradigm

Ken'ichi Akiyama, Yoshikazu Ueda

Ibaraki University

4-12-1 Nakanarusawa, Hitachi, Ibaraki, 316 JAPAN

Recently, an end-user computing is coming popular. It is many occasions that an apprentice programmer has to make programs with a computer programming language. The present languages, however, are so various and complex that beginners can not handle them sufficiently.

Therefore, we aim at the development of a visual programming language which even beginners can make program easily with. We design the language and make a prototype of its support environment based on the concepts of a visualization and object-oriented paradigm. And we consider the programming level in support environment according to the level of user's skill.

1．はじめに

現在、コンピュータは様々な分野で利用されている。そのコンピュータには、機能を極限まで引き出したソフトウェアが要求されてきた。そのため、プログラミング言語をより使いやすいものへと発展させるより、プログラマ一人一人のプログラミング技術の向上に重点が置かれてきた。その結果、プログラミング言語は多様化、複雑化し、扱いやすいとは言えないものになった。

ところが、近年ではエンドユーザが自分の望む環境を手にするため、自らプログラミングを手懸けることが少なくない。さらに、近年の技術向上により、コンピューティングコストが劇的に低下し、ソフトウェアでの機能向上を図らずとも一段上のハードウェアを用いることで要求する機能を満たすことが可能となってきている。

エンドユーザが自分で必要とするプログラムを作成できるようになれば、これまでユーザとプログラマの間に存在した食い違いもなくなり、ユーザは自分が望んだ通りの環境を手にすることができる。また、そのようになればプログラマはハードウェアの機能を極限まで生かしたプログラムの生産に力を注ぐことができる。

本研究では、これらの理由のために望まれている、初心者でも容易にプログラムを作成できるようなプログラミング言語の開発を目的とする。

現在のプログラミング言語について捉えてみたとき、何が初心者に対して分かりにくいかを考えてみると、プログラムを始めから一つ一つ順を追っていかなければ何をしようとしているのか分からない、一つ一つの命令は分かるが全体として何をやっているのか理解できない、などの点が挙げられる。また、プログラムというものはその場で使い捨てにすることなく、部分的な機能やプログラム全体を保存しておき、再利用すべきものであるが、初心者には再利用をどのように行っていけばいいのか分からないのが普通であろう。

このような点を踏まえ、ここでは視覚的プログラミングとオブジェクト指向の考え方に着目し、プログラムを理解しやすく作りやすいプログラミング言語と、再利用を行いやすいプログラミング環境の構築を試みる。

2．視覚的プログラミング言語

2．1　視覚的プログラミング言語の特徴[1]

プログラムを作りやすいプログラミング言語を開発するという試みは最近になって始められたものではない。長年、人々は容易に扱えるプログラミング言語を手にするため、既存言語の改良や新しい言語の設計を重ねてきた。その結果、様々なプログラミング言語が生み出され、初期の頃にあった言語に比べると格段に扱いやすいものになっている。しかし言語設計は進化し続けていくものであり、大きな改革と呼べるようなものはなかった。言語構造は相変わらず一次元的で文章型のままであった。

これに対し、視覚的プログラミング言語は先に述べたプログラミング言語の進化の過程から逸脱している。プログラムの記述に図的な表現を使うことによって文章型の記述方式から離脱し、同時に線形的記述からも離れている。

最近、グラフィクス関係のハードウェアやソフトウェアの低価格化により、絵をコンピュータとの対話手段として用いることが可能となってきている。

2．2　視覚的プログラミング言語の分類[1]

視覚的プログラミング言語は設計原理に基づき、

a）アイコンを中心とした言語
b）フローチャートなどを利用した言語
c）表形式の表現を使用した言語

の三つに大別される。

この中で、b）のフローチャートなどを取り入れた言語は、言語というより既存のプログラミング言語のためのチャートエディタのようになってしまう。c）の表形式の言語はデータ処理用など

の分野においては力を発揮するが、プログラミング言語としては扱いにくい。

そこで本研究では、この中から視覚的プログラミング言語として最も利用し易いと思われるアイコン記述形式のプログラミング形式に目を向け、これに従った視覚的プログラミング言語を構築する。

2．3　アイコンを利用した視覚的表現系

アイコンを使用した視覚的表現系（プログラミング言語を含む）は以下のように大別される。

形式1．アイコン（部品）＆フロー（線）形式

アイコンとフローの二種類のパーツを使った記述形式。

形式2．特殊アイコン＆フロー形式

形式1の特殊な形でアイコンの位置がある程度確定している形式。

形式3．アイコンのみの形式

アイコンのみを用い、フローを使用しない形式。

形式4．アイコンに意味のない形式

形式3の特殊な形で、アイコンが意味を持っていない形式。

この分類を見ると、形式1～3と形式4の間に非常に大きな隔たりがあることが分かる。すなわち、形式1～3が何らかの意味を持ったアイコンをつなぎ合わせて一つの表現形態を作るのに対し、形式4は全く意味のないパーツのみを用いてある意味を持った表現を構成しなければならない。

この二種類の表現形式（形式1～3と形式4）のうち、後者はプログラミング言語を含め、表現系として用いるには事実上不可能である。その理由として、記述した表現がそれを見る人によって全く違ったものとして捉えられてしまう、という点が挙げられる。

前者の方法にはこのような問題点は存在しない。そのため表現系を構築するには前者の方が向いていると言える。

前者の内で、形式2と形式3は形式1の特殊な形として表せる。そこで、ここで開発する視覚的プログラミング言語は形式1を使用して構築することにする。形式1はアイコンとフローを自由に配置して表現（プログラム）を記述することができる。これは初心者にとっては戸惑いを覚えることかも知れない。しかし、始めから制約を持った表現系（プログラミング言語）よりも自由度の高いものの方が将来的に見て望ましいと考え、形式1を取り上げることにした。

2．4　これまでの視覚的言語との相違

視覚的プログラミング言語はこれまでにも数種類のものが開発されている。これらは視覚的プログラミング言語のプログラムを視覚的に記述する特性を利用し、プログラムの記述に重点を置き、プログラムの実行に関しては二次的に考えられている。そのため、プログラムがどのように動くのかを示すトレーサは存在するが、インタプリタやコンパイラなどを使って実行するものではない。そのため、プログラムの学習用として使用するには向いているが、実際に実行させるプログラムを作るには不十分であった。

本研究で開発する視覚的プログラミング言語はプログラムを視覚的に記述するのみに留まらず、記述したプログラムをコンパイルして実行可能形式に変換することまで考える。すなわち、ここで開発する言語は、プログラムの記述だけでなく実行にも重点を置いた視覚的プログラミング言語である。

3．オブジェクト指向パラダイム

3．1　プログラムの再利用とオブジェクト指向

プログラムを作成するとき、過去に作成したプログラムの一部あるいは全部を利用すると効率よく作成できることがよくある。このようなことか

ら、再利用の概念が生まれ、汎用的に使えそうなプログラムはライブラリとして保存しておくなどの方法が採られてきた。しかし、初心者が後に再利用することをあらかじめ考えてプログラムを作るのは極めて困難である。

プログラムを再利用するための一つの有効な手段としてオブジェクト指向がある。オブジェクト指向ではオブジェクトを一つの単位としてプログラムを記述する。そして、各々のオブジェクトはカプセル化され、他のオブジェクトから完全に独立して動作する。そのため、一つのオブジェクトを他のプログラムに持っていっても何の支障もなく動作する。つまり、オブジェクトを利用することによって容易に再利用を実現することができる。

3．2　視覚的プログラミング言語とオブジェクト指向

アイコン指向の視覚的プログラミング言語を考えた場合、アイコン間を線（フロー）でつないだネットワーク状の記述形式となることが予想される。このような記述形式の場合、プログラムの動作はネットワーク状に配置された部品（アイコン）間の通信により進んでいく形が適当であると考えられる。

そこで、現存のプログラミングパラダイムの中から通信によるプログラム進行に適したもので最も一般的であると思われるものとしてオブジェクト指向型プログラミングパラダイムが考えられる。オブジェクト指向のプログラムはオブジェクトとオブジェクト間のメッセージ通信で記述される。オブジェクトをアイコンに、メッセージ通信をアイコン間のフローに割り当てることで自然にアイコン指向視覚的プログラミング言語として適用できる。

以上の二つの点を考慮に入れ、プログラミングパラダイムとしてオブジェクト指向パラダイムを取り上げ、これを取り入れた視覚的プログラミング言語を設計する。

3．3　クラス階層

クラス階層は図１のように構成した。このクラス階層に示したクラスはシステムが予め持っているクラスで、ユーザはこれらのクラスを使用して新しいクラスやプログラムを作成する。作成したクラスやプログラムはこのクラス階層に追加される。なお、クラス階層の構成には Smalltalkのクラス階層を参考にした[2]。

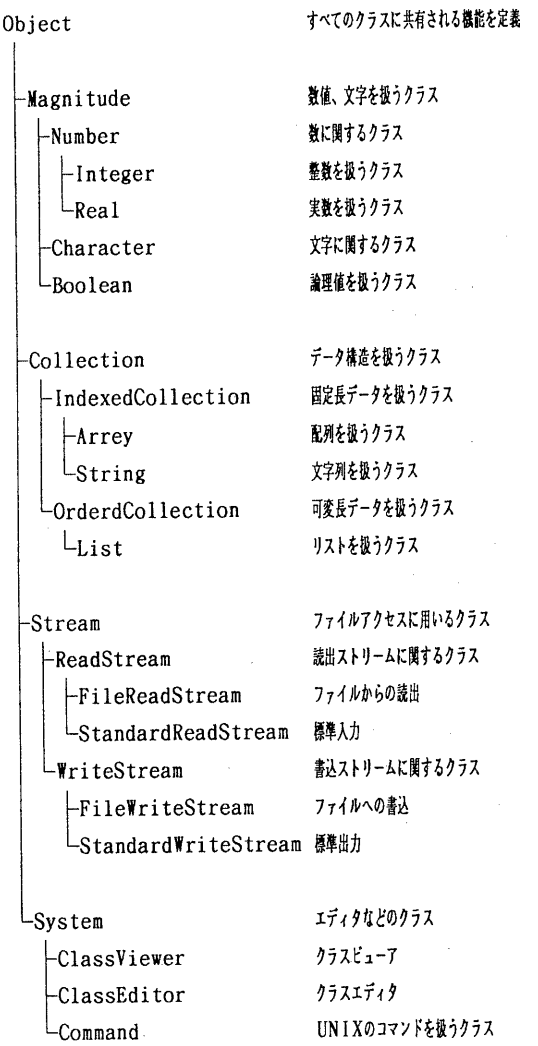


図１．クラス階層

4．ユーザプログラムの設計基盤

4．1　基本クラス

前章で述べたように、ここではオブジェクト指向パラダイムに基づいて視覚的プログラミング言語を設計する。オブジェクト指向に基づいたプログラミングでは、プログラムや新しいクラスはそれまでに作られているクラスから生成されるインスタンスを組み合わせることによって作成される。しかし、プログラムやクラスを記述するための最も基本的なクラスは他の方法であらかじめ与えておかなければならない。ここでは後述するように既存言語への変換という形で実行形式への変換を実現するので、変換する既存言語を用いて基本クラスの記述をしておく。

4．2　ユーザレベルに対応したプログラム作成

本研究では、初心者がプログラムを容易に作れるようなプログラミング言語の開発であることは始めに述べた通りである。しかし、上で挙げたような基本的なクラスのみを使ってプログラムを作ることは初心者には難しい。

そこで、オブジェクト指向と再利用の特性を用い、上級者が予め特定の機能を持った有用なクラスを作り、クラス階層に加えておく。初心者はこのクラスを利用して基本的なクラスを必要とすることなく、目的のプログラムを作ることができる。さらに、作成したプログラムをそのままクラスとしてクラス階層に加えることによってさらに使い易い環境を整備できる。

実際には初心者と上級者という二つのユーザ階層ではなく、更に数段のユーザレベルを設け、初心者が使い易いものへと徐々にクラスを具体的にする、ということができるような環境を整える。

図２に例として１～１０までの整数の和を求めるプログラムを作るときの各ユーザレベルにおいて作成するプログラムを模式的に示す。

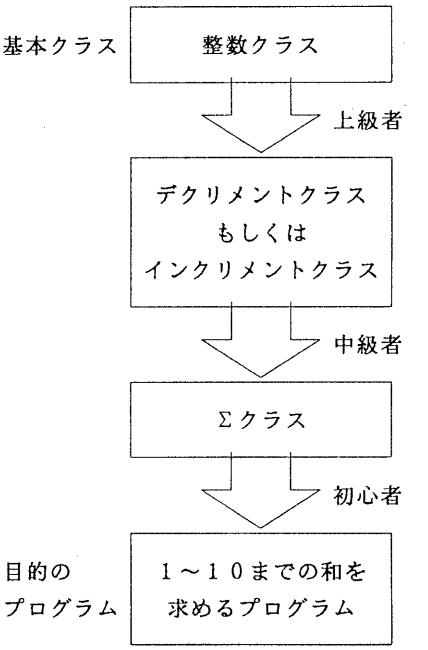


図２．ユーザレベルに応じたプログラム作成

4．3　テキストによるプログラムの記述

それほど複雑でないプログラムやクラスを記述するには視覚的言語よりもテキスト言語の方が簡単なことが多い。特に、上級プログラマであるほどこのことが言える。

このようなことを考え、プログラムをテキストでも記述できるようにテキスト用の文法も用意する。

4．4　メッセージの記述形式

オブジェクト指向プログラミングパラダイムの重要な要素としてオブジェクトの他にメッセージがある。この研究でのオブジェクトはメッセージを受け取ると適当な処理を行い、次のオブジェクトに送るメッセージを出力する、という動作をするので、基本的にユーザがメッセージを直接指定する必要はない。オブジェクトが生成するメッセー

ジに関わらず特定のメッセージを指定したい時には、メッセージを表すフローにメッセージ名を直接記入してメッセージを強制的に変更する。

4．5　プログラムの実行

プログラムは記述しただけではその能力を発揮できない。何らかの形で実行させる必要がある。ここでは、実行効率やシステムの依存性などを考慮に入れ、汎用的な既存言語への変換を行い、その言語のコンパイラを使ってコンパイルするという方法を取る。変換先の既存言語としては、汎用性の極めて高いＣ言語の拡張言語であり、オブジェクト指向言語でもあるＣ＋＋言語を取り上げることにする。

5．ユーザプログラムの表現と動作の仕組み

5．1　オブジェクトの表現と動作

オブジェクトは図３のような構造になっている。

オブジェクトは通常複数のメソッドを持ち、それぞれのメソッドは条件部とメソッド本体から成っている。一つのメソッドは一つのメッセージに対応した動作を表す。そのため、オブジェクトは受け付けられるメッセージの数だけメソッドを持っていることになる。

条件部は、メソッドがメッセージを受け取ったときにメッセージ引数やその時点のオブジェクトの属性の状態を調べ、このメソッドを実際に動かすかどうかを決定する。動作するときにはメソッド本体を実行し、しないときにはエラーメッセージを返す。

メソッド本体はメソッドの実際の動作を記述したものである。実行が終了したときには、何らかのメッセージを生成してそのメッセージを出力する。

オブジェクトは他にメッセージの入力端子をもっている。これは、このオブジェクトに動作が移ってきたとき、メッセージによって定められたオブジェクトを起動する役割を担う。

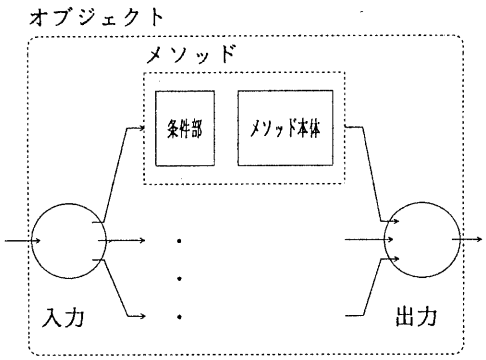


図３．オブジェクトの構造

5．2　プログラムの動作

プログラムは、あるオブジェクトが動作することにより動き始める。一つのオブジェクトの動作が終了すると、そのオブジェクトが生成したメッセージを次に実行されるオブジェクトへと送り、プログラムの実行を続けてゆく。一つのオブジェクトから同時に複数のオブジェクトにメッセージを送る場合、メッセージを受け取った複数のオブジェクトが並列実行される。前節で述べた条件部でメッセージが不適合と判断されると、その部分のプログラムの流れは停止する。また、あるオブジェクトからメッセージが外に出ていない時にも停止する。このようにして、全てのメッセージ流がなくなったときにプログラムの実行が終了する。

プログラムは実際には図５のように動く。プログラム全体を管理しているプログラムマネージャが存在し、プログラムのオブジェクト接続情報に従って、動作させるべきオブジェクトにメッセージを送る。各オブジェクトの生成したメッセージはそのオブジェクトからのリターンメッセージとしてプログラムマネージャが受け取り、これを再び接続情報に従って次に実行するオブジェクトへと送る。

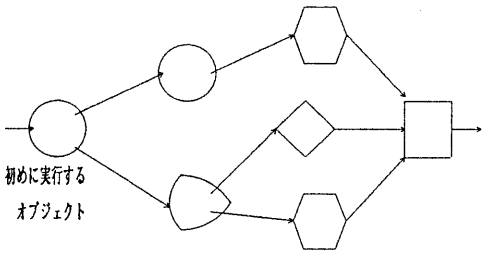
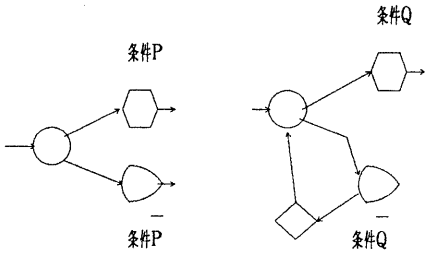


図４．プログラムの流れ



ａ．選択構造の表現　　ｂ．反復構造の表現

図６．制御構造の表現

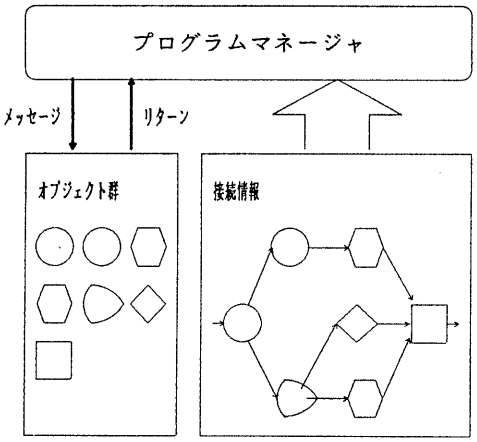


図５．プログラム動作の実際

５．３　制御構造の表現

プログラムを実行する上で必要になってくるのが選択や反復などの制御構造である。

ここで開発する視覚的プログラミング言語では、先に述べたオブジェクトは条件に合致しないメッセージを受け取ったときには動作しない、という特徴を利用し、制御構造を実現する。具体的に言うと、あるオブジェクトから二つのオブジェクトへとメッセージを送り、その二つのオブジェクトの条件部が互いに相反するようなものになっていれば選択構造が実現できる（図６ａ）。同様にして、二つのオブジェクトの出力メッセージのうち、一つをもとのオブジェクトにつなげば反復構造を実現できる（図６ｂ）。

この表現法により、制御構造を専門に扱うクラスを定義することなく、視覚的に制御構造を記述することが可能である。

６．システムの設計

６．１　システム構成

システムは図７のような構成を取る。プログラムやクラスはエディタで記述し、ユーザクラスライブラリに格納される。記述に使用するクラスはシステム／ユーザの各クラスライブラリから供給される。記述したプログラムはコンバータを通すことによって既存言語（Ｃ＋＋）のソースファイルに変換される。

以下の節ではエディタ、コンバータ、クラスライブラリについての説明を行う。

６．２　エディタ

本研究で開発するプログラミング言語は視覚的に記述するため、これまでのプログラミング言語のようにプログラミングにテキストエディタを使用する訳にはいかない。そこで、アイコンの配置とアイコン間に線を引く機能を持った専用のプロ

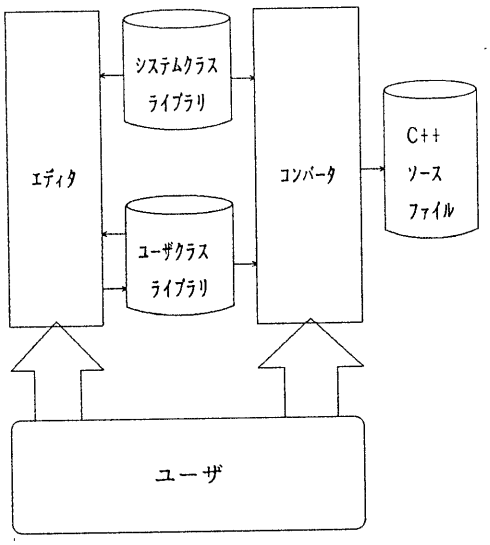


図７．システム構成

グラム記述用エディタを作成し、プログラムの記述に使う。

６．３　コンバータ

プログラムは先に述べたようにＣ＋＋言語のソースファイルに変換してからコンパイルする。コンバートは基本クラスをＣ＋＋のソースファイルとして記述しておくことによって実現する。基本クラス以外のクラスは基本クラスや基本クラスから作られた他のクラスから作られるので、基本クラスのみＣ＋＋との対応を取っておけばプログラム全体をＣ＋＋に変換できる。

６．４　クラスライブラリ

クラスライブラリはクラス階層やクラスそのものを格納しておくファイルで、システムクラスライブラリとユーザクラスライブラリの二つが存在する。

システムクラスライブラリはあらかじめ用意してあるクラスで、基本クラスもこのライブラリに含まれる。ユーザクラスライブラリはユーザが作成したクラスを格納する。

７．まとめ

７．１　考察

今回用意したクラスは基本的なものばかりなので、初心者が使うためにはその前に上級者の手によってクラス階層をある程度整えなければならない。しかし、クラス階層が充実すれば、初心者にもプログラム作成が容易なプログラミング言語であると考える。

また、ユーザレベルを考慮するため、必ずしも視覚的言語レベルにこだわらずテキスト言語レベルの利用を可能にした。さらに、再利用の観点からオブジェクト指向のクラスライブラリだけでなく、ＵＮＩＸのコマンドなどをカプセル化したオブジェクトとして利用できるように考慮した。

７．２　今後の課題

現在のシステムでは複数のオブジェクトにメッセージを送るときにはオブジェクトを並列に実行させず、メッセージ条件の合致した初めのオブジェクトのみを動作させるようになっている。これを実際に並列に実行させることが大きな問題点として挙げられる。

また、これ以外にも

・クラス階層の充実
・より扱いやすいエディタの開発
・より高性能なコンバータの開発
・デバッガの開発

といった問題を解決することにより、言語だけでなくプログラミング環境もより扱いやすいものとすることができるであろう。

参考文献

[1] Nan C. Shu著, 西川博昭訳：
ビジュアル・プログラミング, 日経ＢＰ社

[2] 加藤木和夫著：Smalltalk/Vによる
オブジェクト指向プログラミング, 日刊工業新聞社