ソフトウェア工学 96－4
（1994．1．20）

オブジェクト指向言語のための視覚的プログラミング支援環境

萩庭 崇　永田 守男
慶應義塾大学理工学部管理工学科
E-mail: hagi@ae.keio.ac.jp, nag@ae.keio.ac.jp

オブジェクト指向技術が普及中の現在、既存の言語によるソフトウェア作成の経験があってもオブジェクト指向に関しては初心者の人が多く存在し、そのような人のための環境は整っているとはいえない。

そこで、本研究では、オブジェクト指向に関する初心者を対象とするオブジェクト指向プログラミングの支援環境を提案し、試作した。オブジェクト指向言語 Smalltalk を取り上げ、コーディングの場面での支援を考える。特に初心者が把握しづらいと考えられるクラスの階層構造に注目し、マウスで図を描きながら視覚的にクラス階層を作成、変更できる支援環境を試作した。また、メソッドの誤った再定義を監視、警告する支援も同時に行った。これらの機能に関して評価実験をし、有効性を確認した。

A visual environment for object-oriented programming

Takashi Haginiwa and Morio Nagata

Dept. of Administration Engineering
Faculty of Science and Technology, Keio University
3-14-1 Hiyoshi, Yokohama 223, Japan

Object-oriented programming requires different skills from those of traditional structured programming. Thus, a good environment for beginners of object-oriented programming should be provided.

We have designed and implemented a visual environment of object-oriented programming for beginners.

If programmers draw a diagram of the tree of the hierarchy of classes visually by using our tool, the tool creates automatically the relationship between superclasses and subclasses.

Moreover, in order to prevent careless mistakes to override methods, the prototype environment in the Smalltalk language checks written methods.

1 はじめに

オブジェクト指向技術を使えば各モジュールをオブジェクトとして独立させやすく、その結果、プログラムの生産性と品質を向上させることができるものと期待されている [3]。このようなオブジェクト指向プログラミングに関して、支援環境がこれまでも提案されてきたが [2]、それらはオブジェクト指向の考え方に習熟した技術者に向いたものが多かった。しかし、オブジェクト指向プログラミングにおいては旧来と異なる新しい概念が導入されており、従来の言語によるソフトウェア作成の経験が豊富な技術者であってもオブジェクト指向に関しては初心者である人が多く存在する。ところが、そうした初心者に対する支援環境は十分に整備されているとはいえないのが現状である。

そこで本研究では、オブジェクト指向プログラミングに関する初心者を対象とし、特にオブジェクト指向言語 Smalltalk[1] を取り上げ、分析、設計が終わった後のコーディングの場面を想定した支援環境を提案し試作する。支援環境は視覚的でインタラクティブなものとし、支援環境自体も Smalltalk で作成する[1]。視覚的な支援環境は、近年のビジュアル・プログラミング [4, 5, 7] が目指している方向でもあり、初心者に分かりやすいと考えた。支援対象の言語として Smalltalk を取り上げたのは、オブジェクト指向の考え方を追求した言語である Smalltalk で有効な手法は、他のオブジェクト指向言語においても有効だと考えるからである。また、実装言語として Smalltalk を選んだのはインタラクティブな環境を作成するためと、十分なクラス・ライブラリが揃っていると判断したためである。

オブジェクト指向言語にはクラス階層があり、各クラスの階層構造の把握は慣れないうちは困難である。そこでマウスを操作して、クラス階層を図として描いていくことによってプログラミングを行う視覚的な環境を作成した。テキストだけの情報より、グラフィックを用いた図の方が関係構造の表現を容易にすると考えるからである。

さらに、図だけでは表現できない関係構造の支援も行った。オブジェクト指向言語におけるメソッドの再定義は、オブジェクト指向の重要な特徴の一つである継承を生かす特性である。しかしクラス階層の認識やクラス・ライブラリの知識が不十分な場合には、プログラマが不適切な再定義を行ってしまう可能性がある。そこでそのような場合にはプログラマに警告し、誤りの可能性を排除するようにした。

本稿では、作成した支援環境の以上のような特徴と、さらにこれを利用した実際のプログラミングにおける実験を通した評価について述べる。

2 クラス階層作成の視覚化

2.1 視覚化の目的

オブジェクト指向プログラミングにおけるクラスによる抽象化と、継承によるクラスの階層構造は、モジュールの拡張性と再利用性に貢献し、非常に重要である。しかし、初心者がテキストのみの情報からこうしたクラス階層を的確に把握し、さらに自分で新しいクラスを作成することは困難であると考えられる。 Smalltalk においても、クラス定義は通常システム・ブラウザ (図 1) によって行なわれ、階層構造はテキストで表示される。

そこでこうしたオブジェクトの関係構造の把握の困難さを排除するために、クラス階層作成の視覚化を行うことにした。つまり、図を描くことによってクラス階層を作成できる環境を目的とした。クラス階層は木構造になっていてツリー状の図として自然に表現することができ、また、こうした構造を把握する際に頭の中で同様の図をイメージしているとすれば、視覚化によってクラス階層の把握を容

[1] Objectworks\Smalltalk,release4.1(ParcPlace Systems inc.), 日本語機能 kit\Smalltalk release1.2, ObjectBits\Smalltalk ver.2.0 (Fuji Xerox Information Systems Co., Ltd.) を使用している。

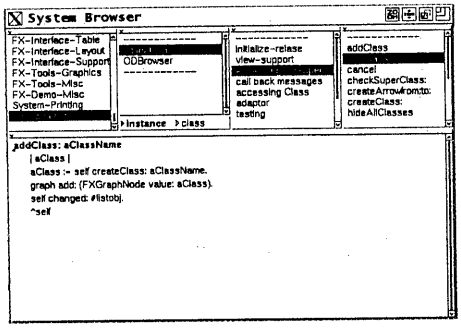

図 1: システム・ブラウザ

易にすることができ、新しい階層構造を構築する支援となるはずである。

2.2 従来の視覚的環境について

クラス階層の木構造を視覚化するツールはいくつか存在し、Smalltalk だけではなく、Objective-C や C++用のものもある。こうしたツールでは、確かにクラス階層を視覚的に表現しているし、そのクラスのメソッドを見ることができるようになっているものもある[8]。

ただし、これらはあくまでも既存のクラスを表示するものであり、再利用のためのクラスの検索やライブラリの把握を目的としたものである。こうしたツールでは図をその場でユーザーが修正することはできず、階層の視覚化はできていてもクラス作成の作業そのものは従来のままで、初心者のクラス作成の支援としては不十分である。

一方オブジェクト指向(クラス階層の視覚化)に限らない分野でのビジュアル・プログラミングの研究や商品についての報告もある。例えば、Pict[5] システムはアイコンの組み合わせによってプログラムを作成する。Tinkertoy[4] は Lisp に基づいた実行可能なグラフィックスである。また、PROGRAPH[7] は図的表現を用いた関数的なデータフロー指向言語である。

こうした言語は、プログラムの制御構造をアイコンの組み合わせによって表すことを主目的としており、本研究のようなクラス階層の概念を扱うわけではない[2]。しかし、将来、我々のシステムの中でクラスの中のメソッドを視覚化するような時には、これらの研究が参考になるかもしれない。

2.3 視覚化の実現方法

クラス階層の木構造の視覚化は、それをツリー状に図2のようにグラフィックで表示することで行う。

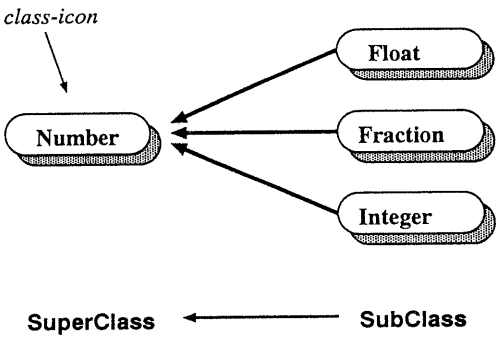

図 2: クラス階層の視覚化の例

クラスは長方形の中にクラス名を表示し、クラス間の関係構造はその間を矢印でつなぐことで表現することにした。このクラスを表す長方形のグラフィックのことを**クラスアイコン**と呼ぶことにする。基本的には画面の左側がスーパークラスでその右側にサブクラスが位置するものとする。クラスアイコンはマウスで選択し、ドラッグすることで自由にウィンドウ内の位置を動かすことができる。これは Macintosh などのドロー系のツールでよく使われる操作方法である。各クラスに対する操作は、基本的に一つのクラスをマウスの左ボタンで選択し、中ボタンでポップアップメニューを呼び出していろいろな機能を選択するという形にした。

例えばプログラマが ClassB のサブクラスとして新しいクラス ClassE を作成しようと

[2]PROGRAPH はクラス階層を持っている。

したとする。この時まず、メニューから **"新しいクラスを作る"** という項目を選択し、クラス名 "ClassE" を入力する (図 3)。

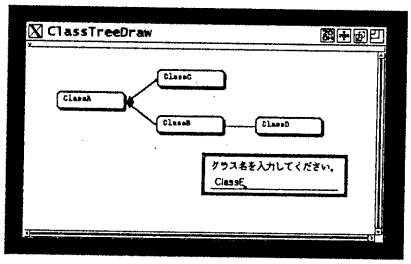

図 3: クラス名の入力

クラス名を入力し終ると、画面に ClassE を表すクラスアイコンが表示される。Smalltalk ではすべてのオブジェクトはクラス Object を頂点とした木構造になっているので、スーパークラスを決めていないこの時点では、このクラスは単に図形としての意味しか持っていない。

次に、メニューから **"スーパークラスを決める"** という項目を選択し、スーパークラスにしたい ClassB のクラスアイコンをマウスで選ぶ (図 4)。

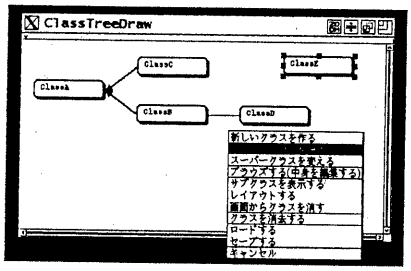

図 4: メニューの選択

スーパークラスを選択し終ると、ClassB と ClassE の 2 つのクラスアイコン間に矢印が描かれ、クラス・サブクラスの関係が画面上にグラフィックで示される (図 5)。

ここで重要なのは、ClassB と ClassE の 2 つのクラスアイコンが矢印でつながれた時点

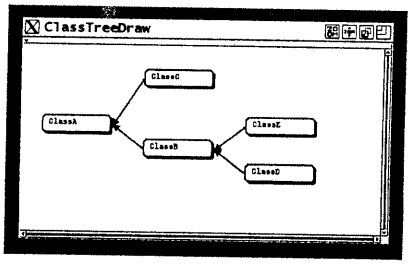

図 5: ClassE の成立

で新しいクラスとその階層関係が Smalltalk に**自動的に定義される**ということである (クラス・カテゴリーはデフォルトでスーパークラスと同じカテゴリーに設定される)。通常はシステム・ブラウザでテキストを打ち込んでクラスを定義するが、支援環境の場合はスーパークラスをクリックする以外の操作はする必要がない。すなわち、クラス・サブクラス間の矢印は**単なるグラフィックデータではなく**、Smalltalk 上できちんと意味付けがされているのである。これにより、ユーザーは図を描き足すという感覚でクラスを作成することができる。スーパークラスの変更なども同様にマウスで図を変更することによって行なうことができ、当然その結果は Smalltalk のシステムに反映される。

画面上に最初はツリー状の図は表示されていない。Smalltalk には数百のクラスが存在し、すべてを表示するのは限られた画面のなかでは大変であるし、また無駄であると考えるからである[3]。既存のクラスを表示させるには **"新しいクラスを作る"** をメニューから選択し、そのクラス名を入力すればよい。さらに、補助的な機能としてツリー状の図を自動的にレイアウトする機能も付け加えた。自分でクラスのアイコンを一つ一つ動かしてレイアウトしても同じことだが、無駄な手間を省くことができる。

また、すでに画面に表示されているクラス

[3] 画面はスクロールに対応しているので、すべてのクラス階層を表示することは可能ではある。

のサブクラスを見たい場合には、クラスアイコンをクリックし、メニューから “**サブクラスを表示する**” を選択すれば画面にきちんとレイアウトして表示される。再帰的な検索は行わないが、この機能により Smalltalk に最初から用意されている数百のクラス群の検索や任意のクラスの階層関係を調べる目的に使用できる。こうした機能は本来ここで目的とするクラス階層作成の支援とは直接関係ないものであるが、既存のクラス階層を参考にしたいという時に別のツールを用いるよりは同じ環境の中に組み込まれていた方が便利だと考え、支援環境の中に取り込むことにした。

3 メソッドの再定義に関する支援

上記の視覚化だけではカバーしきれない関係構造もある。そうしたものに対する支援も考えなければならない。そこで初心者に支援すべきで、かつ本研究で作成する支援環境で視覚化ができなかったものとしてメソッドの再定義の問題を取り上げた。

3.1 メソッドの再定義の問題点

あるオブジェクトがメッセージを受け取ると、オブジェクトは自分自身にそのメッセージに対するメソッドが記述されているかを調べ、もし見つからなければスーパークラスへそのメッセージを送る。そこで見つからなければ、さらにそのスーパークラスへとメッセージは送られていく。逆に、スーパークラスと同じ名前のメソッドを定義すればその機能についてはスーパークラスから継承されず、そのクラス独自の機能として再定義することができる。

しかし、クラスの作成者が十分にスーパークラスについての知識を持たなかった場合、誤って同名のメソッドを定義してしまう可能性もある。このような危険性の排除のために例えば Eiffel[3] では再定義の際には専用の宣言を行うが、Smalltalk ではそうした特別な構文は用意されておらず、再定義に制限はない。また、こうしたメソッドの関係構造はクラスの関係構造の図だけでは支援することができない。そこで、メソッドの再定義に関する支援を別に行うことにした。

3.2 メソッドの再定義に関する支援の実現方法

メソッドの再定義を視覚化とは別に支援するためには、まず、再定義が行われているかどうかを判定しなければならない。これは、あるメソッドが Smalltalk に登録される時に、同名のメソッドがスーパークラスで既に定義されているかどうかを調べればよい。その結果、再定義されていればユーザーに警告し、必要ならばスーパークラスにおけるメソッドを別ウィンドウに表示するようにした。図 6 は、クラス Object のサブクラスの定義中にメソッドの再定義を行なった際に、警告が発せられた例である。

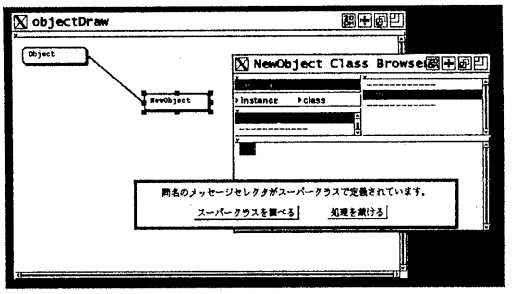

図 6: メソッドの再定義の警告の例

本研究で作成した支援環境では、メソッドの定義はクラスアイコンを選択した後で専用のクラス・ブラウザを開いて行うようにしたが、通常のクラス・ブラウザのサブクラスとして定義したものなので操作方法などが変わることはなく、違和感なく使うことができる。クラス・ブラウザではメソッドを記述した後にメニューから “**accept**” を選択することによってメソッドを承認し Smalltalk に定義をするが、本支援環境における拡張されたクラス・ブラウザではそのメソッドが新しく登録

された場合に限り、“**accept**” の直後に再定義の検査を行う。検査の結果スーパークラスで同名のメソッドが定義されていればダイアログビューを開いてそれをユーザーに警告する。さらに、スーパークラスのメソッドを調べるかどうかを聞いてくるので “**調べる**” と答えると (マウスでクリックすると)、そのメソッドのメソッド・ブラウザが別のウィンドウとして開かれる。ユーザーはそれを見ながら、メソッドを修正したりメッセージセレクタを変更すればいいわけである。十分にその意味が分かっていて再定義を行っている場合には、単に警告を無視すれば何の問題も発生しない。

4 支援環境によるプログラミングの評価

4.1 実験方法

客観的な評価をするために被験者に簡単なプログラムを組んでもらう実験を行なった。対象は特に Smalltalk の経験者に限らず、簡単なオブジェクト指向の考え方を知っている程度の人で、4人とした。この人数は実験としては必ずしも十分ではないが、とりあえず実験をすることに意味があると考え、身近な人達に被験者になってもらった。題材として QuadWorld のようなグラフィック・ライブラリ (3〜4 階層) を2つ用意し、システム・ブラウザと試作した支援環境で作成してもらった。評価は、2つの環境の操作方法や使用感などをアンケート形式で回答してもらうことによって行なった。QuadWorld とはオブジェクト指向入門 [6] で例題として使われているグラフィック・ライブラリで、各種の四辺形 (正方形、矩形、平行四辺形、ひし形など) を画面に描き、またそれを操作 (移動、回転など) することのできるものである。

この実験に先だって、クラス階層の図を見ながらプログラミングを行う予備実験も行なったが、支援環境に比べ従来のシステム・ブラウザの方が操作は多少複雑になるものの、たいして差はないということが分かった。クラス階層に関する詳細設計が終っている場合には、システム・ブラウザのテンプレートを書き換えて定義するという作業自体は多くのプログラマにとって大変な負担というわけではなさそうだった。

そこで4人に関してはクラス階層を図として与えない場合について実験を行なった (作成すべきクラス名だけを資料として与えた)。クラス階層を決定する過程をプログラミングにおける詳細設計として考えると、プログラマが詳細設計を行なう場合に、支援環境が有効であるかどうか調べたかったからである。

また、メソッドの記述については最小限にとどめ、1クラスにつき2〜3メソッドとし、その分クラス階層の作成に重点を置くことにした。これは時間的な制約や、被験者が Smalltalk の文法に不慣れであることを考慮したためと、メソッドの再定義の支援機能についてはこの程度のプログラムではあまり効果を期待できないことが既に予備実験で判明していたからである。

4.2 実験結果

以下に、システム・ブラウザに関するアンケートの主な結果をまとめる。

- 定義する時に文字を多く打つので手間がかかる。
  面倒くさい。 2人
- 定義を行なうこと (テンプレートの書き換え) 自体は、難しくはない。 2人
- 最初に見通しを立てておかないとないと作れない。 2人
- クラスの関係は分かりにくい。 ... 3人

次に、試作した支援環境に関するアンケートの主な結果をまとめる。

- 定義方法はマウスを使うので楽。 . 2人
- 見通しを立てる作業が目で見てできるのがいい。 2人

- クラスの関係は分かりやすい。 ... 3人
- 矢印のデザインが見にくい。 1人
- メソッドの定義のとき継承を考えるようになった。 1人

また、もう一度同じようなプログラミングを行なうとしたら、システム・ブラウザと支援環境のどちらがいいかという問いには、

- クラス階層が複雑になれば支援環境の方がよい。 1人
- 詳細設計もするのであれば支援環境の方がよい。 3人

というのが、主な意見だった。

4.3 実験に関する考察

アンケートの結果と被験者とのディスカッションでは、クラス名だけを示してクラス階層関係を作ってもらった場合、いわば詳細設計も含めてプログラミングを行なってもらった場合には支援環境の方がよいという声が多かった。これはプログラマがクラス階層を考える過程が評価に影響していることを意味している。

支援環境でクラス階層を作る時に、多くの被験者は次のようにした。まず、すべてのクラスアイコン(画面上でクラスを表す長方形のグラフィック)を画面に並べる。次にマウスを使い、クラスアイコンを適当に試行錯誤しながら動かして位置を決める。そして最後に各クラスアイコンをつなげていった。こうした作業過程はそのままクラス階層構築におけるプログラマの思考過程を表していると考えられる。

これに対して、システム・ブラウザでクラス階層を作っていく場合には、プログラマは頭の中でクラスの階層関係の図を描き、関係構造をはっきりさせてから定義を行なっていると考えられる。つまり、支援環境ではその頭の中で行なわれている部分を、画面の上で目で確認しながら作業できるために楽に感じるのではないだろうか。

以上の点から、試作した支援環境はクラス作成において、特にクラス階層の詳細設計に関する支援という面で有効であるといえる。

5 既存の視覚化ツールとの比較

クラス階層の木構造を視覚化するツールはいくつか存在し、Smalltalk だけではなく Objective-C や C++用のものもあるということは、2.2節で述べた通りである。ObjectBits\Smalltalk ver.2.0 に含まれている ClassTreeExample (図7) は、こうしたツールの代表的な例といえる。これらのツールはク

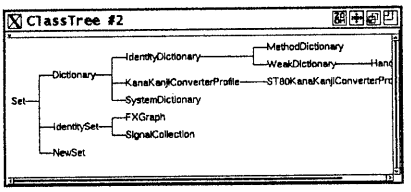

図 7: ClassTreeExample の実行例

ラス階層の木構造を図で表示し、そのメソッドを修正したりでき、既に定義済みのクラス階層の把握という意味では効果があるといえる。しかし、あくまで再利用のためのクラス検索を目的としているため、図に手を加えることによって新しいクラスを追加したり、クラス階層を変更したりすることはできない。

評価の結果、画面上で試行錯誤しながら作業できるという点が支援環境の大きなアドバンテージであったが、既存のツールではそうした機能は不足している。試作した支援環境の目的はクラス階層作成時における支援であり、こうしたツールとは目的が違うので当然のことかもしれない。いずれにしても、本研究の目的の達成には既存のツールでは力不足であるといえるだろう。

6 結論

本研究では、オブジェクト指向プログラミングの中で初心者が把握しづらいと考えられるクラス階層構造を支援対象とし、それを視覚化することによって支援することにした。具体的にはクラス階層の木構造を図で表し、マウスで図を描き足す感覚でクラス階層の作成、変更を行えるような支援環境を試作して評価した訳である。

結果として、システムに付属するブラウザや、既存のツールよりもインタラクティブな環境を試作することができた。また、視覚化された環境がクラス階層の詳細設計に有効なことが分かった。これは、クラス階層を図の上で試行錯誤しながら作成できることがプログラマーの思考過程を助け、負担を減らしたためと考えられる。

図で表せない関係構造の支援として、メソッドの再定義に関する支援を行った。これは初心者がスーパークラスに関する知識不足などから誤ってメソッドを再定義してしまうのを拡張したブラウザによって監視し、それを警告するものである。この支援は簡単なクラス階層（およびメソッドの内容）ではあまり有効な結果は得られなかったが、ある程度クラス階層が複雑化したときには有効であると考えられる。

試作した支援環境は以上の点から、クラス作成、特にクラス階層の詳細設計に関して初心者に有効なものになった。しかし、インスタンスの関係構造やメソッド記述の支援などまだ拡張の余地が残されているといえる。また、問題の視覚化による表現が人間の思考過程を助ける場合があることが分かったので、今後の課題として、さらに様々な関係構造の視覚化による支援なども検討したい。

参考文献

[1] Adele Goldberg and David Robson: Smalltalk-80 The Language and its Implementation, Xerox Palo Alto Research Center (1983).

[2] Adele Goldberg: Smalltalk-80 The Interactive Programming Environment, Xerox Palo Alto Research Center (1984).

[3] Bertrand Meyer: Object-oriented Software Construction, Interactive Software Engineering, Inc. (1988). (邦訳：「オブジェクト指向入門」二木厚吉 監訳／酒匂 寛・酒匂 順子 共訳 アスキー出版局 1990)

[4] Edel, M., "The Tinkertoy Graphical Programming Environment" Proceedings of IEEE Compsac, (Oct. 1986), pp. 466-471.

[5] Glinert, E.P., and Tanimoto, S.L., "Pict: An Interactive Graphical Programming Environment" IEEE Computer, Vol.17, No.11 (Nov. 1984), pp. 7-25.

[6] Kurt J.Schmucker: Object-Oriented Programming for the Macintosh, Productivity Products International, Inc. (1986). (邦訳：「オブジェクト指向プログラミング」大谷 和利 監訳 SOFT BANK CORP.)

[7] P.T.Cox, F.R.Giles, T.Pietrzykowski, "Prograph: A Step Towards Liberating Programming From Texual Conditioning", 1989 IEEE Workshop on Visual Languages, Rome, Italy, (October 4-6 1989), pp. 150-156.

[8] 三ツ井欽一, 久世和資, Shahram Javey: 拡張可能な C++ソースコードブラウザ - 基本設計, 第 45 回情報処理全国大会, 7Q-05, (1992).