ソフトウェア工学 83－19
（1992. 2. 7）

複数の機能を備えた多重オブジェクト

岡本康介† 佐藤康臣† 宮永靖之† 平川正人‡ 市川忠男‡

広島大学大学院† 広島大学工学部‡

従来のオブジェクト指向プログラミングでは、複数のクラスの持つ機能を１つのオブジェクトで表現する場合、多重継承や複合オブジェクトによるクラスの定義が行なわれてきた。しかしこれらの手法では、意味的、構造的に対象世界を自然にモデル化できない場合がある。本稿では、新たなモデル化の手法として多重オブジェクトを提案する。多重オブジェクトを用いることで、従来のシステムに見られない“選択的継承”の概念が実現可能となる。多重オブジェクトは、複数の要求を満足するために複数のクラスをドメインとして持ち、それらを切替えながら動作する。このようにドメインはプログラム実行中に動的に変更可能なため、要求の変化に柔軟に対応できる。

Multi-Object in an Object-Oriented Environment

K. Okamoto, Y. Sato, Y. Miyanaga, M. Hirakawa, and T. Ichikawa

Faculty of Engineering, Hiroshima University
Kagamiyama 1–4–1, Higashi–Hiroshima, 724, Japan

This paper introduces Multi-Object which acts as an instance of multiple classes in an object-oriented environment. In ordinary object-oriented programmings, either a multiple-inheritance or a composite object is used to utilize facilities of different classes. However, these methodologies are not always successful in specifing the structure of a domain suitably. Multi-Object provides a new methodology for modeling which we call “alternative-inheritance”. It behaves as an instance of different classes in the domain, depending on the context.

1 はじめに

近年、ソフトウェア開発の生産性、ソフトウェアの信頼性、保守性を向上させるため多くの研究が行なわれている。それらのアプローチの1つとしてソフトウェアの部品化、再利用があげられ、その有効な手法としてオブジェクト指向プログラミングが提唱されている[1][2]。オブジェクト指向プログラミングでは、クラスによるソフトウェアの部品化が行なわれ、新たなクラスを作成するにあたっては、継承機構により既存のクラスを再利用可能である。こうした利点に加え、オブジェクト指向プログラミングでは、オブジェクトを中心に考えることで、対象領域を自然な形で計算機内にモデル化できる。

従来のオブジェクト指向プログラミングでは、複数のクラスの機能を表現するオブジェクトを作成しようとした場合、多重継承や複合オブジェクトによるクラスの定義が行なわれてきた。しかしこれらの手法だけでは、意味的、構造的に自然なモデル化が不可能な場合がある。また、名前の衝突を回避するために、不必要なプログラミングが行なわれる。本稿では、複数の機能を備えたオブジェクトをモデル化する新たな手法として、多重オブジェクトを提案する。多重オブジェクトを用いることで、従来のオブジェクト指向プログラミングには見られない“選択的継承”の概念が実現可能となる。

多重オブジェクトは、多重継承と同様に1つのオブジェクトで複数のクラスの持つ機能を利用したい場合に用いられる。多重オブジェクトは複数のクラスの機能を満足するため、複数の状態を保持している。多重オブジェクトはこれらの状態を実行条件により切替えながら動作する。多重継承と違い、オブジェクトの能力を拡張し、複数の状態を保持することで実現するため、それらの機能を柔軟に変化させることが可能である。また、個々のクラスを基本単位として考えるため、優れたモジュール性を維持したまま部品の合成が可能となり、名前の衝突も回避しやすい。実際に多重オブジェクトを昨年度開発した制約オブジェクト指向言語ISL-xscheme[3]上で実現した。

以下、2章で一般的なオブジェクト指向の概念と特徴、3章で制約オブジェクト指向言語ISL-xscheme、4章で多重オブジェクトの概念、有効性を述べ、そして5章で具体的なプログラミングにおける多重オブジェクトの使用例について述べる。

2 オブジェクト指向

オブジェクト指向言語はモジュール性に優れ、ソフトウェア開発の生産性が高い。オブジェクト指向プログラミングの特徴として、

- クラスによるプログラムの部品化が行なわれる
- 継承機構により、プログラムの共有、再利用が積極的に行なわれる
- オブジェクトを中心に考えることで、問題の構造をプログラムの構造に自然に反映することができる
- 動的束縛機構を利用して、柔軟性に富んだプログラミングが可能である

などがあげられる。

古典的なオブジェクト指向においては、クラスによるオブジェクトの分類を行なうため、1つのオブジェクトは実行を通して、常に1つのクラスに属する必要があった。したがって、複数のクラスが持つ機能を1つのオブジェクトで表現する場合、クラス階層を利用した多重継承によるクラス定義、あるいはオブジェクトの構造による複合オブジェクトが使用される。

2.1 多重継承

多重継承では、単一継承に比べて継承の機能がより強力である。また、複数のクラスが共有できる性質をスーパクラスとすることで、情報の共有化が促進できる[2][4][5]。しかし多重継承の場合、多機能部品として生成されるインスタンスは常に同一のクラスに属するため、クラス定義時に名前の衝突に対処しなくてはならない。このときスーパクラスの順序づけや名前の変更が行なわれるが、不必要にプログラミングの困難さを増大させることになる。

2.2 複合オブジェクト

複合オブジェクトは、複数のクラスのインスタンスを構成要素としてインスタンス変数に保持する。複合オブジェクトは集約化の概念であり、全体をまとめたクラスと要素クラスとの関係は“is a part of”の関係である。

3 制約オブジェクト指向言語 ISL-xscheme

本章では、我々が開発しているオブジェクト指向言語ISL-xscheme[3]について述べる。ISL-xschemeはフリーソフトウェアxschemeをC言語とschemeを用いて拡張したものである。

3.1 ISL-xschemeの特徴

古典的なオブジェクト指向言語では、オブジェクト間の関係を手続き的にしか記述できないため、密接に関連

する複数のオブジェクトのモデル化は苦手である。そこでオブジェクト指向言語に、関係の宣言的な記述である制約表現を導入する研究が数多く行なわれている。ISL-xscheme も制約表現を導入することで、オブジェクト間の関係を処理手順に依存しない形で記述することができる。このため、プログラマは対象領域に存在する密接に関連するオブジェクトを、より自分の思考に近い形でモデル化できる。

また実行可能な抽象部品であるジェネリックオブジェクトを用いることで、抽象的な要求をコード中に記述可能である。詳細は 3.3節で説明するが、ジェネリックオブジェクトは、関連する制約を満足するように自らの状態を変化させ、周囲のオブジェクトと協調して動作する。こうしたジェネリックオブジェクトを用いることで、ソフトウェア開発の初期段階における抽象的な要求、すなわち確定的でない要求を自分の思考に近い形でプロトタイピングに直接反映することができる。

3.2 クラス定義と制約

クラス定義は図 1に示す通り、クラス名、インスタンス変数、クラス変数、スーパクラスを引数として、Class クラスに "new" メッセージを送ることで行なわれる。また本システムで扱う制約は

class invariant クラスに属する全てのインスタンスに適用され、そのクラスの持つ本質的な性質を表す

precondition メソッドを正しく実行するための必要条件を表す

postcondition メソッド実行後に要求される状態を表す

の 3 種類である。これらは図 1に示す通り、他のオブジェクトとの関係を宣言的に記述する。メソッドの定義は図 1に示す通り、メソッド名、引数、本体、precondition 、postcondition を引数として、クラスに "answer" メッセージを送ることで行なわれる。こうして定義された制約は、オブジェクトにメッセージが送られた時に評価される。まずメソッド実行前に class invariant, precondition が評価される。共に満足する場合、オブジェクトはメソッドを起動し実行する。実行後、postcondition ならびに class invariant を評価することにより、メソッドの正当性を検査する。

現在のシステムは継承機構として単一継承を採用している。クラスは階層的に管理され、上位クラスのインスタンス変数とメソッドおよび制約は、その下位クラスに継承される。

```
; クラスの定義
(Class 'new 'Driver
       '(license) '() 'Person)
; class invariant の定義
(Driver 'definvariant!
        '((observe traffic-regulation)))
; メソッドの定義
(Driver 'answer 'goto '(<whereto> <car>)
         ; メソッド本体
         '((self get-into-car <car>))
           (drive <whereto>))
         ; precondition
         '((license-to-drive <car>))
         ; postcondition
         '((arrived-at <whereto>)))
```

図 1: クラス定義例

3.3 ジェネリックオブジェクト

我々が提案したジェネリックオブジェクトは、インスタンス生成時に任意のクラスをドメインクラスとして与えることで、そのドメインクラス以下の全てのクラスの総称的なインスタンスとして振舞うことができる [3] 。ジェネリックオブジェクトを生成するためのクラスである Generic クラスは図 2のように定義されており、以下のインスタンス変数を持つ。

current-object 現在の状態を保持する

domain current-object が属し得るクラスの範囲を規定する

また、ジェネリッククラスには、"送られたメッセージを受理しなくてはならない (accept-method current-object)" 、"関連する制約を満足しなければならない (satisfy-constraint current-object)" が class invariant として定義されている。これらの制約により、抽象部品であるジェネリックオブジェクトは

- 他のオブジェクトからのメッセージの送信
- 周囲のオブジェクトとの関係

によって具体的なインスタンスへオブジェクト変換され、実行可能となる。このようにジェネリックオブジェクトは他のオブジェクトと協調して柔軟な動作をするが、動作が固定されていないため、文脈によってオブジェクトの振舞いが異なることが問題点として指摘されている。

ジェネリックオブジェクトの設計目標は、抽象的なソフトウェア部品を含んだプログラムを実行可能にすることであり、プログラマのあまり関心のない部分での使用を前提としてきた。本研究で提案する多重オブジェクトは、プログラムの中心部分で利用可能なジェネリックオブジェクトである。多重オブジェクトは、生成時にプログラマの関心のある複数のクラスをドメインとして与えることで、複数の機能を持つオブジェクトとして生成される。多重オブジェクトは状況に応じてドメイン内の適切なクラスのインスタンスとして振舞う。つまり、多重オブジェクトはジェネリックオブジェクトのドメインを興味のある部分に制限したものである。また、多重オブジェクトの生成時に与えられたドメインは動的に変更可能であるため、ジェネリックオブジェクトの特徴である柔軟性を失うことはなく、要求の変化に柔軟に対応できる。さらに多重オブジェクトの属するクラスは文脈により決定されることから、文脈に依存した状態の変化をプログラミングに積極的に利用することできる。

```
(Class 'new 'Generic
       '(current-object domain ...)
       '()
       'Object)
```

図 2: Generic クラスのクラス定義

4 多重オブジェクト

本章では多重オブジェクトの概念を説明し、多重継承、複合オブジェクトとの比較を行なうことで、その有効性について述べる。

4.1 多重オブジェクトの概念

多重オブジェクトとは、複数の要求を満足するために複数の顔を持ち、それらを必要に応じて使い分けるオブジェクトである。多重オブジェクトは生成時に与えられた複数のクラスをドメインとし、それらのインスタンスを複数の顔として保持している。多重オブジェクトは文脈に応じて、それらの顔の中から適切なものを選び動作する。多重オブジェクトの概念は選択的継承として理解できる。

例として、図 3に示す部品階層を考える。ここではクラス階層中に Person, Driver, Student, Worker クラスが存在する。Student, Worker クラスは共に “work” メソッドを持ち、それぞれ “勉強する” 、 “働く” という意味を持っている。また Student, Worker クラスのそれぞれには、 “授業に出席する” 、 “出勤する” ためのメソッドとして、 “attend-lecture”, “attend-office” が定義されている。この時、次の要求を考える。

> 「 Student と Worker の機能を持ったインスタンスを生成したい。但し、それらの機能は状況により使い分けるもので、同時に満足する必要はない。また、“自動車免許を取得する” ことで Driver としての機能を得ることができ、“ 学校を卒業する” ことにより Student の機能を必要としなくなる。」

以下では、多重オブジェクトを用いて、上記の要求を満足する Sample クラスを定義する。

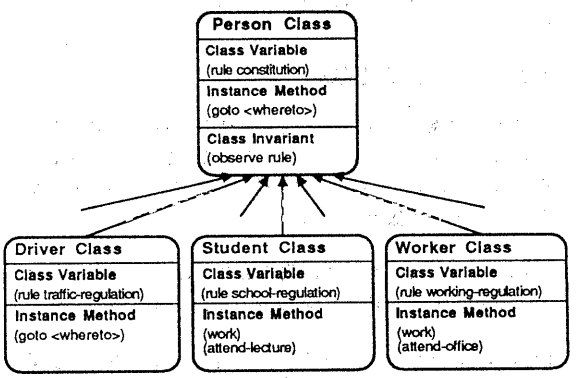


図 3: Person クラスのクラス階層

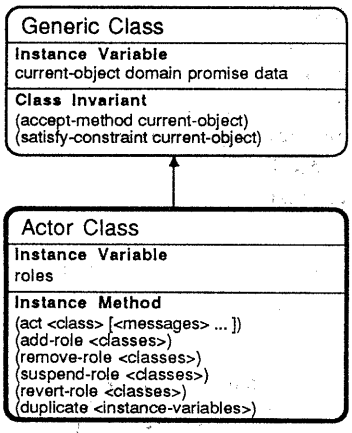
4.2 多重オブジェクトの実現

はじめに多重オブジェクトの実現法について述べる。多重オブジェクトは Actor クラスとその下位クラスのインスタンスである。

4.2.1 Actor クラス

Actor クラスは Generic クラスのサブクラスとして定義されている (図 4) 。このため Generic クラスのインスタンス変数および class invariant を継承している。インスタンス変数 domain には多重オブジェクトが振舞うべきクラスのリストが保持されている。また、Actor クラスではインスタンス変数として roles が定義されており、複数のクラスのインスタンスを保持している。多重オブジェクトは、文脈、関連するオブジェクトとの関係や送

られてくるメッセージにより、roles の中から適切なインスタンスを current-object として選ぶ。また Actor クラスには、図 5のようなメソッドが定義されている。




```
(Class 'new 'Actor '(roles) '() 'Generic)
```

図 4: Actor クラスのクラス定義

```
; 機能を切替えて<message> ... を送る
(<actor> 'act <class> [<message> ...])
; 新たな機能を付加する
(<actor> 'add-role <classes>)
; 機能を削除する
(<actor> 'remove-role <classes>)
; 機能を一時停止する
(<actor> 'suspend-role <classes>)
; 停止していた機能を復帰させる
(<actor> 'revert-role <classes>)
; 記憶域を多重化する
(<actor> 'duplicate
         <instance-variables>)
```

図 5: Actor クラスのメソッド

4.2.2 多重オブジェクトの構造

多重オブジェクトは、Actor クラスまたはその下位クラスとして定義されたクラスに、"new" メッセージを送ることで生成される。この時、引数に複数のクラスをドメインとして与える。多重オブジェクトは、ドメインとして保持している複数のクラスについて、それらの共通のスーパクラスで定義されたインスタンス変数の記憶域を共有する。生成された多重オブジェクトは、インスタンス変数 roles に、ドメインクラスのインスタンスの共有部分と差分とを環境として保持している。差分として保持されている環境は、その親の環境へのリンクを辿ることで、完全なオブジェクトとしての環境となる。また、共有部分は同一の環境で保持されているため、共有部分の変更は多重オブジェクトの内部で、他の roles にも影響を及ぼす。ただし "duplicate" メソッドにより、記憶域の多重化も可能である。

4.3 Sample クラス

Sample クラスは、Actor クラスのサブクラスとして、図 6のように定義できる。図 6で定義された "isnew" メソッドは、Sample クラスのインスタンスの初期状態を決定するためのもので、生成時のドメインである (Student Worker) を引数として、Actor クラスの "isnew" メソッドを呼び出している。また、"自動車免許を取得する" ための "get-driving-license" メソッドでは Driver の機能を付加し、"学校を卒業する" ための "graduate" メソッドでは Student の機能を削除している。

こうして定義した Sample クラスをインスタンシエイトして多重オブジェクト foo を生成し、図 7のようなメッセージを送信する場合を考える。多重オブジェクト foo は、初期状態では Student と Worker の両方の機能を持ち、それらの機能を切替えながら動作する。始めに foo は、"attend-lecture" メッセージを受け取ることで、Student の顔を選択する。したがって "work" メッセージを受け取ると、Student クラスの "勉強する" という意味の "work" が起動される (図 8)。次に、"attend-office" メッセージを受け取ることで Worker の顔を新たに選択するため、次の "work" メッセージにより起動されるのは Worker クラスの "働く" という意味のメソッドである (図 8)。このように多重オブジェクトの状態は、それまでに受け取ったメッセージ、すなわち文脈に依存する。したがって、この例のように、オブジェクトの状態の変化を積極的にプログラミングに利用可能である。また foo に "get-driving-license", "graduate" メッセージを送ってドメインを変更することで、それぞれ Driver の機能の付加、Student の機能を削除が行なわれる。またこうしたドメインの変更は、直接オブジェクトの状態に反映される (図 9)。

4.4 従来手法との比較

従来のオブジェクト指向言語では、複数のクラスの機能を 1 つのオブジェクトで利用するために、多重継承あ

```
; Actor クラスのサブクラスとして定義
(Class 'new 'Sample '() '() Actor)
; インスタンスの初期化を行なう
(Sample 'answer 'isnew '()
   '((send-super 'isnew
                 '(Student Worker))))
; add-role メソッドにより
; Driver の機能を付加する
(Sample 'answer 'get-driving-license '()
   '((self 'add-role Driver)))
; remove-role メソッドにより
; Student の機能を削除する
(Sample 'answer 'graduate '()
   '((self 'remove-role Student)))
```

図 6: Sample クラスの定義

るいは複合オブジェクトとしてクラスの定義が行なわれてきた [4][5][2] 。これらの場合、生成されるオブジェクトは唯一のクラスに属するため、複数の機能を同時に満たすことを意味している。これに対して多重オブジェクトは、生成時に与えられた機能を同時に満足するわけではなく、文脈により定められるオブジェクトの状態や実行条件により、ドメイン内の適切な機能を選択して動作する。

4.4.1 多重継承

多重継承の場合、複数クラスの機能を利用は複数のクラスの共通のサブクラスを定義することで行なう。それらの機能が同時に満足される必要がない時、多重オブジェクトは多重継承に比べて自然な形で対象をモデル化できる。多重継承ではクラス定義時に機能が決定されるため、要求の変化が生じた時、クラス定義を変更する必要があり、柔軟性に欠ける。このような場合、多重オブジェクトでは動的にドメインを変更することで容易に対応できる。

さらに多重継承の場合、オブジェクトが複数の状態を持つわけではないので、クラス定義時に名前の衝突に対処しなくてはならない。このときスーパクラスの順序づけや名前の変更が行なわれるが、不必要にプログラミングの困難さを増大させることになる。これに対して多重オブジェクトの場合、メッセージを受け取った時、受理可能なクラスに属している限り名前の衝突は生じない。また、ドメイン内の個々のクラスを基本単位として考えるため、メソッドとインスタンス変数の整合性が保証される。これらの理由から、多重オブジェクトでは、既存のクラスを再利用する際に不必要なプログラミングを行なう必要が少ないため、多重継承に比べて再利用性に優れているといえる。

```
; 多重オブジェクト foo の生成
(define foo (Sample 'new))
; 授業に出席して勉強する
(foo 'attend-lecture)
(foo 'work)
; 出勤して働く
(foo 'attend-office)
(foo 'work)
; 自動車免許を取得する
(foo 'get-driving-license)
; 学校を卒業する
(foo 'graduate)
```

図 7: 多重オブジェクトの生成と実行

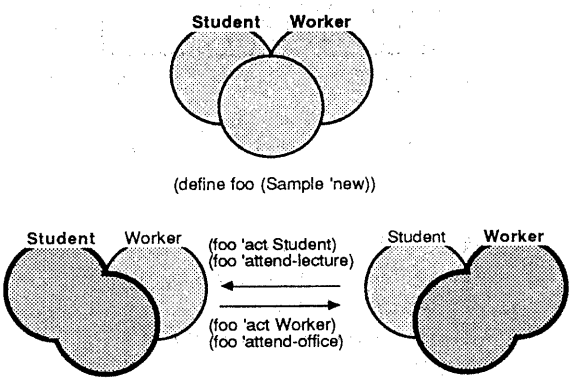


図 8: 多重オブジェクトの機能の切替え

4.4.2 複合オブジェクト

複合オブジェクトによるクラス定義では、複数のクラスの機能の利用は、インスタンス変数として複数のクラスのインスタンスを定義することで行なわれる。複合オブジェクトの概念は“集約”であるため、多重オブジェクトの概念である“選択的継承”とは異なる。複合オブジェクトでは要素クラスのオブジェクトは、それらをまとめるクラスの構成物であるため、他のオブジェクトから直接参照することはできない。一方、多重オブジェクトで

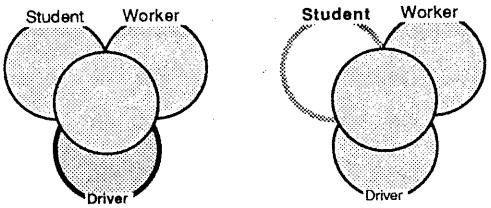


図 9: 機能の付加、削除

は、ドメイン内のあるクラスとして振舞っている場合、そのクラスから生成された通常のオブジェクトと同様に振舞う。さらに、複合オブジェクトが多重オブジェクトのように共有部分を持つ場合、共有部分に関するプログラミングが必要となる。

5 多重オブジェクトの使用例

本章では、メニュー作成の例を用いて、多重オブジェクトを利用したプログラミング例を述べる。

5.1 メニュークラス

クラス階層中に図 10に示すようなメニュークラスが存在するものとする。SimpleMenu クラスは全てのメニューに共通な性質を持つ。インスタンス変数 menuKey は“キー”の配列を保持しており、menuAction は対応する“処理”の配列を保持している。また、“open” メソッドは menuKey を画面上に表示するメソッドで、“close” メソッドはメニューを閉じるメソッドである。“action” メソッドは menuKey に対応する menuAction を実行する。

KeyboardMenu クラスはテキストベースのメニューであり、キーボードからの操作が可能である。キーボードの上下移動キーが押されると “cursor-up”, “cursor-down” メソッドが実行され、menuKey を上下に移動する。リターンキーにより “action” メソッドが起動され、選択されている menuKey に対応する menuAction が実行された後、メニューを閉じる。また、このクラスにはインスタンス変数として配列 hotKey が定義されており、ここに保持されているキーが押されることで、直ちに対応する menuAction が実行される。

MouseMenu クラスはウインドウベースのメニューで、マウスによる処理が可能である。このサブクラスである PopUpMenu クラスではマウスボタンが押されることにより、メニューが開かれる。メニュー内のマウスの位置にしたがった menuKey が選択され、マウスボタンを離すことにより、選択された menuKey に対応する menuAction が実行され、メニューを閉じる。また menuKey 以外の部分でマウスボタンを離すとメニューを閉じる。

5.2 メニューオブジェクトの生成

本節ではインスタンスレベルでの多重オブジェクトの使用例を示す。要求は次の通りである。

> 「ファイルの入出力が必要なプログラムで、これらの処理をメニューにより選択したい。また、テキストベースとウインドウベースでの動作を実現するため、双方に対応したメニューを利用したい。」

この要求を満足するインスタンスは、KeyboardMenu クラスと PopUpMenu クラスの両方の機能を持つ多重オブジェクトとして、図 11のように生成できる。なお、本プログラムのイベント待ちループでは、プログラム画面の上部に表示されている “File(F1)” に対応する “F1” キーがキーボードから入力されると (fileMenu 'act KeyboardMenu 'open) が実行され、画面内でマウスボタンが押されると (fileMenu 'act PopUpMenu 'open) が実行される。従って “F1” キーによりオープンされる fileMenu では KeyboardMenu クラスの機能が選択され、キーボードからの入力を受け付ける。またマウスボタンを押すことでオープンされる fileMenu では、PopUpMenu クラスの機能が選択され、マウスによる操作が可能となる。

次に、多重継承を用いて同じ要求を満たす KeyboardMouseMenu クラスを定義することを考える。KeyboardMouseMenu クラスは、KeyboardMenu クラスと MouseMenu クラスの共通のサブクラスとして定義される。“open”, “close” メソッドは 2 つのスーパクラスから共に継承するため、名前の変更が必要となる。また “eventloop” メソッドについても名前の衝突が生じるが、このメソッドは SimpleMenu クラスの “open” から起動されるため、名前の変更が不可能である。従って KeyboardMouseMenu クラスで再定義する必要がある。

この例のように複数の機能を同時に必要としない場合、多重オブジェクトが利用可能である。多重オブジェクトを用いることで、既存の部品を再利用する際に、不必要なプログラミングが減少すると考えられる。またインスタンスレベルでの部品合成が容易になるため、1 つのクラスでは最低限必要な機能実現し、インスタンスの段階で多機能部品を合成することができる。さらに新しい機能を持つクラスが作成された場合、その機能を容易に追加できるので、要求の変化に柔軟に対応できると考えら

れる。これらの理由から、多重オブジェクトを用いることで、モジュール性に優れたクラス設計が可能である。

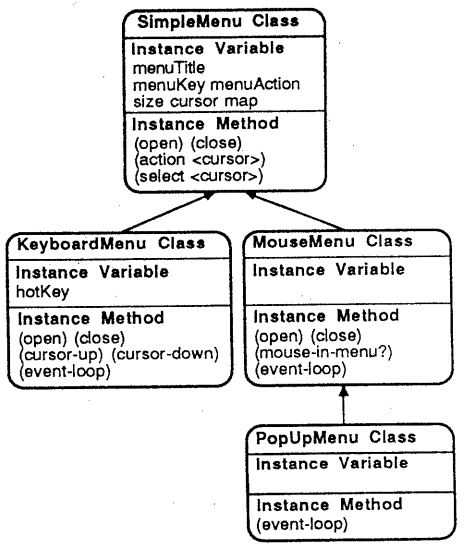


図 10: メニュークラスのクラス階層

```
(define fileMenu
   (Actor 'new (KeyboardMenu PopUpMenu)))
(fileMenu 'size! 3)
(fileMenu 'menuTitle! ``File'')
(fileMenu 'menuKey!
          '(``Open'' ``Close'' ``Quit''))
(fileMenu 'menuAction!
          '((fileDialog 'open)
            (currentFile 'close)
            (exit)))
(fileMenu 'hotKey! '(#\O #\C #\Q))
```

図 11: fileMenu の生成

6 おわりに

本稿では、複数の要求を満足するために複数の顔を持ち、それらを切替えながら動作する多重オブジェクトを提案した。多重オブジェクトを用いることで複数の機能を1つのオブジェクトで表現できるため、対象の自然なモデリングが可能である。また、ドメインを動的に変更可能なため、要求の変化に柔軟に対応できる。さらにオブジェクトレベルの部品合成が容易になるため、モジュール性に優れたクラスの設計を行なうことが可能である。

今後の検討課題として、

- 型制約の導入によるビュー機能の実現 [6][7]
- システムによるドメインの決定
- 名前の衝突に関する考察

などがあげられる。

参考文献

[1] A. Goldberg and D. Robson, *Smalltalk-80 –The Language and It's Implementation–*, Addition-Wesley, 1983.

[2] B. Meyer, *Object-Oriented Software Construction*, Prentice-Hall, 1988.

[3] 森本、佐藤、岡本、宮永、田中、市川, "オブジェクト指向環境におけるジェネリックオブジェクト," 情報処理学会ソフトウェア工学研究会, February, 1991.

[4] D. A. Moon, "Object-Oriented Programming with Flavors," *ACM Proceedings of OOPSLA'86*, vol. 21, No. 11, pp. 1–8, 1986.

[5] 井田昌之、元吉文男、大久保清貴編, Common Lisp オブジェクトシステム –CLOS とその周辺–, bit 別冊, 共立出版, 1989.

[6] J. Richardson and P. Schwarz, "Aspects: Extending Objects to Support Multiple, Independent Roles," *Proceedings of ACM SIGMOD Conference on Management of Data*, pp. 298–307, Denver, Colorado, June, 1991.

[7] S. Abiteboul and A. Bonner, "Objects and Views," *Proceedings of ACM SIGMOD Conference on Management of Data*, pp. 238–247, Denver, Colorado, June, 1991.