オブジェクト指向プログラミングのためのクラス検索システム†

垂水浩幸†† 阿草清滋†† 大野豊††

筆者らは，オブジェクト指向型言語を用いたプログラミングにおけるソフトウェアの部品化・再利用の支援を行うプログラミング環境を開発中である．本論文ではその中の部品検索システムについて述べる．ここで用いるプログラミング言語は，オブジェクト間の関係やオブジェクトの性質を表現することを考慮して筆者らが提案したオブジェクトモデルに基づいて開発された言語 MOMO である．まず利用者は自由な語いと前述のモデルにより定められた関係などを用いて，作成したいオブジェクトとそれらのオブジェクトで実行したいオペレーションを記述する．部品検索システムは MOMO のクラスライブラリである MOMOCLI から，利用者の要求するオブジェクトやオペレーションを実現するためのクラスおよびメソッドを検索する．検索システムは一種のプロダクションシステムであり，与えられた検索ルールに従って動作する．検索ルールには，前述のモデルで定義されているオブジェクト間の関係などのマッチングを取るものや，ユーザが用いた語いを辞書で調べて候補を探すものなどがある．検索結果は図的に表示され，利用者はその表示を見ながら，候補クラスや候補メソッドから適当なものを選択することができる．本検索システムを利用することにより，既存のクラスについて学習する労力が軽減され，初級者および中級者のプログラミングを支援することができる．

1. はじめに

ソフトウェアの生産性と信頼性の向上のため，ソフトウェア部品の再利用技術がソストウェア工学の重要な研究課題となっている．

オブジェクト指向型言語ではクラスと呼ばれるモジュール単位でソフトウェアを記述する．クラスはデータ型に関する記述と手続きに関する記述を共に持っており，モジュール性がよいため，ソフトウェア部品としてすぐれている．またクラス間に性質の継承を定義でき，継承関係を用いた部品の整理や，部品の利用環境に合わせた特殊化を行うことができる．

実際，オブジェクト指向型言語のプログラミングではクラスの再利用が行われている．例えば，Smalltalk-80[1] の場合，整数や文字を表すような基本的なクラスはもとより，ウインドウやメニューを構築するためのクラスなどが豊富に用意されており，利用者がこれらのクラスを利用することが前提となっている．したがって，これらのクラスを熟知しているユーザは効率よくプログラミングをすることができる．

しかし，現実には利用可能なクラスを検索することが困難であり，たとえ検索できたとしてもその利用法を学習するのに手間がかかる．Smalltalk-80 で中規模以上のプログラムを作成しようとすると，必ずこのような学習が必要となるが，効率的な学習方法はない．ブラウザ (browser) というツールが用意されているが，これを用いても最終的にはマニュアルを読んだり提供されているソースプログラムを読んだりして学習しなければならない．また，必要とするクラスやメソッドを検索するための手段としてカテゴリ (category) という分類がなされているが，これはシステム側で一方的に与えられる一階層の分類であるため，どのカテゴリにどんなクラスやメソッドが存在するのかを予想するのは困難である．

このような状況を解決し，クラスの検索，学習の手間を軽減することができれば，オブジェクト指向型言語によるプログラミングは効率化される．我々の MOMOCLI プロジェクト[2)~5)] はクラスの検索，学習の行いやすいオブジェクト指向プログラミング環境の構築を目標としたものである．本論文では MOMOCLI プロジェクトで作成した環境のうち，検索を担当する部分について特に述べる．

2. MOMOCLI プロジェクトの概要

2.1 発想

本研究では，ライブラリに蓄えられているクラスやメソッドの名前や利用法について詳しく知らない利用者でも使えるようなプログラミング環境の構築を目的としている．そのためには，利用者が作成したいソフトウェアについての要求を記述するための言語を用意し，その言語で記述された要求記述をもとにしてクラスやメソッドを検索しなければならない．

検索を実現する方法としては，形式性の高い要求仕様記述言語を用いて，仕様の形式的意味とライブラリ

† A Class Retrieval System for Object Oriented Programming by HIROYUKI TARUMI, KIYOSHI AGUSA and YUTAKA OHNO (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科

中のクラスの形式的意味のマッチングを取るという方法が考えられる．しかし，このような方法で仕様の記述やクラスの検索をするためには，細かい意味の違いを形式的に表現する必要がある．そのような仕様を記述することは利用者にとって容易ではないので，形式的アプローチは利用者の学習負担の軽減という目的にはそぐわない．

検索という観点だけから考えると，利用者に記述してもらうのは完全な仕様である必要はない．仕様の中の何らかの特徴的な部分が記述されていれば，その記述をもとに検索ができるはずである．

本研究では利用者にオブジェクトの関係や性質を記述してもらうことにした．これらの関係や性質が直観的に理解しやすく，かつ形式的に定義されているものであれば，利用者の記述した直観的関係・性質とライブラリ中にあるクラスに記述されている形式的関係・性質とのマッチングをとることができる．使える関係・性質の種類を少数に制限すれば，記述法を学習する負担も少なくてすむ．

我々は，知人/構成変数モデル[6),7)]を定義し，このモデル上で「知っている (KNOW)」，「持っている (HAVE)」という2つのオブジェクト間関係を定義し，さらに各オブジェクトの上で定義されるオペレーションと他のオブジェクトとの関係として「作用する (EFFECT)」，「依存する (DEPEND)」，「取り込む (GET)」，「返す (RETURN)」，「使用する (USE)」，「操作する (HANDLE)」を定義した．さらにオペレーションの性質として「自己作用 (SELF-EFFECT)」と「自己依存 (SELF-DEPEND)」を定義した．

もちろん，これらの関係や性質の単純なマッチングで検索を行った場合，候補の数が大量になることが多い．そこで，オブジェクトやオペレーションの名前として利用者が用いると予想される語いを辞書に登録しておき，単語のマッチングを検索の参考にする．さらに，検索のための知識をプロダクションルールとして蓄えておき，これを用いることにより検索の精度を上げることにする．ライブラリの適用問題分野を限り，辞書に登録する語いやルールを問題分野ごとに特殊化すれば，より検索の精度が上がり，実用的なものにすることができる．

結局，利用者の要求の記述には，

(1) 作成したいソフトウェアに登場するオブジェクトの名前
(2) 各オブジェクト上で定義されるオペレーションの名前
(3) 知人/構成変数モデルで定義されたオブジェクト間関係，オペレーションとオブジェクトの関係，性質名

などを記述してもらうことにした．この記述は部品データベースへの質問と考えられるので，本論文ではこれを質問記述と呼ぶ．

なお，本論文中で「オブジェクト」「オペレーション」とは質問記述中に登場するソフトウェアの構成概念を表し，「クラス」「メソッド」とはそれぞれオブジェクトとオペレーションを実現する部品を表している．

2.2 言語 MOMO

本プロジェクトでは知人/構成変数モデルを仮定しているため，既存のオブジェクト指向型プログラミング言語を利用することはできない．そこで，知人/構成変数モデルにもとづく言語 MOMO[8)] を設計した．MOMO のライブラリが MOMOCLI であり，これが本プロジェクトの名前になっている．

他のオブジェクト指向型言語と同じように，MOMOCLI の中のあるクラスの性質は，そのすべてのサブクラスによって継承される．継承は，知人/構成変数モデルで定義された関係や性質についても成り立つ．これらの関係や性質を検索のよりどころにするため，一つのクラスとそのすべてのサブクラスは，検索時には同一の性質を持ったクラスの集合として認識している．MOMO ではこの集合を型として，型付けを行っている．

MOMO では多重継承を許している．あるクラスが複数のスーパクラスを持つということは，そのクラスがそれぞれのクラスの性質を併せて持っていることを示す．つまり，個々のスーパクラスがクラスの性質を独立に表現しているわけで，これは検索のためのインデックスが複数存在することを意味している．これはすなわち各クラスが検索される機会を増やすことになるため，検索には好都合である．

また，MOMO のクラスには辞書に登録すべき語いを列挙することができる．例えば，Window というクラスには window, view, frame といった語が登録されている．view は Smalltalk 流のウインドウの呼び方であり，frame は SunView 流の呼び方である．

MOMO はコンパイラ言語である．実際にはいったん C++[9)] に変換された後，コンパイルされる．MOMO から C++ へのトランスレータを MOMOCO と呼ぶ．

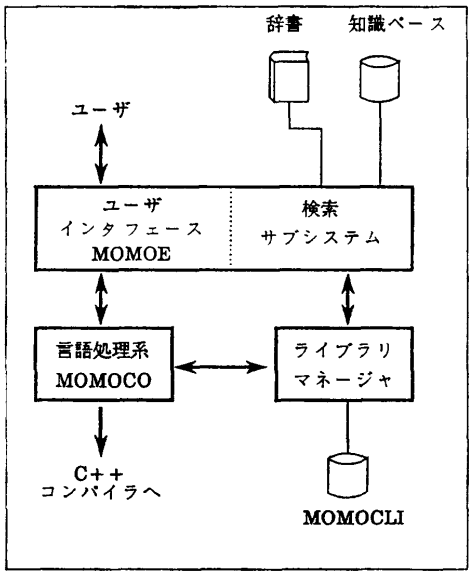


図 1 MOMOCLIプロジェクトで作成したプログラミング環境
Fig. 1 The programming environment developed by MOMOCLI project.

2.3 システムの構成

図1に本プロジェクトで作成したプログラミング環境を示す．この環境は SUN ワークステーション上で実現されている．

```
System          Stock chart
Object          KABUKA-CHAATO
Object-Relation         HAVE            KABUKA-GURAFU
Object-Relation         HAVE            DEKIDAKA-GURAFU
Object-Relation         HAVE            BANGOU
Operation               kaku
Operation-Relation              EFFECT  DISPLAY

Object          KABUKA-GURAFU
Object-Relation         HAVE            ASHI
Object-Relation         KNOW            KABUKA-HYOU
Operation               kaku
Operation-Relation              EFFECT  DISPLAY
Operation-Relation              USE     KABUKA-HYOU

Object          DEKIDAKA-GURAFU
Object-Relation         HAVE            JIKU
Object-Relation         HAVE            SEN
Object-Relation         KNOW            KABUKA-HYOU
Operation               kaku
Operation-Relation              EFFECT  DISPLAY
Operation-Relation              USE     KABUKA-HYOU

Object          KABUKA-HYOU
Object-Relation         HAVE            HIZUKE
Object-Relation         HAVE            HAJIMENE
Object-Relation         HAVE            TAKANE
Object-Relation         HAVE            YASUNE
Object-Relation         HAVE            OWARINE
Object-Relation         HAVE            DEKIDAKA

Operation               kaku
Operation-Type                  SELF_EFFECT
Operation-Relation              GET     HIZUKE
Operation-Relation              GET     TAKANE
Operation-Relation              GET     YASUNE
Operation-Relation              GET     HAJIMENE
Operation-Relation              GET     OWARINE
Operation               yomu
Operation-Type                  SELF_DEPEND
Operation-Relation              GET     HIZUKE
Operation-Relation              RETURN  TAKANE
Operation-Relation              RETURN  YASUNE
Operation-Relation              RETURN  HAJIMENE
Operation-Relation              RETURN  OWARINE

Object          JIKU
Adjective               line
Adjective               scaled

Object          DEKIDAKA
Candidate-Class         Integer

Object          HIZUKE

Object          BANGOU
Candidate-Class         Integer

Object          TAKANE
Candidate-Class         Integer
Object          YASUNE
Candidate-Class         Integer

Object          HAJIMENE
Candidate-Class         Integer

Object          OWARINE
Candidate-Class         Integer

Object          ASHI
Object-Relation         HAVE            HIGE
Object-Relation         HAVE            JISSEN

Object          HIGE
Adjective               line
Adjective               straight

Object          JISSEN
Adjective               rectangle
Adjective               colored

End
```

図 2 質問記述の例
Fig. 2 Example of query description.

MOMOCLI は検索サブシステムからだけではなく MOMOCO からも参照されるので，その両方に対して効率よくサービスを行うため，MOMOCLI の管理を行うサーバプロセス「ライブラリマネージャ」を独立に作成した．ライブラリマネージャと検索サブシステムおよび MOMOCO との通信はソケットによる．

検索サブシステムと，ユーザインタフェース MOMOE の両者は1つのプロセスとして実現されている．MOMOE は検索サブシステムとユーザの対話を実現するウインドウベースのインタフェースで，SunView により実現されている．また，MOMO のプログラムの編集も MOMOE 上で行われる．

次章において，検索サブシステムの詳細を述べる．

3. 検索サブシステムの構成とその動作

3.1 質問記述の記法

検索サブシステムに入力されるものは質問記述である．質問記述には2.1節の(1)-(3)で述べた内容が記述される．現在，簡単な質問記述言語が定義されている．質問記述の例を**図2**に示す．また質問記述の読み方を付録に示す．

2.1 節で述べた以外に，オブジェクトの別名やオブジェクトの性質を示す形容詞を利用者は自由な語いで記述できる．これらは辞書をひくのに使われる．

また，あらかじめ各オブジェクトを実現するクラスや各オペレーションを実現するメソッドを指定することができる．例えば，明らかに整数で実現できるオブジェクトには Integer をクラスとして指定すればよい．あらかじめクラスやメソッドが指定された場合には，それだけ検索の範囲が狭くなるので検索効率が上がる．

質問記述の内容は MOMOE の画面上で視覚的に表現される（後掲の図7参照）．将来は図2のような質問記述言語によらず，MOMOE の上で絵を描きながら質問記述を作成していけるようにする予定である．そうすることによりユーザインタフェースが向上し，質問記述を少しずつ修正しながらインタラクティブに検索を繰り返すことも容易になる．

3.2 MOMOCLI の構造

MOMOCLI は2つの部分からなる．その1つは通常のクラスの集合であり，もう1つはオペレーションクラスの集合である．これらを**図3**に示す．

通常のクラスとはインスタンスを生成することができ，利用者の作成したいオブジェクトを実現するクラスの候補として検索される対象になるものである．これらのクラスは，Object と名付けられたクラスを最上位とした多重継承による半順序関係をなす．

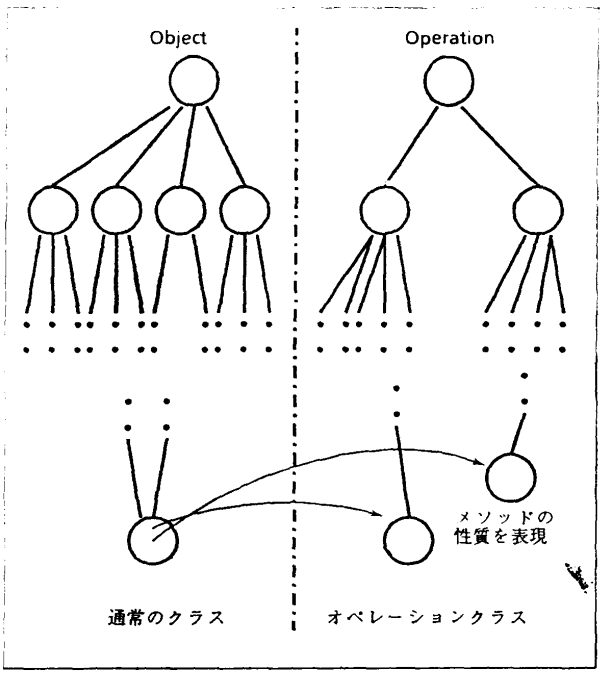


図3 MOMOCLI の構造
Fig. 3 Structure of MOMOCLI.

通常のクラスには，

(1) クラス名
(2) スーパクラスの名前
(3) クラスについてのコメント
(4) 辞書に登録する語い
(5) 知人の定義
(6) 構成変数の定義
(7) クラス変数の定義
(8) メソッドの定義

が記述される．多重継承を行う場合には，スーパクラスを複数指定する．知人は KNOW の関係に対応し，構成変数は HAVE の関係に対応する．ここで，メソッドの定義とは，

(8-1) メソッドを起動するためのメッセージの仕様
(8-2) メソッドが属するオペレーションクラス
(8-3) メソッドについてのコメント
(8-4) メソッドを実現するプログラム

から成っている．ただし，(8-2) は MOMOCLI に登録するクラスにのみ必要であり，利用者が作成するクラスには必ずしも必要ではない．通常のクラスの例を

```
class Window {
"I represent a window. In MOMO, we don't use MVC paradigm like Smalltalk."
superclass      Form
dictionary_entry        view, window, frame.
acquaintance    <Window> superWindow.
component       <OrderedCollection> subWindows;
                <Integer> borderWidth;
                <Form> borderColor, insideColor;
                <Rectangle> boundingBox;
                <PopUpMenu> redButtonMenu,
                yellowButtonMenu, blueButtonMenu;
                <Form> closedIcon.
instancemethods
<Window> openAt: <Point> aPoint extent: <Point> bPoint
"WindowOpen#1"
{
        .
        .
        .
}
        .
        .
        .
}
```

図 4 MOMO 言語によるクラスの例
Fig. 4 Example of MOMO class.

図 4 に示す．今後，単にクラスと表記した場合には通常のクラスを示すものとする．

オペレーションクラスとは，メソッドの性質について知人/構成変数モデルの立場から記述したものであり，継承の関係によって整理されている．クラスの上で定義されている各メソッドは，いずれかのオペレーションクラスによって，その性質が記述されている．オペレーションクラスの継承における最上位クラスは Operation と名付けられている．

オペレーションクラスには以下の項目が記述されている．

（1）クラス名
（2）スーパクラスの名前
（3）辞書に登録する語い
（4）メソッドの持っている関係および性質

ここで，メソッドの持っている関係とは，知人/構成変数モデルで定義されている EFFECT, DEPEND, GET, RETURN などの関係であり，メソッドの性質とは SELF-EFFECT と SELF-DEPEND である．これらの関係や性質は，サブクラスに継承される．ここでも多重継承は許される．

オペレーションクラスの例を図 5 に示す．

オペレーションクラスを導入した理由を述べる．利用者は「このようなオブジェクトを作りたい」という要求を出すだけでなく，「このようなオペレーションを実現したい」という要求を出すこともある．このような要求を満たすメソッドを検索する場合，すべてのクラスに定義されているすべてのメソッドを順に探索するのでは効率が悪い．クラスの検索に先立ってメソッドの検索をするためには，メソッドについての知識を何らかの形で整理しておかなくてはならず，それを実現するものがオペレーションクラスである．

```
class DrawLineByBitBlt {
superclass      DrawLine, BitBltOperation.
dictionary_entry        draw, draw-line.
relation                GET Point.
relation                DEPEND Point.
}
```

図 5 オペレーションクラスの例
Fig. 5 Example of operation class.

例えば，画面に何かを表示するというオペレーションを実現するメソッドは数多くのクラスで定義されており，それぞれの仕様，例えば引数の数や型などはまちまちである．利用者が「画面に表示を行いたい」という要求を出したとき，これらすべてのメソッドを同時に検索できれば，検索されたメソッドが定義されて

```
if is-candidate($M1,$v1) & is-operation-on($v1,$n1)
      & is-candidate($C1,$n1) & ! is-implemented-on($M1, $C1)
    then print("Rule E2 applied ", name($M1), " is removed from", name($v1),
        name($n1), name($C1)),
        rm-candidate($M1,$v1).
```

図 6 ルールの例
Fig. 6 Example of rule.

いるクラスが,「画面に表示を行うオブジェクト」を実現するための候補クラスとなる. この場合, 画面に表示するという性質, つまり画面に EFFECT するという性質を持ったオペレーションクラスがまず検索され, そのオペレーションクラスからのリンクをたどってメソッドが検索される. 個々のメソッドの細かい性質の差はそのオペレーションクラスのサブクラスによって分類されることになる.

Operation のサブクラスには, AskStateOperation と ChangeStateOperation の 2 つがあり, 前者は SELF-DEPEND の性質を持つものを表現し, 後者は SELF-EFFECT の性質を持つものを表現する.

オペレーションクラスの作成は, 現在はライブラリ作成者によって人手で行われているが, 将来は半自動化可能である. 自動化可能な部分はオペレーションの性質の記述であり, 自動化不可能な部分は辞書エントリとスーパクラスの決定である. 現在, 部品登録作業支援ツールを作成中であり, このツールはすでに登録されているオペレーションクラスをブラウジングする機能を備えており, 新しく登録するオペレーションクラスのスーパクラスの決定を容易にする.

現在の MOMOCLI には通常のクラスが 93 個, オペレーションクラスが 374 個登録されている. どちらもそのうちの約 1 割が多重継承を行っている. 通常のクラスのうち, Integer などの基本クラスと Set や Array などの Collection 系のクラスの仕様は, Smalltalk-80 のものを参考にした. この MOMOCLI は, ビットマップディスプレイ上でのユーザインタフェース構築に問題領域を設定してあり, ウインドウやグラフを実現するためのクラスが多く含まれている.

3.3 辞　　書

辞書は MOMOCLI 中の各クラスのソース中に記述されている「辞書登録用語い」から作成される, 単語からクラス名への対応表である. UNIX の簡易データベース構築ツール dbm によって実現されている.

辞書に登録する単語はいかなる自然言語のものであってもよい. ここでは, 英語を採用している.

3.4 検索ルール処理系

検索ルールのインタプリタは, 検索サブシステムに内蔵されており, C で記述されている. このインタプリタはワーキングメモリ上に質問記述を展開し, ルールにしたがって各オブジェクトおよびオペレーションに対する候補クラスの追加・削除, 候補の確実度の変更などを行う. ルールはあらかじめ決められた順に適用され, 一度適用されたルールは二度と適用されない (よってルールの競合も起こらない) ので, プロダクションシステムとしての能力は強いものではない.

検索作業は辞書引きも含めてすべて検索ルールの適用によって行われる. したがって検索サブシステムはルールインタプリタのほかには入出力手続きしか含まない.

検索ルールの文法は, この検索サブシステムのために特に定義した簡単なものである. ルールの例を図 6 に示す. $n1, $C1 などは質問記述中の特定の要素にマッチする変数であり, 例えば $n1, $n2, … などは質問記述中のオブジェクトに, $C1, $C2, … などはオブジェクトに対する候補クラスにマッチする. 変数の一覧を表 1 に示す.

オペレーションに対する候補は, 検索中はオペレーションクラスのまま取り扱われる. これらが候補メソッドに変換されるのは利用者に候補を提示する直前である. 利用者にはオペレーションクラスの存在を意識させない.

3.5 検索ルールの内容

検索ルール内容を要約すると次のようになる. 以下

表 1 ルールに用いる変数の一覧
Table 1 Variables in rules.

記　　号	マッチする対象
$n1, $n2, ...	オブジェクト
$v1, $v2, ...	オペレーション
$r1, $r2, ...	質問記述中の関係
$s1, $s2, ...	文字列
$C1, $C2, ...	クラス
$M1, $M2, ...	オペレーションクラス
$R1, $R2, ...	MOMOCLI 中の関係

のルールが順に適用されることによって検索が実行される．

（1） 質問記述中に示されたオブジェクト名，オブジェクト名の別名や形容詞，オペレーション名，について辞書を参照して候補を得る．

（2） 得られた候補の持っている関係や性質が質問記述で示されている関係や性質とマッチしていたら，その候補の確実度を上げ，また関係の対象に対して候補を追加する．例えば，クラスAがオブジェクトXの候補クラスであり，かつクラスBにはA型の知人があり，かつオブジェクトYがXを知っている (KNOW) という関係があれば，BをYの候補クラスとする．

（3） 質問記述中の関係から，候補オペレーションクラスを求める．例えば，質問記述中にあるオペレーションOについて “EFFECT X” という記述があり，オブジェクトXの候補としてクラスAがあるならば，A型に対して EFFECT を起こすようなオペレーションクラスをOの候補とする．

（4） 特殊なケースとして，3つ以上の構成変数を持つオブジェクトに対しては，OrderedCollection と Array を候補とする．これは，これらのクラスで十分実現可能な単純なデータ構造が比較的よく使われるのにもかかわらず，名前や関係では検索しにくいという経験則によるものである．

（5） オペレーションの候補になっているオペレーションクラスに対応するメソッドが，そのオペレーションの定義されているオブジェクトのいず

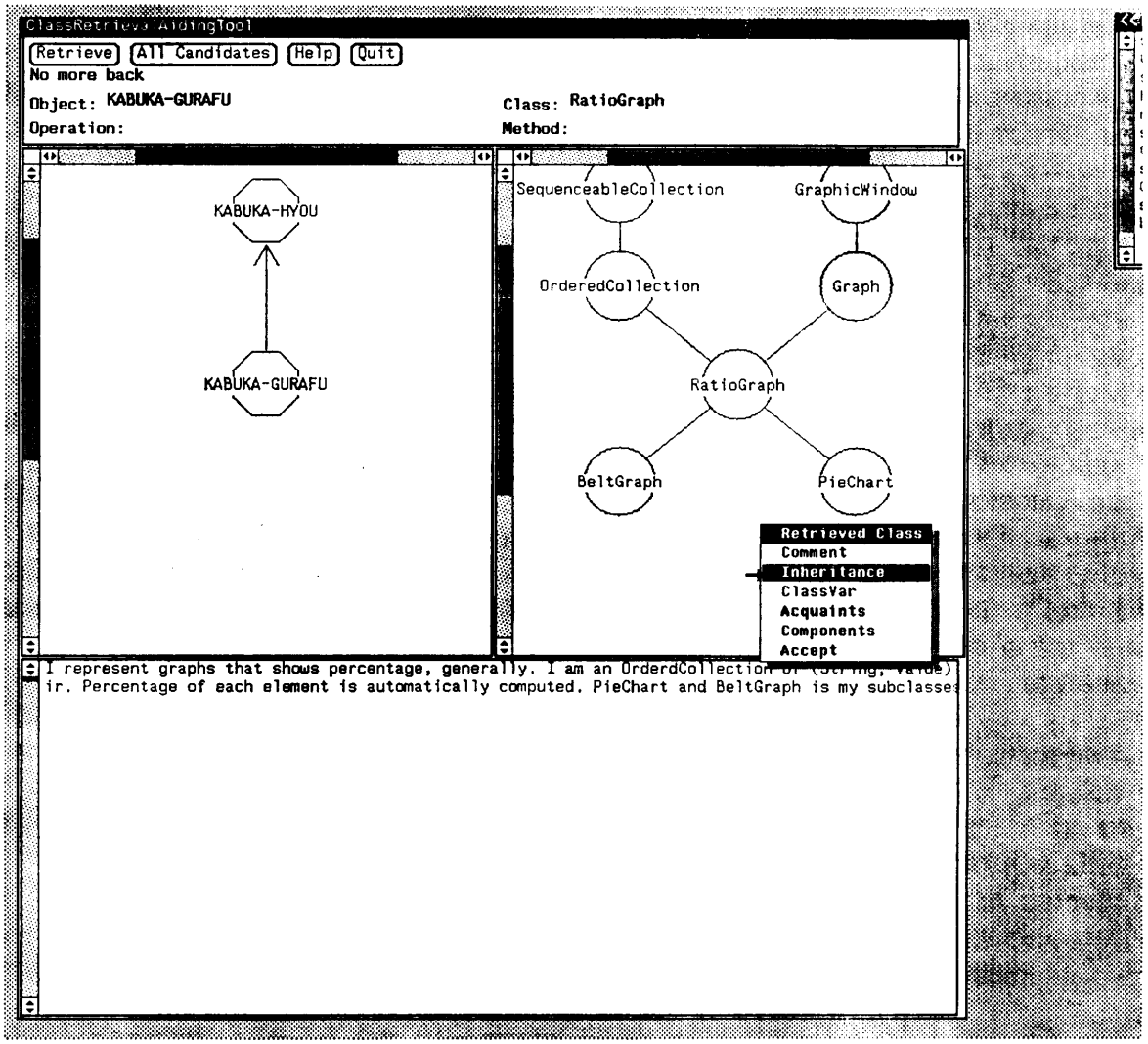


図 7 MOMOE の画面の例
Fig. 7 Example of MOMOE window.

れの候補クラスにおいても実現されえない場合，そのオペレーションクラスは候補から除外する．逆に，うまく実現されている場合は候補クラスの確実度を上げる．

(6) あるオブジェクトの候補クラスがA，Bの複数あり，BがAのサブクラスになっている場合は，Bの確実度をAより大きくする．

これらは，すべて汎用的なルールであり，このほかにここで用いている MOMOCLI の適用問題分野であるユーザインタフェースに関する特殊なルールを追加できる．

3.6 検索結果の確認

検索の結果は，MOMOE を通して利用者に提示される．先に述べたように，オペレーションに対する候補のオペレーションクラスは利用者には提示されず，それらはオブジェクトに対する各候補クラス上で定義されている候補メソッドとして提示される．

利用者は MOMOE の機能を用いて，候補クラスの仕様，KNOW や HAVE の関係，コメント，継承関係，および候補メソッドの仕様，コメント，DEPEND や EFFECT などの関係などを知ることができる．MOMOE の画面例を**図7**に示す．この図では左に質問記述の内容の一部，右に候補クラスについての情報（この場合は継承についての情報）を図示しており，また下に候補クラスについてのコメントをテキスト表示している．これらの情報を得て，利用者はどの候補を採用するかを決定する．

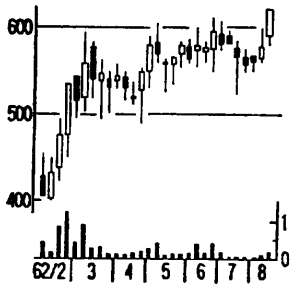
候補の決定は，一度にすべてのオブジェクト・オペレーションについてする必要はない．一部のオブジェクトやオペレーションについて候補を決定した後，残りのオブジェクト・オペレーションについての候補の検索をやり直すことができる．2回目以降の検索では，すでに決定した候補の情報を用いて候補を絞ることができるため，より有効な検索ができる．

現在，検索サブシステムの最終出力は検索結果そのものである．将来は，検索結果をもとにしたプログラムの雛形を出力することにより，プログラミング作業がより効率的に行えると期待できる．

4. 例　　題

ここでは，**図8**に示すような株価チャートを画面上に表示する，株価チャート作成問題を例とする．この問題における質問記述はすでに図2に示した．

株価チャートは株価の動きを示す KABUKA-GURAFU と出来高の大小を示す DEKIDAKA-GURAFU からなっている．これらは，それぞれ KABUKA-HYOU を知っていて，オペレーション kaku を実行するときにこれを参照する．KABUKA-HYOU は HIZUKE, HAJIMENE, TAKANE, DEKIDAKA などの要素からなる．このうち HAJIMENE, TAKANE, DEKIDAKA などは Integer であることがあらかじめ指定されている．



(出典：日本経済新聞 1987 年 8 月 20 日付)

図 8 株価チャートの例
Fig. 8 Example of stock chart.

この例題の質問記述では，オブジェクトの名前としては，故意に日本語のローマ字による名前を用いているところがある．これは，英語名を用いると3.5節で述べた辞書引きのルール(1)によって容易に候補が得られるため，例題としての興味が失われるからである．

第1回の検索の結果を**表2a**に示す．KABUKA-CHAATO の候補と KABUKA-HYOU の候補は3.5節のルール(3)により得られたものである．また，DISPLAY, JIKU, HIGE, JISSEN, の候補はオブジェクト名または形容詞表現から辞書をひいて得られたものである．KABUKA-GURAFU および DEKIDAKA-GURAFU の候補は KABUKA-HYOU の候補 Array を知っているという関係や，JIKU の候補 ScaledLine を持っているという関係によって得られたものである．

ここで，利用者は KABUKA-HYOU の候補 Array と OrderedCollection について，MOMOE を使って学習し，Array のサブクラス Table を採用することに決定し，第2回目の検索を行う．その結果，Table を利用する BarGraph 上のメソッド display などが，DEKIDAKA-GURAFU 上のオペレーション kaku の候補として得られる（表2b）．

表 2a 1回目の検索結果（部分）
Table 2a Result of the first retrieval (partially).

オブジェクト	候補クラス
KABUKA-CHAATO	OrderedCollection Array
KABUKA-GURAFU	Graph
DEKIDAKA-GURAFU	DiagonalGraph Graph
BANGOU	Integer
DISPLAY	DisplayScreen DisplayMedium DisplayText DisplayObject
ASHI	BarGraph LineGraph Window DiagonalGraph DisplayText ClosedFigure Path
KABUKA-HYOU	Array OrderedCollection
JIKU	ScaledLine HorizontalLine VerticalLine LineGraph Line Text Spline

表 2b 2回目の検索結果（部分）
Table 2b Result of the second retrieval (partially).

オブジェクト	候補クラス	候補メソッド
DEKIDAKA-GURAFU	DiagonalGraph	redraw (for kaku) display (for kaku) open (for kaku) openAt: extent: (for kaku)

5. 評価と比較

ここで示した例題を始めとした10個程度の例題で検索実験を行った．各オブジェクトに対して得られた候補クラスは最大9個であり，そのうち，通常上位3候補以内，悪くても5候補以内に最も適当と思われるクラス，または最も適当と思われるクラスのスーパクラスが得られることがわかった．候補の得られない場合もあったがこれは適当なクラスが実際に存在しない場合がほとんどである．ただしこれらの例題は筆者ら自身によって記述された質問記述によっているため，前述の例題のように辞書にある単語をなるべく使用しないなどの工夫をしているものの，設計思想や質問記述に用いる用語が固定化されているために実験が成功しているという危険もある．今後はより多くの人に利用してもらって辞書や知識ベースを強化していく必要がある．

我々の提唱したモデルに基づく形式的マッチングによる検索が本研究の主眼ではあるが，検索システムの能力が辞書の品質に依存することは事実である．しかし，本研究で用いたモデルやあるいは他のモデルを用いた形式的マッチングのみによって検索する手法には限界があり，辞書の導入はやむをえないものである．ここで示した例題で見られるように，辞書による検索とモデルのマッチングによる検索を共用することにより，互いの欠点を補うことができる．利用者がオブジェクト等につけた名前のうち，1つでも辞書に登録されている語にヒットするものがあれば，そこを手がかりにして検索を開始することができる．

あらかじめ与えられたクラスだけを検索の対象にするのであれば，本検索システムは，検索機能の強力なオンラインマニュアルと言うことができる．現在，クラス登録ツールの作成を進めており，これを組み込むことによってソフトウェア部品の再利用システムとして完全なものになる．登録ツールでは，多重継承による矛盾のチェックやオペレーションクラスの作成支援などを行う．

オブジェクト指向型言語のクラスの階層構造は，シソーラスまたはフレーム型の知識ベースと解釈することができ，それがオブジェクト指向型言語の1つの利点となっている．既存のオブジェクト指向のシステムでこの利点を生かしたものは少ないが，本研究では，クラスのライブラリにメソッドの性質を表現するオペレーションクラスをつけ加えて，クラスライブラリの知識ベース的側面を活用している．

1章で述べたように，本研究の動機は Smalltalk のクラス検索機能の貧弱さの解消であった．本研究で作成したシステムの検索機能は，辞書による検索だけをとってみても Smalltalk のカテゴリ名による検索機能を包含している．また MOMOE はブラウジング機能を備えており，現在は Smalltalk のブラウザよりもクロスリファレンス機能などの点で劣っているものの，将来機能拡張は可能である．

本研究に類似した研究成果としては，PARIS シス

テム[10]，モジュール機能の論理表現に基づいたモジュール検索[11]や Embley の抽象データ型再利用システム[12]がある．

PARIS は Partially Interpreted Schema と呼ばれる部品（ソースコードの一部をパラメータ化したもの）を再利用可能条件の形式的マッチングのみによって検索するものである．また文献 11) はモジュールの機能を述語によって表現し，利用者の与えた述語からの推論によってモジュールを検索するものである．これらは，辞書による検索機能を含めた本研究と比較すると検索能力が劣ると予想される．しかしながら言語に依存しない点が優れている．

文献 12) はキーワードやコメント文と部品間の関係を用いて抽象データ型を検索するもので，本研究とよく類似しているが，この自動検索システムはキーワードとコメント文のみを検索のキーとしており，関係は人手によってのみフォローされる．この点は関係も自動検索の手がかりとしている本研究と異なる．一方，この研究も言語に依存しない点が優れていると言える．

本研究がこれらの研究と比較して優っているのは，プロダクションルールを用いて検索戦略を柔軟に変更可能とした点と，継承機能を用いることにより利用者の必要に応じた抽象レベルの部品を検索できるようにした点である．

6. おわりに

現在，言語 MOMO の処理系はテスト段階にあり，このプログラミング環境は全体としてまだ実用の域には達していない．しかし，検索サブシステムはほぼ完成しており，検索のみに限定すれば利用可能である．

今後は前述のクラス登録ツールの作成のほか，MOMOE のクロスリファレンス能力の強化，検索結果からプログラムのスケルトンを生成する機能の追加などを行うことにより，より快適なプログラミング環境を得られることが期待できる．

謝辞 本システムの開発に当たり，岡村和男（現在松下電器産業），尾形哲（現在郵政省），河津正人（現在日本電気），菊田英明（日立計測エンジニアリング）各氏の協力を得た．

参考文献

1) Goldberg, A. and Robson, D.: *Smalltalk-80, The Language and Its Implementation*, Addison-Wesley, Massachusetts (1983).
2) 垂水浩幸，阿草清滋，大野 豊：MOMOCLI プロジェクト—全体像—，第 34 回情報処理学会全国大会論文集，5 U-1 (1987).
3) 河津正人，垂水浩幸，阿草清滋，大野 豊：MOMOCLI プロジェクト—ユーザインターフェース MOMOE—，第 34 回情報処理学会全国大会論文集，5 U-2 (1987).
4) 菊田英明，垂水浩幸，阿草清滋，大野 豊：MOMOCLI プロジェクト—ライブラリマネージャー，第 34 回情報処理学会全国大会論文集，5 U-3 (1987).
5) 垂水浩幸，阿草清滋，大野 豊：MOMOCLI プロジェクト—検索例—，第 35 回情報処理学会全国大会論文集，1 R-3 (1987).
6) Tarumi, H., Agusa, K. and Ohno, Y.: Acquaintance/Instance Variable Model for Object-Oriented Programming, *Proc. IEEE 9th COMPSAC*, pp. 69-73 (1985).
7) 垂水浩幸，岡村和夫，阿草清滋，大野 豊：クラス再利用支援のためのオブジェクトモデル，コンピュータソフトウェア，Vol. 3, No. 3, pp. 61-70 (1986).
8) 垂水浩幸，阿草清滋，大野 豊：オブジェクト指向型言語 MOMO の特徴とその実現方式，日本ソフトウェア科学会第 2 回大会，4 A-1 (1985).
9) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley, Massachusetts (1986).
10) Katz, S. et al.: PARIS: A System for Reusing Partially Interpreted Schemas, *Proc. 9th ICSE*, pp. 377-385 (1987).
11) Yoshida, H. et al.: Retrieval of Software Module Functions Using First-Order Predicate Logical Formulae, *Lecture Notes in Computer Science*, Vol. 221 (Proc. 4th Logic Programming Conference), pp. 117-127, Springer-Verlag, Berlin (1985).
12) Embley, D. W. and Woodfield, S. N.: A Knowledge Structure for Reusing Abstract Data Types, *Proc. 9th ICSE*, pp. 360-368 (1987).

付録 質問記述の読み方

行頭のキーワード	意味
System	プログラムの名前
Object	オブジェクトの名前
Object-Relation	オブジェクト間の関係：***HAVE*** または ***KNOW***
Adjective	オブジェクトの性質を表現する任意の英語形容詞
Alias	オブジェクトの別名
Candidate-Class	候補クラスの指定

Operation	各オブジェクトが実行するオペレーションの名前
Operation-Type	オペレーションの性質: ***SELF_EFFECT*** または ***SELF_DEPEND***
Operation-Relation	オペレーションと他のオブジェクトとの関係: ***GET, RETURN, EFFECT, DEPEND, USE, HANDLE*** のいずれか
Candidate-Method	候補メソッドの指定

(昭和62年9月 4 日受付)
(昭和63年1月19日採録)

垂水 浩幸(正会員)

1960年生. 1983年京都大学工学部情報工学科卒業. 1985年同大学院工学研究科修士課程情報工学専攻修了. 同年より同大学院博士後期課程に在籍. 情報システム工学講座に所属. ソフトウェア工学に関する研究に従事している. ソフトウェア再利用技術, ソフトウェア設計技法, ソフトウェア開発環境等に興味を持つ. 日本ソフトウェア科学会会員.

阿草 清滋(正会員)

1947年生. 1970年京都大学工学部電気第二学科卒業. 1972年同大学院工学研究科電気工学第二専攻修士課程修了. 1974年より京都大学工学部情報工学科に勤務. 1986年から87年までカリフォルニア大学客員研究員. 現在, 助教授. 工学博士. ソフトウェア工学に関する研究に従事. とくに要求分析, ソフトウェア仕様化技法, ソフトウェア部品化技法, ソフトウェア開発モデルなどに興味をもつ. また, マンマシンシステム, 分散処理システム等に関する研究も行っている. 1985年度情報処理学会論文賞. 電子情報通信学会, 日本ソフトウェア科学会会員.

大野 豊(正会員)

1924年生. 1946年東京大学工学部機械工学科卒業. 同年より国鉄鉄道技術研究所に勤務, 座席予約システム, 新幹線運転管理システムなどの研究・開発に従事. 1972年より京都大学工学部情報工学科教授, 情報システム工学講座担任. 工学博士. 京都大学情報処理教育センター長を兼任. 現在, ソフトウェア工学, システム性能評価, 分散データベース, コンピュータグラフィックスなどの研究を行っている. 1960年電気学会進歩賞, 1968年電子通信学会業績賞, 1971年紫綬褒章, 1975年情報化個人表彰, 1985年度情報処理学会論文賞. 計測自動制御学会, 日本機械学会等の会員. 本学会理事, 副会長, 計測自動制御学会理事, 第3回日米コンピュータ会議議長, 第6回ソフトウェア工学国際会議議長, 第12回巨大データベース国際会議議長, 日本ソフトウェア科学会理事長等を歴任. 1985年よりシグマ・システム開発委員会委員長. 1987年より本学会会長. 著書「オンラインリアルタイムシステムの計測」ほか.