ソフトウェア工学 113－5
（1997．3．7）

図形表現とテキスト表現を併用したプログラム実行状態の可視化

太田 一郎　井上 勝行　魚井 宏高　首藤 勝
大阪大学大学院基礎工学研究科

概要

オブジェクト指向プログラミングの普及に従い，そのデバッグの難しさが指摘されるようになってきている．その一つは実行時環境におけるインスタンスの状態の把握の難しさであり，もう一つはクラス継承から生ずるコンテキスト追跡の難しさである．前者の問題に関しては実行時環境におけるオブジェクトの可視化によって支援を行う試みが行われている．筆者らは後者の問題に関しても可視化の手法を適用することが有効ではないかと考え，制御構造の可視化を行う実行時環境 *Runtime9* の開発を行った．*Runtime9* はテキスト表現と図的表現を併用した折衷様式の表現を用いて制御構造の直感的な把握を助け，本来は異なるコンテキストのプログラムコードを実行系列に従って仮想的に一つの制御構造として可視化することで実行時のコンテキスト追跡を支援する．

Presenting the Runtime Information of Programs in Mixture of Textual and Graphical Representations

Ichiro OHTA　Katsuyuki INOUE　Hirotaka UOI　Masaru SUDO
The Faculty of Engineering Science, Osaka University

Abstract

Programs written in object oriented manner are getting popular today. It's also getting apparent, however, that debugging such programs needs the profound understanding of 1) their runtime instances and 2) control flow. With visualization techniques, several studies have been done to support the better understanding of runtime instances. Having thought those techniques also could help understanding control flow, we developed *Runtime9*, the runtime environment for Java*. With mixture of graphical and textual representations, *Runtime9* visualizes control structures of methods invoked in form of one virtual control structure, enabling us to trace program execution easily.

1.はじめに

オブジェクト指向プログラミングは今や常識的な手法として定着しつつある．ところがオブジェクト指向プログラミングが普及するに従い，そのデバッグの難しさが指摘されるようになってきた．一般的にオブジェクト指向プログラムにおいてデバッグの対象となる誤りは，おおむね以下の何れかに分類できると考えられる．

・クラス設計の誤り
・メソッドの実装の誤り
・メソッドの使用方法の誤り

テスト段階においてクラス設計に誤りが直接見

* The registered trademark of Sun Microsystems, Inc.

つかるということは考えづらく，通常はまずメソッドの実装や使用方法に関する誤りが発見されると考えられる．そうした誤りがないにも関わらず，プログラムが予期しない動作をする場合にクラス設計に疑いが生じる．そこでデバッグ作業においてはまずメソッドの実装や使用方法の誤りを発見する必要がある．これら実装段階の誤りを発見する作業の鍵となるものとしては次の2点があげられる．

(a) オブジェクトの状態の把握

あるオブジェクトのもとでメソッドの振る舞いが正しいかどうかを判断するには，そのオブジェクトの状態について十分に把握しておく必要がある．ところがオブジェクトは単純なデータ型の変数と異なり，複雑な状態を持ちうるためにその把握は容易でない．また，オブジェクト間に見られるhas-aやusesのような関係の概念も問題をさらに複雑にしている．これらの関係はいずれもオブジェクトへの参照メンバという形で実装され，オブジェクトの内部状態を表現するパラメータの一つとして取り扱われる．デバッグ時にオブジェクトの内部状態を検査する場合，これらのオブジェクト参照がどのような関係を意味しているのか，あるいは参照先のオブジェクトの状態がどうなっているのか知りたいという要求がある．しかし従来から用いられているソースコードレベルデバッガの大半はこれらの要求に答えるという点で機能的に不十分である．

(b) コンテキスト追跡

メソッドが正しく用いられているかどうかを判断するためには，どのような状態のコンテキストからメソッドが呼び出されているのか，制御の流れを追跡して把握する必要がある．一般的な制御構造は連接，分岐，反復から成り，反復や分岐の条件に注目すれば制御の流れを追跡できる．これを妨げるものとして関数や手続き呼び出しによって起こるコンテキストの切り替えがあげられるが，構造化プログラムの場合は階層的に手続きが構成されていることが多く，しかも呼び出される手続きは静的に決定されるためコンテキストの切り替えは予測しやすい．一方オブジェクト指向プログラムでは，差分定義に起因するコンテキスト切り替えが頻繁に起きることに加え，起動されるメソッドも動的に決定されることが多いことから追跡がしづらい．暗黙のメソッド呼び出しや委譲，例外処理といった機構もそれを助長している．

実行時のオブジェクトの状態の把握が難しいという問題に関して，可視化技術を適用することで改善を試みている事例としてはVion-Duryらの分散オブジェクト可視化システムである Virtual Images [1]，Pauwらのシステム [2]，Bulestone社のLOOK! [3]などがあげられる．しかし，実行時のすべてのオブジェクトを平等に可視化するこれらのシステムでは，個々のオブジェクトを識別することが難しい．重要なオブジェクトに着目してその状態を把握するためにはその情報への基点となる何らかのポインタが必要になるが，一般的にはコンテキストに表れる識別子を用いてその識別を行うものと考えられる．識別子はコンテキストのスコープに支配されるため，オブジェクトの状態を把握する上で結局はコンテキストの状態を理解することが重要な鍵となってくる．

ところが先ほども述べたように，オブジェクト指向プログラムではコンテキストの切り替わりが頻繁におき，コンテキストの状態を把握しづらい．そこで筆者らは，実行中のコンテキスト追跡を支援するために，コンテキストの制御構造を単体で可視化するのではなく，本来は異なる複数のコンテキストの制御構造を一つの制御構造として結合した形で可視化する表現を考えた．筆者らが**畳み込み表現**と呼ぶこの表現を用いて制御構造を可視化することで，コンテキストのどの時点で別のコンテキストに切り替わるのか，あるいは逆に現在実行されているコンテキストにどこで切り替わったのかを把握することが容易になる．

しかし図的表現を用いて可視化する場合，その表現に必要なスクリーン上の面積が大きくなるという問題が存在する．本研究では図的表現に必要な

面積は得られる直観性とのトレードオフと考え，すべてを均一に可視化表現するのではなく，着目している部分のみに図的表現を用いることで表現に必要な面積を抑え，その有効な利用を試みる．図的表現と従来のテキスト表現を使い分けるこの折衷様式の形式を**混在表現**と筆者らは呼んでいる．

2.関連研究

オブジェクト指向プログラムの動的な振る舞いを可視化することで，デバッグや保守に役立てようという試みは数多くなされている．オブジェクト指向分析，設計では，オブジェクトの系の動的な振る舞いを視覚的に表現するものとしてシナリオダイアグラムと呼ばれる図がよく用いられる．このシナリオダイアグラムに基づいて実行時のオブジェクトの振る舞いを可視化するシステムの例としては，Koskimies と Mössenböck の Scene がある [4]．また，メッセージ送受信の可視化に加え，インスタンスの生成，クラス階層の情報の可視化を行う例としては Bluestone 社の LOOK! [3]，Jerding らの Groove [5]などがあげられる．これらのシステムにおいては実行時のすべてのオブジェクトを平等に取り扱って可視化するため，個々のオブジェクトの識別が難しいという問題がある．前節でも述べたが，一般的に識別子を通してオブジェクトの判別を行う場合が多いことを考えると，最終的にはそれらに着目しつつコンテキストの状態を把握することが重要になってくる．

本研究では，図的表現とテキスト表現の折衷様式である混在表現を可視化に取り入れているが，混在表現をプログラミング環境に導入した例として Erwig と Meyer の研究 [6]があげられる．Erwig らは図的表現とテキスト表現のそれぞれの長短所が相補的であることに着目し，図形とテキストを混在させてプログラムを記述するためのメタフレームワークについて提案している．プログラマが好みの言語に対して自由に図形表現を取り入れることが望ましいと考え，属性文法の文法記号の一つとして図形を等価変換する枠組みについて述べているが，実行時環境への適用はまだ考慮されていない．

3.本研究で用いる可視化技法

本節ではオブジェクト指向プログラムの実行時のコンテキスト追跡を支援するために，その作業を妨げる要因について考察し，それを踏まえた上で具体的な可視化表現を検討する．妨げる要因は動的束縛とクラス継承による差分定義であると前節で簡単に述べた．まず，動的束縛について述べると，呼び出されるメソッドを実行時に決定するこの機構は遅延束縛とも呼ばれ，ソフトウェアを構成するオブジェクト間の結びつきを緩やな状態に保って再利用性を向上させる働きがある．ところが呼び出し先が動的に決定されるという性質のために，コンテキスト追跡を行う上では制御の移り先となるコンテキストが予想しづらい．また継承コンストラクタの暗黙的な呼び出しや，委譲によるメソッド呼び出しといった明示的なメソッド呼び出しによらないコンテキスト切り替えもこの問題をさらに複雑なものにしている．次にクラス継承によるプログラムの差分定義について見てみると，これには共通の処理を行う部分のコードを基底クラスのメソッドとして集約することで，プログラム記述量と保守コストを減少させるという利点がある．しかしコンテキスト追跡をする場合，共通処理を行うメソッドを実行しているコンテキストが，どのメソッドのコンテキストから呼び出されているかをとっさに判別しづらいという問題がある．以下では，これらのコンテキスト追跡の難しさを考慮した制御構造の可視化方法について述べる．

制御構造の可視化を行うにあたり，その基本方針として本研究では PAD を採用した．候補としてはフローチャートや NS 図も考えられたが，フローチャートは制御の流れの表現には向くが構造の表現には不適切である．また NS 図については構造を表現できるが，制御構造における条件の支配関係について PAD の方がわかりやすいと思われた．以下では PAD をベースとして，本節の最初で述べたコンテキスト追跡の難しさを考慮した可

視化の手法を提案する．既に述べたように，オブジェクト指向プログラムではコンテキスト間の呼び出し関係が把握しづらいためにコンテキスト追跡が難しい．そこで本研究ではコンテキスト間のつながりを視覚的に表現することを考え，**制御構造の畳み込み**と呼ぶ表現を可視化方法に導入した．その基本的なアイデアは，制御構造の可視化を行う場合にメソッドの呼び出し部分をそのメソッドの内容（制御構造を可視化したもの）で置き換えるというものであり，本研究ではこの表現を**畳み込み表現**と呼んでいる．図 1 は畳み込み表現の一例である．同図(a)はあるクラス B とその基底クラス A の初期化メソッドを PAD で表現したものであり，同図(b)はクラス A の初期化メソッド呼び出し部分をその内容で置き換えたものである．このように本来は異なる制御構造を仮想的に一つの制御構造とみなして可視化して，コンテキストとコンテキストの結びつきを強調することで，コンテキスト追跡の作業を支援できるものと考えられる．

ところが制御構造の可視化に限らず，可視化表現はその表現に必要な面積が大きいという問題がある．特に本研究で行う制御構造の可視化では，畳み込み表現の導入によりこの傾向が助長されることが予想される．しかし可視化表現が必要とする面積は，その導入によって得られるわかりやすさとトレードオフの関係にある．そこで可視化表現に必要な面積が大きくなることについては是認するかわりに，できるだけ面積を効率よく利用して情報を提示する方法はないか考察した．一般的に制御構造は階層を形成するが，デバッグやトレース作業の際にプログラマがある時点で注目しているのはその一部分であり，しかもその着目点は作業が進行するにつれ変化する．例えばデバッグ作業を開始したばかりであればプログラムの大まかな制御の流れについて着目するであろうし，その調査が進むにつれ制御構造のより深い階層へと関心を移すであろう．こういった仮定から，制御構造において着目されている部分に焦点を当てることが考えられる．着目されている部分を中心に PAD によって可視化を行い，それ以外の部分については必要な面積が小さい別の表現を用いて可視化するが，そのような表現としてはデバッグという作業の性質上，プログラムコードを省略せずに表現できるものが好ましい．本研究ではそのような要求から，注目していない部分をテキストを用いて表現することにした．図 2 は Boyer-Moore の文字列検索アルゴリズム [7]の一部を PAD で表現したものである．手続きを構成する複文の階

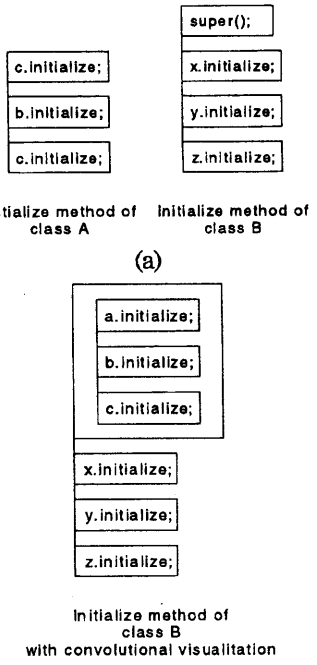


図 1畳み込み表現の例

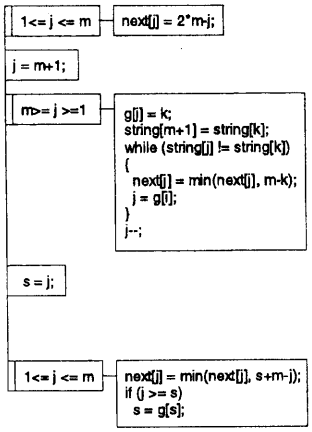


図 2 混在表現の例

層のうち，最も深い階層に位置する複文についてのみテキストで表現されている．筆者らはこのようにテキスト表現と図的表現の折衷様式の表現形態を**混在表現**と呼んでいる．

本研究では，以上に述べた可視化表現を用いて制御構造を可視化する実行時環境 *Runtime9* の開発を行った．次節ではその設計の概要について，可視化表現を実際にどのように導入するかという点を中心に述べる．

4.*Runtime9* における可視化表現の適用

Runtime9 を設計するにあたり，筆者らが基本機能として想定したものを以下にあげる．

- 動的に実行位置とスコープ中にあるオブジェクトの状態を表示．これはデバッガやトレーサが持つべき基本的機能である．
- 実行経過を溯る，あるいは進めることで任意の時点での実行状態を再現．実行状態の変遷に関してより詳しく調査することを支援する．
- 制御構造は，畳み込み表現と混在表現によって可視化．制御構造の可視化の方法は，対象となるプログラミング言語に依存する．
- 可視化におけるおけるテキスト表現と図的表現の使い分けは基本的に自動．ただし任意の指定も可能．

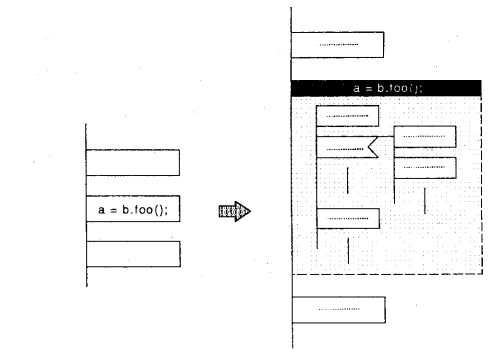
以下では主にトレース対象となるプログラミング言語に対して，どのように可視化を適用するかについて述べる．*Runtime9* はその対象となるプログラミング言語として Java を採用しているが，前述したようにその制御構造の可視化は PAD をベースとして行う．PAD では連接 (concatenation)，反復 (iteration)，選択 (selection) に対応するアイコンを用いて構造化プログラムを視覚的に表現するが，Java はこれらのアイコンでは十分に表現できない制御構造を持っているため *Runtime9* ではそれらに対応する PAD 表現を拡張している．

次に畳み込み表現をどのように適用するかについて述べる．Java の言語仕様 [8]によると，メソッドの呼び出しが現れるのは式中においてのみである．そこでコンテキストにおいてメソッドの呼び出しが行われる際に，その式を表すブロックをメソッドの内容で置き換えることで畳み込み表現を適用する．

図 3 はこの方針に基づいて，コンテキスト中に現れた b.foo()というメソッド呼び出しに畳み込み表現を適用した例である．式中に複数のメソッド呼び出しが式に含まれる場合は，それぞれの呼び出し時に対応するコンテキストの可視化表現が矩形内に表示される．畳み込み表現において制御構造を囲む矩形は一つのコンテキストを表しており，以降ではこの矩形を**コンテキストボックス**と呼ぶ．畳み込み表現をメソッド呼び出しに適用していけば，このコンテキストボックスが入れ子になった表現になる．実行中のコンテキストに対応するコンテキストボックスと，そのコンテキストボックス内で実行されている部分をそれぞれハイライト表示することで実行時の制御の様子を示す．

以上のように畳み込み表現で可視化した制御構造に対して，混在表現を適用する．前に述べたように，デバッグやトレース作業中にプログラマが着目するのは制御構造の一部分であることが多い．*Runtime9* はこの仮定を踏まえ，制御構造の階層（複文）を単位にしてテキスト表現と可視化表現を使い分ける．以下はこれらの表現の適用の制御方針について述べる．

制御構造は既に述べたように，コンテキストボックスが入れ子になった形で表現される．実行中のコンテキストボックスはハイライト表示されるが，



A Method Invocation in an expression　　A Method invocation, folded out.

図 3 メソッド呼び出しへの畳み込み適用例

この実行中のコンテキストボックスのことを**カレントコンテキストボックス**と呼ぶ．利用者が最も注目する部分はこのカレントコンテキストボックスであると考え，利用者のビューに対する操作は，基本的にカレントコンテキストボックスを対象として適用するものとした．畳み込み表現と混在表現の自動／手動制御も，基本的にカレントコンテキストボックスを対象として行われる．

これらのうちまず畳み込み表現に関しては，実行位置がメソッド呼び出しにさしかかった時点で自動的に適用する．畳み込み表現は本来継承メソッド呼び出しの際のコンテキスト追跡を支援するためのものであるが，実用的には継承関係に無いクラスのメソッドに対しても畳み込み表現を適用できた方が良い．これは，利用者がコンテキストボックス上にある式を表すブロックをダブルクリックすることによって行う．展開されたコンテキストボックスは再度のダブルクリックで縮退する．

一方混在表現の自動制御は，複文のネスティングに着目して行う．手続きは複文の階層により構成されるが，実行時には実行中の文の階層に着目することが多いと考えられる．そこで，最初は手続きを構成する一番浅い階層にある文についてのみ図的表現を用いて可視化を行い，実行が進んで深い階層に制御が到達するのに従ってその階層を可視化する．ただし利用者が大局的な流れだけに注目したい場合や，逆にまだ未到達の階層の文に注目したい場合などを想定して，ある範囲内で可視化するような指定手段も提供している．またこのような制御手段以外にも，実行位置を基準にして図的表現を適用するレベルを変更する窓関数的なフィルタも用意する．これは実行位置から離れてあまり注目されていないと思われる部分にテキスト表現を多用することで，面積を効率よく利用しようというものである．*Runtime9* のプロトタイプの実装では文の実行順序に着目し，実行位置より前，もしくは後ろの文になるほどテキスト表現の割合が高くなるフィルタを用意している．

5.実装について

本節では前節で述べた設計に基づいて行った *Runtime9* プロトタイプの実装について簡単に述べる．*Runtime9* はトレース対象であるプログラムの実行トレース情報を予め収集する．この際，対象となるプログラムソースにトレース出力コードを挿入した中間プログラムを生成するサブシステムが CodeAGENT である．CodeAGENT はトレース対象となるプログラムソースを読み込み，実行位置とオブジェクトの変更履歴情報を出力するコードを挿入した中間プログラムを作成する．CodeAGENT は，これらのトレース情報出力コードを挿入するパーサ部分と，実行時にそれらのコードが用いるランタイムライブラリから構成されている．一方 *Runtime9* もまたソースプログラムの構文解析を行い，制御構造に関する情報を得て制御構造の可視化を行っている．*Runtime9* は可視化した制御構造をベースとして，中間プログラムが出力したトレース情報をもとに実行時の模様を再現する．以上の関係のモデル図が図 4である．CodeAGENT と *Runtime9* のいずれも Java を用いて実装されており，Java プログラムの実行が可能なすべてのプラットフォームで動作する．

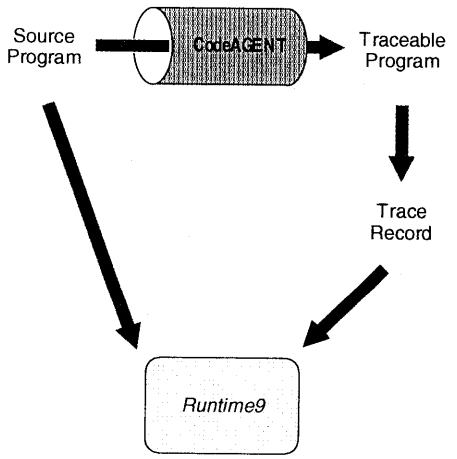


図 4 ***Runtime9*** プロトタイプの実装モデル

6.評価と考察

畳み込み表現の評価を行うにあたり，まずjava.awt.FileDialogクラス（以下FileDialog）のコンストラクタを例にあげる．FileDialogはObjectからComponent, Container, Windowを経てDialogから派生している．FileDialogのコンストラクタでは逆にDialog, Window, Container, Componentの順にそれぞれのコンストラクタが呼び出されるが，Window()とComponent()のコンストラクタを除いて，すべて明示的に親クラスのコンストラクタを呼び出している．図5はFileDialogのコンストラクタを可視化した例である．コンストラクタの呼び出し部分が畳み込み表現によって可視化されており，それぞれのコンストラクタが実行されるコンテキストのつながりがわかりやすくなっているだけでなく，親クラスの初期化処理を含んだFileDialogの初期化処理全体を一望できる．

次に挙げるのは，共通処理を行うコードのコンテキスト判別を助ける例である．抽象基底クラスを定義して派生クラス間で共通する処理を集約している場合，畳み込み表現を用いることで集約された共通コードがどのコンテキストから呼び出されているかをわかりやすく可視化することができる．例としてあげるFigureクラスは図形オブジェクトを表す抽象クラスであり，java.awt.Graphicsクラス（以下Graphics）のインスタンスに対して描画を行うdrawOnメソッドを持つ．Figureは主に描画イメージのキャッシュ領域の管理を行い，drawOnメソッドではキャッシュイメージの有効性をチェックしてGraphicsインスタンスに転送する処理のみを行う．そのためFigureの派生クラスではdrawOnメソッドをオーバライドするなどして，キャッシュイメージに対して個々のクラスに応じた描画を行う必要がある．派生クラスにおけるdrawOnメソッドの基本的な処理の流れは，1)キャッシュバッファ描画用のGraphicsインスタンス取得 2)クラス固有の描画処理 3) Figureのdrawonメソッド呼び出し，となっている．これらのメソッドを実行するコンテキストを，畳み込み表現により可視化したのが図6である．Figureのdrawonメソッドを実行しているコンテキストと，そのメソッドを呼び出しているコンテキストの関係がわかりやすくなっている．

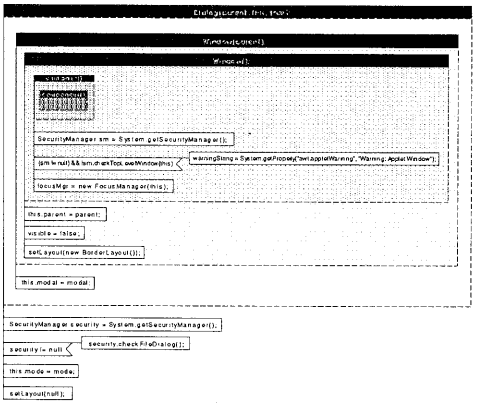


図 5 FileDialogのコンストラクタ

以下では，評価で得られた知見をもとに考察を行う．まず，カプセル化の概念を考慮した際の畳み込み表現の是非について述べる．カプセル化はオブジェクト指向プログラミングにおける重要な概念であり，オブジェクトの内部状態に関する詳細な情報を隠蔽することで，外部からそのオブジェクトを一つのブラックボックスとして利用できるようにする．しかし実行時において畳み込み表現

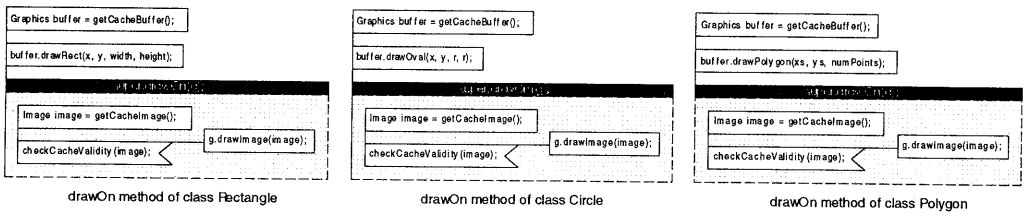


図 6 Figureの派生クラスのdrawOnメソッド

を用いると，コンテキストの区別なく隠蔽されているメソッドの振る舞いが可視化されてしまう．内部の振る舞いに注意を払う必要が無いというカプセル化の利点が失われてしまうのではないかという疑問が生じるが，これについては次のように考えることができる．畳み込み表現は本来継承したメソッド呼び出しの表現に適用することを想定している．このような継承メソッドの呼び出しによって基底クラスのコードを再利用する場合，その内部の詳細な振る舞いについての知識が必要とされる場合が多い．このような状況では派生クラスから見て基底クラスはカプセル化されているとは見なせないだけでなく，実行時には分散する記述を一望できた方がむしろ好ましい．

次に *Runtime9* の実装上の問題点について触れておく．*Runtime9* は実行時のトレース情報をもとに実行時環境を再現する実装方式を採用しているため，実行時のオブジェクトの状態については再現できるものの，メソッド実行に伴うファイル入出力や画像描画といった副作用は再現されないという問題がある．またこれとは別にトレース対象となるプログラムのそれぞれについて，トレース情報出力のためのコードを埋め込んだ中間プログラムを生成する手間が必要である．筆者らはこの作業を自動化するためのに，専用の Java プロジェクトマネージャ JPM を開発している [9]

7.まとめ

本研究では，実行時環境におけるオブジェクト指向プログラムのコンテキスト追跡を支援するための可視化の手法について延べ，その手法を用いた実行時環境 *Runtime9* の開発を行った．本来は異なったコンテキストに属する複数の制御構造を，仮想的に一つの制御構造と見なして可視化する畳み込み表現を用いることで，実行時環境におけるオブジェクト指向プログラムのコンテキスト追跡を支援できるものと考えられる．また図的表現とテキスト表現の折衷様式である混在表現を用いることで，可視化表現に必要な面積を効率的に利用できる．

今後の予定としては，制御構造の理解をより効果的に行うための混在表現，畳み込み表現の適用方法に関する考察があげられる．また *Runtime9* に関連するものとしては，マルチスレッド環境への対応や，他のオブジェクト指向言語への適用が考えられる．

参考文献

[1] Vion-Dury, J. and Santana, M.: "Virtual Images: Interactive Visualization of Distributed Object-Oriented Systems", *9 th Annual Con-ference on OOPSLA'94*, 1994.

[2] Pauw, W. D., Helm, R., Kimelman, D. and Vissides, J.: "An Architecture for Visualizing the Behavior of Object-Oriented Systems", *8 th Annual Conference on OOPSLA'93*, 1993.

[3] Bluestone, Corp.: LOOK!, `http://www.bluestone.com`, 1996.

[4] Koskimies, K.: "Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs", *18th International Conference on Software Engineering (ICSE)*, 1996.

[5] Jerding, D. and Stasko, J. T., " Visualizing Message Patterns in Object-Oriented Program Executions", *Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-15*, 1996.

[6] Erwig, M. and Meyer, B.: "Heterogeneous Visual Languages - Integrating Visual and Textual Programming", *Proceedings of 11th IEEE Symposium on Visual Languages*, 1995.

[7] 石畑 潔: "岩波講座ソフトウェア科学3アルゴリズムとデータ構造", 岩波書店, 1989.

[8] Gosling, J., Joy, B. and Steele, G.: "The Java Language Specification", *Corporate and Professional Publishing Group*, Inc., 1997.

[9] 竹谷 英奉: "視覚的プログラミング環境 *Runtime9* におけるトレース情報支援ツールの実装", 大阪大学基礎工学部情報工学科特別研究報告, 1996.