2007－SE－155 (19)
2007／3／23

オブジェクト指向に基づくプログラム動作の可視化

高橋 裕康† 紫合 治‡
†東京電機大学大学院情報環境学研究科
‡東京電機大学情報環境学部

本研究では，従来の手続き型プログラムの可視化システムとは異なり，オブジェクトの状態の変化を可視的に表現することにより，より容易にオブジェクト指向に基づくプログラム動作を捉えることができるシステムの開発を行った．システムの実装手法として，アスペクト指向を用い，対象となる JAVA 言語プログラムを直接的に変更せずに可視化することを実現した．

Visualizing a Program Operation based on Object-Oriented Programming

Hiroyasu Takahashi† and Osamu Shigo‡
†Graduate school of Information Environment Engineering ,Tokyo Denki University
‡School of Information Environment, Tokyo Denki University

Abstract

We developed a program visualization tool to understand the dynamic behavior of complex Object-Oriented programs. The system show a diagram of objects and its relation, which is dynamically changed along the program execution. We used Aspect-Oriented Programming to instrument JAVA source code.

1 はじめに

近年，社会の急激な IT 化とともに，学校や会社等でも，プログラミングに携わる人が増加している．それら多くの人々は，教育や仕事の場を通し，オブジェクト指向プログラミングを利用している．これは，オブジェクト指向プログラミングを利用することにより，プログラムの保守性や再利用性の向上というメリットを得ることができるためである．しかしながら，オブジェクト指向によるオブジェクトのインスタンス化やガベージコレクションといったプログラム動作により，簡易な手続き型プログラムのコードとは異なり，初心者がプログラムの動作を捕らえることは容易ではない．そこで本研究では，プログラム動作の可視化を行う既存のシステムを調査し，よりオブジェクト指向プログラムに特化した可視化システムの開発に取り組んだ．また，本可視化システムの開発には，対象となるプログラムのソースコードを直接変更することを回避するために，アスペクト指向による可視化情報の抽出を行った．

2　プログラムの可視化

ここで言うプログラムの可視化とは，任意のプログラムコードに対して，そのプログラムの動作をわかりやすく図的に表現することによって，主にデバッグ等を容易にする方法である．

ここでは，手続き型とオブジェクト指向型のプログラミング言語に対する既存の可視化システムを調査し，本研究で開発を行う可視化システムが用いる新たな可視的表現方法についての考察を行う．

2.1　手続き型プログラムの可視化

手続き型プログラムに対する可視化では，C 言語プログラムのデバッガである insight[1]や TurboDebugger[2]に見られるように，対象となるプログラムコード内の任意の箇所にブレークポイントを定め，その箇所の変数の値を出力することによってプログラム動作を可視的に表現する．これにより，プログラム内で変数が正常にやりとりされているかどうかを確認することを可能にする．

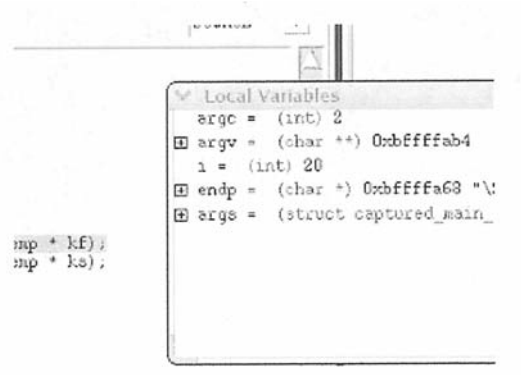


図 1. insight の実行画面

2.2　オブジェクト指向プログラムの可視化

オブジェクト指向プログラムに対する可視化では，手続き型プログラムに対するものに加えて，JAVA 言語プログラムの設計時に，描いた UML の図からプログラムコードの雛形を自動生成してくれるツール Together[3]や独自のライブラリを用いて，プログラムコード内の変数の値の変化をアニメーションとして表示する Jeliot[4]等，手続き型プログラムに対する可視化より様々な形態をとっている．

2.3　従来の可視化システムの改善案

従来の手続き型あるいはオブジェクト指向のプログラムに対する可視化システムとは，個々の変数の値の変化を確認するには良いが，オブジェクト指向では，より大きな単位であるオブジェクトの生成，変化，消滅等のオブジェクトの振舞いを知ることが重要になる．

そこで本研究では，従来の可視化システムの多くに見られる，変数の値の変化に重点を置くプログラム動作の可視化ではなく，オブジェクト単位での可視化表現を取り入れ，オブジェクト指向特有のオブジェクトのインスタンス化やガベージコレクションの処理の可視化を行った．

3　プログラム動作の可視化システム

3.1　システム構成

本可視化システムは，任意の JAVA 言語で記述されたプログラムの実行に対して動作し，可視化するために必要な情報を抽出する抽出部，抽出した情報を受け取り，描画を行う描画部からなる（図 2 を参照）．描画部が用いる図の表現方法として，UML のオブジェクト図に似た表現用いた．また，抽出部の設計手法として，JAVA 言語用のアスペクト指向設計を実現するライブラリである javassist[5]を適用した．

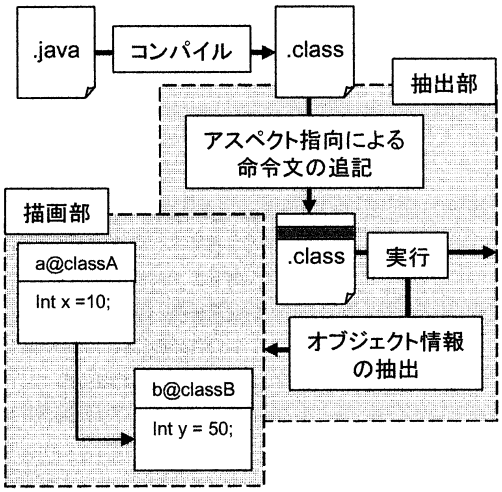


図 2. 可視化システムの構成図

3.2 情報の抽出

3.2.1 アスペクト指向の利用

本可視化システムでは, 前でも述べたように情報の抽出を行う箇所にアスペクト指向によるプログラム設計を用いた. 中でも, javassist を用いたバイトコード変換によるアスペクト指向を適用した. 図 3 は, アスペクト指向を用いない場合の命令文の追加手法だが, ここでは図 4 のように, アスペクト指向設計を用いることにより, 対象となるプログラムコードを直接変更することを回避した.

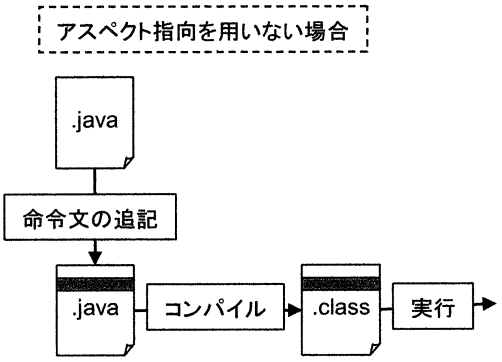


図 3. アスペクト指向を用いない抽出

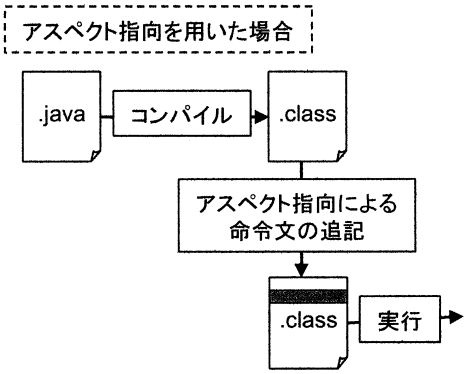


図 4. アスペクト指向を用いた抽出

3.2.2 抽出情報

ここでは, プログラムの動作を可視化する上で, 対象となるプログラムコードから抽出する必要がある情報を挙げる. これらは, オブジェクトを可視化することに重点を置いた上で, そのオブジェクトに関する, オブジェクトの生成やオブジェクト内のフィールド変数へのアクセス等, またスタティックメソッドの実行といった命令に対するものである.

① スタティックメインメソッドの実行

メインメソッドの実行時には, 他のメソッドとは異なり, メソッド内でのオブジェクトの生成に対して, オブジェクトの関連を表現するために, メソッド内の変数を抽出する.

② オブジェクトの生成

生成されたオブジェクトの ID とクラス名およびオブジェクト内のフィールド変数の値等, またオブジェクトを生成したオブジェクトの ID とクラス名を抽出する.

③ フィールド変数の書き込み

書き込みが行われるフィールド変数が属する

オブジェクトのIDとクラス名，また更新される情報を抽出する．

3.2.3 抽出方式

本可視化システムの抽出方式には，アスペクト指向による手法を用い，javassist ライブラリの利用によるバイトコード変換を適用した．

javassistとは，JAVA言語のバイトコード変換を容易に実現することを目的としたライブラリであり，アスペクト指向設計エンジンとして研究が行われている．ここでは，それを利用することによって，対象となるプログラムコードのコンパイル実行後に，生成されたバイトコードに対して，バイトコード変換を行うことで，他のアスペクト指向設計での，コンパイラによる命令追記制限等の問題点（プライベート変数の参照ができない等）を回避した．

これにより，オブジェクトの生成時やフィールドの書き込みアクセス時に，図の描画に必要な情報の抽出を行う．ただし，スタティックなメイン関数の実行にともなう情報の抽出は，他の関数とは区別し，特別に命令を設けることで必要な情報の抽出を行う．ここでは，javassist を利用した抽出命令の例を挙げる．

図5は，任意のJAVA言語プログラムのバイトコードに対して，javassist が提供する機能を用いて，新たに情報を抽出するためのプログラムコードを埋め込むプログラムコードの例である．

ここでは，まずJAVA言語プログラムが使用する情報の格納域であるクラスプールの情報を抜き出し（**ClassPool.getDefault()**），さらにそのクラスプールの中からメインの引数で与えた，情報の抽出を行う対象となるプログラムのクラス名からクラスに関する情報を抜き出す（**cpool.get(args[0])**）．そのクラスの振舞いに対して行うべき処理の規定（**class VPOEditor**）を追加する（**clazz.instrument(newVPOEditor)**）ことで，対象のバイトコード内に存在するメソッド実行やフィールド変数への書き込みアクセス，またオブジェクトの生成時といった場合に，情報の抽出を行う命令を個々に記述することが可能である．またここでは，それらの命令を終えた後，更新したクラス情報を上書きして保存するのを避けるために，元のディレクトリとは異なる場所保存を行っている（**clazz.writeFile("./out")**）．

```
01:import javassist.*;
02:public class VPO_Aspect {
03:   public static void main(String[] args) {
04:      ClassPool cpool =ClassPool.getDefault();
05:      CtClass clazz = cpool.get(args[0]);
06:      clazz.instrument(new VPOEditor());
07:      clazz.writeFile("./out");
08:      System.out.println("VPO finished.");
09:   }
10:   static class VPOEditor extendsExprEditor{
11:    public void edit(FieldAccess arg0) {
12:     (フィールドアクセス時の追加処理)
13:    }
14:    public void edit(NewExpr arg0) {
15:     (オブジェクト生成時の追加処理)
16:    }
17:    public void edit(MethodCall arg0) {
18:     (メソッド呼び出し時の追加処理)
19:    }
20:   }
21:}
```

図5. javassistによるコード例

図6は，オブジェクトのフィールド変数に対して，書き込みアクセスが行われる時の情報の抽出を行うプログラムコードの例である．（このコードはVPOEditorクラスに含まれる）

ここでは，オブジェクトのフィールドがアクセスされる時に，edit メソッドの引数（FieldAccess arg0）に実行されるフィールドアクセスについての情報が渡される．そのフィールドアクセスが書き込みアクセス（isWriter()）である場合に，元の命令文を表す特殊変数（$proceed（$$））とそのオブジェクト自身をあらわす特殊変数($0)，またそのフィールド名(getFieldName())とフィールドの値をあらわす特殊変数($1)，さらには編集箇所のクラス名(getClassName())，行番号(getLineNumber())を加えて，対象のプログラムのバイトコード内のものと置き換えている（replace()）．これにより，対象となるバイトコードの実行時には，元の処理に加えて，追記した情報群を出力する．なお，"$"から開始されるコードは javassist が提供するものである．

```
01:public void edit(FieldAccess arg0) throws
CannotCompileException {
02:   String className = arg0.getClassName();
03:   String fieldName = arg0.getFieldName();
04:                String location =
arg0.getFileName()+":"+arg0.getLineNumber();
05:   if(arg0.isWriter()){
06:      arg0.replace(
07:         "{System.out.println(
08:
¥"[FieldAccess]_¥"+$0+"+"¥""+"_"+className+"_"
+fieldname
09:
+":"+"¥""+"+$1+"+"¥""+""+location+"¥""+");$proce
ed($$);}"
10:      );
11:   }
12:}
```

図 6. フィールドアクセスに対するコード例

3.3 描画

3.3.1 描画要素

本可視化システムが描画を行う要素は，オブジェクトの ID やクラス名，またオブジェクトが持つフィールド変数名とその値など，またオブジェクト各オブジェクト間の関連性である（表 1 を参照）．これらの要素は，描画部が対象のオブジェクトの動作より随時情報を抽出し，それに連動して動的に描画を行う．

表 1. 描画要素一覧

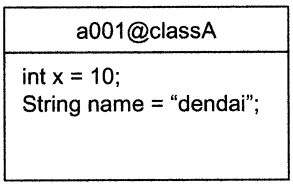
動作	必要となる情報	描画要素
オブジェクトの生成	オブジェクト ID，属するクラス名 オブジェクトが持つフィールド変数名，値 親のオブジェクトとの関連性	オブジェクト ID，属するクラス名 オブジェクトが持つフィールド変数名，値 親のオブジェクトとの関連性
フィールド変数への書き込み	オブジェクト ID，属するクラス名 変更されたフィールド変数名，値	変更されたフィールド変数名，値
オブジェクトの関連性の損失	オブジェクト ID，属するクラス名	

3.3.2 フィールドの表示方式

ここでは，本可視化システムが，対象となるプログラムの動作にともなったオブジェクトの情報を抽出し，図を描画する時のオブジェクトが持つフィールド変数の表現方法について述べる．ここでは，オブジェクト内に含まれる，変数，変数の配列，クラス，クラスの配列の情報に分類して，実際に描画を行う図の表現方式を述べる．

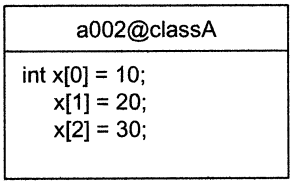
① 基本型の変数

ここでは，int 型や String 型の変数を基本型と定め，これらの変数の名前と値は図のように表現する．



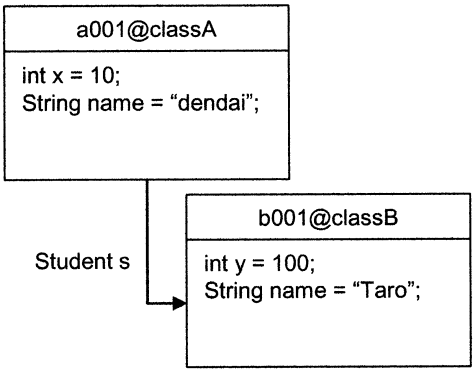
② 基本型の変数の配列

基本型の変数の配列やリストは，基本型の変数単体と同様に，図のように表現する．



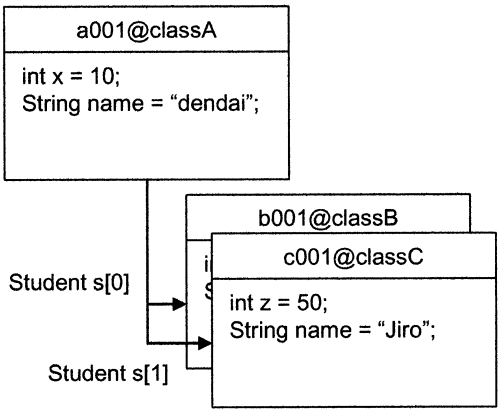
③ クラス型の変数

オブジェクトが持つクラス型のフィールド変数は，各オブジェクト間を矢印を用いて表現する．また，矢印のラベルにはオブジェクトが格納されるフィールド変数の名前を記述する．



④ クラス型の変数の配列

オブジェクトが持つクラス型のフィールド変数の配列やリストは，単数の場合に加えて配列数のオブジェクトとそれに対するラベルを記述し表現する．



3.4 実行操作

3.4.1 実行方式の選択

本システムは，対象となる JAVA 言語プログラムがグラフィカルユーザインターフェースを用いて開発されている場合を考慮し，可視化の処理を対象となるプログラムの実行中に動的に動作可能なように，一度ログを吐き出して可視化を行うバッヂ式ではなく，リアルタイムに可視化を行う方式をとっている．

3.4.2 レイアウト

本システムは，機能の一つとしてレイアウト機能を実装する．これは，対象となるプログラム動作に関して複数のオブジェクトが描画されていくにつれて，利用者が任意のタイミングで自由に各オブジェクト情報のレイアウトを調整し確認しやすくする機能である．また，現在のシステムでは初期状態として生成されたオブジェクトが完全に重ならないように，座標をずらし描画を行っているだけだが，今後は利用者にとって見やす

いレイアウトデザインを検討して実装を行う予定である.

3.4.3 フィルタリング

本システムは, 機能の一つとしてフィルタリング機能を実装する. これは, 対象となるプログラムを大きいものに指定した場合に, 生成されるオブジェクト数が膨大になり, 図が非常に見難くなってしまう問題を防ぐものである. システム利用者が所属するクラス名等の条件を設けて実行することで, 条件に沿ったオブジェクト情報のみを図示する機能である.

3.4.4 実行

図 7 と図 8 は, 本可視化ツールの実行画面である. ここでは, プログラミング教育の教材[6]にでてくる Composite パターンの説明に使われる簡単なプログラムコードに対して実行を行った場合のものである. 図 7 は, 対象のプログラムコードの実行途中, 図 8 は, 実行終了時の本可視化システムの画面である.

本システムの実行画面には, 既存の対象となるプログラムの動作に対して抽出を行ったオブジェクトの情報を, プログラムの実行に関連して動的に図に表現している. 各図形内には, オブジェクトの ID とそのオブジェクトが属するクラス名, またそのオブジェクトが持つフィールド変数の名前とその値が描かれる. また各オブジェクト間には, それぞれのオブジェクトがいずれのオブジェクトと関係を持っているのかがわかるよう, 矢印を用いて表現を行っている.

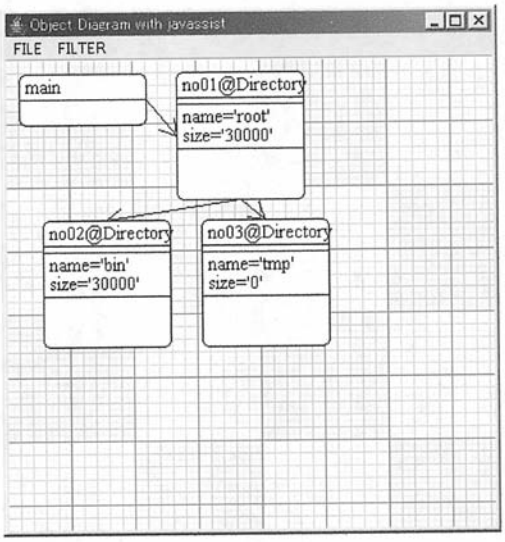


図 7. 本システムの実行画面 1

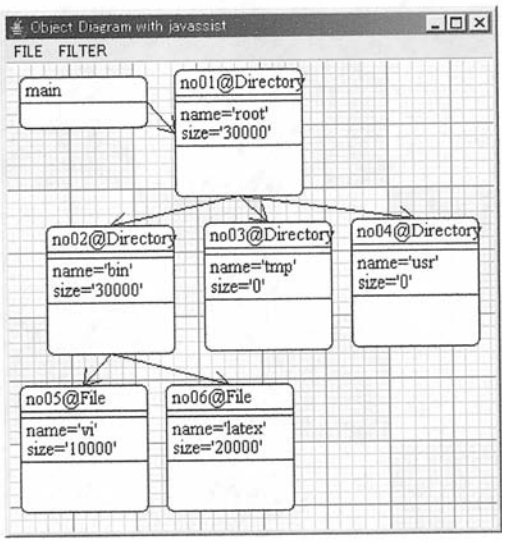


図 8. 本システムの実行画面 2

4 評価および問題点

本可視化システムの実行例で見られるように, 各オブジェクトの変化を可視化することで, オブジェクト指向プログラムの動作を捉えることが容易になることがわかった. 本来の目的として, 初心者向けのプログラミング教育での支援ツー

ルとして開発を行った本可視化システムは，初心者が理解しづらい UML のクラス図等の学習に効果があることはわかった．しかしながら，今度適用していく大きなプログラムのデバッグ等には，任意の箇所でのブレークポイントやステップ実行による機能や，拡張クラスまでのトレース機能等，より機能的な可視化システムの開発が必要になると考えられる．

5 おわりに

本研究では，従来の手続き型プログラムとオブジェクト指向プログラムに対する可視化の調査を行い，既存の可視化システムとは異なる可視的表現を適用したオブジェクト指向プログラムの可視化システムの開発を行った．

従来の可視化システムとは異なり，オブジェクトの状態の変化を可視化することに重点をおいた図の描画方法を取り入れた．また，情報の抽出を行う箇所の実装には，アスペクト指向設計を用い，可視化を行う対象のプログラムコードを直接変更せずに，可視化を行うために必要な情報を抽出するプログラムを実現した．本可視化システムをいくつかのプログラムに対して実行を行い，プログラム全体のオブジェクトの状態の変化からプログラム動作が捉えやすくなるという成果を得た．

今後は，本可視化システムを様々な構造のプログラムに対して実行し検証を行うことで，システムの改良を進めていきたい．また，プログラミングに携わる多くの実験者方に，実際に本可視化システムを利用してもらい，本可視化システムの評価を行うことも検討している．

参考文献

[1] insight
(http://sources.redhat.com/insight/)

[2] TurboDebugger
(http://www.borland.com/jp/products/cbuilder/turbodebugger.html)

[3] Together
(http://www.borland.com/jp/)

[4] Jeliot
Roman Bednarik, Mike Joy, A.Moreno, Niko Myller, Shanghua Sun and Erkki Sutinen. To appear in Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC'2005), 28 - 30 November 2005, Vienna, Austria.
(http://www.cs.joensuu.fi/jeliot/)

[5] javassist
Shigeru Chiba and Muga Nishizawa
Proc. of 2nd Int'l Conf. on Generative Programming and Component Engineering (GPCE '03),
LNCS 2830, pp.364-376, Springer-Verlag, 2003.
(The performance of Javassist)
(http://www.csg.is.titech.ac.jp/~chiba/javassist/)

[6] JAVA 言語で学ぶデザインパターン入門，第 5 版，結城浩著，ソフトバンクパブリッシング株式会社，2005.8.