社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－SE－148（3）
2005／5／31

IIAnalyzer: オブジェクト間の相互作用を分析するための リバースエンジニアリングツール

神　谷　年　洋[†,☆]

ツール IIAnalyzer は，Java プログラムの実行履歴から，実行中に生成されるオブジェクト間の相互作用を特定し図示することで，プログラムの動的な構造，性質の理解を補助する．本稿では，IIAnalyzer の機能の一つ，プログラムの実行中の振る舞いの類似性を検出する機能ついて説明する．

A reverse engineering tool IIAnalyzer(instance-interaction analyzer)

Toshihiro Kamiya[†,☆]

The tool IIAnalyzer performs analyzing and visualizing of interaction between the object that are instanciated in Java program, in order to support understanding of dynamic structure and characteristic of the program. In this paper, I will present the function to detect similarities of behavior in a program execution.

1. はじめに

オブジェクト指向プログラムが実行されるときには，多くのオブジェクトが生成され，それらが相互作用することで，プログラムの機能が実現されている．したがって，プログラムの開発や保守といった作業を行うためには，作業者は，プログラムの実行時に生成されるオブジェクトやそれらの相互作用を理解することが必要である．オブジェクト指向モデリング・設計のための言語である UML[7] では，オブジェクトやそれらの関係を記述することができるコラボレーション図やシーケンス図を提供している．リバースエンジニアリングツールとして，プログラムの実行履歴からシーケンス図を生成ものもある．

オブジェクト指向プログラムを分析する上での問題点のひとつに，オブジェクト指向プログラミング言語では動的なディスパッチが多用され，静的な解析手法では必要とされる分析が行えないことがある．たとえば，特定の変数の値に影響したすべての文を求める手法であるバックワードスライスを求める場合，静的な手法[3] を用いると，動的な手法[1] を用いた場合よりも多くの文がその変数の値に影響している，という結果が得られる．この理由は，静的な手法では，動的なディスパッチの部分では，呼び出されうるすべてのメソッドが考慮されるのに対して，動的な手法では，実際に呼び出されたメソッドのみが考慮されるから，である．

ツール IIAnalyzer は，オブジェクト指向プログラミング言語のひとつである Java で記述されたプログラムを対象としたリバースエンジニアリングツールである．プログラムの実行履歴から，実行中に生成されるオブジェクト間の相互作用を特定し，図示することで，プログラムの動的な構造，性質を理解することを目的とする．現在までに実装されている機能には，
(1) プログラムの実行中の振る舞いの類似性の検出，
(2) プログラムの機能の位置の特定，がある．本稿では，特に前者の機能について説明する．

この「振る舞いの類似性検出」機能によって，そのプログラムのソースコードに含まれるコード片の機能的な類似を調べることができる．また，振る舞いの類似性は，その実行に用いたテストケースを，内在する類似性や，他のテストケースとの類似性といった観点から評価することに使えるかもしれない．

2. 振る舞いの類似性

本研究では，基本的に，プログラムの振る舞いを，そのプログラムの実行履歴から判定する．たとえば，

† 科学技術振興機構 さきがけ
Presto, Japan Science and Technology Agency
☆ 現在，産業技術総合研究所
Presently with National Institute of Advanced Industrial Science and Technology

ある実行履歴の２つの部分列が，同じ順序で，同じオブジェクトの同じメソッドを呼び出している場合，それらの部分列はまったく同じである（以降,「厳密な一致」）とみなすことができる．あるプログラムの実行中では，ループや再帰呼び出しによって，類似した振る舞いが繰り返される．また互いに類似したコード片を実行することによっても，類似した振る舞いが起きる．その一方で，同じコード片を複数回実行する場合であっても，含まれている分岐で異なった部分をたどることにより，振る舞いとしては異なることもある．

ただし，このような「厳密な一致」は，抽象度が高いレベルの分析を行いたい場合，たとえば，振る舞いの類似を機能の類似を判断するために用いようとする場合には厳しすぎる条件である．厳密な一致に従えば，同じループを異なる回数実行する部分列は異なっていると判定されるが，そのような差異は無視したいこともある．また，メソッド m からメソッド n を経由してメソッド p を呼び出す部分列と，メソッド m' から直接メソッド p を呼び出す部分列があったとき，前者の部分列のメソッド n が単に委譲をするだけのもの（ほとんど何もせず，単に p を呼び出すだけのもの）であったとすれば，これらの部分列は同じであるとみなしたい．

他の応用，たとえば，実行履歴からシーケンス図を作成する場合でも，ループや再帰による繰り返しをまとめてコンパクトなシーケンス図を作成する研究においても，厳密な一致よりも緩い繰り返し判定基準を用いることで，シーケンス図の圧縮率の大幅な向上が得られている[2)]．

3. 類似した振る舞いの検出・可視化手法

3.1 振る舞いの類似性判定手法

厳密な一致よりも緩い類似性の判定条件として，本稿では，実行履歴の部分列が直接あるいは間接的に実行する単位操作（後述）の種類が一致していること（以降,「呼び出し単位操作の一致」）を用いることを提案する．単位操作とは，プログラミング言語によって基本的な操作と見なされるものであり，具体的には，ライブラリの基本的なメソッド，および，ユーザー定義のクラスのメソッドのうち他のメソッドを呼び出していないものとする．たとえば，Java の場合，String クラスをはじめとする JDK(Java Development Kit) で提供されるクラスで実装されているメソッドは単位操作となる．

さらに，本論文では，類似性判定の対象となる実行履歴の部分列としては，あるメソッドの開始からその終了に対応する部分列とする．シーケンス図（**図 1**）では，メソッド呼び出し（メッセージ送信）の開始から，その生存線（縦の線）をたどって，制御が返ってくるまでの時点となる．この制約は，上述の類似性判定には不要だが，後述する図示のために導入された．この制約により，類似性の判定の対象も，類似性判定のために用いられる単位操作も，どちらもメソッドになる．扱う対象がメソッドだけになることで，結果の図示が容易になり，また，図示することで，振る舞いの類似や差異の直感的な理解が可能になる．

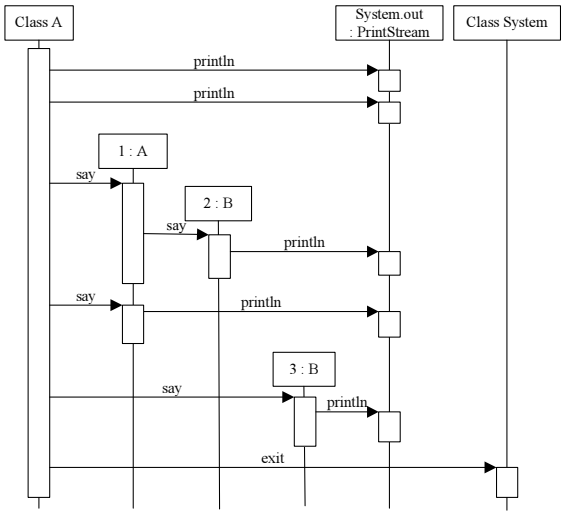

図 1 シーケンス図の例
Fig. 1 The corresponding sequence diagram

3.2 類似した振る舞いの図示手法

類似性判定の具体例を視覚的に表現することで分析をサポートするための多層集約インスタンスコールグラフ (multi-layered summarized instance call graph; 以降，MSI-CG) を提案する．

MSI-CG の説明を行うため，まず，補助的にインスタンスコールグラフ (以降，I-CG) を導入する．I-CG とは，あるプログラムの実行履歴の中での個々のメソッド呼び出しを頂点とし，呼び出し元から呼び出し先へ，有向辺を引いたものである．頂点は，根（最初に呼び出されるメソッド）に対応する頂点は矩形，それ以外は楕円で表し，インスタンスの ID とクラス名，メソッド名をラベルとして付加する．さらに，単位操作のメソッドの呼び出しである頂点は灰色で表す．**図 2** に，図 1 と同じシーケンスを表す I-CG の例を示す．

「厳密な一致」は I-CG を使って図示することができる．I-CG の 2 つの頂点について，その頂点から有向辺をたどることで到達可能な部分グラフ（以降,「そ

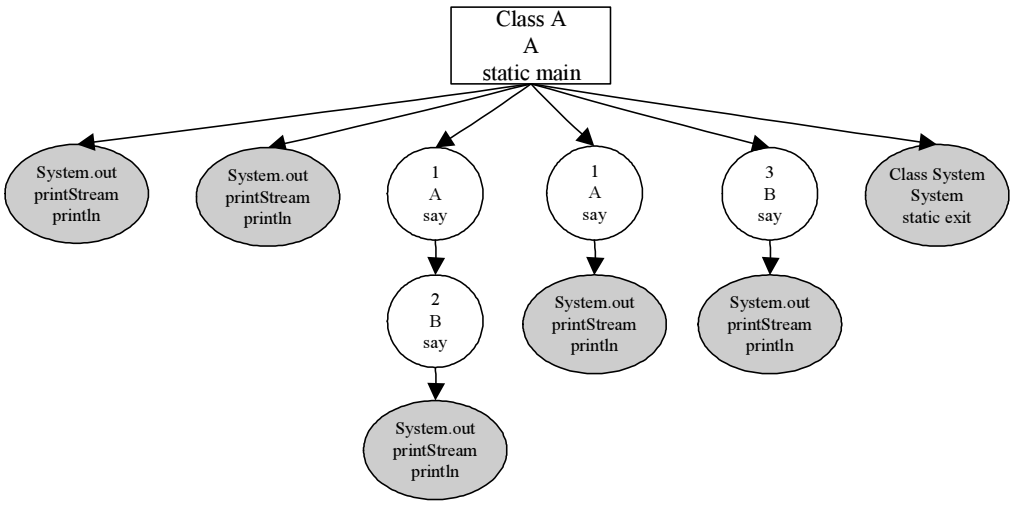

図 2 図 1 のシーケンスに対応するインスタンスコールグラフ (I-CG)
Fig. 2 An instance call graph(I-CG) correspoinding to the sequence of Fig. 1

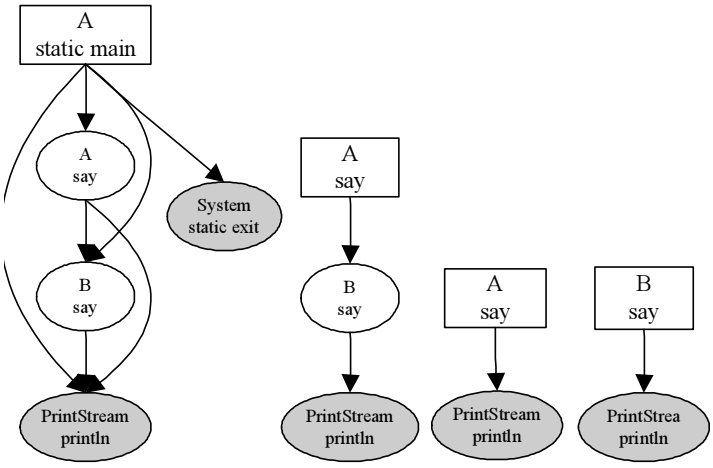

図 3 それぞれの頂点を根とした集約インスタンスコールグラフ (SI-CG)
Fig. 3 The sammarized instance call graphs from nodes of Fig. 2

の頂点を根とする部分グラフ」）が同一であることを意味する．

次に，集約インスタンスコールグラフ (以降，SI-CG) を導入する．SI-CG とは，I-CG の頂点のうち，クラス名とメソッド名が同じ頂点を一つの頂点で置き換えたもの（集約したもの）である．「呼び出し単位操作の一致」は，I-CG の 2 つの頂点を根とする部分グラフを取り出したあと，それらをそれぞれ別個に集約した SI-CG について，それらに含まれている単位操作に対応する頂点の集合が同じであることを意味する．注意すべき点として，順番を逆にして，集約したあと部分グラフを取り出すと，結果が変わり，同じメソッドの異なる呼び出しの差異を図示することができなくなる．

図 3 に，図 2 の I-CG の各頂点を根とする部分グラフから生成した SI-CG を示す．一つの頂点が持つオブジェクト ID は複数となり煩雑になるため，図には描かなかった．また，基本操作の頂点を根とするもの

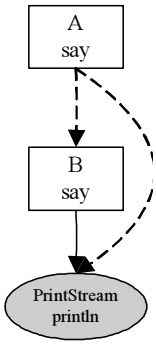

図 4 3つの SI-CG を合成してできる MSI-CG
Fig. 4 The MSI-CG generated from the three SI-CGs

は自明のため省略した．クラス A のメソッド say を根とする SI-CG が 2 つあるのは，このメソッドの呼び出しが 2 回行われていることを意味する．前の注意に背いて，集約を行ってから部分グラフの取り出しを行うと，これらは 1 つにまとまってしまう．

MSI-CG とは，上述の SI-CG を何枚か「重ね合わせ」て一つの図として表現したものである．重ね合わせの規則は，頂点については，元の SI-CG の間で対応する頂点（クラス名とメソッド名が同じ頂点）のうち，一つでも根に相当するものがあれば，矩形の頂点とする．さもなければ，楕円の頂点とする．辺については，2 つの頂点 p, q について，元の SI-CG のうちで p と q を両方含むものが，全部 p から q への辺を含むなら，その辺を実線で引き，全部ではなく一部がその辺を含んでいるなら，点線で，さもなければ，その辺を引かない．

元になる SI-CG は任意に選ぶことができるが，特に，「呼び出し単位操作の一致」した頂点を根とする SI-CG を重ね合わせて作った MSI-CG は，類似した振る舞いを比較するために用いることができる．具体的には，まず，実行履歴の部分列のそれぞれは，矩形の頂点を根とする部分グラフによって読み取ることができる．次に，それら部分列が共有する単位操作は，灰色の頂点によって示されている．さらに，点線で示されている辺は，一部の部分列では起こったが，一部では起きなかったメソッド呼び出しを意味している．

図 4 に示す MSI-CG は，図 3 に示された 4 つの SI-CG のうち，「呼び出し単位操作の一致」の意味で振る舞いが類似している右の 3 つを重ね合わせて作成したものである．この図から，クラス A のメソッド say（以降，「A#say」と表記）と，B#say は，ともに PrintStream#println を，最終的には呼び出していることが読み取れる．また，A#say は，B#say を経由して PrintSream#println を呼び出す場合と，直接呼び出す場合の 2 通りあることも読み取れる．

4. 実　　験

IIAnalyzer は，実行履歴を取得する部分と，実行履歴を解析する部分，視覚化する部分からなっている．履歴取得部分は約 1600 行の C++コードで実装され，JVMTI[6] を利用した JavaVM のエージェント（プラグイン）である．解析部分は約 2 千行の C++コードで実装され，類似した振る舞いの検出を行う．可視化部分は 750 行の C++コードと 141 行の AWK スクリプトで実装されて，ビットマップの生成のために GraphViz[5] を利用している．検出された類似の振る舞いについて，4 節で説明した MSI-CG を生成する．

IIAnalyzer の類似した振る舞いの検出および可視化機能を評価するため，オープンソースプロダクトである findbugs-0.86[4] に適用した．結果，約 700 万回のメソッド呼び出しを含む実行履歴から，約 200 個の類似した振る舞いが検出された．

図 5 は，ツールが出力した MSI-CG のうちの一つについて，その一部を拡大したものである．図右の 2 つのメソッド TypeAnalysis# meetInto(TypeFrame fact, Edge edge, TypeFrame result)void と ValueNumberAnalysis#meetInto(ValueNumberFrame fact, Edge edge, ValueNumberFrame result)void（パッケージはともに edu.umd.cs.findbugs.ba）は，図から，それらが直接呼び出している基本操作の集合が同じであり，間接的に呼び出している基本操作の集合も同じであることがわかる．ソースコードを参照すると，一見すると全く違うコードのように見えるが，それぞれのメソッドのエラー処理をしなかった場合のシーケンスを追っていくと，確かにほぼ同じ処理を行っていることが確認できた．

5. ま と め

本稿では，オブジェクト指向プログラミング言語のひとつである Java で記述されたプログラムを対象とするリバースエンジニアリングツールである IIAnalyzer の機能のうち，プログラムの実行中の振る舞いの類似性の検出機能について，検出のアルゴリズム，および，検出結果を図示する方法について述べた．さらに，オープンソースプロダクトへの適用実験により，IIAnalyzer が実際に類似した振る舞いを検出し分析できることを確認した．

参 考 文 献

1) H. Agrawal and J. Horgan, “Dynamic Program Slicing”, ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 246-256 (1990).
2) 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎, “Java 実行履歴からのシーケンス図生成ツール”, 情報処理学会 組込みソフトウェアシンポジウム 2004 論文集, pp108-111 (2004).
3) M. Weiser, “Program Slicing,” IEEE Transactions on Software Engineering, SE-10(4), pp. 352-357 (1982).
4) Findbugs, http://findbugs.sourceforge.net/
5) GraphViz, http://www.graphviz.org/
6) JVM Tool Interface, http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/
7) UML Resource Page, http://www.uml.org/

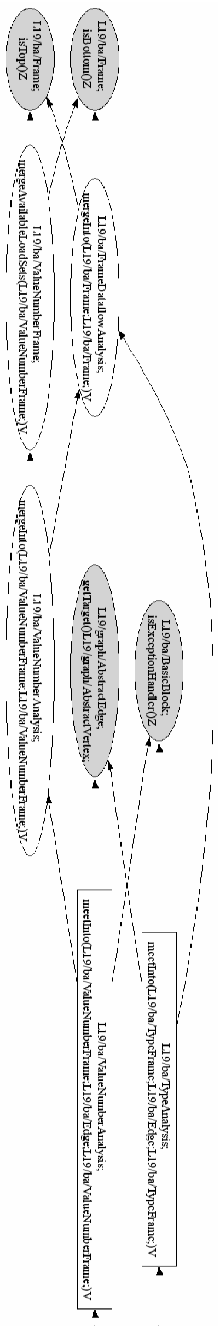

図 5 findbugs-0.86 の実行履歴から検出された類似した振る舞いの一つの MSI-CG の一部
Fig. 5 The part of the MSI-CG generated from an execution trace of findbugs-0.86