ソフトウェア工学 135－9
（2001．11．22）

Javaプログラムにおけるコールグラフの構築方法

西見 浩一　　趙 建軍
福岡工業大学情報工学部情報工学科
〒811-0295 福岡市東区和白東3-30-1
Email: zhao@cs.fit.ac.jp

概 要

ソフトウェア開発の設計の難しさは、ソフトウェアの設計者にとって常に問題になっており、プログラミング技術の進歩とともにソフトウェア開発も複雑で難しいものになっている。そのため、ソフトウェアのプログラム理解、デバッグ、テストなどの為の解析ツールが必要とされている。また、近年、オブジェクト指向の考え方を取り入れている・特定のプラットフォームに依存しない言語仕様など様々な利点を備えたJava言語が普及するにつれ、Javaソフトウェアに対する理解・テスト・デバッグなどの為の解析支援技術とツールの開発は重要である。本論文では、Javaバイトコードに基づいて解析支援ツールの一つとしてコールグラフの構築方法を提案する。

Efficient Call Graph Construction for Java Programs

Kouichi Nishimi　　Jianjun Zhao
Department of Computer Science and Engineering
Fukuoka Institute of Technology
3-30-1 Wajiro-higashi, Higashi-ku, Fukuoka 811-0295, Japan
Email: zhao@cs.fit.ac.jp

Abstract

In this paper, we present an efficient approach to constructing the call graph of Java programs. Comparing to the traditional call graph construction algorithms that derived the call graph from the source code level, our approach is to derive the Java call graph from the bytecode level of Java classfile. The constructed call graph can be useful in program understanding of Java Software.

1．はじめに

ソフトウェア開発の設計の難しさは、ソフトウェアの設計者にとって常に問題になっており、プログラミング技術の進歩とともにソフトウェア開発も複雑で難しいものになっている。そのため、ソフトウェアのプログラム理解、デバッグ、テストなどの為の解析ツールが必要とされている。また、近年、オブジェクト指向の考え方を取り入れている・特定のプラットフォームに依存しない言語仕様など様々な利点を備えたJava言語が普及するにつれ、Javaソフトウェアに対するプログラム理解・テスト・デバッグなどの為のJavaバイトコード解析支援の技術とツールの開発は重要である。ここで、ソフトウェア解析の一つのキーとなるものにコールグラフが上げられる。コールグラフとは、アプリケーションプログラム中の メソッドの呼び出し関係をグラフとしてアプリケーションプログラム全体を表現したものである（ここで言われるメソッドとはC言語では関数、prolog言語では手続きと呼ばれているものと同じである)。このコールグラフを利用することで、複雑な内容で理解することが困難なプログラミングに対しても容易にそのプログラム中に使用されているメソッドの呼び出し関係を把握することが可能となる。このことはモジュール間のつな

```
class Test1 {
        static int getPlus(int x,  int y){
                return x+y;
        }
        static void getInt() {
                System.out.println("10 + 20 = " +
                getPlus(10, 20));
        }
        public static void main(String[] args){
                Test1 tes = new Test1();
                tes.getInt();
        }
}
```

● ソースコード

バイトコード変換

```
new [Test1]
dup
invokenonvirtual [Test1.<init>]
astore_1
invokestatic [Test1.getInt]
return
```

● main メソッド部分のバイトコード

```
getstatic [java/lang/System.out]
new [java/lang/StringBuffer]
dup
ldc1 ["10 + 20 = "]
invokenonvirtual [java/lang/StringBuffer.<init>]
bipush 10
bipush 20
invokestatic [Test1.getPlus]
invokevirtual [java/lang/StringBuffer.append]
invokevirtual [java/lang/StringBuffer.toString]
invokevirtual [java/io/PrintStream.println]
return
```

● getInt 部分のバイトコード

図1. Java バイトコードへの変換例

がりを理解することにつながり、オブジェクト指向のプログラミングを構築する上で参考にすることができる等のように有効に利用できるといえる。Java はプラットフォーム独立性の機能を生かし機種に依存することなく Java ソフトウェアを供給できる等の利点が挙げられる。インターネットの WEB 上でも Java を利用したコンテンツを見る機会が多くなり、フリーウェア・シェアウェアとして配布されているソフトウェアも少なくなく現在幅広く普及しつつある。これらの Java で作成された物はソースプログラムではなく Java 仮想マシン (Java Virtual Machine : JVM) 上で実行できる形式にコンパイルされた class ファイル形式のものであることが多い。我々はこの class ファイルを解析することによってコールグラフを構築する方法を本論文で提案している。ソースプログラムを解析することによりコールグラフを構築する方法をとった研究もある。しかし、class ファイルにはバイトコード形式で情報が記述されておりソースプログラムを解析するよりも効率よく情報を取り出すことができるという利点がある。また、実際にはソースプログラムよりもコンパイル済みの class ファイルの方が配布される可能性が高いため、class ファイルを解析対象として選ぶことで利用価値も高くなると考えられる。

本論文では class ファイル内の Java バイトコードを静的に解析することで得られる情報を元にコールグラフ解析を進める手法を説明する。本論文の構成を述べる。まず第2節では Java 仮想マシンと Java バイトコードについて説明する。第3節でコールグラフ構築の説明に入り本論文で提案するコールグラフ構築方法を述べる。第4節でコールグラフ可視化ツールの概要

を述べる。第5節で提案したコールグラフ構築方法を用いて作成したプロトタイプの実行を交えて説明する。そして、第5章でまとめに入る。

2．Java 仮想マシン(Java virtual machine)

Java 仮想マシン (JVM) は Java プラットフォームの土台となる抽象的な計算機である[1]。この Java 仮想マシンには特定のバイナリ・フォーマット、すなわち class ファイル・フォーマットに関する情報のみが保持されており、プログラミング言語 Java に関する情報は保持されていない。class ファイルには、Java 仮想マシンの命令(すなわちバイトコード)、シンボル・テーブル、他の補助的な情報が収められている。Java では、ソースコードを中間コードである class ファイルに一旦変換し、この class ファイルのバイトコード命令を Java 仮想マシンのインタープリタで解釈しながら実行する。class ファイル・フォーマットは明確に定義されており、プログラムを解析しやすいものとなっている。ソースプログラムと比べて Java バイトコードを解析することの利点は Java バイトコードからの方がデータを解析しやすい点にある。例を挙げると、ソースプログラムにおいて繰り返し処理に for 文や while 文が挙げられる。これらはほぼ同じような命令だが、ソースプログラムから解析する場合には記述が明確に違う為それぞれの場合に対して対処する必要がある。それに対し Java バイトコードでは、この `for` 文と `while` 文の動作は `goto` 文と if 文であらわされており動作に沿った処理を行う為、別個のものとみなす必要がない。これは処理の軽減につながり、プログラムを解析する上で効率がよい。ソースプログラムから Java バイトコード命令への変換例を図1に示す。この例では例題のソースプログラムの中の `main` メソッドと `getInt` メソッドの処理部分のバイトコード変換を示している。

3．コールグラフ構築方法

以下にコールグラフの構築の流れを簡単に示す。

① Java バイトコードから `main` メソッドから始まる制御流れ情報を生成する。

② 各メソッド呼び出しのキーとなる `invokeinterface`、`invokevirtul`、`invokestatic`、`invokespecial` の Java 仮想マシン命令に着目し各種関連情報を取り出す。

③ 上記の命令で呼び出されたメソッドに対して①・②と同じ解析を処理が終了するまで行う。

④ 取り出されたメソッド情報に基づいてコールグラフを生成する。

これより各工程の操作について詳しく述べる。

① classファイルのJavaバイトコード情報を読み取り Java バイトコードの制御流れを読み取る。 Java バイトコードから情報を読み取る方法は我々の以前の研究で開発されているツール[2,3,4]を利用している。図に例を示す。この段階では class ファイルの `main` メソッドを対象に解析を進めていかなければならない。Javaで記述されたプログラムは一般的に `main` メソッドから実行されるという特徴がある。例外もあるが、この様な理由によりプログラム全体のメソッド呼び出し関係を正確に構築する為には `main` メソッドから解析する必要がある。

② 制御流れからメソッド呼出し命令の情報を取り出す。Java 仮想マシンのバイトコード命令は約200命令ほど用意されている。このバイトコード命令の中でメソッドの呼び出しに対応した命令は `invoke` の接頭辞をもつ `invokeinterface`、`invokespecial`、`invokestatic`、`invokevirtual` の4つである。この4つの命令の内容を以下に示す。

- `invokeinterface`
  インタフェース・メソッドを起動する
- `invokespecial`
  スーパークラス、private、インスタンス初期化メソッド起動といった特殊な取り扱いが必要なインスタンス・メソッドを起動する
- `invokestatic`
  クラス(`static`)メソッドを起動する
- `invokevirtual`
  クラスに基づくディスパッチを行い、

インスタンス・メソッドを起動する

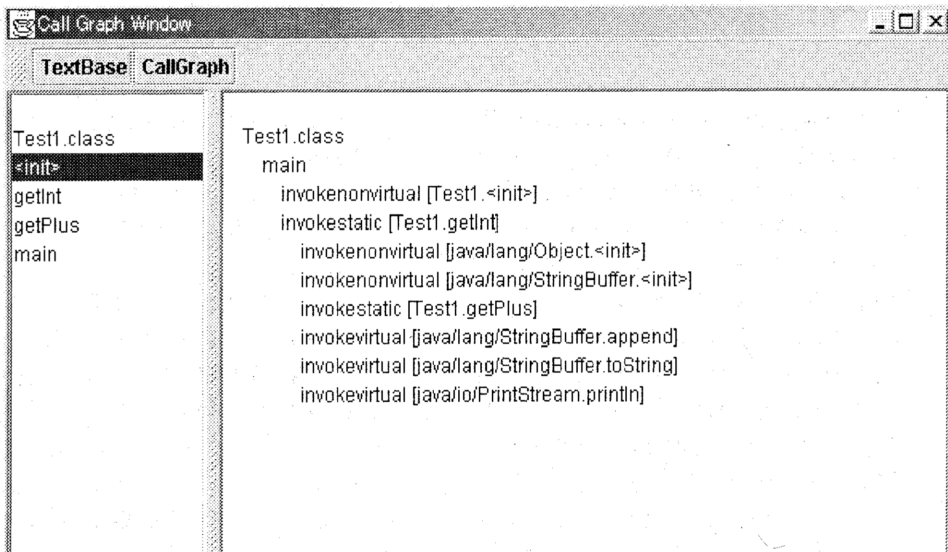

図２．　コールグラフ可視化ツールのテキストベース表示

Javaにおいてメソッドが呼び出された場合、そのメソッドの種類に応じて上記のバイトコード命令が実行される[1]。これらのバイトコード命令に着目し、命令が実行される際に参照される呼び出し元・呼び出し先の情報を取り出し・保存しコールグラフを構築する際の材料とする。同一クラス内にないメソッドであってもここで抽出される情報を元に他のクラスの中からメソッド情報の解析を行うことができる。

③　①・②の作業を実行・終了した後、①でmainメソッドに対して行った処理と同じ処理を②の作業で取り出した呼び出し先のメソッドに対して行う。このとき、呼び出されたメソッドに関して②の作業で保存した情報と比較し、一度操作したメソッドは重複して呼び出さないようにする。これにより再起呼び出し・コールバック等の場合に解析処理が無限ループに陥ることを防ぐことができる。また、JavaプラットフォームのAPIに含まれるメソッドに関してはすでに規定のものとなっているため自明のものとして新たに解析する必要はないとし、解析処理を行わず次のメソッドの解析処理に移る。この操作を続けていくうちに発見されるメソッドが次第に収束していき新たなメソッドが発見されなくなった時点でソフトウェアの実行上で呼び出されうるメソッドの呼び出し関係が全て取り出すことができたとみなし、この操作を終了する。

④－最後にバイトコード解析により得られた情報を元にメソッドの呼び出し関係を構築する。構築された呼び出し関係情報を元に理解を容易にできるようにグラフ形式で視覚的に表示する。

我々はこのコールグラフ可視化ツールのプロトタイプを作成した。次章でこのプロトタイプについて利用方法を交えながら説明を行う。

４．コールグラフ可視化ツールの概要

これより我々が試作したコールグラフ可視化ツールの概要について述べる。前述した解析方法に従いコールグラフ情報を生成し、この情報を元にコールグラフを構築し、視覚化できるよう表示させるためのツールである。図２にプロトタイプのメイン画面を示す。画面左上の[TextBase] [CallGraph]ボタンがあり、画面中央の左側のウィンドウにクラス内のメソッドを表示し、画面中央の右側にメソッドごとに利用されているメソッドの情報を表示させている。ここで画面右側のメソッド表示について説明する。

<u>invokestatic</u> [<u>Test1</u> . <u>getInt</u>]

invokestatic ：　呼び出されるメソッドの種類ごとのバイトコード命令。２節②で説明した４種類がある。

Test1　：　対象となるメソッドを含んでいるクラスの名前。java/～の物はJavaプラットフォームのAPIの物である。

GetInt　：　対象となるメソッド名。

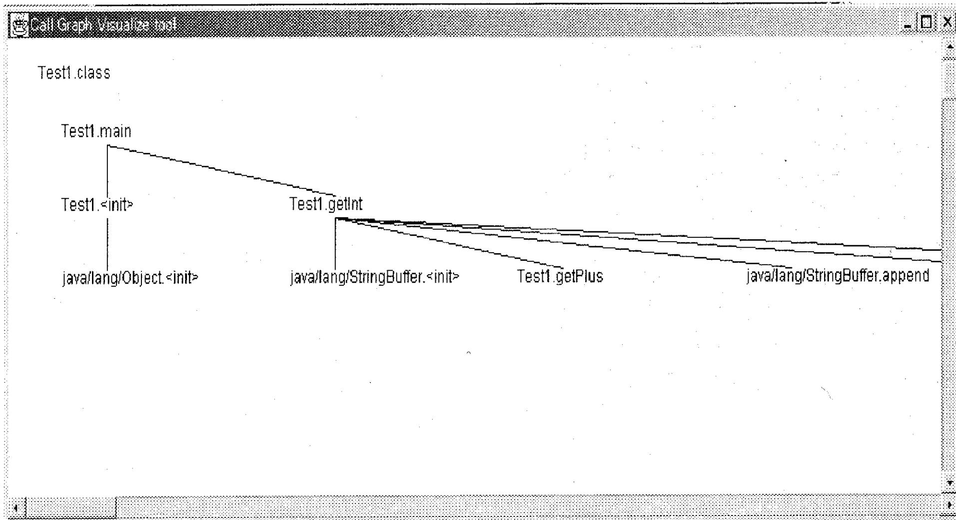

図3． コールグラフ可視化表示例

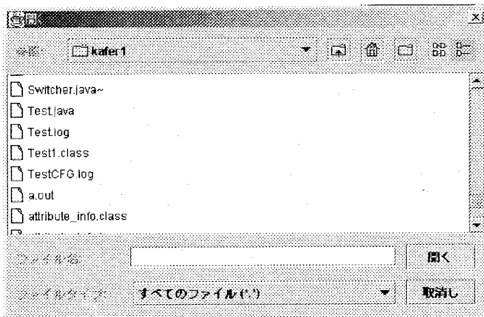

図4． ダイアログボックス

図1で示したプログラム例を処理した結果を示しながらこのツールの操作手順を以下に示す。

① 開始ボタン[TextBase]を押すとダイアログボックスよりクラス一覧（図 4）が表示されるので、対象となるクラスファイル(main メソッドを含むもの)を選択する。

② 選択されたクラスファイル内のメソッド一覧が画面左側のウィンドウに表示される。このメソッドをクリックすることでクリックされたメソッドに対しての解析を行う。ここでは main メソッドを選択する。main メソッド以外を選択することも可能である。main メソッド以外で適切だと思われるメソッドがあればそのメソッドを選択することもできる。

③ 選択されたメソッド(main)に対して解析を行いコールグラフ構築情報を生成する。ここでそれぞれ呼び出されるメソッドの情報を画面右側のウィンドウに表示する。

④ [CallGraph]ボタンをクリックすることで生成されたコールグラフ構築情報を元にコールグラフを可視化表示する。図3に実際にコールグラフを可視化表示した例を示す。

複雑な作業を省き手軽に利用できるようにシンプルな操作方法で利用できるように設計した。実際に目的のコールグラフを表示させるための操作はクリック操作を4回行うだけでよい。

5．まとめ

本論文では、class ファイルのバイトコードを解析することでコールグラフを構築する方法を提案した。このツールにはまだいくつかの課題が残されている。まず、このツールは Java の class ファイルを解析することが前提なので、Java で作成されたプログラムでも class ファイル形式でないものは解析できない。また、Swing や AWT 等の GUI（グラフィカルユーザインタフェース）機能を利用したソフトウェアでは見栄えのよいソフトウェアを容易に作成することが可能となるが、main メソッドにはプログラム呼び出しの処理が含まれず、実行されるメソッドを直接指定し、メソッド関係を呼び出さなければコールグラフを構築できないという問題点もある。この問題には対象となる class ファイル全体を解析対象とし抽出したメソッド全体の呼び出し関係を構築することで解決すると考えられる。試作したコールグラ

フ解析ツールにもまだ問題点が残っている。コールグラフを構築しグラフ形式で表示させる為の描画アルゴリズムが未完成の為、コールバックや再起呼び出し部分の表示を出力する際に呼び出し関係を表している枝が重なってしまい視認しづらくなる等のグラフの見易さの点に問題がある。これらのような問題点も現段階では挙げられるが、一般的な Java ソフトウェアの class ファイルに対しては今回試作したコールグラフ可視化ツールにおいて正常なコールグラフを構築できることを確認できたので、有用なツールとして利用することが可能である。前述した問題点を解決し、よりよいツールへと発展させることを課題としてあげる。

参考文献

[1] ティム・リンドホルム、フランク・イェリン 著、野崎裕子 訳著、"The Java 仮想マシン 仕様"、アジソン・ウェスレイ・パブリシャーズ・ジャパン、1997 年 12 月.

[2] J. Zhao、"Analyzing Control Flow in Java Bytecode," Proc. 16th Conference of Japan Society for Software Science and Technology, pp.313－316, Japan, September 1999.

[3] J. Zhao, "Dependence Analysis of Java Bytecode," Proc. 24th IEEE Annual International Computer Software and Applications Conference (COMPSAC'2000). pp.486－491, IEEE Computer Society Press, Taipei, Taiwan, October 2000.

[4] J. Zhao, "Static Analysis of Java Bytecode," Proc. 2001 International Software Engineering Symposium, pp.383 － 390, Wuhan, China, March 2001.