# 動的解析情報を利用したJavaプログラムからのシーケンス図の作成

谷口考治[†]　石尾隆[†]　神谷年洋[‡]　楠本真二[†]　井上克郎[†]
[†]大阪大学大学院情報科学研究科　[‡]科学技術振興機構さきがけ

## 1．はじめに

オブジェクト指向プログラムでは，オブジェクトが相互にメッセージを交換することによってシステムが動作する．そのため，プログラムの動作を理解するには，複数のオブジェクトがそれぞれどのように通信しているのかを理解する必要がある．しかし，動的束縛等実行時に決定される要素が多いことや，１つの機能に多数のオブジェクトが関与すること等から，ソースコード等の静的情報のみからその動作を理解する事は困難である[2]．

そこで我々は，プログラムの振る舞いの理解を支援するために，Java プログラムの実行履歴から UML のシーケンス図[3]を作成する手法を提案する．具体的にはまず，解析対象とするプログラムを実行し，メソッド呼び出しの実行履歴を取得する．その後，実行履歴中に含まれる繰り返し等の冗長な部分を圧縮する．その結果を元にシーケンス図を作成することで，オブジェクト間のメッセージ通信を簡潔に利用者に提示する．

本稿では，実行履歴の取得と圧縮手法について述べ，その結果の評価を行う．

## 2．シーケンス図作成までの手順

本手法では以下に示す４つの過程を経てシーケンス図を作成する．

Step1：解析対象プログラムへの入力決定
　解析対象となるプログラムへの入力を決定する．

Step2：実行履歴の取得
　Step1 で決定した入力を元にプログラムを動作させ，メソッド呼び出しの実行履歴を取得する．

Step3：実行履歴の圧縮
　Step2 で取得した実行履歴は冗長な情報を多く含む．そのため，メソッド呼び出し構造を解析，圧縮し，図として表現できるサイズに加工する．

Step4：シーケンス図作成
　Step3 の結果を元にシーケンス図を作成する．

本稿では Step2 と Step3 について詳細に述べる．

Extracting UML Sequence Diagram from Execution Trace of Java Program.
†Koji Taniguchi, †Takashi Ishio, ‡Toshihiro Kamiya, †Shinji Kusumoto, and †Katuro Inoue.
†Graduate School of Information Science and Technology, Osaka University
‡Presto, Japan Science and Technology Corporation

```
Gemini(0).main(java/lang/String){
    GeneralManager(44753296).GeneralManager(){
        MDI(44736792).MDI(GeneralManager){
            MDI(49860968).initComponents(){
                MDI(49860968).initMenuBar(){
                    MDI$2(44662040).MDI$2(MDI){
                    }
                    MDI$3(44666320).MDI$3(MDI){
                    }
                    MDI$4(44682200).MDI$4(MDI){
                    }
                }
```

図１：実行履歴の例

## 3．実行履歴の取得

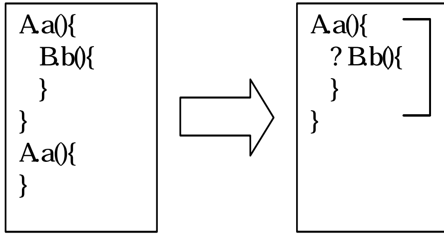
本手法では実行履歴として，プログラム実行中に発生するメソッド呼び出し情報を用いる．具体的には個々のメソッド呼び出しについて，メソッド開始時にクラス名，オブジェクト ID，メソッド名，引数のシグネチャを記録し，終了時にメソッド終了記号を記録する．これらの情報を用いることで，実行時に呼び出されたオブジェクトとメソッドを特定し，呼び出し構造を再現することが可能となる．

これまでにJavaプログラムを対象とした実行履歴取得システムを実装した．これにはJava Virtual Machine Profiler Interface (JVMPI)[1]を利用している．実際に取得した実行履歴の例を図１に示す．

## 4．実行履歴の圧縮

実行履歴中にはループや再帰構造の中で発生するメソッド呼び出しが全て記録されている．これらをそのままシーケンス図として表現する事は非常に冗長である．そこで，実行履歴中からこれらを検出，圧縮し，簡潔に図示できるようにする必要がある．またその際にはシーケンス図として表現しやすい構造に圧縮することが重要である．

そこで，繰り返し構造と再帰構造を検出，圧縮する方法として，以下に示すR1からR4まで




B.b()は繰り返し中で実行されない場合がある

図２：欠損構造を含む繰り返しの例

の４つのルールを考案した．

以下それぞれについて説明する．

R1：完全な繰り返し

実行履歴中から，完全に同一な構造が繰り返されている箇所を検出し，圧縮する．このルールでは繰り返し回数を記録しておけば，元の実行系列の内容を損なうことがない．

R2：オブジェクトが異なる繰り返し

実行履歴中から，オブジェクト ID のみが異なる構造が繰り返されている箇所を検出し，圧縮する．但し，圧縮結果として表現される呼び出し構造は，同一クラスのオブジェクト群に対しての呼び出しを表しており，元の実行系列全体を正確に表現していない．

R3：欠損構造を含む繰り返し

実行履歴中から，内部構造の一部に欠損を含むような繰り返しを検出し，圧縮する．欠損を含むような繰り返しの圧縮例を図２に示す．従って，欠損部分の呼び出しは，圧縮した結果「実行される場合と実行されない場合がある」としか表現できないため，元の実行系列の内容を正確には表現できない．

R4：再帰構造

実行履歴中の呼び出し構造において再帰的に呼び出されているメソッドを検出し，圧縮する．さらに再帰構造を簡潔に表現するために，欠損を許容した圧縮を行う．具体的には，再帰の各階層の中から，他の階層の和集合となりうる呼び出しの集合を選び，R3 と同様の欠損表現を用いて再構成を行う．また，ここではオブジェクト Id を考慮せず，同一クラスの同一メソッドであれば再帰として扱うものとする．このルールは圧縮効果そのものよりも，再帰構造の階層差を緩和し，他の圧縮ルールの効果を高めることを主な目的としている．

## 5．実験と評価

考案した４つの圧縮ルールを R4，R1，R2，R3 の順に実行履歴に適用し，圧縮効果を測定した．実験対象となる実行履歴は，テキストエディタ jEdit，コードクローン分析ツール Gemini，スケジュール管理プログラム Scheduler，そして実行履歴圧縮を行う本ツール LogCompactor の４つの Java プログラムから取得したものを用いた．それぞれの実行履歴の圧縮前と各ルール適用後のメソッド呼び出し回数を表１に示す．

まず R4 は圧縮効果よりも再帰構造の階層差の緩和を目的としているため，圧縮効果は高くない．次に R1 では，完全に一致する繰り返し部分が少ないためか，あまり圧縮効果は得られなかった．しかし，単純な繰り返しが多い Gemini では高い効果を発揮した．R2 は実験で用いた全ての実行履歴に対して高い圧縮効果を示しており，非常に有効であると思われる．最後の R3 では，圧縮前のメソッド呼び出し回数が少ないものに対しては効果を発揮している．これは，呼び出し回数が多くなり階層が深くなるほど，単純な欠損構造にはならないためと考えられる．

## 6．まとめ

本稿ではオブジェクト指向プログラムの理解支援を目的として，動的解析から得られたメソッド呼び出しの実行履歴の圧縮方法について述べた．

今後の課題は，圧縮した実行履歴からシーケンス図を作成する機能の実装である．

## 参考文献

[1] Java Virtual Machine Profiler Interface http://java.sun.com/j2se/1.4/ja/docs/ja/guide/jvmpi/jvmpi.html

[2] M. Lejter, S. Meyers, and S. T. Reiss. Support for Maintaining Object-Oriented Programs. IEEE Trans. Softw. Eng., 18(12):1045-1052, December 1992.

[3] Unified Modeling Language (UML) 1.3 specification. OMG, March 2000.

表１：圧縮ルール適用結果

| 実行履歴 | 元のメソッド数 | R4 | R1 | R2 | R3 | 圧縮率 |
|---|---|---|---|---|---|---|
| jEdit | 228,764 | 217,351 | 178,128 | 16,889 | 16,510 | 7.22 |
| Gemini | 208,360 | 205,483 | 57,365 | 1,954 | 1,762 | 0.85 |
| scheduler | 4,398 | 4,398 | 3,995 | 238 | 147 | 3.34 |
| LogCompactor | 11,994 | 8,874 | 8,426 | 208 | 105 | 0.88 |