The University of Osaka
Institutional Knowledge Archive

Title	Java実行履歴からのシーケンス図生成ツール
Author(s)	谷口, 考治; 石尾, 隆; 神谷, 年洋 他
Citation	組込みソフトウェアシンポジウム論文集. 2004, 2004(10), p. 108-111
Version Type	VoR
URL	https://hdl.handle.net/11094/51102
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

Java 実行履歴からのシーケンス図生成ツール

谷口考治† 石尾隆† 神谷年洋‡ 楠本真二† 井上克郎†

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，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．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 のシーケンス図を作成する手法を提案し，ツールとして実装した．本稿では，実行履歴の取得と圧縮手法，圧縮結果からのシーケンス図生成手法について述べ，4 つの Java プログラムに対して適用することで，ツールの評価を行った．

A Sequence Diagram Generation tool from Java Program Execution

Koji Taniguchi†, Takashi Ishio†, Toshihiro Kamiya‡, Shinji Kusumoto†, and Katuro Inoue†

A software system developed by object-oriented programming operates by message exchanges among the objects allocated by the system. To understand such system behavior, we need to grasp how the objects are communicating. However, it is difficult to understand the behavior of such system only from the source code. We propose a method of extracting a sequence diagram from an execution trace of a Java program in order to understand the behavior of the program. Also, we implemented the method as a tool and evaluated it through some case studies.

1. はじめに

近年，組み込みソフトウェアの分野においても，オブジェクト指向技術を用いてプログラムを開発することが増えてきている．高いモジュール性や再利用性が期待されているからである．また，UML[6]や Java 言語の普及，組み込みソフトウェア向けの開発環境の整備などにより，今後，オブジェクト指向を取り入れる傾向はますます高まってくると思われる．

オブジェクト指向言語で開発されたプログラムでは，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，動的束縛など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などから，ソースコードなどの静的情報のみからその動作を理解することは困難である[3]．

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 のシーケンス図を作成する手法を提案し，ツールの作成を行った．

一般的に，プログラムの実行履歴は膨大な量になる．これをそのまま用いてシーケンス図の作成を行うと，巨大な図ができあがってしまう．この図からプログラムの処理の概要を理解することは困難である．そのため，提示する情報量の削減を行う必要がある．本手法では，取得した実行履歴中から，繰り返しなどの特定のパターンを検出し，その部分を圧縮，抽象化して簡潔な表現に置き換えることで，全体の情報量を削減し，プログラムの動作を簡潔に表現するシーケンス図の作成を行う．また，元の実行系列も記憶しておき，注目したい各部分については部分的に圧縮結果を展開しながら閲覧することを可能にすることで，圧縮したことによる情報の損失の影響を小さくしている．

本ツールが行う具体的な処理は，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パターンを検出し，圧縮する．そして，その結果を元にシー

†大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
‡科学技術振興機構さきがけ
Presto, Japan Science and Technology Agency

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，本ツールが行う実行履歴の圧縮と，圧縮結果からのシーケンス図作成について，処理の詳細を述べる．

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膨大な量になる．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 から Step4 について述べる．

3. 圧縮手法

3.1. 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，引数の型を記録し，メソッド終了時にメソッド終了記号を記録する．引数の型を取得するのは，メソッドのオーバーロード時にどのメソッドが呼ばれたかを特定するためであり，実行時に引数として与えられた値については取得しない．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．

3.2. 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しても，プログラム全体の動作を理解することは困難である．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

実行履歴中の完全に同一な呼び出し構造が繰り返されている箇所を検出し，圧縮する．このルールによる圧縮は，元の実行系列の情報を損なわない．

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 のみが異なる呼び出し構造が繰り返されている箇所を検出し，圧縮する．ただし，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すことになり，呼び出されたオブジェクトを特定することはできなくなる．この手法によって，同じ ID のオブジェクトが、シーケンス図上部に複数表れることがある．結果のシーケンス図は，繰り返しによって統合されるオブジェクトの組み合わせの数だけ，横に大きくなってしまう．この問題に対して，同じオブジェクトを共有する統合されたオブジェクトを，さらに 1 つのオブジェクト群に統合して表現する．この結果，ある 1 つのオブジェクトは，単体のオブジェクトと，統合されたオブジェクト群の，高々2 つのオブジェクトで表現される．

R3：欠損構造を含む繰り返し

実行履歴中から，呼び出し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．欠損している呼び出しは，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，元の実行系列の構造を正確には表現しなくなる．

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．ここではオブジェクトの違いを考慮せず，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．さらに再帰構造を簡潔に表現するために，呼び出しの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，他の階層全てを包含するような階層の集合を選び，それらを組み合わせて再帰構造の簡潔な表現を作成する．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．

4. シーケンス図の作成

圧縮した実行履歴を元にシーケンス図の作成を行う．繰り返し回数などの，圧縮結果を元にした情報を注釈として表現することで，より分かりやすい図を作成する．

まず，圧縮されなかった部分はメソッド呼び出し構造

を，そのままシーケンスとして表現する．

R1 によって圧縮された部分には，通常のシーケンスに加えて，繰り返しを圧縮した部分に繰り返しを示す情報とその回数を表記する．

R2 によって圧縮された部分では，複数のオブジェクトを統合したオブジェクトへのシーケンスが存在するため，図中の上部に並ぶオブジェクト列の中に統合されたオブジェクト群を示すオブジェクトを追加し，それに対するシーケンスを引いていく．

R3 により圧縮された部分では，発生する場合としない場合がある呼び出しがある．その部分については，それが呼ばれるシーケンスと呼ばれずに素通りするシーケンスの 2 通りを引く．

R4 によって圧縮された部分は，統合されたオブジェクトへのシーケンスを含む再起呼び出しを含んでいる．そのため，再起呼び出しが発生する部分へ戻るようなシーケンスを引く．このシーケンスは，時間軸の過去へ戻るような表現になるため，左側への曲線の矢印で表すことによって，通常のシーケンスと区別できるようにする．

5. ツールの詳細

5.1. ツールの実装

3, 4 節の手法をツールとして実装した．本ツールは，実行履歴を取得するプロファイラ，実行履歴の圧縮を行う圧縮部，圧縮結果からシーケンス図を生成するシーケンス図生成部の 3 つの部分から構成されている．ツールの概要を図 1 に示す．

実行履歴を取得するプロファイラは Sun の Java VM に用意されている Java Virtual Machine Profiler Interface (JVMPI)[1]を利用し，C 言語で記述されたダイナミックリンクライブラリとして実装した．このプロファイラを Java VM にコマンドラインオプションとして渡して解析対象となる実行プログラムを実行することで，実行プログラムで発生する各メソッド呼び出しについて，3.1 節で述べた情報をスレッドごとに実行履歴ファイルに記録できる．JVMPI を利用していることから，解析対象のプログラムは JVMPI の仕様に準拠している JavaVM 上で動作することが必要である．対象プログラムを特殊なハードウェアやシミュレーション環境上のみでしか動作させられない場合は，現在の実装では，実行履歴が取得できない．この点については今後の課題である．

しかし，本ツールが言語および実行系に依存しているのはこのプロファイラだけであり，実行履歴取得システムを作成できれば，他言語，他実行系へのツールの拡張も可能であると考える．

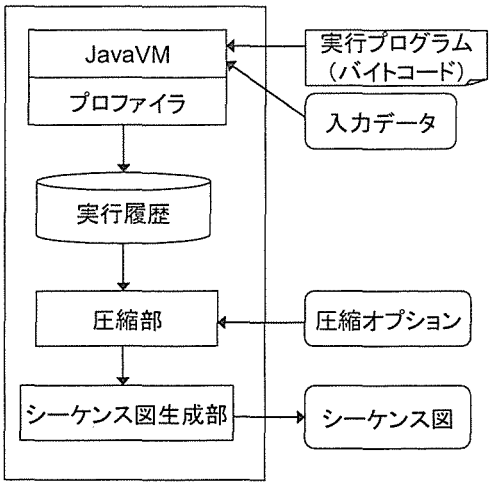

図1：ツールの構成図

圧縮部では 3.2 節で述べた圧縮ルールに従って，プロファイラが収集した実行履歴ファイルから実行履歴を読み込み，圧縮処理を行う．この時，ユーザは圧縮オプションを与えることによって，どのルールを用いて圧縮するか，圧縮処理中の比較条件では呼び出し構造の何階層下までを比較して同一性の判定を行うかなどの条件を指定できる．また，圧縮処理後，圧縮された部分を展開して元の実行履歴を取り出すことも可能であり，注目したい部分について，より詳細に解析することができる．

シーケンス図生成部では，圧縮結果を元に 4 節の方針にしたがってシーケンス図を生成する．後述する適用実験において，ある実行プログラムから生成されたシーケンス図を図 2 に示す．この図では R2 によって圧縮された部分のシーケンスが示されている．生成されるシーケンス図には，圧縮部による圧縮が行われた場所に注釈が表示される．図中の A, B, C は注釈の例であり，それぞれ，R2 によって圧縮された実行履歴中の繰り返されている部分，その繰り返し回数，統合されたオブジェクト群などを表している．現在の UML の仕様では繰り返しなどが表現できないため，この注釈は独自の形式を用いて表現している．今後策定される予定の UML2.0[7]では繰り返しが表現できるため，策定されればその形式に対応する予定である．

5.2. ツールの利用手順

ユーザは，本ツールのグラフィカルユーザインタフェースを用いて対話的に分析を行う．

まず，ユーザは解析対象とする実行プログラムと，そ

のプログラムを動作させる入力データを用意する．次に，本ツールを起動し，プロファイルボタンから対象プログラムを実行すると，ツールがメソッド呼び出しの実行履歴をファイルに保存していく．対象プログラムの実行が終了すると，記録された実行履歴のスレッドの一覧が表示され，その中から任意のスレッドを指定して，圧縮部に読み込ませる．

実行履歴の読み込みが終了した後，ユーザは圧縮ボタンを押して圧縮処理を行う．この時，圧縮オプションを指定することにより，4 つのルールの中から適用する圧縮ルールとその順番，繰り返しの圧縮ルールが呼び出し構造を比較する際に考慮するメソッド呼び出しの階層数を任意に指定できる．

そして最後に，シーケンス図生成ボタンを押すと，シーケンス図を表示するウィンドウが開く（図 2）．ユーザは表示されたシーケンス図を見ることにより，プログラム実行時のメソッド呼び出しの流れを理解することができる．

さらに，圧縮操作とシーケンス図の作成は対話的に行うことができる．すなわち，ユーザはシーケンス図で圧縮された部分，たとえば，繰り返しになっている部分を指定して展開し，繰り返し圧縮を行う前のシーケンス図を表示させることで，繰り返し 1 回ごとの詳細な振る舞いを見ることもできる．

6. 適用結果と考察

4 つの Java プログラム，テキストエディタ jEdit[2]，コードクローン解析ツール Gemini[5]，スケジュール管理ツール scheduler[4]，本ツールの実行履歴圧縮部 LogCompactor に対して，本ツールの適用実験を行った．まず，各プログラムについて実行履歴を取得し，その後，圧縮手法の適用，シーケンス図の生成を行った．scheduler では圧縮前の実行履歴のメソッド呼び出しの数が 4398 回であったが，圧縮により 147 回まで圧縮できた．図 2 は，その結果から作成されたシーケンス図である．この回数ならシーケンス図全体に目を通すことが可能であると考える．また，208360 回のメソッド呼び出しがあった Gemini の実行履歴では，圧縮処理により 1762 回まで圧縮できたが，まだ繰り返しになっている部分が含まれていることも分かった．このような部分をさらに圧縮できるような手法を考案することによって，大きな実行履歴からも，理解しやすいサイズのシーケンス図が生成できると考える．

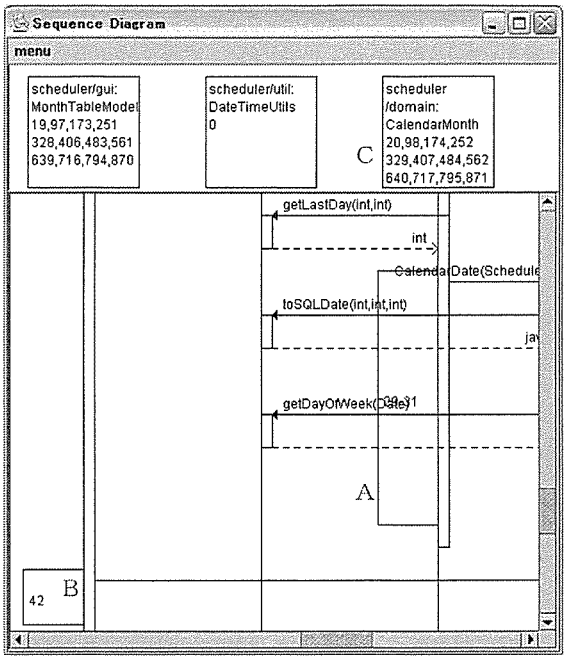

図 2：scheduler から生成したシーケンス図

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] jEdit. http://www.jedit.org/

[3] M. Lejter, S. Meyers, and S. P. Reiss. Support for Maintaining Object-Oriented Programs. IEEE Transaction of Software Engineering, 18(12):1045-1052, December 1992.

[4] 月刊 Java World 2002 年 10 月号. scheduler. IDG Japan. ftp://ftp.idg.co.jp/pub/jw/0210/scheduler_sample.zip

[5] Y. Ueda, T. Kamiya, S. Kusumoto, K. Inoue. Gemini: Maintenance Support Environment Based on Code Clone Analysis. Proceedings of Eighth IEEE Symposium on Software Metrics (METRICS2002), pp.67-76, Ottawa, Canada, June 4-7, 2002

[6] Unified Modeling Language 1.5 Specification. OMG, March 2003.

[7] Unified Modeling Language 2.0 Specification Nearing Completion. OMG.