特別研究報告

題目

# オブジェクトの動的支配関係解析を用いたシーケンス図の縮約

指導教員
井上 克郎 教授

報告者
伊藤芳朗

平成 20 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 19 年度 特別研究報告

# オブジェクトの動的支配関係解析を用いたシーケンス図の縮約

伊藤芳朗

## 内容梗概

オブジェクト指向プログラムは動的に決定される要素が多く，実行時の動作を理解するのは難しい．そのため動作の理解には静的なソースコードなどよりも，実行履歴などの動的な情報を用いて，シーケンス図などに可視化する手法が有用である．しかし，プログラムの開始から終了までには，多くのメソッド呼び出しが行われ，実行履歴は膨大な量となってしまい，そのままでは可視化しても人間の読解に適したサイズにはならない．この問題に対して，情報を簡潔に提示するための様々な手法が提案されているが，これらの手法は，繰り返されるメソッド呼び出しや，開発者にとって興味のないメソッド呼び出しを図から除去することを主な手段としている．

本研究では，オブジェクトをグループ化し，グループ外部から参照されないオブジェクトを隠すことで，オブジェクトグループ間の通信のみを可視化する手法を提案する．具体的には，実行履歴に出現するオブジェクト群の相互の呼び出しついての支配関係を計算しグループ化を行う．グループ間の情報のみを取り出して可視化することにより，簡潔なシーケンス図を生成する．

本手法の評価実験として，提案手法を実装したシステムを作成し，業務用 Web アプリケーションなどの実行履歴に適用した．その結果，実行履歴に含まれるオブジェクトのうち，平均で約 4 割がグループ化によって隠されることを確認した．我々の研究グループで開発しているシーケンス図生成システム Amida に対してオブジェクトのグループ化を行った実行履歴を与えることで，縮約された図を表示することができ，また，Amida が実装する既存手法と組み合わせた適用が可能であることを確認した．

## 主な用語

実行履歴
シーケンス図
オブジェクトのグループ化
支配関係解析

目 次

# 1　まえがき

オブジェクト指向プログラムでは，実行時に動的に生成されるオブジェクトが相互にメッセージを交換する事によってシステムが動作するが，どのオブジェクトがどのようにメッセージ通信を行うかは，実行時に動的に決定される．このようなシステムの振る舞いを理解するためには，プログラムの実行履歴を可視化することが有効である [15].

プログラムの実行履歴は一般に膨大な量になる [7, 8, 9]．そこで繰り返しや再帰構造を圧縮することで，簡潔なシーケンス図を生成する手法が提案されている．しかしこれらの手法の多くは，図中で繰り返されるメソッド呼び出しや，開発者にとって興味のないメソッド呼び出しを図から除去することで実現されており，図の正確さが損なわれるという弱点も備えている．

本研究では，オブジェクトをグループ化し，グループ外部から参照されないオブジェクトを隠すことで，オブジェクトグループ間の通信のみを可視化する手法を提案する．具体的には，実行履歴上に現れるオブジェクト群の呼び出し関係グラフについて支配関係の計算を行い，外部から参照されないオブジェクトを識別する．

支配関係とは，根となる頂点がただ 1 つ $root$ であるような有向グラフ上の 2 頂点 $v_1, v_2$ に対して定義される関係で，$root$ からは $v_1$ を通過しなければ $v_2$ に到達できないとき，$v_1$ は $v_2$ を支配するという．提案手法では，支配関係を，実行履歴から得られるオブジェクト間のメソッド呼び出しグラフに対して計算する．オブジェクト $v_1$ が別のオブジェクト $v_2$ を支配するという関係が得られたとき，$v_1$ と $v_2$ をグループ化し，オブジェクト $v_2$ を図中から取り除いても，$v_2$ への呼び出しは必ず $v_1$ を経由しているため，グループ外部のメソッド呼び出し関係は影響を受けない．この特徴を用いることで，オブジェクトグループ間の通信を削除することなくオブジェクトのグループ化を行い，簡潔なシーケンス図を作成する．

我々のチームはこれまでに，実行履歴からシーケンス図を生成するシステム Amida [12] を開発してきている．Amida は，オブジェクト指向プログラムにおけるオブジェクト間のメッセージのやり取りなど，プログラムの動作の理解支援を目的としており，プログラムの動的解析から得た実行履歴を基にシーケンス図の作成を行う．この Amida に入力する実行履歴を解析してオブジェクトのグループ化を行い，グループ内のメッセージ通信を削除し，グループ間のメッセージ通信のみを取得するツールを作成した。そしてこのツールを実際に業務用 Web アプリケーションなどの実行履歴に適用した結果，実行履歴に含まれるオブジェクトのうち，平均で約 4 割がグループ化によって除去されることを確認した．Amida が縮約された図を表示することができ，また，Amida が実装する既存手法と組み合わせた適用が可能であることを確認した．

以降，2 章ではこの研究の背景を述べ，3 章で提案手法の説明をする．4 章ではオブジェ

クトのグループ化の判定方法の実装について説明し，5 章では行った適用実験について述べる．最後に 6 章で本研究のまとめと今後の課題について述べる．

## 2　背景

### 2.1　実行履歴の可視化

オブジェクト指向プログラミングでは，クラスを用いて処理の抽象化を行い，また継承や多態性などを利用して，オブジェクト間のメソッド呼び出しとしてソフトウェアの機能を実現する．その一方で，開発者がソフトウェアの機能の詳細を読解する場合，たとえばあるメソッド呼び出し文が，動的束縛の結果，実際に呼び出しうる複数のメソッド定義を発見するといった労力の増大が生じており，支援ツールの必要性が指摘されている [5, 11, 15]．

特定の機能を実現するためのオブジェクトの相互作用は，設計段階において，主に UML のシーケンス図によって記述される [14]．シーケンス図によって表現される動作シナリオは，そのソフトウェアにおける特定のタスクを実現する方法を記述しており，ソフトウェアの理解や再利用にも適した単位である [9]．

### 2.2　UML シーケンス図

シーケンス図とは Unified Modeling Language(UML) [13] で定義されているインタラクション図の 1 つで，オブジェクト間のメソッド呼び出しやオブジェクトの生成などのメッセージ通信を，時系列に沿って示すことができる図である．横軸はオブジェクトの種類を表しており，図 1 のように，図の上部には，図中に記述されるメッセージ通信に関連するオブジェクトが横方向に並べられる．縦軸は時間軸を表しており，下方に行くほど時間が経過していく．各オブジェクトの下部には縦方向に点線が引かれており，これがオブジェクトが生存する区間を示している．さらに，個々のメッセージ通信について，時系列順に，送信元のオブジェクトから，送信先のオブジェクトに対して矢印を引く．送信されたメッセージがメソッド呼び出しだった場合は，メッセージを受けたオブジェクトは，そのメソッドの実行区間を縦長の長方形で表す．メソッドが終了する時点で，呼び出し元のオブジェクトへ戻り辺の矢印を引く．メッセージがオブジェクトの生成だった場合は，生成されるオブジェクトをその高さに書く．このようにして，オブジェクト間のメッセージの様子を時系列に沿って表現する．

UML で書かれた設計図を活用することが困難な場合もある．たとえば，開発が進行していく中でソフトウェアの機能に変更が加えられたとき，開発者が設計図の更新を怠ると，設計図がソフトウェアの最新の状態を正しく反映しなくなる [4]．また，ソフトウェアを実装していく段階で，設計図には登場しないようなクラス，メソッドが追加されることもある [12]．

そこで，開発されたソフトウェアからオブジェクト間の相互作用を検出し，UML のシーケンス図を含む様々な形式によって可視化する方法が研究されている．ソースコードを用いた解析によって得られる結果は動的束縛などを静的に解決できる範囲に限られるため，実際

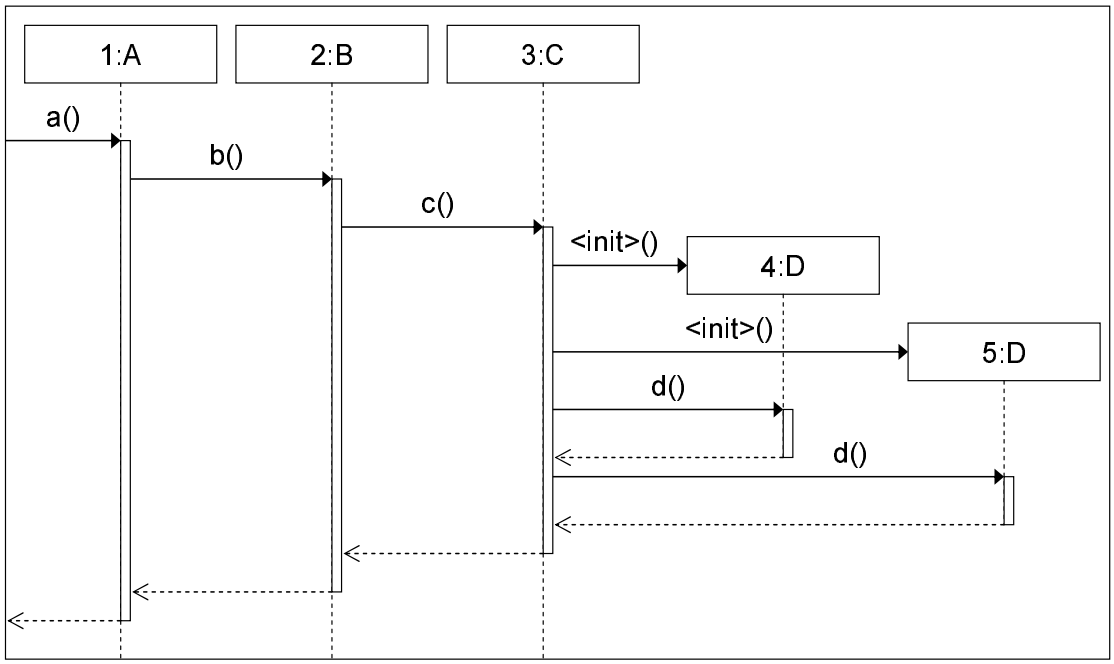



図 1: シーケンス図

にソフトウェアを実行し，どのオブジェクトが呼び出されたかを記録した実行履歴を解析し，可視化する手法が広く研究されている．

実行履歴は，プログラムの実行開始から終了までの，膨大な数のメソッド呼び出しの系列である．オブジェクトの ID およびメソッド名の系列があれば，シーケンス図として可視化することができる [12].

### 2.3 Amida

我々の研究グループは，オブジェクト指向プログラムにおけるオブジェクト群の動的な振る舞いを視覚的に表現し，プログラムの理解支援を行うために，プログラムの動的解析から得た実行履歴を基にシーケンス図の生成を行うツール Amida を作成してきている [12]. Amida は Java プログラムのメソッド呼び出しを実行履歴として取得するプロファイラと，その実行履歴を解析してシーケンス図を生成，GUI にて表示するビューアからなるツールである．図 2 は Amida の GUI の図で，図の左側にシーケンス図を表示する．ただし，実行履歴から生成したシーケンス図は膨大サイズとなってしまうため上部にオブジェクトを表示し，その下にシーケンス図の一部分を表示する．図の右側はシーケンス図の全体図の表している．

Amida にはシーケンス図上の繰り返し処理や再起呼び出しなどのループの部分を圧縮し

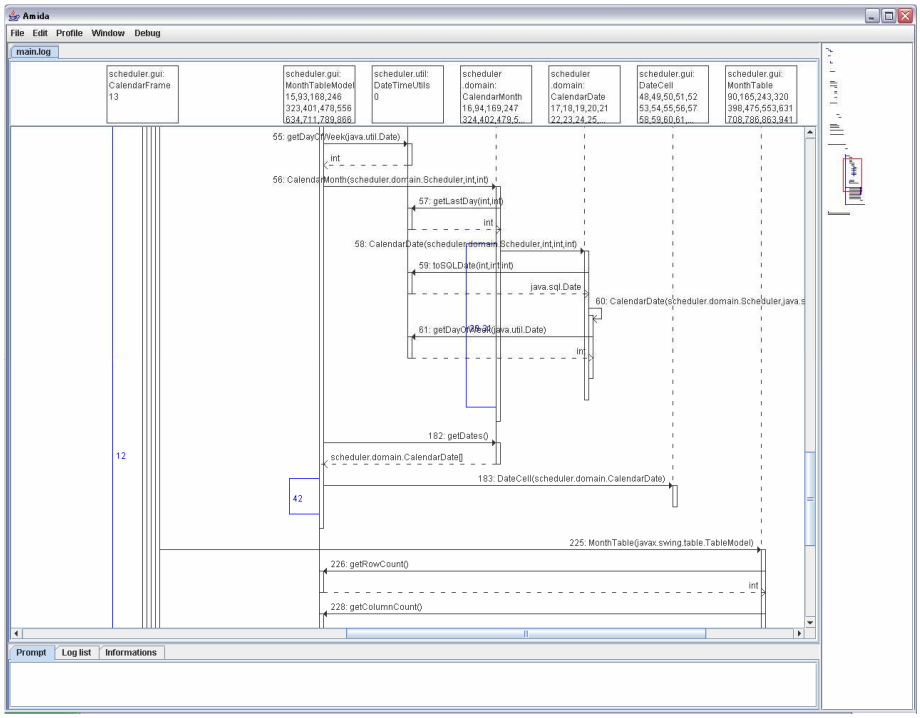



図 2: シーケンス図生成システム Amida

て表示する機能が実装されている．ループの部分を実際に圧縮して表示すると，図 3 のように表示される．図の赤の線で示されている範囲でループの圧縮が行われている．ループを圧縮することで，シーケンス図全体を圧縮する．図 4 の右の図ではループ部分を圧縮表示している．その圧縮部分を元に戻したものが左の図である．この部分だけでおよそ 10 倍の差がある．

Amida の特徴は上記のように，膨大な量となる実行履歴に対して繰り返し処理や再起呼び出しの検出を行い，簡潔な図を生成する点である．実行履歴を単純にシーケンス図として可視化しただけでは人間の読解に適したサイズにはならないことは広く知られており，他にもパターン検出を用いた圧縮処理 [8] や，実装の詳細である可能性が高いメソッドの自動的なフィルタリング [3]，概要を把握するための縮小表示 [6] など，様々な手法が研究されている．

これらの手法の多くは図中から開発者にとって興味のないメソッド呼び出しを取り去ることで実現されており，図の正確さが損なわれるという弱点も備えている．本研究で提案するオブジェクトのグループ化は，グループ外部から参照されないオブジェクトを隠すことでオブジェクトのグループをより大きな処理単位として可視化する．これにより，生成されるシーケンス図の正確性を損なうことなく，処理全体の流れを把握することが可能となる．

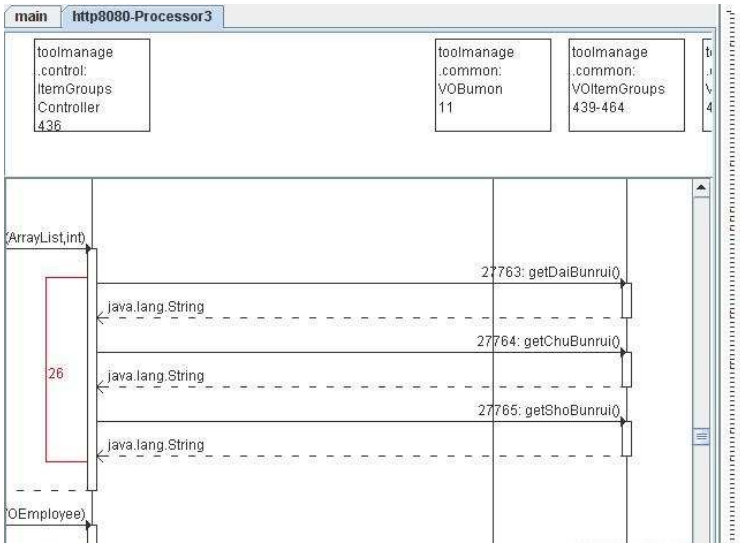



図 3: ループの圧縮

## 3 提案手法

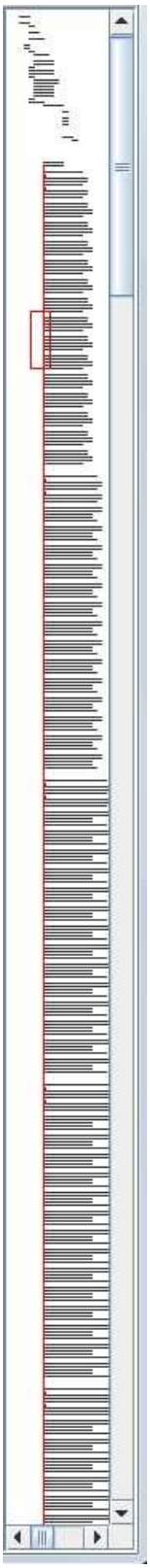
本研究では，実行履歴中に含まれるオブジェクトのグループ化を行い，グループ外部から参照されないオブジェクトを隠すことで，シーケンス図中に登場するオブジェクトの数を削減する手法を提案する．具体的には，実行履歴に出現するオブジェクト群の相互の呼び出しについての支配関係を解析し，その結果に基づいてグループを構築する．

提案手法は，Java プログラムの実行履歴を可視化するシステムである Amida の存在を念頭に置いて，Java プログラムの実行履歴に対応した手法となっているが，オブジェクト ID が取得可能な実行環境を持つプログラミング言語などにも適用可能である．

### 3.1 動的解析による実行履歴の取得

シーケンス図は，オブジェクトの生成とオブジェクト間のメソッド呼び出しについて，時系列に沿って表現する図である．オブジェクトの生成のメッセージとメソッド呼び出しのメッセージは，シーケンス図上では別の形式で表現されるが，本研究で対象としている Java 言語においては，オブジェクトの生成は，コンストラクタの呼び出しとみなせるため，実行履歴上は同様のものとして扱える．以降は，メソッド呼び出しという時は，コンストラクタの呼び出しも含めることとし，オブジェクトの生成のメッセージをメソッド呼び出しのメッセージと同種のものとして扱う．

対象とするプログラムに対して動的解析を行い，オブジェクト間のメソッド呼び出しに関する情報を実行履歴として記録する．具体的には，個々のメソッド呼び出しについて，メ



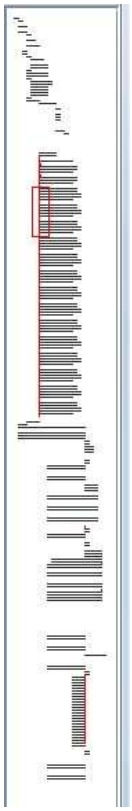


図 4: シーケンス図の全体図

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，返り値の型を記録する．コンストラクタの呼び出しは，<init>という名前のメソッド呼び出しとして表現している．また，メソッド終了時には終了記号を記録する．引数の型を記録するのは，メソッドがオーバーロードされて同名のメソッドが複数存在する場合に，メソッド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記録しない．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，メソッドの呼び出し構造を再現することが可能となる．

実行履歴を取得するシステムは，対象とするプログラムの実行中に個々のメソッド呼び出しを捉え，実行履歴を保存するファイルに「返り値の型 クラス名 (オブジェクト ID). メソッド名 (引数の型){」，という形式で記録する．static メソッドの呼び出しについては，オブジェクト ID を 0 番とする．閉じ括弧「}」のみの行は，それぞれ対応する開き括弧のメソッドの終了を表している．本研究では，オブジェクトの支配関係を求めるためオブジェクト ID とクラス名，オブジェクト間の呼び出し関係だけに注目する．

例として Amida の実行から取得した実行履歴を図 5 に示す．この図は Main クラスの main メソッドが MainFrame クラスのオブジェクトを作成する経過の一部を示している．

### 3.2 オブジェクトのグループ化

実行履歴にはプログラムの開始から終了までに発生したすべてのメソッド呼び出しが記録されているため，膨大な量となっている．これ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のは困難である．そこでシーケンス図に出現するオブジェクト群を外部のオブジェクトから参照されないようにグループ化し，グループ間のメッセージ通信だけを抽出することで，実行履歴全体の流れを把握できるシーケンス図を生成する．

オブジェクトをグループ化して，1 つのグループを 1 つのオブジェクトと扱うことでシーケンス図に出現するオブジェクトの数を減らすことができる．また，グループ内のメッセージ通信を非表示にすることでシーケンス図に出現するメッセージ通信の数も減らすことができる．シーケンス図に出現するオブジェクトの数が減れば，シーケンス図は横方向の縮約が，メッセージ通信の数が減れば，シーケンス図は縦方向の縮約が可能となる．

本手法で提案するオブジェクトのグループ化は次の条件を満たすように行う．

- 1 つのグループは 1 つの代表となるオブジェクトを持つ．
- グループ外部からのメッセージ通信はグループの代表となるオブジェクトを呼び出す．
- グループも 1 つのオブジェクトとみなし，グループの階層化を可能とする．

```
void amida.Main(0).main(java.lang.String[]){
void amida.sequencer.gui.MainFrame(1).<init>(){
void amida.sequencer.gui.SearchDialog(2).<init>(){
amida.sequencer.gui.MainFrame amida.sequencer.gui.MainFrame(0).getInstance(){
}
amida.sequencer.gui.MainFrame amida.sequencer.gui.MainFrame(0).getInstance(){
}
void amida.sequencer.gui.SearchDialog$1(3).<init>(amida.sequencer.gui.SearchDialog){
}
void amida.sequencer.gui.SearchDialog$2(4).<init>(amida.sequencer.gui.SearchDialog){
}
}
void amida.logcompactor.gui.WorkingSetFrame(5).<init>(java.lang.String){
void amida.logcompactor.gui.WorkingSetCanvas(6).<init>(int){
}
void amida.logcompactor.gui.WorkingSetCanvas(7).<init>(int){
}
}
void amida.logcompactor.gui.LogTextAreaFrame(8).<init>(){
void amida.logcompactor.gui.SearchDialog(9).<init>(){
void amida.logcompactor.gui.SearchDialog$1(10).<init>(){
}
void amida.logcompactor.gui.SearchDialog$2(11).<init>(){
}
```

図 5: 実行履歴の例

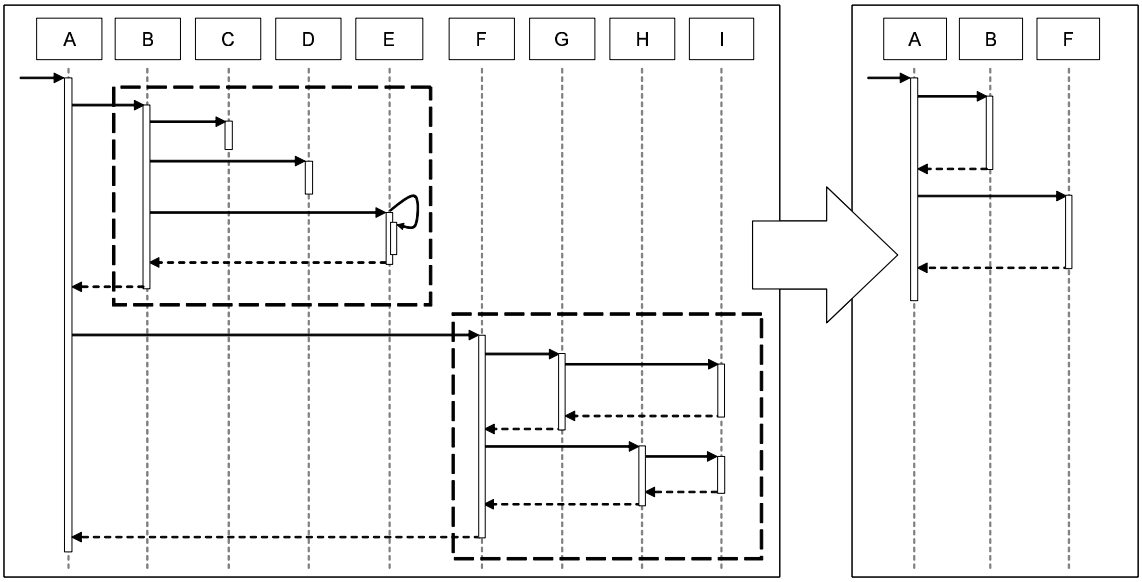



図 6: オブジェクトのグループ化によるシーケンス図の縮約

この条件を満たしたオブジェクトのグループ化では，グループの代表ではないオブジェクトは同じグループに含まれるオブジェクトからのみメッセージを受け取る．そのため代表オブジェクト以外のオブジェクトをすべて取り除いても，グループ外部のメッセージ通信は影響を受けず，シーケンス図全体の整合性は保たれる．

階層化したグループを作成するのは，グループ化の解除を行うときにシーケンス図が一気に拡大するのを防ぐためである．グループ内のオブジェクトやメッセージ通信を見たい場合には，グループの解除を行うしかないが，そのグループに大量のオブジェクトが含まれていた場合，シーケンス図が拡大してしまい，縮約した意味がなくなってしまう．それを防ぐために，階層的なグループ化を行い，グループの中身のオブジェクトを表示する場合でも，そのすべての情報を一度に表示するのではなく，開発者にとって必要最小限の表示で済ませるためである．

本手法はオブジェクトのグループ化を行うが，本来オブジェクトに属さない static メソッドも実行履歴に含まれている．数学関数を提供する Math クラスやコレクションに対する操作を提供する Collection クラスなどが多数の static メソッドを定義しており，static メソッドの呼び出し元は相互に関連性がない場合がある．グループ化するときに関連性が低いオブジェクトを同じグループとして判断してしまう可能性を避けるため，static メソッドの呼び出しは，呼び出しごとに個別の仮のオブジェクトが存在するとみなしてグループを判断する．

プログラムの中には複数スレッドで処理を行うものがある．その場合はスレッドごとに区別して実行履歴を取得する．グループ化を行う際は，複数スレッドで共通のオブジェクトが

存在したとしても，複数スレッドの実行履歴を同時に可視化することが困難であるため，グループ化は実行履歴中のスレッド単位で行う．

### 3.3 動的支配関係解析

本研究では，実行履歴に含まれるオブジェクト間の呼び出し関係からオブジェクトの動的な支配関係を求めることで，グループ化すべきオブジェクト群を識別する．

実行履歴中で，あるオブジェクト $o_1$ が別のあるオブジェクト $o_2$ のメソッドを 1 度でも呼び出すとき，$o_1$ は $o_2$ に対する呼び出し関係を持つという．実行履歴呼から呼び出し関係を求めることで，ソースコードから取得できる静的な呼び出し関係を使用せず，ソースコードが存在しないプログラムでもシーケンス図の生成を行うことができる．また，リストやツリーなどの動的に構築されるデータ構造内部で生じる呼び出し関係にも対応できる．

支配関係とは，有向グラフついて，根（入次数が 0 である頂点）がただ 1 つ $root$ であるときに，グラフ上の 2 頂点 $v_1, v_2$ に対して定義できる関係である．$root$ からは $v_1$ を通過しなければ $v_2$ に到達できないとき，$v_1$ は $v_2$ を支配する（dominate）という．この関係は，通常，プログラムの制御フローグラフに対して解析されるもので，コンパイラがプログラムを最適化するために使用する [10]．なお，$v_1$ が $v_2$ を支配するとき，$v_1$ は $v_2$ のドミネータ（dominator）であるという．オブジェクト間の呼び出し関係グラフは， Java プログラムの場合はただ 1 つのエントリポイントである main メソッドが存在するため，main メソッドを根 $root$ とみなすことで支配関係を定義することができる．

オブジェクト間の呼び出し関係グラフに対し，支配関係解析を適用する．本研究では，反復計算によるアルゴリズム [2] を用いた．支配関係には推移性が成り立っており，$v_1$ が $v_2$ を支配し，かつ $v_2$ が $v_3$ を支配するとき，$v_1$ は $v_3$ を支配する．この性質を用いて，支配関係は支配関係木（dominance tree）によって表現される．我々が用いたアルゴリズムは，この性質を利用して，すべてのオブジェクト $o_k$ について，対応するイミディエイトドミネータ（immediate dominator）$idom(o_k)$ を計算する．イミディエイトドミネータは，あるオブジェクトにとって，支配関係木での直接の親を意味する．図 7 に，呼び出し関係グラフの，図 8 に，呼び出し関係グラフから求められる支配関係木の例を示す．

得られたオブジェクト間の支配関係木を用いて，支配関係木の各接点 $o$ について，直接の子であるような接点をグループメンバーとし，$o$ 自身がグループ代表であるようなグループを作成する．図 9 に，図 8 で示した支配関係木からグループを作成した例を示す．このようにして作成されたグループは，次のような性質を持つ．

- あるグループ $G$ について，グループの代表を $d$ とすると，$\forall o \in G, d \neq o \Rightarrow idom(o) = d$ が成り立つ．よって，グループの外部から，グループの代表 $d$ を経由しないような

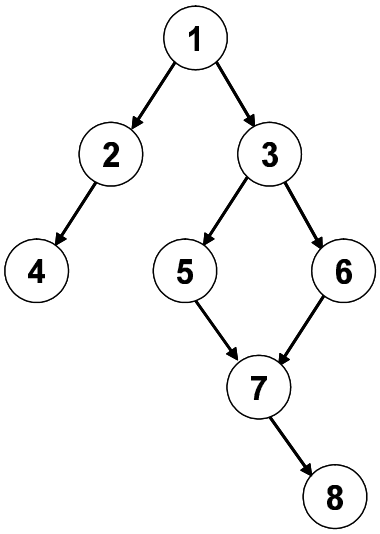



図 7: 呼び出し関係グラフ

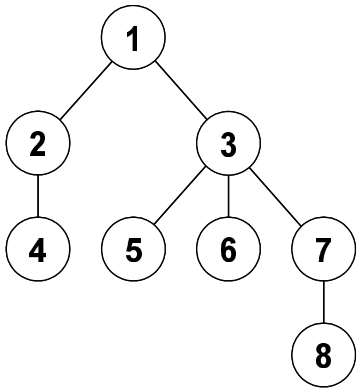



図 8: 図 7 のグラフに対して得られる支配関係木

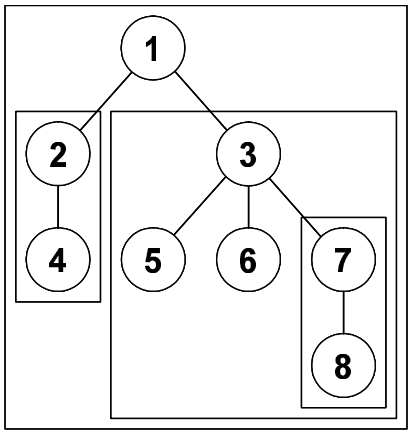



図 9: 図 8 の支配関係木から作られたグループ

グループ $G$ のメンバーへのメソッド呼び出しは存在しない.

- グループは，グラフの根である $root$ を支配関係木の根とした階層的な構造となる.

上記の性質は，3.2 で述べた，グループ化の条件を満たしている.

グループは階層的に構築されているため，実行履歴中に登場したすべてのオブジェクトは，開始頂点 $root$ を代表とした 1 つの巨大なグループとなる．これでは，シーケンス図に表示する情報が 1 つのオブジェクトだけになってしまい，プログラムの理解支援とならない．それを避けるため，本研究では，$root$ の直接の子であるようなオブジェクトグループ群を，最終的な出力結果とした.

### 3.4 シーケンス図の生成

実行履歴に対して，支配関係解析を行うと，各グループの代表となるオブジェクトが得られる．グループ代表オブジェクトと，グループ代表に対するメソッド呼び出しのみを実行履歴から抽出することで，最終的なシーケンス図の出力を行うことができる.

なお，上記のような対応を行うと，最も外部にあるグループだけが意味を持ち，グループ化が階層的であることによる効果は特にない．しかし，今後，シーケンス図生成システム Amida 中で，対話的にオブジェクトグループのメンバーを展開しながらシーケンス図を閲覧する，あるいは，特定のグループ内部のシーケンス図だけを可視化するといった対応を行っていく予定である.

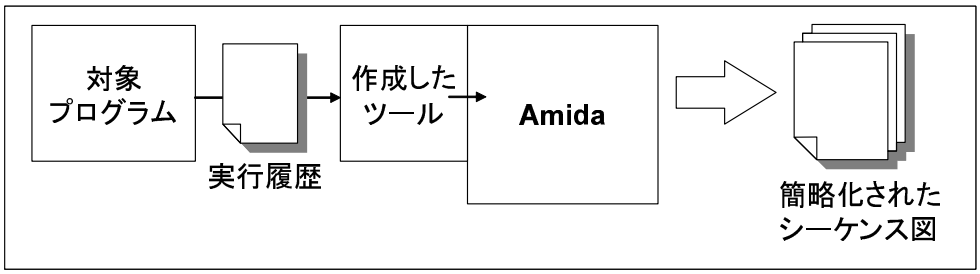



図 10: ツールの構成

## 4 実装

3 章で述べた手法をツールとして実装した．手法のうち，実行履歴の取得とシーケンス図の生成に Amida [12] を使用する．Amida は Java プログラムを対象とし，プログラムのメソッド呼び出しを実行履歴として取得し，それを解析してシーケンス図を生成，GUI にて表示する．

作成したツールと Amida の関係を図 10 に示す．今回実装したツールは Amida が取得した実行履歴に動的支配関係解析を適用し，支配関係木において main の直接の子となるグループのメッセージ通信のみを取り出した実行履歴を Amida の使用する形式で出力する．main の直接の子となるグループ間のメッセージ通信だけを取り出すのは，本手法のグループ化を最大限適用した場合に，シーケンス図の縮約がどれほど行えるのかを確認するためである．

このツールの処理の流れは以下の 3 つのステップからなる．

1. オブジェクトの呼び出し関係の取得
2. グループの解析
3. 実行履歴の出力

### オブジェクトの呼び出し関係の取得

実行履歴のオブジェクト間のメソッドの呼び出し関係からオブジェクトの呼び出し関係を求める．あるオブジェクトのメソッド呼び出しから，そのメソッドの終了記号までに，別のオブジェクトのメソッド呼び出しが行われた場合，そのオブジェクトを呼び出していると判断する．出現したオブジェクトには，後のステップで支配関係を解析するために，呼び出し元のオブジェクトには呼び出したオブジェクトの ID を，呼び出されたオブジェクトには呼び出し元のオブジェクトの ID をそれぞれ記録しておく．ここでは，出現したオブジェクト

のクラス名，オブジェクト ID，オブジェクトの前後の呼び出し関係だけを記録しそれ以外の情報は記録しない．

Amida は実行履歴を取得する際にスレッドごとにメソッド呼び出しを区別している．そのためオブジェクトはスレッドごとに別のものとして呼び出し関係を求める．またオブジェクト ID が 0 番のオブジェクトはクラス名が同じでも別オブジェクトとして考えるため，出現順に内部的に ID を割り当てることで区別する．

これを実行履歴の最後まで行ってすべてのオブジェクトの呼び出し関係を取得する．

### グループの解析

取得したオブジェクト間の呼び出し関係に対して，支配関係解析を適用して，支配関係木を作成する．支配関係木からオブジェクトのグループ化を行い，シーケンス図に表示するオブジェクトを求める．

シーケンス図に表示するオブジェクトは支配関係木の根を代表オブジェクトとしたグループのメンバーとなるため，実装時には支配関係木の直接の子となるオブジェクトを取得する．

### 実行履歴の出力

グループの解析で求めた支配関係木の根を代表オブジェクトとするグループのメンバーとなるオブジェクトへのメッセージ通信を表示するため，再度実行履歴を読み込み，オブジェクト ID を判断して出力するオブジェクトのメソッド呼び出しだけを出力し，それ以外の情報は削除する．メソッドの終了記号もメソッド呼び出しとの対応を調べ，該当するものだけ出力する．

# 5　適用実験

4 章で述べたツールを用いて，適用実験を行った．本章ではその内容および結果と結果に対する評価を述べる．

## 5.1　実験目的

本手法によって，シーケンス図の縮約が実際に行えているかを確認し，支配関係解析の効果は実行履歴の差異への依存が大きいのかどうかを調べる．また，本手法を適用した前後で，実行履歴にどのような差が出るのかを調べる．

## 5.2　実験対象

実験対象として以下の実行履歴を用意した．

### 業務用 Web アプリケーション

このアプリケーションは Struts フレームワーク [1] を用いた Web アプリケーションで，データベースアクセスに DAO を用いる，要求機能単位での実行遷移が画面表示によって切り替わる，などの特徴がある．本実験で使用した実行履歴は 4 種類あり，処理 A，B，C，D で別の処理を実行している．

処理 A は，ログイン，物品検索，物品詳細情報取得の順に，処理 B はログイン，情報メンテナンス，備考メンテナンス，ログアウトの順に，処理 C は，ログイン，情報メンテナンス，備考メンテナンス，備考内容変更，ログアウトの順に，処理 D はログイン，情報メンテナンス，備考メンテナンス，1 行追加，元に戻す，ログアウトの順に処理を行った時の実行履歴である．

### ソースコード解析プログラム

我々の研究室で開発中のツール．与えられたソースコードに対する解析を行うプログラムである．本実験で使用した実行履歴は 4 種類あり，処理 A，B，C，D で別の処理を実行している．

処理 A は対象を指定しないで実行したときの，処理 B は対象ファイルを指定して実行したときの，処理 C は処理 B とは別のファイルを指定して実行したときの，処理 D は処理 C と同じ対象だが，特定のメソッド呼び出しを除去したものを指定して実行したときの実行履歴である．

図書管理システム

Struts フレームワークを用いた Web アプリケーションで，同じ設計仕様に基づいて学生 4 グループが異なる実装で作成した 4 つのシステムである．

システム A，B，C，D はそれぞれのシステムで，どのシステムも同じ処理を実行して取得した実行履歴である．

### 5.3 実験方法

取得した複数の実行履歴に 4 章で作成したツールを適用し，ツールを使用する前後でどれほどシーケンス図の縮約ができたのかを調べる．また，実行履歴や生成されたシーケンス図などにどのような変化が現れるのかを調べる．

### 5.4 実験結果

それぞれの実行履歴のツールの適用前後のオブジェクト数とメソッド呼び出し数を表 1，2，3 に記す．複数スレッドで処理が行われているものはスレッドごとに分けて結果を出した．表中の出現オブジェクト数は実行履歴をシーケンス図にしたときに図に表れるオブジェクトの数を表し，メッセージ通信数はシーケンス図に表れるメッセージ通信の合計を表している．また，圧縮率は以下の式によって計算した．

$$\text{圧縮率} = \frac{\text{適用後の出現オブジェクト数またはメッセージ通信数}}{\text{適用前の出現オブジェクト数またはメッセージ通信数}} \times 100(\%)$$

表を見ると多くのスレッドで縮約でき，手法が有効であることが確認できる．圧縮率だけに注目すると，まったく圧縮できずに圧縮率が 100 %となっているものから，適用後の値が 1 となるほど圧縮されてしまうものまであるのが分かる．図書管理システムの表 3 では，システム A，B，C は比較的に近い圧縮率を示しているが，システム D だけ極端に圧縮されている．

業務用 Web アプリケーションと図書管理システムはすべての実行履歴で main スレッドは縮約できなかった．これらの実行履歴では共通して，main スレッドはプログラムの起動時と終了時にしか処理が行われておらず，また出現するオブジェクト間にはメッセージ通信が存在しなかったため，グループ化が行われなかったのが原因である．

逆に，業務用 Web アプリケーションと，ソースコード解析プログラムのスレッドの中には出現オブジェクト数が 1 となってしまうものがある．これは実行履歴を解析して得た支配関係木が図 11 のようになってしまっているのが原因である．実行履歴中にあるすべてのオブジェクトがオブジェクト A を代表オブジェクトとするグループのメンバーとして縮約されてしまい，ある特定のオブジェクト A へのメッセージ通信のみしかシーケンス図上に表

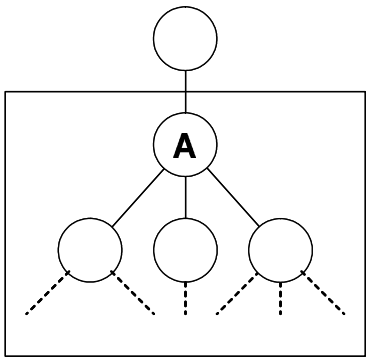



図 11: 支配関係木の表示オブジェクトが過度に削減される例

示されなくなってしまう．ただし，これは提案手法の適用方法に問題がある．今回作成したツールはすべてのグループ化が終わった時点でのグループ間のメッセージ通信のみを取り出している．このような場合でもグループ化を解除して，グループ内の情報を確認できるようにツールを作成することで，必要な情報が取得できる．

### 5.5 考察

図書管理システムの実行履歴うちシステム A の実行履歴について，本手法を適用する前と後で出現するオブジェクト比較した．図 12 は，ツールを使用する前後の実行履歴に出現するオブジェクトのクラス名を抽出したものである．図の左側は手法適用前の実行履歴に出現するオブジェクトの一覧で，右側が適用後である．手法適用後に残っているオブジェクトのオブジェクト名には関連性があり，残っているオブジェクトは主に Action や jsp という名前を持ったオブジェクトであった．

図書管理システムでは，Action オブジェクトはユーザから命令された処理を実行し制御する役割を持ち，jsp オブジェクトは処理の結果を画面に表示する役割を持っている．これらのオブジェクトはサーバから直接呼び出しを受けて動作するため，本手法を適用してもシーケンス図に表示されると考えられる．そして，これらオブジェクトは全体の動作の中でも重要な処理を行う部分であるので，これらのオブジェクトがシーケンス図に表示されることで，全体の動作を問題なく把握できると考えられる．

残ったオブジェクトの中には Data と名前を持つオブジェクトも存在する．このオブジェクトはデータを保持するための一時オブジェクトだと考えられるが，シーケンス図からは消えず，さらにそのオブジェクトへ，データの設定や取得といった繰り返し処理が行われていた．これらの動作はシステム全体の動きからすると重要ではない情報と考えられるので，シーケンス図に残すよりは隠したほうが，全体の動作を把握しやすくなると考えられる．

逆に，手法を適用した前後で消えたオブジェクトは，Manager や DAO という名前を持ったオブジェクトである．実行履歴を調べると Manager オブジェクトは Action オブジェクトや jsp オブジェクトから呼ばれて，DAO オブジェクトを呼び出しデータオブジェクトを取得するといった処理を行っていた．そのため，Manager オブジェクトや DAO オブジェクトはシーケンス図から消えたと考えられる．

図書管理システムで画面表示を行う際には，ユーザからの命令を Action オブジェクトが実行し，Manager オブジェクトなどを使用して，必要な情報を集め，jsp オブジェクトがその情報を画面に出力するという流れになっている．このとき，Action オブジェクトが Manager オブジェクトを通して得た Data オブジェクトは，その後，jsp オブジェクトによって呼び出され，オブジェクト内のデータを取得し，画面出力に使用する．つまり，Data オブジェクトは 2 つのオブジェクトから参照されるため，処理全体から見ると重要なオブジェクトではないが，本手法ではシーケンス図に表示されることになる．

addstock.AddStockAction
addstock.Cache
addstock.Item$ImageSize
addstock.Item
addstock.ItemManager
addstock.ShowAddStockAction
addstock.StockDAO
addstock.StockManager
logon.DynaLogonAction
logon.LogoffAction
logon.LogonDAO
logon.LogonManager
stockstate.ChangeStockStateLooku
pDispatchAction
stockstate.ListBorrowedAction
stockstate.ListStockAction
stockstate.ListWatchAction
stockstate.ShowStockAction
stockstate.StockData
stockstate.StockStateDAO
stockstate.StockStateManager
util.ConnectionUtil
util.Resources
util.SequenceNumber
web.AuthenticationFilter
web.ImageServlet
addstock.addStock_jsp
common.doctype_jsp
common.footer_jsp
common.header_jsp
logon.logon_jsp
stockstate.listBorrowed_jsp
stockstate.listStock_jsp
stockstate.listWatch_jsp
stockstate.showStock_jsp

addstock.AddStockAction
addstock.ShowAddStockAction
logon.DynaLogonAction
logon.LogoffAction
stockstate.ChangeStockStateLookupDis
patchAction
stockstate.ListBorrowedAction
stockstate.ListStockAction
stockstate.ListWatchAction
stockstate.ShowStockAction
stockstate.StockData
util.ConnectionUtil
web.AuthenticationFilter
web.ImageServlet
addstock.addStock_jsp
common.doctype_jsp
common.footer_jsp
common.header_jsp
logon.logon_jsp
stockstate.listBorrowed_jsp
stockstate.listStock_jsp
stockstate.listWatch_jsp
stockstate.showStock_jsp

図 12: 実行履歴に出現するオブジェクトの変化

表 1: 業務用 Web アプリケーションの適用結果

| 処理 | スレッド | 出現オブジェクト数 | | | メッセージ通信数 | | |
|---|---|---|---|---|---|---|---|
| | | 適用前 | 適用後 | 圧縮率 | 適用前 | 適用後 | 圧縮率 |
| A | main | 7 | 7 | 100.0 | 11 | 11 | 100.0 |
| | http8080-Processor4 | 176 | 4 | 2.3 | 10645 | 30 | 0.3 |
| | http8080-Processor3 | 96 | 1 | 1.0 | 5253 | 1 | $\leq 0.1$ |
| B | main | 8 | 8 | 100.0 | 12 | 12 | 100.0 |
| | http8080-Processor4 | 163 | 4 | 2.5 | 10091 | 27 | 0.3 |
| | http8080-Processor3 | 251 | 1 | 0.4 | 14177 | 2 | $\leq 0.1$ |
| | http8080-Processor2 | 72 | 1 | 1.4 | 2323 | 1 | $\leq 0.1$ |
| C | main | 8 | 8 | 100.0 | 12 | 12 | 100.0 |
| | http8080-Processor4 | 163 | 4 | 2.5 | 10091 | 27 | 0.3 |
| | http8080-Processor3 | 251 | 33 | 13.1 | 19990 | 1369 | 6.8 |
| | http8080-Processor2 | 72 | 1 | 1.4 | 2323 | 1 | $\leq 0.1$ |
| D | main | 8 | 8 | 100.0 | 12 | 12 | 100.0 |
| | http8080-Processor4 | 315 | 4 | 1.3 | 16303 | 36 | 0.2 |
| | http8080-Processor3 | 247 | 1 | 0.4 | 14159 | 1 | $\leq 0.1$ |
| | http8080-Processor2 | 7 | 1 | 14.3 | 20 | 1 | 5.0 |

表 2: ソースコード解析プログラム

| 処理 | スレッド | 出現オブジェクト数 | | | メッセージ通信数 | | |
|---|---|---|---|---|---|---|---|
| | | 適用前 | 適用後 | 圧縮率 | 適用前 | 適用後 | 圧縮率 |
| A | | 94 | 20 | 21.3 | 4916 | 3182 | 64.7 |
| B | main | 9199 | 28 | 0.3 | 796024 | 1837 | 0.2 |
| | Thread-0 | 9 | 1 | 11.1 | 22 | 1 | 4.5 |
| | Thread-1 | 3 | 1 | 33.3 | 16 | 1 | 6.3 |
| | Signa | 2 | 1 | 50.0 | 4 | 3 | 75.0 |
| | plugin-1 | 31 | 1 | 3.2 | 127 | 3 | 2.4 |
| C | main | 7988 | 34 | 0.4 | 1151338 | 1925 | 0.2 |
| | Thread-0 | 9 | 1 | 11.1 | 22 | 1 | 4.5 |
| | Thread-1 | 3 | 1 | 33.3 | 16 | 1 | 6.3 |
| | plugin-1 | 31 | 1 | 3.2 | 135 | 3 | 2.2 |
| D | main | 4571 | 34 | 0.7 | 704257 | 1913 | 0.3 |
| | Thread-0 | 9 | 1 | 11.1 | 22 | 1 | 4.5 |
| | Thread-1 | 3 | 1 | 33.3 | 16 | 1 | 6.3 |
| | plugin-1 | 31 | 1 | 3.2 | 127 | 3 | 2.4 |

表 3: 図書管理システムの適用結果

| システム | スレッド | 出現オブジェクト数 | | | メッセージ通信数 | | |
|---|---|---|---|---|---|---|---|
| | | 適用前 | 適用後 | 圧縮率 | 適用前 | 適用後 | 圧縮率 |
| A | main | 2 | 2 | 100.0 | 4 | 4 | 100.0 |
| | http-8080-Processor25 | 97 | 56 | 57.7 | 1037 | 715 | 68.9 |
| | http-8080-Processor24 | 97 | 55 | 56.7 | 999 | 669 | 67.0 |
| | http-8080-Processor23 | 120 | 78 | 65.0 | 1331 | 1002 | 75.3 |
| B | main | 2 | 2 | 100.0 | 4 | 4 | 100.0 |
| | http-8080-Processor25 | 72 | 30 | 41.7 | 757 | 395 | 52.2 |
| | http-8080-Processor24 | 127 | 81 | 63.8 | 1527 | 1148 | 75.2 |
| | http-8080-Processor23 | 124 | 80 | 64.5 | 1509 | 1168 | 77.4 |
| C | main | 2 | 2 | 100.0 | 4 | 4 | 100.0 |
| | http-8080-Processor25 | 204 | 136 | 66.7 | 2451 | 1885 | 76.9 |
| | http-8080-Processor24 | 65 | 26 | 40.0 | 700 | 358 | 51.1 |
| | http-8080-Processor23 | 65 | 32 | 49.2 | 707 | 396 | 56.0 |
| D | main | 2 | 2 | 100.0 | 4 | 4 | 100.0 |
| | http-8080-Processor25 | 109 | 2 | 1.8 | 1268 | 37 | 2.9 |
| | http-8080-Processor24 | 47 | 3 | 6.4 | 462 | 8 | 1.7 |
| | http-8080-Processor23 | 125 | 2 | 1.6 | 1453 | 48 | 3.3 |
| | http-8080-Processor22 | 116 | 2 | 1.7 | 1319 | 36 | 2.7 |

## 6　まとめ

オブジェクト指向プログラムの動作理解には実行履歴などの動的な情報を可視化する手法が有効であるが，一般に実行履歴は膨大な量となるため，視覚化した情報も膨大なものとなってしまう．本研究は実行履歴上に現れるオブジェクトの支配関係を用いてオブジェクト群をグループ化することで簡潔なシーケンス図を生成する手法を提案した．

今後の課題としては，開発者が対話的にグループ化とグループ化の解除を制御できようにすることが挙げられる．現状，実装したツールではそれができないため，グループ内の情報が欲しいときや，極端なグループ化が起こった場合は，Amida 単体を使用するか実行履歴を直接見るかしか方法がない．そのため，提案手法を Amida に組み込み，Amida の GUI 上でグループ化とグループ化の解除を行う方法を検討している．さらに，Amida に実装されているループの圧縮機能と組み合わせることでさらにシーケンス図を縮約することが期待できる．それ以外にも，他の関連手法と組み合わせることでシーケンス図の縮約を行う方法を考えている．

## 謝辞

本研究において，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします．

本研究において，適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に深く感謝いたします．

本研究において，逐次適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 助教に深く感謝いたします．

本研究において，様々な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 渡邊 結 氏に深く感謝いたします．

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします．

## 参考文献


[1] Apache Struts Project． http://struts.apache.org/

[2] Cooper, K. D., Harvey, T. J. and Kennedy, K.: A Simple, Fast Dominance Algorithm. http://www.cs.rice.edu/ keith/EMBED/

[3] Hamou-Lhadj, A. and Lethbridge, T.: Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System, Proceedings of the 28th International Conference on Software Engineering, pp.181-190, 2006.

[4] LaToza, T. D., Venolia, G. and DeLine, R.: Maintaining Mental Models: A Study of Developer Work Habits. Proceedings of the 28th International Conference on Software Engineering, pp.492-501, Shanghai, China, 2006.

[5] Lejiter, M., Meyers, S. and Reiss, S. P.: Support for Maintaining Object-Oriented Programs. IEEE Transactions on Software Engineering, Vol.18, No.12, pp.1045-1052, 1992.

[6] Pauw, W. D., Jensen, E., Mitchell, N. Sevitsky, G., Vlissides, J. M. and Yang, J.: Visualizing the Execution of Java Programs. Revised Lectures on Software Visualization, International Seminar, pp.151-162, 2002.

[7] Pauw, W. D., Lorenz, D., Vlissides, J. and Wegman, M.: Execution Patterns in Object-Oriented Visualization. Proceedings of the 4th Conference on Object-oriented Technologies and Systems, pp.219-234, April 1998.

[8] Reiss, S. P. and Renieris, M.: Encoding program executions. Proceedings of the 23rd International Conference on Software Engineering, pp.221-230, May 2001.

[9] Richner, T. and Ducasse, S.: Using Dynamic Information for the Iterative Recovery of Collaborations and Roles. Proceedings of the 18th International Conference on Software Maintenance, pp.34-43, October 2002.

[10] 佐々 正孝，滝本 宗宏：静的単一代入形式を用いた最適化 (導入編)．コンピュータソフトウェア，Vol.25，No.1(2008)，pp.19-29．

[11] Spinellis, D.: Abstraction and Variation. IEEE Software, Vol.24, No.5, pp.24-25, 2007.

[12] 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎: プログラムの実行履歴からの簡潔なシーケンス図の生成手法．コンピュータソフトウェア，Vol.24，No.3(2007)，pp.153-169.

[13] Unified Modeling Language(UML)1.5 specification. OMG, March 2003.

[14] Unified Modeling Language(UML)2.0 Infrastructure Specification. http://www.omg.org/

[15] Wild, N. and Huitt, R.: Maintenance Support Object-Oriented Programs. IEEE Transactions on Software Engineering, Vol.18, No.12, pp.1038-1044, December 1992.