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手法の提案

伊藤 芳朗† 渡邊 結† 石尾 隆† 井上 克郎†

† 大阪大学 大学院情報科学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3

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には，実行履歴情報を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るため，そのままシーケンス図として可視化した場合，人間の読解に適したサイズにはならない．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取り出した簡潔なシーケンス図を生成する．評価実験の結果，実行履歴中の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シーケンス図から取り除けることを確認した．

Reducing a sequence diagram by dynamic dominance analysis for objects

Yoshiro Ito† Yui Watanabe† Takashi Ishio† Katsuro Inoue†

† Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

To understand the behavior of an object-oriented system, visualizing its execution trace as sequence diagrams is effective because the system involves dynamically determined elements. However, sequence diagrams generated from execution traces are too large to understand because the traces include many method calls. This paper proposes a method to vizualize objects and method calls among groups of objects as sequence diagrams by replacing a group of objects with a representative. We conducted a case study as a reduced sequence diagram. As a result, it shows the tool removes about 40 percent of the objects from the execution traces on average.

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こと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 [14].

プログラムの実行履歴は一般に膨大な量になる [8, 9, 10]．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，グループ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関係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 [13] を開発してきている．Amida は，オブジェクト指向プログラムの動作の理解支援を目的としており，プログラムの実行履歴を基にシーケンス図の作成を行う．本研究では，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．

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, 12, 14].

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．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 [10].

2.2 UML シーケンス図

シーケンス図とは Unified Modeling Language(UML) 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．この図を作成することで，設計者はシステムが実際に動作する際のオブジェクト間の関連を，設計時にイメージすることができ，また，他の設計者や開発者にオブジェクトの動作を分かりやすく示すことができる．

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，

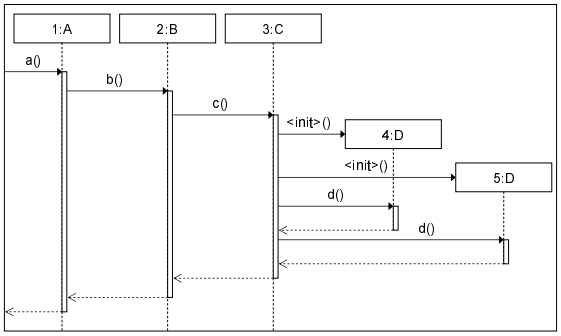

図 1: シーケンス図

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 [13].

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 と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 [13].

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 を開発してきている [13]．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 のスクリーンショットである．実行履歴から生成したシーケンス図は膨大なサイズとなってしまうため，ウィンドウの右側にシーケンス図の全体図を表示している．これによって，利用者は，現在表示している領域が，全体のどの部分に当たるかを確認することができる．

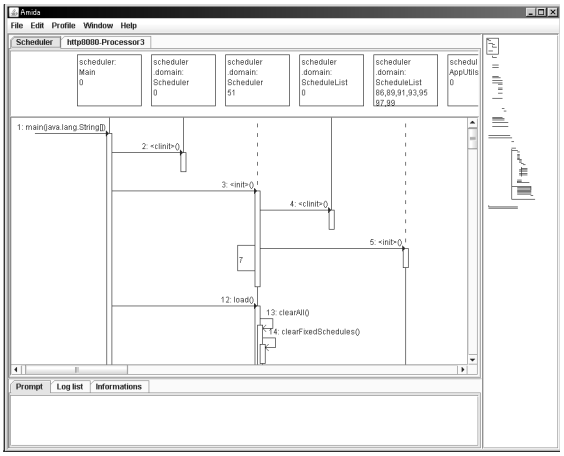

図 2: シーケンス図生成システム Amida

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て表示する機能が実装されている．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，圧縮することで簡潔な図を生成することができる．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 [9] 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 [7] 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．

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であれば適用可能である．

3.1 動的解析による実行履歴の取得

シーケンス図上では，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，本稿におけるメソッド呼び出しという表記は，コンストラクタの呼び出しも含めるものとし，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．

本手法では，まず，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，メソッド実行開始時 (動的束縛の解決後) 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．このとき，static メソッドの呼び出しについては，オブジェクト ID として，static な呼び出しであることを示す特別な値 ϕ を記録する．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．

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．

3.2 オブジェクトのグループ化

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

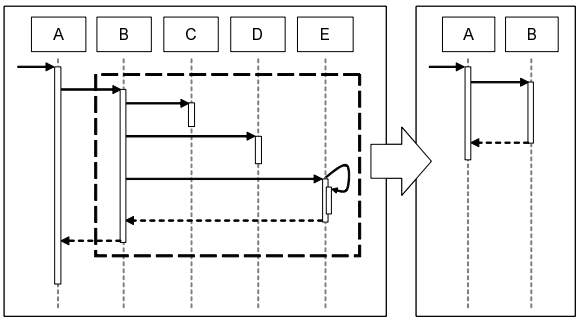

図 3: グループ化によるシーケンス図の縮約

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．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に縮約され，メッセージ通信の数が減れば，シーケンス図は縦方向に縮約される．

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．

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．

グループを階層化するのは，開発者がグループ内部の通信を調査したいとき，段階的にグループ内部の情報を図として提示していくためである．

オブジェクトのグループ化を行うとき，static なメソッドの扱いが問題となる．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 Collections クラスなどは多数の 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のスレッドで処理を行うものがある．本研究では，スレッドを区別して実行履歴を取得する．グループ化を行う際は，複数のスレッドで共通のオブジェクトが存在したとしても，複数の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，有向グラフについて，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 [11]．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 4 に，呼び出し関係グラフとそのグラフから求められる支配関係木の例を示す．

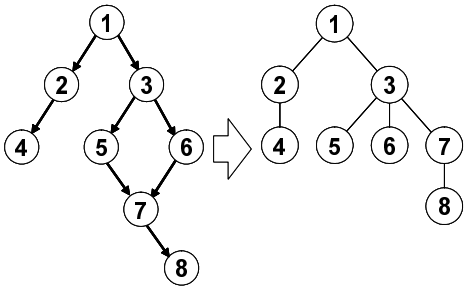

図 4: 呼び出し関係グラフと支配関係木

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．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．

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．

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 main 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，main 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．

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．

4 適用実験

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，本手法の適用前後で，実行履歴にどのような差が出るのかを調べるために実験を行った．

4.1 実験方法

実験対象として Struts フレームワーク [1] を用いた Web アプリケーションである図書管理システムを用意した．これは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，すべての機能を網羅する同一のシナリオを用いて，各システムの実行履歴を取得した．

Amida [13] を使用して取得した実行履歴に本手法を適用し，支配関係木において main の直接の子となるグループ間のメッセージ通信のみを抽出し，本手法を適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た．また，グループの代表となるオブジェクトと，除去されるオブジェクトが何であるのかを調べるため，実行シナリオの各機能に対応した実行履歴の区間を開発者に指定してもらい，各区間ごとに本手法を適用し，オブジェクトと機能の対応関係を調査した．

4.2 オブジェクト数の変化の調査

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を表 1 に，メソッド呼び出し数を表 2 に記す．表 1 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ている．このオブジェクト数には main など static メソッドに割り当てた仮オブジェクトの数は含んでいない．表 2 の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．

$$\text{圧縮率} = \frac{\text{適用後のオブジェクト数}}{\text{適用前のオブジェクト数}} \times 100(\%)$$

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．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．

すべての実行履歴で main スレッドは縮約の効果を得られなかった．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．

表 1: 出現オブジェクト数の適用結果

システム	スレッド	出現オブジェクト数		
		適用前	適用後	圧縮率
A	main	2	2	100.0
	http1	97	56	57.7
	http2	97	55	56.7
	http3	120	78	65.0
B	main	2	2	100.0
	http1	72	30	41.7
	http2	127	81	63.8
	http3	124	80	64.5
C	main	2	2	100.0
	http1	204	136	66.7
	http2	65	26	40.0
	http3	65	32	49.2
D	main	2	2	100.0
	http1	109	2	1.8
	http2	47	3	6.4
	http3	125	2	1.6
	http4	116	2	1.7

4.3 オブジェクトの種別の調査

4 つの図書管理システムの実装のうちシステム A の実行履歴について，縮約によって取り除かれるオブジェクトの種別と，ソフトウェアの機能の関係について調査した．

ある機能を実行したときに出現するオブジェクトは，その機能を実行している間のみに出現するオブジェクトと，他の機能の実行履歴にも出現するオブジェクトの 2 種類に分けられる．特定の機能の実行履歴にのみ出現するオブジェクトは，その機能を実行するために使われるオブジェクトで，複数の機能の実行履歴に出現するオブジェクトは，そのプログラム全体の処理に関連したオブジェクトである．これらのうち，複数の機能の実行履歴に出現するオブジェクトは，プログラム全体を理解するために重要なオブジェクトである．

シーケンス図に表示されたオブジェクトのうち，複数の機能の実行に出現するオブジェクトについて調べた．このようなオブジェクトの名前には共通点があり，“Action” もしくは “jsp” という単語が含まれていた．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の役割から，プログラム理解のために表示されるべきオブジェクトと判断できる．そして，実際にこれら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 [6].

表 2: メッセージ通信数の適用結果

システム	スレッド	メッセージ通信数		
		適用前	適用後	圧縮率
A	main	4	4	100.0
	http1	1037	715	68.9
	http2	999	669	67.0
	http3	1331	1002	75.3
B	main	4	4	100.0
	http1	757	395	52.2
	http2	1527	1148	75.2
	http3	1509	1168	77.4
C	main	4	4	100.0
	http1	2451	1885	76.9
	http2	700	358	51.1
	http3	707	396	56.0
D	main	4	4	100.0
	http1	1268	37	2.9
	http2	462	8	1.7
	http3	1453	48	3.3
	http4	1319	36	2.7

シーケンス図に表示されたオブジェクトの中に，“Data” と名前を持つオブジェクトが存在した．実行履歴を調べると，この Data オブジェクトは一時オブジェクトとして動作しているが複数のオブジェクトから参照されているため，本手法では，シーケンス図に表示されてしまう．このオブジェクトは各機能ごとに生成されていて，複数の機能で使用されることはなかった．そのため，このオブジェクトは重要ではないと判断できたが，本研究で用いた支配関係解析では，あるオブジェクトが特定の機能に属しているかどうかを判定できないため，除去の対象とはならなかった．このようなオブジェクトの取り扱いは，今後の課題である．

一方で，グループ化によりシーケンス図から除去されたオブジェクトの詳細についても調査を行った．その結果，除去されたオブジェクトの多くは，“DAO” や “Item” という文字列を含むクラスのインスタンスであった．“DAO” という名前を持ったオブジェクトはデータベースから読み込んだデータを Data オブジェクトに設定する処理を行っていたが，特定の Action オブジェクトや jsp オブジェクトからしか呼び出されていないため，シーケンス図から消えたと考えられる．“Item” という名前を持ったオブジェクトはデータの検索時に検索条件を保持する一時オブジェクトであった．そのため，DAO オブジェクトと同じく，特定のオブジェクトからしか参照されず，シーケンス図から取り除かれた．これらのオブジェクトは，各機能の実装の詳細であるため，プログラム全体の動作を理解する場合には重要度が低いと判断することができる．したがって，これらのオブジェクトがシーケンス図から除去されたことは妥当である．

5 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．提案手法を Amida に組み込み，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．また，本手法で得られるオブジェクトの階層構造について，開発者の考えるクラスの重要度や，ソフトウェアの複雑度との関係を調査したいと考えている．

謝辞

本研究は，日本学術振興会科学研究費補助金若手研究（スタートアップ）（課題番号:19800021）の助成を得た．

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] Lienhard, A., Greevy, O. and Nierstrasz, O.:Tracking Objects to Detect Feature Dependencies. Proceedings of the 15th IEEE International Conference on Program Comprehension, pp.59-68, June 2007.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

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