類似した振舞いのオブジェクトのグループ化による クラス動作シナリオの可視化

宗像 聡[†]　石尾 隆[†]　井上 克郎[†]

[†] 大阪大学 大学院情報科学研究科

オブジェクト指向システムにおいて，あるクラスの動作を理解するには，そのクラスのオブジェクトの実際の振舞いをシーケンス図として可視化する手法が有効である．しかし，1 つのクラスから多数のオブジェクトが生成される場合，オブジェクトごとにその動作を逐一確認することは現実的ではない．そこで本研究では，振舞いの類似性に基づいてオブジェクトをグループ化し，そのクラスの代表的な振舞いのみを可視化する手法を提案する．ケーススタディとして，ある Java プログラムの動作理解へと提案手法を適用し，その有効性を確認した．

Visualization of Representative Class Interaction Scenarios based on Clustering of Object Behavior

Satoshi Munakata[†]　Takashi Ishio[†]　Katsuro Inoue[†]

[†] Graduate School of Information Science and Technology, Osaka University

Visualizing an execution trace of an object-oriented system as sequence diagrams is effective to understand the behavior of a class. However, a class may create a large number of instances; a developer is hard to read sequence diagrams for each object. In this paper, we propose to classify objects into groups based on their behavior, and visualize only representative behavior of objects. As a case study, we have applied our approach to understand a Java program.

1　はじめに

オブジェクト指向プログラミングでは，クラスが基本的なソフトウェア部品であり，システムの動作を理解するにはクラスの動作を理解する必要がある．特に保守作業においては，その遂行に必要な少数のクラスの動作だけを効果的に理解するための手法が必要とされている．

オブジェクト指向システムでは，複数のオブジェクトが相互にメッセージ通信を行うことで処理が行われる．オブジェクトとメッセージはシステムの実行時に動的に生成されるため，クラスの動作を理解するには，そのクラスのオブジェクトの実際の振舞いを可視化することが有効である [2, 7, 8]．しかし，解析対象のクラスのオブジェクトが多数ある場合，それぞれの動作を逐一確認することは現実的ではない．同じクラスのオブジェクトが多数存在したとしても，その多くは互いに振舞いが類似していると期待されるため，代表的な振舞いのみを可視化することで，より効果的に理解を行うことができると考えられる．

そこで本研究では，特定クラスの複数のオブジェクトを，その振舞いの類似性に基づいてグループ化することで，代表的な振舞いのみを可視化する手法を提案する．ユーザは各振舞いを比較することで，クラスの動作を効果的に理解することができる．ケーススタディとして，ある Java プログラムに提案手法を適用し，その有効性を確認した．

以降では本手法を提案するにいたった背景を説明し，次に 3 章で本手法について，4 章で実装について，5 章でケーススタディについて述べる．最後に 6 章でまとめと今後の課題を述べる．

2　背景

2.1　オブジェクトの振舞いの可視化

オブジェクトの振舞いの可視化には，UML のシーケンス図が有効である [1, 7, 8]．シーケンス図による可視化の利点は，オブジェクト間の相互作用を時間軸に沿って直観的に確認できることである．しかし実用的なシステムでは，実行履歴には膨大な数のメソッド呼び出しが記録されており，単純に可視化すると図は大規模になり読解が困難になる．そのため，メソッド呼び出し系列内のループ構造や再帰構造を検出して圧縮処理を行うなどの工夫がとられる．

2.2　動作シナリオの抽出

実用的なオブジェクト指向システムでは，出現するオブジェクトの振舞いをすべて確認することは困難である．そのため，特定のクラスの各オブジェクトの振舞いから，クラスの動作シナリオを抽出する手法が提案されている．

代表的な手法に，特定のメソッド呼び出し系列を受理する有限状態オートマトンを用いて，クラスの動作シナリオを表現する手法がある [4, 6]．あるクラスのすべてのオブジェクトのメソッド呼び出し系列を受理する有限状態オートマトンを構築することで，そのクラスが取りうるすべての振舞いを表現することができる．しかし，これは実際の振舞いの近似であり，オブジェクトの具体的な振舞いを確認することはできない．

Salah らは，特定のクラスのオブジェクトのメソッド呼び出し系列を，互いの類似度にに基づきクラスタリングすることで，代表的なクラスの動作シナリオを抽出する手法を提案している [3]．しかし，このクラスタリングに必要な時間計算量は厳密解法で指数時間と大きい．また，動作シナリオは正規表現に似た形式で提示されるが，これもまた実際の動作の近似である．

本手法では，特定のクラスのオブジェクトを振舞いの類似性に基づいてグループ化し，それぞれの代表オブジェクトの振舞いを可視化することで，このクラスの代表的な動作シナリオを提示する．この動作シナリオはオブジェクトの実際の振舞いであるため，繰り返し構造の反復回数などの詳細を確認できるほか，他のクラスとの相互作用を確認することもできる．

3　提案手法

本研究では，開発者が注目するクラスのオブジェクトをその振舞いの類似性に基づいてグループ化することで，そのクラスの代表的な振舞いのみを可視化する．具体的には，実行履歴から特定クラスのオブジェクトが関係したメソッド呼び出し系列を抽出，それらの属性に基づいてオブジェクトをグループ化し，各グループから選んだ代表オブジェクトの振舞いのみをシーケンス図として可視化する．

提案手法は，ある 1 つのクラスの動作例を抽出するだけでなく，2 つのクラス間の相互作用例を抽出するためにも使用できる．基点となるクラスと相互作用する他のクラスについても同様にグループを構築し，互いのグループ間の呼び出し関係を解析する．グループの組み合わせごとに，呼び出し関係のあるオブジェクトのペアを代表として選択し，その動作をシーケンス図として可視化する．

提案手法により，実行履歴から特定クラスのグループごとの代表的な振舞いと，他のクラスとの相互作用の例がシーケンス図として可視化される．ユーザはこれらの図を比較することで，すべてのオブジェクトの動作を逐一確認することなく，クラスの実際の動作についての理解を深めることができる．

3.1 実行履歴の取得

本手法は，オブジェクト指向プログラムの実行時情報を用いる動的解析手法の 1 つである．利用者は，解析したいソフトウェアを実行し，動作中に行われた各メソッド呼び出しについて，実行されたスレッドの識別子，タイムスタンプ，呼び出し元（呼び出した）オブジェクトのクラスと ID，呼び出し先（呼び出された）オブジェクトのクラスと ID，メソッドシグネチャの情報を実行履歴として取得する必要がある．本研究では，Java を対象として実装を行っており，Amida[1] を用いて実行履歴を取得している．

3.2 各オブジェクトの動作コンテキストの抽出

本手法では，あるオブジェクト o の振舞いを，その相互作用に関わるクラスやメソッドの集合に近似する．具体的には，実行履歴から，各オブジェクトについて以下の 4 つの集合を計算する．なお，コンストラクタ呼び出しはメソッド名が ⟨init⟩ であるメソッド呼び出しとして扱う．

- $Use(o)$ ：オブジェクト o に対してメソッド呼び出しを行ったクラスの集合．
- $Used(o)$ ：オブジェクト o からメソッド呼び出しを行なわれたクラスの集合．
- $Methods(o)$ ：オブジェクト o の呼び出されたメソッドの集合．
- $Called(o)$ ：オブジェクト o が呼び出したメソッドの集合．

3.3 グループの構築

本手法では，クラス c_k のオブジェクトをグループ化する基準として，関連クラスに基づくグループ化 $G_c(c_k)$ と，関連メソッドに基づくグループ化 $G_m(c_k)$ を用いる．

関連クラスに基づくグループ化 $G_c(c_k) = \{G_1, \cdots, G_l\}$ は，以下の条件を満たすようにオブジェクトをグループに分類する．

$\forall o_1, o_2 \in G_i (1 \leq i \leq l) \leftrightarrow Use(o_1) = Use(o_2) \wedge Used(o_1) = Used(o_2)$

これはある 1 つのグループに属しているどんな 2 つのオブジェクトも，相互作用するクラス群が完全に一致するということを意味する．どのようなクラスと相互作用するかは設計時に作成されるクラス図やシーケンス図に明記されているが，実際のオブジェクトの動作では，相互作用する可能性のあるクラスのすべてではなく，いくつかのクラスとのみ相互作用する場合がある．たとえば，画面に表示されるデータとそうでないものなど，オブジェクトごとの性質の違いから，相互作用するクラスの組み合わせが異なることが期待され [5]，$G_c(c_k)$ はこのような振舞いの差を認識する．

一方，関連メソッドに基づくグループ化 $G_m(c_k) = \{G_1, \cdots, G_l\}$ は，以下の条件を満たすようにオブジェクトをグループに分類する．

$\forall o_1, o_2 \in G_i (1 \leq i \leq l) \leftrightarrow Methods(o_1) = Methods(o_2) \wedge Called(o_1) = Called(o_2)$

$G_m(c_k)$ で同一のグループに分類されたオブジェクト群は，呼び出されたメソッド，呼び出したメソッドが完全に一致する．この分類基準は，$G_c(c_k)$ による分類と異なり，どのクラスから呼ばれていても，呼び出されるメソッドが一致するとき，オブジェクトを同じグループへと分類する．ユーティリティクラスのように多くのクラスから利用されているクラスでは，どのクラスから利用されていても，振舞いはほぼ同じであることが期待され，利用のされ方の違いをグループ化に反映する．

なお，これらのグループ化は，同値類としての性質を満たしている．そのため，解析対象のクラスのオブジェクト数を n ，グループ化後のグループ数を g とすると，グループの構築を単純に行った場合の時間計算量は $O(gn)$ で済む．オブジェクトが多数あるときは $g \ll n$ であることが期待され

るため，実質的な時間計算量は $O(n)$ である．

3.4 グループの振舞いの可視化

オブジェクトのグループ化結果を用いて，グループに含まれたオブジェクトの振舞いを明らかにする．本研究では，2 種類の可視化手法を用いた．

1 つは，各グループから代表オブジェクトをランダムに選択し，直接呼び出し関係のあるオブジェクト群との相互作用に関するイベントだけを実行履歴から抽出，シーケンス図として可視化する手法である．この手法はあるグループについて，1 つの具体的な動作例を読解したい場合に有効である．ただし，グループ内に含まれているオブジェクトがまったく同一の振舞いをしているとは限らない．適切な代表オブジェクトをグループ内から選択する手法は，今後の課題である．

もう 1 つは，Salah らが提案した，オブジェクトの各グループと周辺クラスとの呼び出し関係を，有向グラフとして可視化する手法である [2]．この有向グラフは，頂点として，オブジェクトのグループと，周辺のクラスを用いる．ある頂点のグループあるいはクラスに属するオブジェクトの内 1 つでも，ある別の頂点に属するオブジェクトへと 1 度でもメソッド呼び出しを行っていた場合に，それらの頂点間に有向辺を作成する．このグラフは，グループごとに相互作用するクラス群の差異を効果的に提示する．

3.5 クラス間の相互作用例の抽出

3.4 節の可視化手法は，あるクラスのグループごとの振舞いの差異を可視化するが，グループ間の呼び出し関係を用いて，2 つの異なるクラス c_s, c_t の間の相互作用を分析することも可能である．具体的には，グループ $G(c_s) = \{s_1, \cdots, s_m\}, G(c_t) = \{t_1, \cdots, t_n\}$ に対して，$P(s_i, t_j) = \{(o_1, o_2) | o_1 \in s_i \wedge o_2 \in t_j \wedge Reachable(o_1, o_2)\}$ を抽出する．ただし，$Reachable(o_1, o_2)$ は，実行履歴中で，少なくとも 1 回，o_1 からのメソッド呼び出しが o_2 に（直接あるいは推移的に）到達するという条件を意味する．

相互作用例の可視化には，オブジェクトのグループ $G(c_k)$ の場合と同様に，$P(s_i, t_j)$ からオブジェクトのペアをランダムに選択，シーケンス図として可視化する手法と，グループ間および関連クラスとの呼び出し関係グラフを抽出し可視化する手法を用いる．

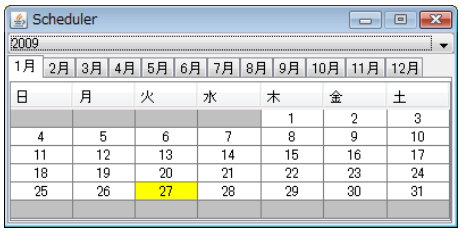

図 1: Scheduler のスクリーンショット

4 実装

本手法を *Amida-OGAN* というツールに実装した．我々のグループでは既に，Java プログラムから実行履歴を取得しシーケンス図として可視化する *Amida* を開発している [8]．実行履歴の取得には *Javassist*[9] を用いており，実行時情報を出力するコードを解析対象のプログラムに埋め込んで，プログラムを実行している．シーケンス図による可視化機能は Amida の機能をそのまま使用しており，メソッド呼び出し関係の可視化には *GraphViz*[¶1] を用いている．

5 ケーススタディ

実装した Amida-OGAN を用いて，Java プログラム *Scheduler* の動作を分析するケーススタディを実施した．

Scheduler は小規模の予定管理用カレンダープログラムである．図 1 に起動直後のウィンドウのスクリーンショットを示す．ユーザが予定を記入したい日付に対応するセルをクリックすると新たにダイアログが開き，予定を編集できる．

本ケーススタディでは，予定データの管理方法を分析するというシナリオで，実際に Scheduler を起動し，それぞれ異なる日付に計 3 つの予定を追加した．この動作で得た実行履歴には 7523 個のメソッド呼び出しイベントと，1975 個のオブジェクトが含まれていた．ただし，*java,javax,sun,com.sun,* パッケージに含まれるクラスのイベントは除外している．

Scheduler は 2 つのクラス，*DateCell* と *CalendarDate* を用いて予定の管理を行っている．DateCell はカレンダーの各日付を表現するセルの役割

[¶1] GraphViz. `http://www.graphviz.org/`

表 1: グループ $G_c(DateCell) = \{s_1, s_2, s_3, s_4\}$

Group ID	#Instances	$Use(o), Used(o)(o \in s_k)$
s_1	882	{MonthTableModel},{}
s_2	33	{MonthTableModel,DateCellRenderer},{}
s_3	90	{MonthTableModel,DateCellRenderer},{CalendarDate}
s_4	3	{MonthTableModel,DateCellRenderer,CalendarFrame},{CalendarDate}

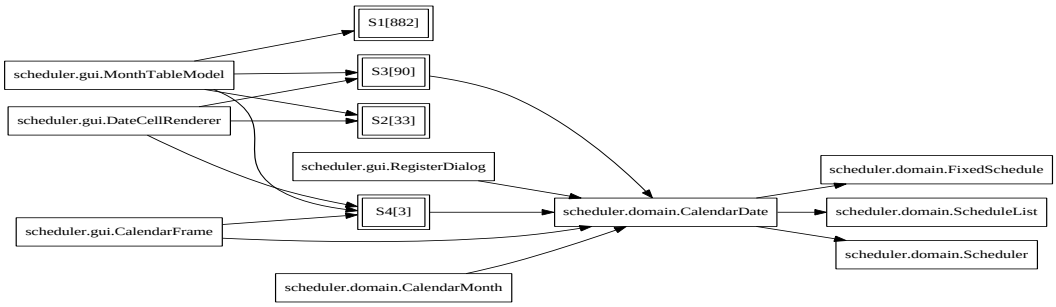

図 2: DateCell をグループ化したクラス間の呼び出し関係グラフ．$G_c(DateCell) = \{s_1, s_2, s_3, s_4\}$

を担うクラスであり．CalendarDate は各日付の予定データを保持するクラスである．取得した実行履歴中には 1008 個の DateCell オブジェクト，730 個の CalendarDate オブジェクトが出現しており，単純に可視化するには多すぎる．そのため，この 2 つのクラスに対して提案手法を適用し，動作シナリオおよび代表的な相互作用の可視化を試みた．

5.1 関連クラスに基づく DateCell の振舞いの分析

DateCell クラスの代表的な動作シナリオを可視化するために，関連クラスに基づくグループ化を適用したところ，DateCell のオブジェクトは 4 グループ $G_c(DateCell) = \{s_1, s_2, s_3, s_4\}$ に分類された．グループの関連クラスとオブジェクト数を表 1 に，グループに属するオブジェクトと他のクラスとの呼び出し関係を図 2 に示す．DateCell クラスと他のクラスの呼び出し関係は，グループ化を行わなければ一見多くのクラスと関係するように見える．しかし，グループ化を行うと，大多数のオブジェクトを含む s_1 は *MonthTableModel* オブジェクトから呼び出されているだけであり，また，CalendarDate オブジェクトに対して実際にメソッド呼び出しを行っているものは s_3 ,s_4 だけであることなど，プログラム内での使われ方がグループごとに大きく異なることがわかる．

DateCell クラスのグループ s_2, s_3, s_4 からそれぞれ代表オブジェクトを 1 つランダムに選び，シーケンス図で振舞いを可視化した．結果を図 3, 図 4, 図 5 に示す．s_1 に含まれるオブジェクトの振舞いは，システム起動時に MonthTableModel オブジェクトからコンストラクタ呼び出しを 1 回受けているだけの単純なものだったため省略した．

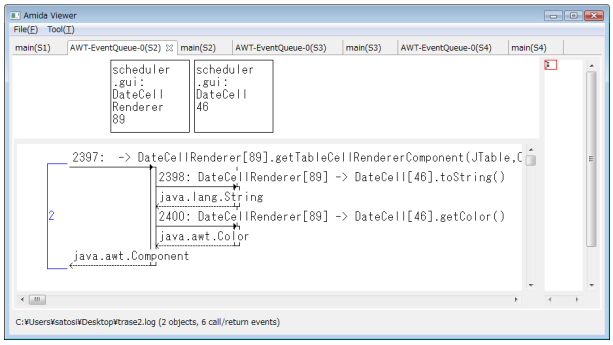

図 3: DateCell クラスのグループ s_2 に属するオブジェクトの振る舞いを可視化したシーケンス図．DateCellRenderer からの描画情報の要求に応じている．

どの図においても，*DateCellRenderer* オブジェクトからセルの描画情報を得るために呼び出されていることが確認できる．しかし，s_3, s_4 では s_2 と異なり，CalendarDate オブジェクトを更に呼び出している．また，s_4 では，ユーザのマウスクリック操作の結果，*CalendarFrame* オブジェクトからもメソッドを呼び出されていることが確認できる．これは DateCell クラスが担っているセルの性質の

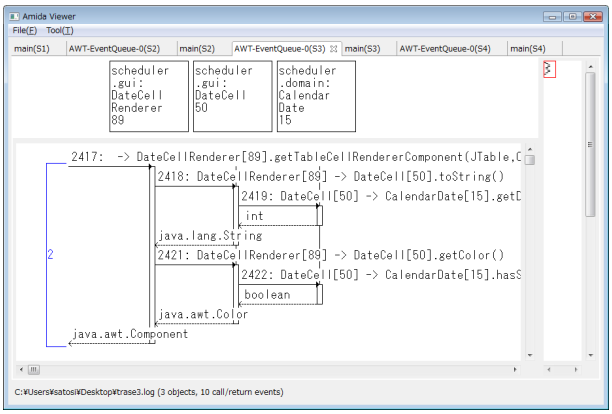

図 4: DateCell クラスのグループ s_3 に属するオブジェクトの振る舞い．DateCellRenderer からの要求に対して，CalendarDate に予定データの有無を問い合わせている．

表 2: $G_c(DateCell) = \{s_1, s_2, s_3, s_4\}$ と $G_c(CalendarDate) = \{t_1, t_2, t_3\}$ の関連

	$t_1(637)$	$t_2(90)$	$t_3(3)$
$s_1(882)$			
$s_2(33)$			
$s_3(90)$		90 組 (1 対 1)	
$s_4(3)$			3 組 (1 対 1)

違いからくる振舞いの差を示していると考えられ，実際により詳しく解析を行ったところ，s_1 はカレンダーとして画面に 1 度も表示されなかったセル，s_2 は図 1 において灰色で表示されている日付の無いセル，s_3 は日付と対応しているセル，s_4 は日付に対応していて，かつ予定が追加されたセルであった．関連するクラスに基づくグループ化の目的は，オブジェクトの性質の違いからくる振舞いの差を識別することであり，この例では目的を達成できていると考えられる．

5.2 クラス間の相互作用例の抽出

次に，DateCell クラスのグループ $\{s_1, s_2, s_3, s_4\}$ と，CalendarDate クラスを関連するクラスに基づいて分類したグループ $\{t_1, t_2, t_3\}$ との間の呼び出し関係を分析した結果を表 2 に示す．表 2 で，グループ ID 横の 内の数値はそのグループに含まれるオブジェクト数を表していて，さらにその横の括弧内は関係性を表している．両クラス間には，2 通りの組み合わせの呼び出し関係だけが存在して

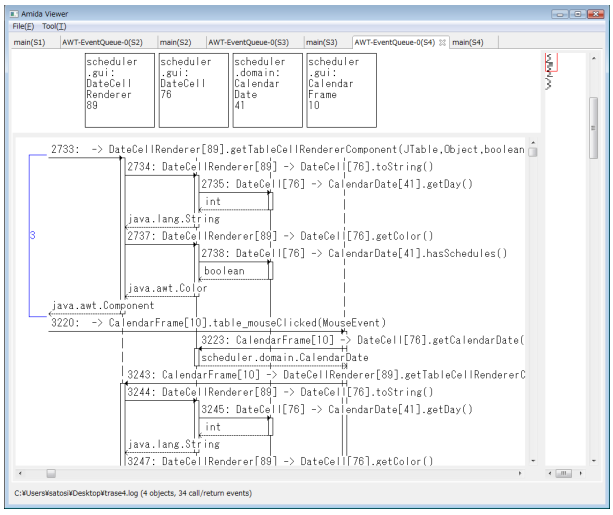

図 5: DateCell クラスのグループ s_4 に属するオブジェクトの振る舞い．ユーザのマウス操作に関するメソッド table_mouseClicked(MouseEvent) が呼び出されている．

おり，s_3 と t_2 の間に 90 組，s_4 と t_3 の間に 3 組，いずれも 1 対 1 の呼び出し関係となっていた．

これらのグループ間の呼び出し関係グラフを，図 6 に示す．CalenderFrame オブジェクトと相互作用する DateCell クラスのグループ s_4 は，予定を追加するダイアログの役割を担う *RegistorDialog* オブジェクトと相互作用する CalendarDate クラスのグループ t_3 とのみ関連していることがわかる．実際にこの組み合わせで呼び出し関係のあるオブジェクトのペアをランダムに 1 つ選び振舞いをシーケンス図に可視化したところ，図 7 に示すように，ユーザ操作により予定データを追加する振舞いが確認できた．

5.3 関連メソッドに基づくグループ化

続いて，CalendarDate クラスから呼び出される *ScheduleList* クラスを分析した．このクラスは予定データをリスト構造で保持するクラスであるが，この関連するクラスに基づくグループ化では分類できず，単一のグループとなった．しかし，関連するメソッドに基づくグループ化では表 3 に示すように，$G_m(ScheduleList) = \{u_1, u_2, u_3\}$ という結果が得られた．

それぞれの代表オブジェクトをシーケンス図に可視化すると，特に u_3 は振舞いが他のグループと

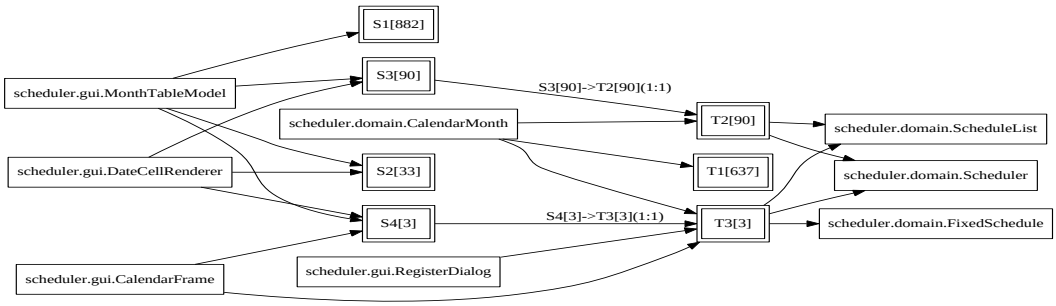

図 6: $G_c(DateCell) = \{s_1, s_2, s_3, s_4\}$ と $G_c(CalendarDate) = \{t_1, t_2, t_3\}$ 間の呼び出し関係グラフ

表 3: グループ $G_m(ScheduleList) = \{u_1, u_2, u_3\}$

Group ID	#Instances	$Methods(o), Called(o)(o \in u_k)$
u_1	90	{⟨init⟩(),iterator(),hasValids(Date)},{}
u_2	3	{⟨init⟩(),iterator(),hasValids(Date),add(Schedule)},{}
u_3	7	{⟨init⟩(),iterator(),hasValids(Date),clear()},{}

大きく異なっていた．この原因は，CarendarDate クラスのグループ $\{t_1, t_2, t_3\}$ との関連を解析することで明らかとなった．表 4 に示すように，u_1 ,u_2 は 1 対 1 で，u_3 は 1 対多で CalendarDate クラスのオブジェクトと関係しており，関係性が異なることがわかる．実際にそれぞれの組み合わせの代表ペアをシーケンス図に可視化したところ，u_1 ,u_2 は各日付ごとの予定データを管理し，u_3 は曜日ごとの予定データを管理していると判明した．このように，関連するメソッドに基づくグループ化は，同じクラス群と相互作用しているオブジェクトの振舞いの違いを識別する場合に有効である．

6 まとめと今後の課題

実行履歴の可視化により特定のクラスの機能を理解しようとするとき，そのクラスのすべてのオブジェクトの振舞いを逐一確認することは現実的ではない．本研究では，ある 1 つのクラスのオブジェクトを，関連する呼び出しの属性に基づいてグループ化し，各グループの代表オブジェクトのみを可視化することで，クラスの代表的で具体的な振舞いのみを確認する手法を提案した．

本研究ではグループの振舞いの可視化にあたって，代表オブジェクトをグループ内からランダムに選択しているが，グループ内に含まれているオブジェクトがまったく同一の振舞いをしているとは限らない．適切な代表オブジェクトをグループ内から選択する手法は今後の課題である．その他にも，より大規模なシステムに適用し，提案手法のスケーラビリティを確認したいと考えている．また，より多くのケーススタディで，グループに含まれるオブジェクトの振舞いを分析し，グループ化の妥当性を検証する必要がある．

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] Ishio, T., Watanabe, Y. and Inoue, K.: AMIDA: a Sequence Diagram Extraction Toolkit Supporting Automatic Phase Detection. Proceedings of the 30th International Conference on Software Engineering, pp.969–970, 2008.

[2] Salah, M. and Spiros, M.: A Hierarchy of Dynamic Software Views: from Object-Interactions to Feature Interacions. Proceedings of the 20th International Conference on Software Maintenance, pp.72–81, 2004.

[3] Salah, M. and Spiros, M.: Scenariographer: A Tool for Reverse Engineering Class Usage Scenarios from Method Invocation Sequences. Proceedings of the 21st International Conference on Software Maintenance, pp.155–164, 2005.

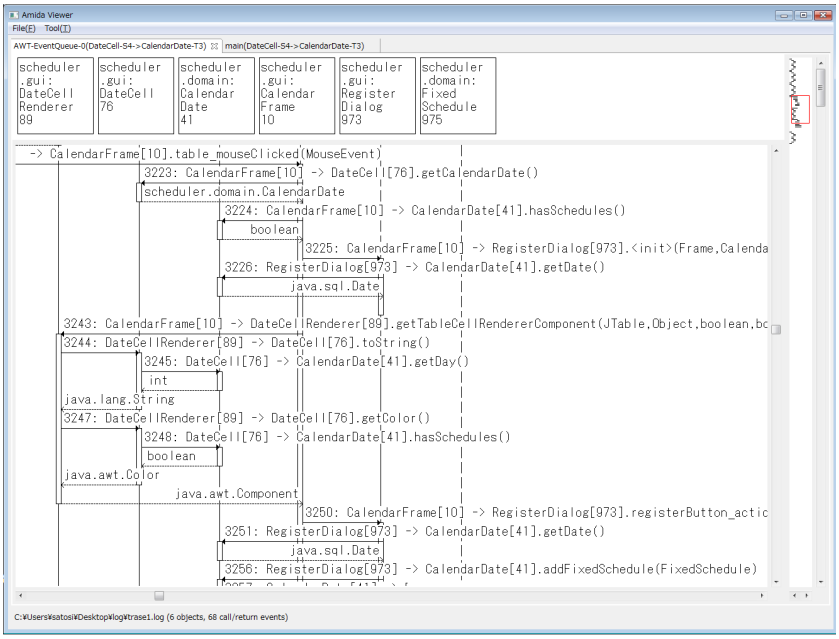

図 7: DateCell クラスのグループ s_4 と CalendarDate クラスのグループ t_3 の相互作用例を可視化したシーケンス図．ユーザのマウス操作に関するメソッド呼び出し後，DateCell から CalendarFrame を経て RegisterDialog に CalendarDate が渡されたのち，RegisterDialog により予定データが追加されている．

表 4: $G_c(CalendarDate) = \{t_1, t_2, t_3\}$ と $G_m(ScheduleList) = \{u_1, u_2, u_3\}$ の関連

	$u_1(90)$	$u_2(3)$	$u_3(7)$
$t_1(637)$			
$t_2(90)$	90 組 (1 対 1)		90 組 (多対 1)
$t_3(3)$		3 組 (1 対 1)	3 組 (多対 1)

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

[5] 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, 2002.

[6] Valentin, D., Christian, L., Andrzej, W. and Andreas, Z.: Mining Object Behavior with ADABU. Proceedings of the 4th International Workshop on Dynamic Analysis, pp.17–23, 2006.

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

[8] 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎: プログラム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, pp.153–169, 2007.

[9] 千 葉 滋: Javassist Home Page, `http://www.csg.is.titech.ac.jp/~chiba/javassist/index.html`.