Vol.2010-SE-167 No.10
2010/3/18

クラス動作シナリオ可視化手法の プログラム理解作業に対する有効性評価

宗像 聡[†1] 渡邊 結[†1]
石尾 隆[†1] 井上克郎[†1]

オブジェクト指向プログラムにおいて，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図として可視化する手法が有効である．しかし，プログラム実行時に多数のオブジェクトが存在するクラスでは，各オブジェクトの振舞いを逐一確認することは労力が大きく現実的ではない．そこで本研究では，注目するクラスのオブジェクト群を，それらの振舞いに基づいて同値分割し，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．同値分割は，観点の異なる 4 つの同値関係に基づいて行う．利用者は調査の目的に応じて，同値分割に使用する同値関係を選ぶことができる．提案手法を既存のソフトウェアに適用し，プログラム理解作業に対する有効性を検証した．

An Assesment of Visualization of Representative Class Interaction Scenarios for Program Comprehension

Satoshi Munakata ,[†1] Yui Watanabe ,[†1]
Takashi Ishio [†1] and Katsuro Inoue[†1]

In object-oriented programming, it is effective to visualize an execution trace of an instance of a class as behavior diagrams for understanding behaviors of the class. However, a class can create a large number of instances; it is hard for developers to read behavior diagrams for each instance. In this paper, we propose a method to classify instances into groups based on their behaviors, and visualize only representative behaviors of their instances. Users can select one of four categorization criteria depends on their purpose of investigation for program comprehension. As an assessment of the efficacy for program comprehension, we have applied our approach to six Java programs.

1. はじめに

クラスは，オブジェクト指向プログラムにおける基本的なソフトウェア部品であり，プログラムを適切にメンテナンスするにはクラスの動作を理解する，つまり，クラスの使われ方や，そのオブジェクトの振舞いを，十分に理解することが不可欠である[9]．クラスの動作理解には，プログラム実行時に収集した実行時情報を解析し，ある 1 つのオブジェクトの実際の振舞いを可視化する手法が有効である[2),3),5),9),11)]．

しかし，これらの手法は，プログラム実行時に多数のオブジェクトが存在するクラスでは，適用することが難しい．同じクラスのオブジェクトであっても振舞いはそれぞれに異なるため[7),8)]，可視化するオブジェクトの選び方により，獲得できる知識が変わるからである．すべてのオブジェクトの振舞いを可視化すれば，実行時情報に含まれる動作については網羅できるが，労力が大きく現実的ではない．

我々は，このようなクラスの動作理解支援を目的に，注目するクラスのオブジェクト群を，それらがプログラム実行時に相互作用したクラスやメソッドの違いに基づいて分類することで，そのクラスの代表的な振舞いのみを UML シーケンス図として可視化する手法を提案した[12)]．しかし，この手法では，利用者が振舞いを調査する時の多様な調査目的を反映することができない．また，UML シーケンス図だけでは提示された複数の振舞いの違いを十分に比較，確認することが難しかった．

そこで本研究では，以前の手法の拡張として，注目するクラスのオブジェクト群を，それらの振舞いの同値性に基づいて同値分割し，各同値類から選んだオブジェクトの振舞いをそのクラスの代表的な振舞いとして，呼び出し関係図，DOPG 図，UML シーケンス図で可視化する手法を提案する．同値分割のために，観点の異なる 4 つの同値関係を定義した．利用者は調査の目的に応じて，同値分割に使用する同値関係と可視化手法を選ぶことができる．また，各振舞いの差を強調して可視化することができる．さらに，特定の同値類に対して，異なる同値関係に基づいて再帰的に分割することで，複数の観点からオブジェクト群の振舞いの違いを調査できる．提案手法を GUI ベースの対話的ツールとして実装し，複数の既存のソフトウェアに適用することで，プログラム理解作業に対する有効性を検証した．

以降では，本手法を提案するに至った背景を説明し，次に，3 章で本手法について，4 章

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

で実装について，5 章で適用実験について述べる．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[10]．クラスの動作理解を目的に，その特定オブジェクトの振舞いを可視化する手法は，これまでに数多く提案されている[2),3),5),9),11)]．

Quante らは，1 オブジェクトについての制御フローグラフである DOPG(Dynamic Object Process Graph) を提案した[5]．DOPG は，実行履歴から生成したプログラム全体の制御フローグラフを，注目オブジェクトについてスライスすることで得られる．DOPG は，ソースコード位置と対応する 8 種類の頂点 (Read，Write，Call，Entry，Return，Start，Final，Condition) と，制御フローに対応する 3 種類の辺 (Conditional，Unconditional，Invocation) から構成される．

Lange らは，オブジェクトと他のクラスの呼び出し関係は，有向グラフとして可視化することで直観的に理解できることを示した[4]．このような有向グラフを可視化したものを，呼び出し関係図と呼ぶ．頂点としてオブジェクトとクラスを用いた場合．ある頂点のオブジェクト，あるいは頂点のクラスに属するオブジェクトの 1 つ以上が，ある別の頂点に属するオブジェクトにメソッド呼び出しを行っていた場合に，この頂点間に有向辺を作成する．

2.2 多数のオブジェクトが存在するクラスの理解

実行履歴解析により単一オブジェクトの振舞いを可視化することでクラスの動作理解を支援する手法は，注目クラスのオブジェクトがプログラム実行時に多数存在する場合，単純に適用することはできない．同じクラスのオブジェクトであってもその使われ方や振舞いは同一であるとは限らないため[7),8)]，オブジェクトが複数ある場合，可視化するオブジェクトの選び方によって，獲得できる知識が変わる．そのため，振舞いの違いを認識してオブジェクトを選ぶ方法を，ユーザに提供する必要があると考える．

Salah らは，メソッド呼び出し系列の文字列表現をオブジェクトごとに生成し，編集距離の近い文字列表現を簡潔に表現できる正規表現を求めることで，クラスのより典型的な使用例を抽出する手法を提案した[8]．クラスの使用例は，そのクラスを再利用する時や，使い方の検証を行う場合には有効だが，本研究のように，プログラム理解を目的としてクラス内部の動作および実装を調査する際には，情報が不足している．しかし，振舞いに基づいてオブジェクトを分類する考え方は，本研究でも踏襲している．

Richner らは，プログラムに関する知識の少ないユーザがプログラムの実行時情報を分析するには，そのスケーラビリティから，分析作業を対話的に支援する必要があると主張している[7]．本研究でも，この主張を踏襲している．

3. 提 案 手 法

本研究では，オブジェクトの振舞いに対して同値関係を定義し，オブジェクト群の同値類それぞれから 1 つずつオブジェクトを選び，それらの振舞いの違いを強調して可視化する手法を提案する．

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，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．同値分割に使用する情報は上記のみであるが，提案手法と組み合わせて使用する可視化手法のために，メソッド呼び出しが発生したスレッドの識別子，メソッド呼び出しの実行順序 (タイムスタンプ) の情報も必要になる．

3.2 オブジェクトの振舞いの同値分割

オブジェクトの振舞いの同値関係を定義するために，まず，4 つの異なる観点に基づく，オブジェクトの動作コンテキストを定義する．あるオブジェクトの動作コンテキスト集合とは，プログラム実行時にそのオブジェクトが相互作用したメソッドやクラスからなる集合である．動作コンテキスト集合として，次の 4 つを定義する．なお，コンストラクタ呼び出しは，メソッド名が定義クラス名であるメソッド呼び出しとして扱う．

本研究では，次の 4 つの動作コンテキスト集合を定義した．

- $Use(o)$: 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 (図 1(a)).
- $Used(o)$: オブジェクト o に，メソッド呼び出しされたクラスの集合 (図 1(b)).
- $Methods(o)$: 動作した，オブジェクト o のメソッドの集合 (図 1(c)).
- $Called(o)$: オブジェクト o が呼び出したメソッドの集合 (図 1(d)).

4 つの動作コンテキスト集合に対応する，4 つの同値関係を定義する．

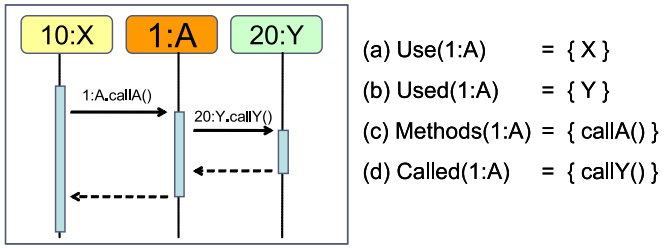

図 1 動作コンテキスト

3.2.1 $E_{use}(o_s, o_k)$

2 つのオブジェクト o_s, o_k に対する同値関係 $E_{use}(o_s, o_k)$ を，次の式で定義する．

$$E_{use}(o_s, o_k) \leftrightarrow Use(o_s) = Use(o_k)$$

クラスはプログラム中でそれぞれに異なる役割を担っており，複数の役割の異なるクラスが協調動作することでシステムの機能は実現される[1),6)]．そのため，呼び出し元のクラスの集合が異なる 2 つのオブジェクトでは，それぞれ異なる機能から使われていた可能性があり，同値関係 $E_{use}(o_s, o_k)$ は，このような参加した機能の違いによる振舞いの差を識別する．

3.2.2 $E_{used}(o_s, o_k)$

2 つのオブジェクト o_s, o_k に対する同値関係 $E_{used}(o_s, o_k)$ を，次の式で定義する．

$$E_{used}(o_s, o_k) \leftrightarrow Used(o_s) = Used(o_k)$$

呼び出し先のクラスの集合が異なる 2 つのオブジェクトでは，それぞれ異なる機能を実現した可能性があり，同値関係 $E_{used}(o_s, o_k)$ はこのような実現した機能の違いによる振舞いの差を識別する．

3.2.3 $E_{methods}(o_s, o_k)$

2 つのオブジェクト o_s, o_k に対する同値関係 $E_{methods}(o_s, o_k)$ を，次の式で定義する．

$$E_{methods}(o_s, o_k) \leftrightarrow Methods(o_s) = Methods(o_k)$$

動作したメソッドの集合が異なる 2 つのオブジェクトは，他のオブジェクトからそれぞれ異なる使い方をされた可能性があり，同値関係 $E_{methods}(o_s, o_k)$ は，このような使われ方の違いによる振舞いの差を識別する．

3.2.4 $E_{called}(o_s, o_k)$

2 つのオブジェクト o_s, o_k に対する同値関係 $E_{called}(o_s, o_k)$ を，次の式で定義する．

$$E_{called}(o_s, o_k) \leftrightarrow Called(o_s) = Called(o_k)$$

表 1 可視化手法

可視化手法	Use	$Used$	$Methods$	$Called$	抽象度
呼び出し関係図	○	○	×	×	高
DOPG 図	△	△	○	△	中
UML シーケンス図	○	○	○	○	低

○は違いを完全に，△は条件的に，×は全く識別できないことを表している．

呼び出したメソッドが異なる 2 つのオブジェクトは，それぞれ異なる実装を処理に利用した可能性があり，同値関係 $E_{called}(o_s, o_k)$ は，このような利用した実装の違いによる振舞いの差を識別する．

3.2.5 再帰的な同値分割

ある同値分割から得た同値類について，異なる同値分割を再帰的に適用することができる．例えば，同値関係 $E_{use}(o_s, o_k)$ に基づくオブジェクト群 O の同値分割 $G_{use}(O) = \{G_1, \cdots, G_l\}$ から得た同値類 $G_i \in G_{use}(O)$ について，さらに同値関係 $E_{methods}$ により同値分割した場合，同値分割 $G_{use,methods}(G_i) = \{G_{i1}, \cdots, G_{im}\}$ が得られる．この分割で得た同値類は，含まれるすべてのオブジェクトで同値関係 $E_{use}, E_{methods}$ が成り立つ．

3.3 振舞いの可視化

オブジェクト群の同値分割により得られる同値類から 1 つずつオブジェクトを選び出し，呼び出し関係図，DOPG 図，あるいは UML シーケンス図として可視化する．この時，同値類ごとの振舞いの違いを強調して可視化する．表 1 に示すように各可視化手法はそれぞれ観点や抽象度が異なり，識別できる振舞いの特徴もそれぞれに異なる．そのため，クラス理解の目的に合わせて，可視化手法と適用する同値関係を選ぶことになる．

4. 実　　装

提案手法をツール Amida-OGAN として実装した．メイン画面を図 2 に示す．GUI は，注目クラスの選択やそのオブジェクト群の分割を行う Explorer，選択した同値類の詳細を表示する Metrics View，選択した同値類の各動作コンテキスト集合を表示する Feature View，可視化方法を選択する Visualize View から構成されている．利用者は GUI を通して分析対象クラス，分割基準，可視化方法を適宜選択することで，対話的にクラスの動作を分析する．実装はすべて Java 言語で行い，コード行数はコメントを含めて約 44000 行であった．

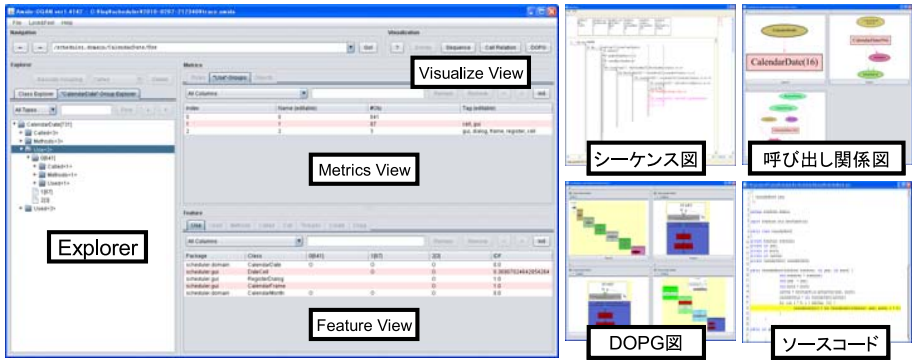

図 2 Amida-OGAN メイン画面

5. 適用実験

提案手法のクラス動作理解に対する有効性を検証するために提案手法を，6 つのオープンソースソフトウェアに適用した．表 2 に，対象ソフトウェアと，各ソフトウェアから取得した実行履歴について，含まれるメソッド呼び出しイベントの個数，オブジェクトが存在するクラスの個数 (括弧内は，オブジェクトが 2 つ以上存在するクラスの個数)，出現したオブジェクトの個数を，それぞれ示す，ただし，java，javax，sun，com.sun パッケージに含まれるクラスについては，そのオブジェクトとメソッド呼び出しイベントを実行履歴から除外している．表 2 から，プログラム実行時にオブジェクトが複数存在するクラスの割合は，全クラスに対して約 24〜71%(平均で約 36%) と，無視できない個数あることがわかる．提案手法は，これらのクラスの動作理解を支援するものである．

適用実験の目的は，次の質問に答えることである．

(1) 可視化するオブジェクトの選び方により，獲得できる知識が変わる場合はあるか？
(2) 振舞いに基づくオブジェクト群の同値分割により，代表的な振舞いを提示できるか？
(3) 各クラスのオブジェクト群は，同値分割により幾つの同値類に分割されるか？

(1)(2) のために，特定クラスの動作を分析するケーススタディを実施した．また，(3) のために，各ソフトウェアのクラスごとの同値分割結果を調査した．

5.1 ケーススタディ:Scheduler

Scheduler は小規模の予定管理用カレンダープログラムである．ユーザが予定を記入した

表 2 分析対象

Software	#Events	#Classes	#Objects
Scheduler	8578	(12)17	1973
JHotDraw	128,796	(97)185	6,286
Apache PMD	633,309	(67)105	8,431
MASU	3,198,125	(80)309	12,414
Apache FOP	3,657,813	(120)502	27,342
Antlr	7,305,687	(42)60	21,022
計	14,932,308	(418)1,178	77,468

い日付に対応するセル (日付セル) をクリックすると，新たに予定登録用のダイアログが開き，予定を編集できる

本ケーススタディでは，GUI と予定データとの相互作用を分析するために，クラス CalendarDate について動作理解を試みる．分析のために，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際の実行履歴を取得した．この実行履歴には 8578 個のメソッド呼び出しイベントと，1973 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．

取得した実行履歴には，クラス CalendarDate のオブジェクトが 731 個出現していた．これらの振舞いをすべて確認することは，労力が大きく現実的ではない．そこで，これら 731 個のオブジェクトに対して，それぞれが参加した機能の違いによる振舞いの差を識別するために，同値関係 E_{use} に基づいて同値分割を行ったところ，3 つの同値類 $\{s_1, s_2, s_3\}$ に分割された．このとき，s_1 に含まれるオブジェクトは 641 個，s_1 に含まれるオブジェクトは 87 個，s_1 に含まれるオブジェクトは 3 個であった．この分割では，同じ同値類に属するオブジェクトは，呼び出し元のクラスの集合が完全一致する．そこで，このような同値類ごとの振舞いの差を確認するため，各同値類の振舞いを呼び出し関係図として可視化した．結果を図 3 に示す．図 3 より，大多数のオブジェクトを含む s_1 では CalendarMonth オブジェクトから呼び出されているだけであることなど，同値類ごとに振舞いが大きく異なることがわかる．詳細にみると，同値類 s_2 ，s_3 では，図 3 の破線で示した箇所に，共通する呼び出し関係が確認できる．同値類 s_3 では，図 4 より，予定データ登録時に用いたダイアログの役割を持つ RegisterDialog オブジェクトから呼び出されていることが確認でき，予定データの追加機能に参加したと予測できる．

より詳細に振舞いを確認するために，クラス CalendarDate の各同値類 s_1, s_2, s_3 の振舞いを，それぞれ DOPG 図で可視化した. スレッド 1 での振舞いは，DOPG 図で確認する

Vol.2010-SE-167 No.10
2010/3/18

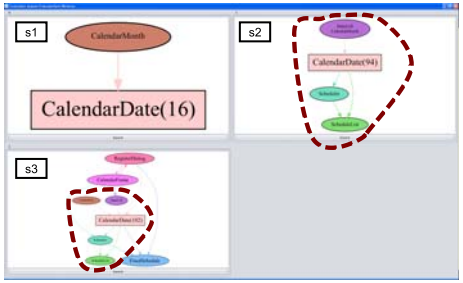

図 3 CalendarDate の各同値類 $\{s_1, s_2, s_3\}$ についての呼び出し関係図．同値類 s_2 ，s_3 では，破線で示した箇所に，共通する呼び出し関係が確認できる．

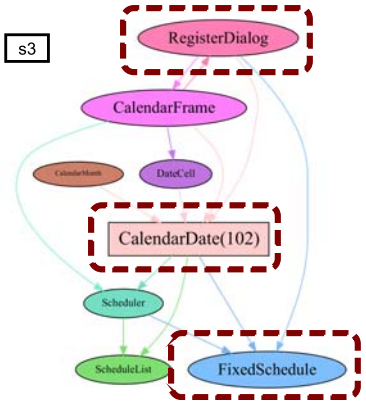

図 4 同値類 s_3 についての呼び出し関係図．登録ダイアグラムと予定データを CalendarDate オブジェクトが仲介している様子が確認でき，予定追加機能に参加したと予測できる．

限り完全に同じ呼び出し系列・同じソースコードから実現されており，CalendarMonth オブジェクトのコンストラクタ内の処理で，各 CalendarDate オブジェクトは生成されていた．スレッド 18 の振舞いは，同値類 s_2 と s_3 で異なっていた．

同値類 s_2 についての DOPG 図から，DateCellRenderer オブジェクトからの描画要求に応えるために，DateCell オブジェクトが CalendarDate オブジェクトに予定データの状態を問い合わせていることがわかった．この振舞いをより詳細に確認するために，同値類 s_2 の振舞いを UML シーケンス図として可視化した．このシーケンス図から，同値類 s_2 は，各日付セルの描画処理に関わったオブジェクト群だとわかった．

同値類 s_3 についての DOPG 図には，同値類 s_2 の振舞いに存在した描画処理に関係する振舞いが含まれていた．これは，図 3 で確認したように，同値類 s_2, s_3 の呼び出し関係図に共通する箇所があることに対応する．また，同値類 s_3 についての DOPG 図の別の部分からは，ユーザのマウス操作を受けて，RegisterDialog オブジェクトから予定データの追加を要求されていることがわかった．この振舞いをより詳細に確認するために，UML シーケンス図として可視化した．このシーケンス図から，同値類 s_3 は，予定データ追加処理に関わったオブジェクト群だとわかった．

ここまでの分析で，クラス CalendarDate について，次のことがわかった．

- 各日付セルは描画に用いる色情報を生成するために，CalendarDate に予定データの状態を問い合わせる．それを受けて，CalendarDate は予定データの状態として，当日の予定データと曜日ごとの予定データの状態を，それぞれ調べる．
- 登録ダイアログは追加ボタンが押されたあと，CalendarDate に予定データの追加要求をし，それを受けて，CalendarDate は予定データベースに予定データを追加する．

より詳細にプログラムを分析したところ，同値類 s_1 はプログラム実行中に表示されなかった日付のデータ，s_2 は表示された日付のデータで描画機能に参加したもの，s_3 は表示された後に予定が追加されたデータで描画機能と予定追加機能に参加したものにそれぞれ対応していることがわかった．同値関係 E_{use} に基づく同値分割の目的は，参加した機能の違いからくる振舞いの差を識別することであるため，このケーススタディでは，目的を達成できていると考えられる．

ケーススタディの考察

本ケーススタディで注目したクラス CalendarDate では，大多数のオブジェクトは GUI と予定データに関わっていなかった．そのため，すべてのオブジェクトからランダムにオブジェクトを 1 つ選択し可視化しただけでは，描画処理と予定追加処理についての知識は獲得できないと考えられる．また，すべての振舞いを確認することは，労力が大きく困難である．クラス CalendarDate の 731 個のオブジェクトを，その振舞いの同値性に基づいて同値分割することで，初期化，描画処理，予定追加処理が含まれる振舞いのみをそれぞれ提示することができた．

クラス CalendarDate のすべてのオブジェクトについて呼び出し関係図を生成し，同型判定をすることで，互いに同型ではない図は全部で 3 つであることがわかった．同型な図を複数読解しても，新たに得られる知識は少ないと考えられる．本ケーススタディでは，各同値類 s_1, s_2, s_3 から可視化した 3 つの呼び出し関係図は，この 3 つの図を網羅していた．そのため，呼び出し関係図を可視化手法に用いた場合には，クラス CalendarDate の理解に効果的な振舞いのみを可視化できたと言える．さらに，可視化した 3 個のオブジェクトを除く 728 個のオブジェクトの振舞いについては，詳細な分析を省略することで分析時間が短縮され，効果的に理解を進めることができたと言える．

しかし，クラス CalendarDate のすべてのオブジェクトの DOPG を解析すると，互いに同型でない DOPG は 6 つであることがわかった．これは (1) 実行スレッドの違い，(2)2 階層以上の呼び出し階層の違いからくるものだった．実行スレッドや推移的な呼び出しの違いからくる振舞いの差は，本手法では識別できない．クラス動作理解において，オブジェクト

の振舞いとして，このような振舞いの差を識別すべきかどうかについて，今後議論を深める必要がある．

ケーススタディ:MASU

ソフトウェア MASU に対しても，同様のケーススタディを行った．結果を簡潔に示す．

MASU は，Java，C++，C#プログラムを静的解析し，プログラムの各種メトリクスを計測するツールである．クラス UnresolvedMethodIndo は AST 解析時にメソッド情報を保持する役割を担っているが，その 84 個のオブジェクトは，$E_{methods}$ で 4 同値類に分割された．それぞれ，戻り値型がプリミティブ型に設定されたもの，引数の情報が設定されたものなど，AST 解析器からの使われ方の違いに対応していた．また，この分割では，全部で 4 つある互いに同型ではない DOPG 図のすべてが提示された．

5.2 分割結果の調査

オブジェクト群を同値分割することで，クラスの動作理解に効果的な，一部のオブジェクトの振舞いのみを提示することができる．しかし，一度に多数の同値類に分割された場合は，提示されるすべての振舞いを確認する労力が増大する．著者らは，ケーススタディやツールの作成にあたり複数人の被験者に使用してもらった時の知見から，一度に効果的に確認できる振舞いの数は，9 以下だと推測している．9 つより多くの振舞いを図として可視化した場合には，各図を一画面に並べると図のサイズが小さくなりすぎ，相互に比較して振舞いの差を確認することが難しくなる．そこで，表 2 に示す 6 つのソフトウェアに対して，同値分割により適度な個数の同値類に分割されるクラスがどの程度存在するのか調査した．

表 3 に，各実行履歴に含まれているクラスごとに，そのオブジェクト群を同値分割した結果を示す．表 3 中の，“呼び出し関係図”，“DOPG 図” 列は，呼び出し関係図，DOPG 図の完全一致に基づいて同値分割した時の同値類数を，それぞれ示している．なお，呼び出し関係図，DOPG 図の完全一致判定は，それぞれをラベル付き有向グラフとみなした時に，グラフが同型かどうかで判断した．表 3 から，全クラスの同値類の総数は，全クラスのオブジェクトの総数に対して約 0.5〜8.4%(平均で約 2.4%) と，かなり少ないことがわかる．1 クラスあたりの同値類数は約 1.4〜2.0 個 (平均で約 1.6 個) となり，一度の同値分割で得られる同値類の個数は，全体的には比較的少数であることがわかる．

また，表 3 から，呼び出し関係図，DOPG 図の完全一致に基づく分割では，各同値分割で得られる同値類数に比べて，より多くの同値類が存在することがわかる．本手法は，利用者が調査したい振舞いの特徴に応じて，各オブジェクトについての呼び出し関係図，DOPG 図を探索するものだとみなせる．しかし，定義した 4 つの同値関係を再帰的に適用するだけでは識別できない図の違いは存在する．これらの図の違いを識別し探索できることが，クラスの動作理解に必要であるかについて，今後検証する必要があると考えている．

表 3 分析結果

Software	$\#E_{use}$	$\#E_{Used}$	$\#E_{methods}$	$\#E_{called}$	#呼び出し関係図	#DOPG 図
Scheduler	23	22	26	22	25	41
JHotDraw	522	272	265	261	528	1,657
Apache PMD	162	193	136	145	261	516
MASU	583	440	421	419	632	1,188
Apache FOP	821	611	680	639	875	1,928
Antlr	180	96	257	172	204	6,091
計	2,291	1,634	1,785	1,658	2,525	11,421

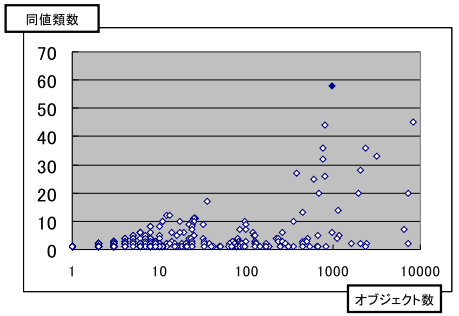

図 5 同値関係 E_{use} に基づく同値分割結果.

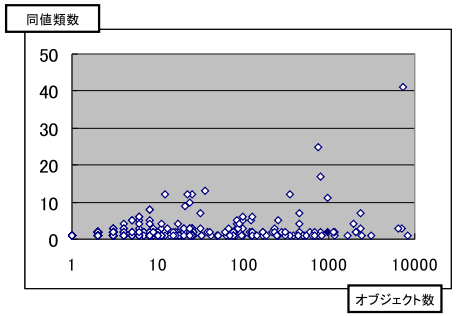

図 6 同値関係 E_{used} に基づく同値分割結果.

5.2.1 同値類が少数のクラス

図 5,6,7,8 に，各同値関係 $E_{use}, E_{used}, E_{methods}, E_{called}$ に基づいて，実行履歴に含まれる各クラスを同値分割した結果を，それぞれグラフで示す．図 5,6,7,8 のグラフは，縦軸が同値類数，横軸がオブジェクト数を表していて，各クラスについて，そのオブジェクト群を同値分割した時の同値類数とオブジェクト数に対応する点をプロットしている．図 5,6,7,8 から，大多数のクラスでは，各同値分割で得られる同値類は 9 個以下であることがわかる．同値関係 E_{use} に基づく同値分割では，同値類が 9 個以下であるクラスの割合は全クラスに対して約 97%，オブジェクトが 50 個以上存在するクラスに対して約 82%，E_{used} に基づく同値分割では，割合はそれぞれ約 99%と約 95%，$E_{methods}$ に基づく同値分割ではそれぞれ約 99%と約 87%，E_{called} に基づく同値分割ではそれぞれ約 99%と約 94%であった．前述のように，これらのクラスは提案手法により効果的に動作理解を支援できるものと考えられ

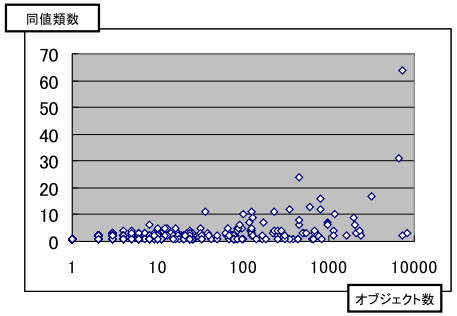

図 7 同値関係 $E_{methods}$ に基づく同値分割結果.

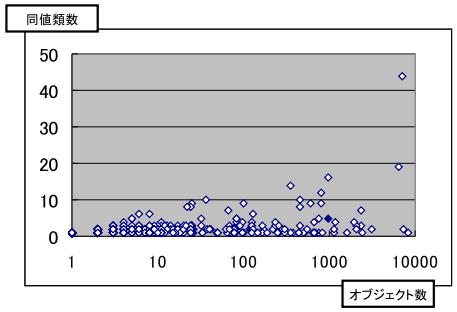

図 8 同値関係 E_{called} に基づく同値分割結果.

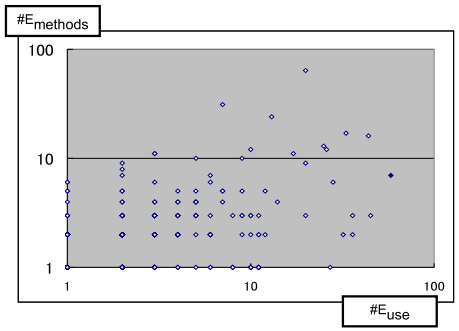

図 9 同値関係 E_{use} と $E_{methods}$ の同値類数の相関

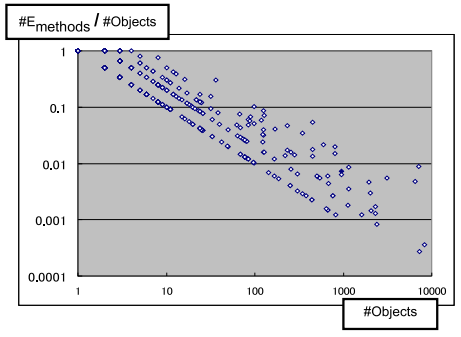

図 10 各クラスの同値関係 $E_{methods}$ に基づく分割数とオブジェクト数の割合.

る．特に，オブジェクト数が 50 以上であっても同値類数が少数であるクラスは，本研究の目的に強く合致する．

このように，大半のクラスで，生成されるオブジェクト数によらず同値類数が一定になるのは，各クラスが行う処理のバリエーションは実際には限られていることを示唆している．つまり，ソースコード上では，実現可能性のあるオブジェクトの振舞いのバリエーションは多数あるが，実際のプログラムの実行では，少数のパターンに従うのではないかと推測される．

5.2.2 同値類が多数のクラス

一方，同値分割が効果的に作用しないクラスも複数あることがわかる．これらのクラスは，(a) 多数のオブジェクトが存在し多数の同値類に分割されているクラスと，(b) 少数のオブジェクトが存在しオブジェクト数に近い数の同値類に分割されているクラスに大別される．

(a) について，プログラム実行時にオブジェクトが多数生成されるクラスは，MASU では StateChangeEvent や NamespaceInfo など，JHotDraw では ResourceBundleUtil や AttributeAction など，データオブジェクト，イベントオブジェクト，イベントハンドラ，ユーティリティの役割を持つクラスである傾向が高かった．これらのクラスは，プログラム内の多くのクラスから横断的に使用される傾向がある．そのため，呼び出し元のクラスの集合の完全一致に基づく同値関係 E_{use} による同値分割で，多数の同値類に分割されたと考えられる．実際に，MASU のクラス StateChangeEvent の 8344 個のオブジェクトは E_{use} に基づく同値分割で，45 個の同値類に分割された．これらの同値類の振舞いの違いを確認したところ，行っている処理は小さく単純であるものの，同値類ごとに相互作用するクラス群が異なっていることがわかった．ソースコード上で，1 つのオブジェクトと相互作用するクラス群を特定することが困難であるため，本手法で，実行時に実際に行われた相互作用を確認することが有効である．しかし，前述のように，多数の同値類に分割される場合には，分析の労力が大きい．その場合，複数の同値関係を用いて，特定の同値類を段階的に分割することができる．

図 9 に，実行履歴に含まれる各クラスを，同値関係 E_{use} に基づいて同値分割した時の同値類数と，$E_{methods}$ に基づいて同値分割した時の同値類数との，相関をグラフで示す．図 9 は，横軸が E_{use} に基づく分割の同値類数，縦軸が $E_{methods}$ に基づく分割の同値類数を表している．図 9 から，同値関係 E_{use} と $E_{methods}$ で，分割で得られる同値類の個数の多寡は，必ずしも一致していないことがわかる．これは他の同値関係の組み合わせでも同様である．一方の分割で得られる同値類の個数が，もう一方に比べて少ないとき，前者の分割を先に適用することで，段階的に調査したい振舞いのオブジェクトを探索できる．クラス StateChangeEvent の 8344 個のオブジェクトは，動作したメソッド集合の完全一致に基づく同値関係 $E_{methods}$ による同値分割により，3 個の同値類に分割される．この同値類の振舞いを DOPG 図や UML シーケンス図で可視化することで，それぞれにプログラム内での使われ方が違うことがわかる．この内，利用者が調査したい振舞いの同値類について E_{use} で再帰的に分割することで，目的の相互作用例を得ることができると考えられる．

このように，図 9 のグラフで，対角線から離れた位置のプロットに対応するクラスでは，効果的な探索手順が存在する可能性がある．ただし，対角線付近のプロットに対応するクラスは，現在のところ提案手法では効果的に探索することはできない．

Vol.2010-SE-167 No.10
2010/3/18

図 10 に，実行履歴に含まれる各クラスを，同値関係 $E_{methods}$ に基づいて同値分割した時の，そのクラスのオブジェクト数に対する同値類数の割合 (同値類数 ÷ オブジェクト数) を，グラフで示す．図 10 は，横軸がクラスのオブジェクト数，縦軸がクラスごとのオブジェクト数に対する同値類数の割合を表している．(b) について，図 10 から，オブジェクトが少ないクラスでは，オブジェクト数と同値類数が近いもの (縦軸の値が 1 に近いもの) は多くあることがわかる．このようなクラスでは，すべてのオブジェクトの振舞いを調査する必要がある．

6. おわりに

プログラム実行時にオブジェクトが多数生成されるクラスでは，各オブジェクトの振舞いを確認する労力が増大し，クラスの動作理解が困難になる．そこで，注目クラスのオブジェクト群を振舞いの同値性に基づいて同値分割する手法を提案した．同値類ごとに振舞いを可視化することで，そのクラスの代表的な振舞いのみを図として可視化することができる．また，各振舞いの違いを強調して可視化することで，効果的にクラスの動作を分析できる．提案手法をツールとして実装し，6 つのオープンソースソフトウェアに対して適用実験を行った．

2 つの適用実験から得られた知見をまとめる．

(1) ケーススタディから，可視化するオブジェクトの選び方により，獲得できる知識が変わる場合が実際にあることを確認した．

(2) Scheduler のケーススタディでは，3 つの機能に参加する可能性があるクラスについて，その 3 つの機能に関連する振舞いのみを提示できた．他のケーススタディでも，各クラスに特徴的な振舞いを提示できた．

(3) 分割結果の調査から，オブジェクトが 50 個以上あるクラスの内，約 84〜95%と大多数のクラスで，各分割数は 9 以下と 1 画面で比較，確認できる数であった．

以上のことから，本手法は，多くのクラスの動作理解に有効である．

今後の課題として，本文で述べたもの以外に，他の分割基準の検証，実際のメンテナンス作業に対する有効性の評価を行う必要があると考えている．

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) G.Arevalo, F.Buchli, and O.Nierstrasz. Detecting implicit collaboration patterns. In *the 11th Working Conference on Reverse Engineering*, pp. 122–131, 2004.
2) V.Dallmeier, C.Lindig, A.Wasylkowski, and A.Zeller. Mining object behavior with adabu. In *the 4th International Workshop on Dynamic Analysis*, pp. 17–24, 2006.
3) T. Gschwind and J. Oberleitner. Improving dynamic data analysis with aspect-oriented programming. In *the 7th European Conference on Software Maintenance and Reengineering*, pp. 259–268, 2003.
4) D.B. Lange and Y.Nakamura. Interactive visualization of design patterns can help in framework understanding. In *ACM SIGPLAN Notices*, Vol.30, pp. 342–357, 1995.
5) J.Quante and R.Koschke. Dynamic object process graphs. In *Journal of Systems and Software*, Vol.81, pp. 481–501, 2008.
6) T.Richner and S.Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In *the 15th International Conference on Software Maintenance*, pp. 13–22, 1999.
7) T. Richner and S. Ducasse. Using dynamic information for the iterative recovery of collaborations and roles. In *the 18th International Conference on Software Maintenance*, pp. 34–43, 2002.
8) M.Salah, T.Denton, S.Mancoridis, A.Shokoufandeh, and F.I. Vokolos. Scenariographer: A tool for reverse engineering class usage scenarios from method invocation sequences. In *the 21st International Conference on Software Maintenance*, pp. 155–164, 2005.
9) T.Systa. Understanding the behavior of java programs. In *the 7th Working Conference on Reverse Engineering*, pp. 214–233, 2000.
10) N. Wilde and R. Huitt. Maintenance support for object-oriented programs. In *IEEE Transactions on Software Engineering*, Vol.18, pp. 1038–1044, 1992.
11) T. Xie and D. Notkin. Automatic extraction of sliced object state machines for component interfaces. In *the 3rd Workshop on Specification and Verification of Component-Based Systems*, pp. 39–46, 2004.
12) 宗像, 石尾, 井上. 類似する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. 情報処理学会研究報告 第 163 回ソフトウェア工学研究発表会, 第 31 巻, pp. 225–232, 2009.