オブジェクト指向プログラムの実行履歴上のフェイズに対応する機能の推定手法の提案

Approach of Extrapolating a Future of a Phase from an Execution Trace of an Object Oriented Program

渡邊 結* 石尾 隆† 井上 克郎‡

あらまし
実行履歴の可視化はオブジェクト指向プログラムの動作理解に有効であるが，調査したい機能の実行範囲を実行履歴から特定する際に発生するコストがその活用を妨げている．そこで本研究では，実行履歴上の区間 “フェイズ” ごとに機能を反映する語群を提示することで実行された機能を推定する手法を提案する．

1 動的解析と可視化によるプログラム理解支援

オブジェクト指向言語で記述されたプログラムの動作理解には，ソースコードではなくプログラムの実行時の実際の動作を実行履歴として記録し，実行履歴から実行時のオブジェクトの振舞いを開発者が読み解きやすい形式で可視化するアプローチが有効である [1].

1つの実行履歴は通常複数の機能の実行を連続して記録したものであり，実行された全てのイベントの列で構成される．そのため，プログラムのある機能の動作を理解したいとき，開発者は該当する実行履歴中から興味のある機能の実行に該当するイベント列を特定する必要がある．この作業は対象プログラムに関する知識が必要であり，加えて1つの実行履歴を1つの図として表現すると，膨大な量のイベント・オブジェクトを全て表示するために，開発者の読解に耐えられないほど巨大化してしまうため，手動で実行全体の概要を読解する作業は負担が大きい．一方，オブジェクトの振る舞いが実行時に動的に決定されることから自動化も困難である．

2 プログラムの機能と実行履歴上のフェイズ

そこで本研究では，与えられた実行履歴から実行された各機能に対応する区間 “フェイズ” を自動で検出し，それぞれのフェイズで実行された機能を反映する情報を端的に提示することで，開発者が関心のある機能の実行に対応するイベント系列を特定する作業コストを軽減する．

フェイズとは実行履歴上から切り出された連続するイベント列であり，入出力処理やデータベースアクセスなど，開発者にとって意味のある処理に対応するものを指す [2].

これまでの研究において，実行履歴から各機能の実行に対応する区間をフェイズとして自動検出する手法を提案，実装し，適用実験として複数の企業アプリケーションに適用した結果，開発者が手動で特定した各機能の実行区間と本手法で自動検出したフェイズが高い適合率を示すことを確認している [3].

3 フェイズの機能推定アプローチ

提案手法では与えられた実行履歴上の個々のフェイズに含まれるイベント群とその差異に着目し，フェイズ中で発生したイベントのシグネチャから抽出した語句の

*Yui Watanabe, 大阪大学大学院情報科学研究科
†Takashi Ishio, 大阪大学大学院情報科学研究科
‡Katsuro Inoue, 大阪大学大学院情報科学研究科

うち出現頻度に基づく重要度において上位のものを提示する．

我々の研究グループの調査では，1つの実行履歴上のイベントシグネチャに含まれるメソッド名・クラス名等の語句のうち，履歴内での出現頻度が上位のものがその実行履歴を取得したプログラムの特徴や実行した機能を反映していた．そこで提案手法では，フェイズ内で Call-Return されたメソッドのうち，フェイズ内での出現頻度が上位のメソッド名を提示する．プログラムの実行時には，たとえ同じクラスのインスタンスであっても，異なる機能で生成されたオブジェクトは異なる振る舞いをする．それらの振る舞いの違いはそのオブジェクトに対する操作であるメソッド呼び出しの差異としてとらえることができると考えられる．従って，フェイズに含まれるイベント群のシグネチャにおいて，メソッド名の部分にそのフェイズに対応する機能を反映した語が含まれることが予想される．また，このとき実行履歴中に含まれる他のフェイズの比較を行うことで，履歴内で実行された複数の機能で登場するメソッド名を候補から除外する．実行履歴上の複数の機能で呼び出されるメソッドは，たとえあるフェイズ内での出現頻度が高くても1つの機能ではなくプログラム全体の特徴を表すメソッド名を持つ可能性が高いためである．

事前実験として，異なる複数の機能を実行した実行履歴に対して，各フェイズに登場するメソッド名の出現頻度を用いてランク付けを行った．まず，単純に出現回数のみを比較したところ，上位のメソッド名にはプログラムのドメインやフレームワークなどプログラム全体の特徴を反映した語を含むものや，逆にライブラリ操作など実装の詳細を反映した語を含むものが多く，また複数のフェイズで共通のメソッド名が上位に表れていたため，この結果はフェイズを比較・選択するために不十分であった．次に，実行履歴中の複数のフェイズで呼び出されるメソッド名を除外したところ，それぞれのフェイズで異なるメソッド名が上位に表れ，またそれらはそのフェイズに対応する機能を実行した際に行われる操作や参照・変更される対象データを表す語を含んでおり，これらの情報からフェイズを比較・選択することができると考えられる．しかしこの単純なフィルタリングでは，あるテーブルに対するデータの追加と削除など同一のデータに対して異なる処理を行う機能に対応するフェイズ間で互いの機能の対象データ名を反映するメソッド名を打ち消しあってしまう場合があることも判明した．そこで，提案手法では，1つの実行履歴に対して含まれる各フェイズを1文書としてメソッド名の出現頻度と逆出現頻度を用いた TF-IDF によるランク付けを採用することで，各フェイズの機能を反映した語句を提示するものとした．

4　今後の展開

今後，提案手法を我々の開発している Java プログラムの実行履歴解析ツール Amida [4] に実装し，手法自体の正確性と計算コストの評価とプログラム理解に対する実効性を確認するための実験を行う予定である．

謝辞　本研究は，文部科学省グローバルＣＯＥプログラム（研究拠点形成費）の補助によるものである．ここに記して謝意を表す．

参考文献

[1] Briand, L.C., Labiche, Y. and Leduc, J.: Towards the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Transactions on Software Engineering, Vol.32, No.9, pp.642-663 (2006).

[2] Reiss, S. P.: Dynamic Detection and Visualization of Software Phases. Proc of International Workshop on Dynamic Analysis, pp.50-55 (2005).

[3] Watanabe, Y., Ishio, T. and Inoue, K.: Feature-level Phase Detection for Execution Trace Using Object Cache. Proc. of International Workshop on Dynamic Analysis, pp.8-14 (2008).

[4] Amida: a Sequence Diagram Extraction Toolkit for Java *http://sel.ist.osaka-u.ac.jp/ ishio/amida/*