フレームワーク誤用による副作用の可視化手法

久米 出[†1] 中村匡秀[†2]
新田直也[†3] 柴山悦哉[†4]

フレームワーク・アプリケーションの重要な問題の一つとして、利用規則が明示化されていないフレームワークの誤用が挙げられる。本論文では誤用によって発生した副作用のデバッグ支援を目的とした新しい可視化手法と実用的なアプリケーションへの適用例を紹介する。

A Visualization Technique for Side Effects Caused by Framework Misuses

Izuru Kume,[†1] Masahide Nakamura ,[†2] Naoya NItta[†3]
and Etsuya Shibayama[†4]

Framework misuse in absence of explicit rules of use is a serious problem in building applications on frameworks. In this paper, we propose a novel visualization technique for debugging side effects caused by framework misuses. We also introduce an example to apply our method to a practical framework application.

1. はじめに

アプリケーションフレームワークはある領域のアプリケーションに共通する処理の骨組の設計と実装を提供する。一般にフレームワークアプリケーションはフレームワークとそれを拡張した**アプリケーション固有部分**より構成される。前者に属するクラスとそのメソッドを**フレームワーククラス**と**フレームワークメソッド**、後者に属するクラスとそのメソッドを**アプリケーション固有クラス**と**アプリケーション固有メソッド**と呼ぶ事にする。

フレームワークを利用するためには**ホットスポット**と呼ばれるフレームワーククラスを正しく拡張し、かつアプリケーション固有メソッドからフレームワークメソッドを正しい方法で呼び出す必要がある。こうした規則はしばしば文書化されていないため、フレームワークの誤用、特にフレームワークメソッドの呼出しに関する誤りが大きな問題となっている[1)]。

我々は先行研究[2)] でフレームワークの誤用による副作用に焦点を当てたデバッグ支援手法を提案した。本手法は誤用に起因する「予期せぬ」副作用を示唆する**兆候**の抽出と、その可視化によってデバッグの効率化の実現を目指している。本手法は Java 言語で記述されたフレームワークアプリケーションを対象としている。

†1 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
†2 神戸大学, Kobe University
†3 甲南大学, Konan University
†4 東京大学, The University of Tokyo

2. 関連研究

一般にデバッグ作業では制御とデータの依存関係を逆に辿る事によって、発生した障害からその原因となった命令文を特定するという、多大な労力を要する作業が必要とされる。こうした作業を支援する既存の代表的な手法として、Zhang 等によるエラー生成の寄与の格付け手法[3)] や、GUI を用いて依存関係を効率的に辿るツール[4)] が挙げられる。

これらの手法を適用するためにはデバッグ作業者がデバッグ作業時に出現するエラーを特定する必要がある。しかしながらフレームワークの実装の知識を持たない利用者にとってはこうした特定は困難である。

3. 副作用の兆候検出と可視化

本手法は障害が発生した実行トレースから**誤用による副作用の可能性のある挙動**を抽出し、それらを障害発生の文脈と関連付ける形で配置する (図 2)。抽出される挙動を**兆候** *(symptom)* と呼ぶ。兆候は図 1 に示すように *Hidden Update*、*Aliasing* 及び *Outdated State* に分類される。*Hidden Update* はあるオブジェ

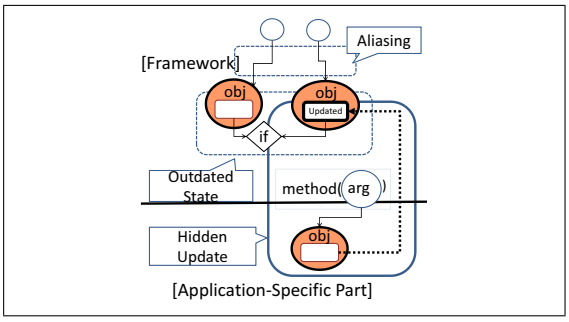

図 1 副作用の兆候
Fig. 1 Side Effect Symptoms

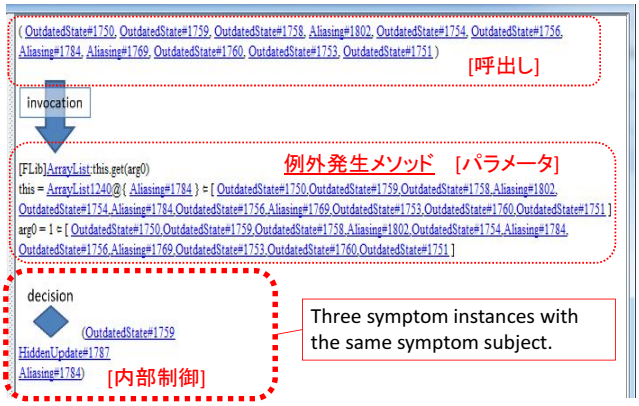

図 2 兆候の配置
Fig. 2 Layout of Symptoms

クト (図 1 の obj) に対するフレームワークとアプリケーション固有部分の**境界を越えた**状態更新とその参照を表現する。メソッドの呼び出し元の実行は obj の状態に依存するが、その状態は呼び出し元から見えない形で変更されている。

Hidden Update がエラーを生成しているか否かはその状態変更がメソッドの呼び出し元にとって予期せぬものであるか否かに依存する。*Aliasing* と *Outdated State* はこの状態変更が予期せぬものである事を示唆する副次的な挙動を表現する。

Aliasing はオブジェクトに対する複数の異なる参照経路の存在を意味する。このような参照経路の存在は予期せぬ状態変更の遠因である。*Outdated State* は同一オブジェクトの新旧の状態の参照結果を同時に利用して条件分岐を行う挙動を表現する。例えばあるリストが既に空になっているにもかかわらず、過去に参照したリストのサイズを信用し、その内容を取得しようとして現在のサイズが 0 であるために例外が発生するような状況は *Outdated State* の一例として挙げられる。

本手法では発生例外に対してまず例外が発生したメソッドに対する (1) 呼び出し、(2) 呼び出しのパラメータの計算、(3) メソッド内部で例外の発生を決定した制御の流れ、のそれぞれに関与した兆候を配置する。作業者は表示された兆候を実現するコードやその呼び出し文脈、さらにそこで参照されるオブジェクトを特定出来る。

本手法を第三者が開発した実用的なフレームワークアプリケーションの不具合に適用した結果を図 2 に示す。赤線で囲まれているのは説明のために付けられた注釈である。この事例ではこれらの情報とフレームワークの API から図 1 で示すような境界を越えた状態変更と参照が実行されている事が特定され、フレームワークの API 情報より、それが誤ったメソッド呼び出しの内部で発生している事が判明した。本手法によってこの不具合を引き起した誤用による副作用を効率的に特定するに至った。

4. おわりに

我々はフレームワークの誤用に起因する副作用の兆候を実行トレースから抽出、可視化する事によってデバッグを支援する手法を提案し、実用的なアプリケーションに適用した事例を紹介した。本ワークショップでは可視化の改良点や評価法に関する意見を頂ければ幸いである。

謝辞 本研究は JSPS 科研費挑戦的萌芽 (No.23650016)、基盤 (C)(No.24500079)、基盤 (B)(No.23300009) の助成を受けたものです。

参 考 文 献

1) Monperrus, M., Bruch, M. and Mezini, M.: Detecting Missing Method Calls in Object-Oriented Software, *ECOOP*, pp.2–25 (2010).
2) Kume, I., Nitta, N., Nakamura, M. and Shibayama, E.: A Dynamic Analysis Technique to Extract Symptoms That Suggest Side Effects in Framework Applications, *Symposium On Applied Computing*, ACM (2014).
3) Zhang, X., Gupta and Gupta, R.: Pruning Dynamic Slices with Confidence, *Conference on Programming language design and implementation*, ACM, pp.169–180 (2006).
4) Ko, A. and Myer, B.: Debugging reinvented: asking and answering why and why not questions about program behavior, *International Conference on Software Engineering*, IEEE, pp.301–310 (2008).