作業者の知識を効率的に活用する全知デバッガの設計

久米 出[†1] 中村匡秀[†2]
波多野賢治[†3] 柴山悦哉[†4]

オブジェクト指向プログラムの障害には不備 (設計の誤り) や予期せぬ副作用に由来するものが少なくない。我々は現在こうした障害の診断を支援する全知デバッガ (omniscient debugger) を開発している。本デバッガには作業者が登録した知識に合わせて障害の発生過程を抽象化する機能が実装される予定である。本稿ではデバッガの概要を紹介すると共に、登録された知識を更新するための作業者とデバッガ間の対話に関する議論を提起する。

Designing an Omniscient Debugger that Leverages Maintainers' Knowledge Efficiently

Izuru Kume,[†1] Masahide Nakamura ,[†2] Kenji Hatano[†3] and Etsuya Shibayama[†4]

Failures of object-oriented programs are often caused by flaws (design errors) and unexpected side effects. We are developing an omniscient debugger that aims to support maintainers' diagnosis of such failures. Our debugger is to implement a novel feature to abstract failure processes according to registered maintainers' knowledge. In this paper, we introduce the overview of our debugger, and discuss the interaction with its users in order to update the registered knowledge.

1. はじめに

実用的なオブジェクト指向プログラムの開発では不備 (設計上の誤り: flaw) や予期せぬ副作用が障害の原因である事が珍しくない。我々が知る限り、こうした種類の欠陥の特定を自動化する事は困難であり、現状では人手によるデバッグが唯一の現実的な解決方法である。しかしながらオブジェクト指向プログラミング固有の問題[1] がその解消を困難にしている。我々はこうした障害の**診断** [⋆1] を支援するデバッガを開発している。我々はプログラム実行をメソッド呼び出しを抽象化した計算木として定義する。診断は計算木の各節点 (メソッド呼び出し) に対する作業者による判定 (正、不正、或いは不備) として定式化される。我々のデバッガは作業者の判定支援を目的とする。

本デバッガは作業者自身の知識に基づいて障害の発生過程を抽象化する新しい機能を備える。加えて事前に登録された実行時の異常 (anomaly) を含む特徴を検出して作業者の注意を喚起する機能も実装される。作業者が自身の知識を適切に登録する事によって、診断時の根拠となる情報の取得と調査すべき節点の特定の効率化の実現が期待される。

2. デバッガの概要

我々はオブジェクト指向プログラムの実行を、計算木 (computation tree)[3),4)] として形式化する。各節点にはメソッドの返り値と実行時に於けるオブジェクトの参照・変更情報が付与される。作業者は付与された情報に基づいて正、不正、不備を判定する。我々のデバッガは障害の発生過程を抽象化する事によって、判定すべき節点の特定と、判定に必要な情報の取得を支援する。作業者は自身の有する知識をデバッガに登録する事によってこの抽象化を調整出来る。

本デバッガは二種類の知識の登録を受け付ける。一つ目はソフトウェア・アーキテクチャである。もう一つはプログラム実行の調査や理解の手掛かりとして利用されるオブジェクトやクラス、そしてそれらに対す

†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 同志社大学, Doshisha University
†4 東京大学, The University of Tokyo
⋆1 文献[2)] に依れば診断とは障害の発生に関して、観測可能かつ妥当性のある予測を提供する仮説である。

る操作 [⋆1] である。適切なオブジェクトを選択し、その操作を観察する事によってプログラム理解やデバッグが効率化される事が知られている[5)–7)]。我々はこれらの要素の観察は診断の判定に対しても有用であると考えている。このようなデバッグ作業に於ける調査、理解、判定の手掛かりとして用いられるオブジェクトとそれに関する情報を本稿では**観測点**と呼ぶ。観測点は作業者の選択に依存し、属人性が高い情報である。

本デバッガは登録情報に従って、観測点同士の因果関係を説明する最小限の情報だけを含むようにメソッド呼び出しとオブジェクトの参照に関する構造を濃縮する。デバッガには予め一般的な異常 (anomaly) を含み、パターン化された特徴が登録されており、観測点の関与が自動的に検査される。適合するパターンへの関与が特定された場合、それらは作業者の注意を喚起する**標識**として表示される。作業者は濃縮された実行内容と標識を参考にしてメソッド呼び出しを判定し、或いは判定材料を得るために調査すべき箇所を特定する。

過去の全ての実行に対するパターンの適合検査を実現するために、本デバッガは全知デバッガ (omniscient debugger)[8)] として実装される。またシステムがフレームワークのアプリケーションとして構築されている場合にはフレームワークの誤用を示唆する**兆候**[9)] が標識に含められる。

3. デバッガと利用者の対話

診断作業の進展に伴い、判定対象メソッドも変更される。同じメソッドに対しても判定を決定する視点の方が変わるかもしれない。このように作業者の側の関心が変遷する場合には、それと連動する形で新たな観測点が円滑に再登録される事が、作業の効率性の面から望ましい。デバッガが観測点候補を作業者に呈示する機能が実装されていれば、判定作業の効率化に大きく貢献すると思われる。

判定対象の切り替えをデバッガの側で認識出来るように作業者とデバッガの対話を設計する事は可能であろう。さらに観測点候補を標識と関連付ける事によって、作業者の操作に伴う形で自然に観測点候補が呈示出来ると考えている。将来の課題として、過去の対話の履歴を解析して潜在的な観測点を推測し、候補として作業者に呈示する機能を実現したいと考えている。

⋆1 メソッド呼び出しとインスタンス変数 (クラス変数) への代入と参照

4. おわりに

本ワークショップでは、具体的なデバッグ作業事例を示しながらユーザインタフェイスと対話の設計、及び対話内容の変遷を説明する予定である。抽象化された実行の表示や、デバッガとの対話のタイミング等、ユーザインタフェイスと対話に関する事柄について何でも議論を頂ければ幸いである。

謝辞 この研究は栢森情報科学振興財団、科研費基盤 (A)24500352、基盤 (B)23300009 及び 26280115、基盤 (C)24500079、挑戦的萌芽 25540150 の助成を受けて遂行された。

参考文献

1) Wilde, N. and Huitt, R.: Maintenance Support for Object-Oriented Programs, *IEEE Transactions on Software Engineering*, Vol.18, No.12, pp.1038–1044 (1992).
2) Zeller, A.: *Why Programs Fail, Second Edition: A Guide to Systematic Debugging*, Morgan Kaufmann (2009).
3) Caballero, R., Hermanns, C. and Kuchen, H.: Algorithmic Debugging of Java Programs, *Electronic Notes in Theoretical Computer Science*, Vol. 177, No. 0, pp. 75 – 89 (2007). Proceedings of the 15th Workshop on Functional and (Constraint) Logic Programming (WFLP 2006).
4) Shapiro, E. Y.: *Algorithmic Program DeBugging*, MIT Press, Cambridge, MA, USA (1983).
5) Ressia, J., Bergel, A. and Nierstrasz, O.: Object-Centric Debugging, *International Conference on Software Engineering*, IEEE, pp. 485–495 (2012).
6) Quante, J. and Koschke, R.: Dynamic object process graphs, *The Journal of Systems and Software*, Vol.81, No.4, pp.481–501 (2008).
7) Quante, J.: Do Dynamic Object Process Graphs Support Program Understanding? – A Controlled Experiment, *International Conference on Program Comprehension*, IEEE, pp. 73–82 (2008).
8) Pothier, G., Tanter, Éric. and Piquer, J.: Scalable Omniscient Debugging, *OOPSLA*, ACM, pp.535–552 (2007).
9) 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, pp. 1176–1178 (2014).