プログラム理解のためのコードリーディング支援ツールの提案と実装

大村 裕　渡部 卓雄

主としてフレームワークを用いて構成されたプログラムを対象とした，コードリーディング支援手法を提案する．提案手法では，抽象度の高いモジュール単位の視点とソースコードレベルの視点を繋ぐツールを用いる．具体的には，ソースコードから生成された UML クラス図およびシーケンス図上に，デバッグ実行で得られた情報をマッピングする．これにより，ソースコードレベルの実行位置と，クラス図およびシーケンス図上の対応する位置を正確に把握することが可能になる．プログラムを理解する際に，従来は抽象レベルの異なる記述間の対応関係を開発者自身が発見する必要があった．提案手法では，ツールによりこの対応関係の構築を自動化することで，異なる抽象レベル間の視点の切替を支援する．本発表では，提案手法がプログラム理解に寄与することを予備的な実験により示す．

1　はじめに

現在のソフトウェアは様々なフレームワークやライブラリを利用して開発されており，開発者はそれらの利用方法について知る必要がある．その際，フレームワークの構成要素やライブラリの内部動作等，ドキュメントやサンプルコードのみからは得ることが難しい情報が必要になるケースもある．例えば，Eclipse [1] のプラグイン開発を行う際には拡張ポイントに関するドキュメントおよびサンプルを参考に開発を進めていくことになる．しかし，実際には新規の拡張ポイントに関するドキュメントの不備や他モジュールの参照方法についてのドキュメントが存在しない場合がある．このような場合，開発者はしばしば使用するフレームワークやライブラリのソースコードを読み，利用方法を学ぶ必要がある．

開発者がソースコードを読む場合，複数の視点からソースコードを読み進めていく．複数の視点には大きく分けて 2 つ存在する．1 つはソースコードレベルの視点であり，もう 1 つはより抽象度の高いモジュール単位の視点である．ソースコードレベルの視点では CASE ツールによる型情報の表示や変数宣言場所の表示，変数・メソッドの一覧表示などの技術が寄与する．また，デバッガによるステップ実行により具体的な処理内容の把握なども挙げられる．モジュール単位の視点ではソースコードから UML [2] クラス図やシーケンス図をリバースするツールなどが寄与する．開発者はこれらのツールを利用し，複数の視点を切り替えてソースコードを読み，理解していく．

しかし，既存の技術，ツールでは個々の視点によるソースコードリーディングに貢献しているが，視点を繋ぐツールが存在しない．そのため，開発者はそれぞれ抽象レベルの異なる記述間の対応関係を開発者自身が発見する必要があった．

2　提案手法と想定効果

上記課題を解決するために，抽象レベルの異なる記述間の対応関係を自動的に構築するツールを提案する．このツールはソースコードから生成された UML クラス図およびシーケンス図上にデバッグ実行に得られた情報をマッピングして表示する．これにより，ソースコードレベルの現在の実行位置と，クラス図およびシーケンス図上の対応する位置を正確に把握することが可能になる．

Design and Implementation of a Source Code Reading Helper Tool for Program Comprehension

Yuu Ohmura, Takuo Watanabe, 東京工業大学・大学院情報理工学研究科・計算工学専攻, Dept. of Computer Science, Tokyo Institute of Technology.

提案手法によるツールの基本的な機能は次のようになる．まず，UML リバースツールにより対象ソースコードの UML クラス図，シーケンス図を得る．次にデバッグ実行によるソースコード上の現在実行位置と共に対応する UML クラス図，シーケンス図上の位置を表示する．

このツールによる効果は以下を想定している．

1. デバッグ実行時にソースコード全体に対する実行位置を見失わない
2. 具体的な実行クラスを元に UML クラス，シーケンス図をデバッグ実行を行いながら再構成できる

膨大なソースコードの実行系列を追跡していると現在の実行場所が一体どのクラスなのかを見失うことがある．しかし，このツールを使用すればソースコードとともに UML クラス図やシーケンス図を表示しているので，現在の実行場所を見失いにくくなる．

また，ソースコードリバースツールで UML クラス図を復元すると現在注目している処理部分のクラス群のみを残し，注目していないクラス群を削除した上で UML クラス図を再構成できるという利点がある．これは特に Java 等のオブジェクト指向言語における継承関係の整理に役立つ．具体的には静的解析によるソースコードリバースツールではリバース対象としたソースコードのすべてを UML クラス図に復元してしまう．これはソフトウェア全体の構造を把握するには有効であるが，特定の処理を理解したい場合には開発者が自らクラスを取捨選択する必要があった．しかし，このツールでは実際の実行系列を UML クラス図上にマッピングしていくのでデバッグ実行を行いながら，特定の処理の説明に最適な UML クラス図に修正していくことが可能となる．

本研究では上記の提案手法を Eclipse 上に実装し，予備評価を行った．

3 実装

本研究では上記提案手法の実験対象言語として Java を選択し，ツール作成のために Eclipse および Eclipse 上で動作する AmaterasUML [3] を利用した．Eclipse は The Eclipse Foundation より提供されて

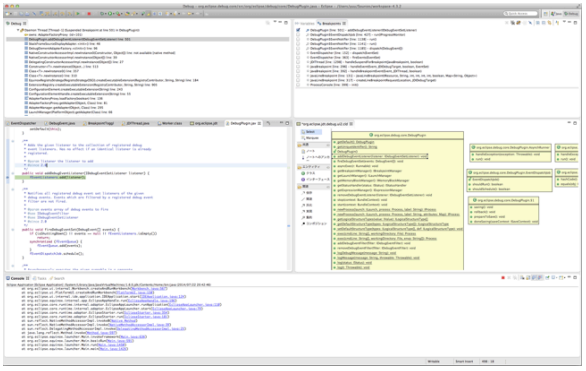

図 1　実行画面

いるオープンソースソフトウェアである．特に Java のソフトウェア開発環境として広く利用されている．また，AmaterasUML は Eclipse 上で動作する UML クラス図やシーケンス図を描画するためのプラグインであり，オープンソースソフトウェアとして提供されている．

本ツールは Eclipse の Debug 環境のソースコードを一部修正し，デバッグ実行時のステップ実行のタイミングでソースコード上の実行位置を差し示す矢印と共に AmaterasUML で作成された UML クラス図やシーケンス図上の該当個所の背景色を赤くしている．実装としては Eclipse のデバッグ実行ではステップ実行時に現在実行しているコードのクラス名やメソッド名が取得できる．この情報を元に UML クラス図やシーケンス図上の該当個所を探し，背景色を変更している．

この機能により開発者は抽象度の異なる記述間の対応関係を発見する必要はなく，ソースコードの処理内容の理解に専念できる．

4 予備評価

予備評価では開発者自身が本ツールにより Eclipse の Debug パースペクティブにおいてデバッグのステップ実行時にスタックトレース表示が実際に変更されている処理部分を確認するという事例で 3 つのポジティブな効果と 1 つのネガティブな効果について述べる．

まず，ポジティブな効果の内，デバッグ実行時の

図 2　実行イメージ 1

実行系列を追跡している場合に現在の実行場所のモジュールレベルでの全体の中での位置の把握について，非常に効果があったと考えられる．特に，1 つのファイル内に複数のクラスが存在する場合や内部クラスが存在している場合にはファイル名のみでクラスが確認できないため，UML クラス図上での位置把握は非常に役に立ったと考えられる．また，無名クラスについても一定の成果があった．無名クラスの場合，AmaterasUML のリバース機能では UML クラス図上での自動生成は行われないが，手動追加により無名クラスを UML クラス図上に追加しておき，実行時には手動追加した無名クラスの該当個所が表示されるようにすることで無名クラスにも対応可能となっている．今回の事例上では DebugPlugin.java が該当する．このファイルは 1 つの public class と 4 つの内部クラス，1 つの無名クラスから構成されている．このようなファイルの場合，本ツールの効果があったと考えられる．

また，UML リバースツールによる静的なリバースでは対象としたソースコードのすべてのクラスが UML クラス図上に表記される．しかし，実際に注目しているのはすべてのクラスではないため，大量の不要なクラスを刈り取る必要がある．それらのクラスを刈り取るには UML クラス図上から削除すればいいだけなため，簡単に行うことができる．ただし，このクラスの刈り取りは現状の手動であるため，自動的に刈り取れるとより効率的に行えると考えられる．例えば，IDebugEventSetListener インターフェースは多くのサブクラスを持つ．単純にソースコードから UML クラス図をリバースするとクラス・インターフェースが表示される．これのクラス群を実行しながら削除していけるのは本ツールの効果があったと考えられる．

最後に UML シーケンス図上への実行位置の表示も UML クラス図上での実行位置把握と同様に処理の流れ全体に対する現在の実行場所の位置の把握として効果があったと考えられる．ただし，インタフェースを介して処理が呼び出される場合，静的解析により UML シーケンス図の復元するとインタフェース部分で図の上では処理が終わってしまう．しかし，デバッグ時の情報を元に UML シーケンス図を構築すればインタフェースを介していても具体的な処理内容まで繋げることが可能となり，非常に有用であった．ただし，現在のツールではインタフェースを介した場合，UML シーケンスを繋げる部分は手動となっている．これを自動化することでより効率化が可能であると考える．例えば，Eclipse Debug パースペクティブではデバック時のイベントに対して Listener を登録しておき，イベント発火時に登録した Listener が実行される部分が多くあったが，このような場合には静的リバースではインタフェースの呼び出しまでしかリバースされない．しかし，本ツールではインタフェースで呼び出される先まで追跡することが可能となる．

次にネガティブな効果について述べる．ポジティブな効果の 2 番目とも関連するが，ソースコードを jar ファイル単位等で単純にリバースすると大量のクラスが表示される．その場合，UML クラス図の実行時の位置の背景色を赤色にする程度では探すのに時間がかかるという問題が生じた．このため，デバッグ実行時における UML クラス図，シーケンス図上での実行位置の見せ方について工夫する必要があると考えられる．

5　今後の課題

本研究では複数の異なる抽象度の記述を自動的に繋げる手法を提案し，実装した．また，予備評価として具体的な効果について考察した．

今後の課題としてはツールの改善および定量的評価がある．ツールの改善としては予備評価時に問題となった細かい修正点以外に非同期プログラムに関する対応を考慮する必要がある．Eclipse Debug パースペクティブでは非同期の処理が数多く存在する．これら

の処理を追跡する場合，デバッグ実行時のステップ実行で実行系列を追跡していても，処理を他のスレッドが引き継いでしまうため，1 回のステップ実行では最後まで処理を追跡していくことはできない．そのため，何度もターゲットプログラムのブレークポイントを設定しなおし，実行するという問題が生じた．この問題に対応するため，注目しているクラスのインスタンスレベルでマークをつけて，ターゲットクラスのインスタンスのメソッドやフィールドにアクセスがあった際に実行を止めるといった機能を実現すると非同期プログラムの実行系列を追跡する場合に非常に有効であると考える．

また，定量的評価に関しては次の評価項目についての評価を行い，開発されたツールが実際のソースコードリーディングに寄与していることを定量的に確認する．

1. ターゲットプログラムの実行回数
2. ブレークポイントの設定変更回数
3. ターゲット到達時間
4. 説明用ドキュメント作成時間

上記，想定評価項目のうちターゲットプログラムの実行回数やブレークポイントの設定変更回数については減少することを想定している．ターゲットプログラムを何度も実行したり，ブレークポイントの設定変更を行う原因としては効果の点でも挙げたモジュールレベルでの実行箇所の位置を見失ってしまうためと考えられる．本ツールを使用すればそのような原因を低減できると考えらるからである．

6 関連研究

プログラムの実行系列からプログラム理解を支援する研究として，実行系列より UML シーケンス図を作成する研究がある [6]．この研究では単純に実行系列から UML シーケンス図を構築すると非常に巨大なシーケンス図が生成されるため，[10] [7] [5] [9] では様々なシーケンス図の分割，圧縮手法を提案している．同様にプログラムの実行系列から拡張オブジェクト図を生成する研究も存在する [8]．しかし，これらの研究ではソースコードとの対応関係については言及していない．特にデバッグ実行を行いながら，異なる視点の対応関係を取るには利用者自らが考慮する必要がある。

また，動作中のシステムを対象に可視化を行った研究も存在する．[4] ではシステム (CPU，メモリ，ネットワーク) という抽象度でシステムそのものを可視化している．ただし，これは CPU やメモリといったハードウェアの利用状況の実行時可視化を行うものである．ソフトウェアの実行状況を可視化する研究としては似ているが，ソフトウェアの抽象モデルとソースコードを結び付けるという本研究とは異なる．

参 考 文 献

[1] : Eclipse, http://www.eclipse.org/home/index.php.

[2] : UML, http://www.uml.org.

[3] : AmaterasUML, http://amateras.sourceforge.jp/cgi-bin/fswiki/wiki.cgi.

[4] Stolte, C., Bosche, R., Hanrahan, P., and Rosenblum, M.: Visualizing application behavior on superscalar processors, *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, IEEE Comput. Soc, 1999, pp. 10–17.

[5] 渡邊結, 石尾隆, 井上克郎: オブジェクト指向プログラムの実行履歴に対する機能単位での自動分割手法, 電子情報通信学会技術報告. . . *SS*, ソフトウェアサイエンス, (2007).

[6] 谷口孝治, 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, (2007), pp. 1–17.

[7] 宗像聡, 石尾隆, 井上克郎: 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, 情報処理学会研究報告, (2009).

[8] 進中原, 治紫合: オブジェクト指向プログラムの動作の可視化, 情報処理学会研究報告. ソフトウェア工学研究会報告, Vol. 2010, No. 9(2010), pp. 1–8.

[9] 大平直宏, 谷口 考治, 石尾隆, 神谷年洋: 動作オブジェクト群の変化に着目したオブジェクト指向プログラムの実行履歴分割手法, 電子情報通信学会, (2005).

[10] 伊藤芳朗, 渡邊結, 石 尾隆, 井上克郎: オブジェクトの動的支配関係解析を用いたシーケンス図の縮約手法の提案, 情報処理学会研究報告 ソフトウェア, (2008).