GUI を持つプログラムの理解支援のための可視化システム

Visualization for supporting extension of GUI programs

佐藤 竜也 †, 志築 文太郎 †, 田中 二郎 †
Tatsuya SATO, Buntarou SHIZUKI, Jiro TANAKA

† 筑波大学コンピュータサイエンス専攻
Dept. of Computer Science, University of Tsukuba
{tatsuya,shizuki,jiro}@iplab.cs.tsukuba.ac.jp

GUI を持つプログラムへの機能拡張を目的とする際に、プログラムを理解する作業を支援する可視化システムを実装した。GUI は、ユーザからの操作に応じて画面を更新する特徴を有する。これを活かし、本システムは操作の度に、操作前後の GUI 画面、及び操作によって実行されたソースコードを併せて可視化する。ソースコードには、操作によって実行された部分とその呼び出し関係を強調し、拡張には理解が必須であるクラス関係と併せて表示する。これにより、開発者は操作単位でプログラムの構成を理解することが可能になる。

1 はじめに

他人が作成した GUI(Graphical User Interface) を持つプログラム (以降、GUI プログラム) に機能追加を行うには、実行の外的振る舞いと対応付けながら、関数呼び出し、クラス階層などの要素を元にプログラム構造を把握する必要がある。GUI は外的振る舞いとして、マウスやキーボードを用いた操作に応じて画面を更新するという特徴を有する。プログラムの実行部分はそれらの操作に依存するため、開発者は GUI への操作をしながら、各要素を把握する必要がある。

GUI プログラムの理解作業を効率的に行うため、デバッガや IDE などの既存ツールを利用することが考えられる。しかし、これらのツールは GUI プログラムの理解を目的としたものではないため、GUI への操作をしながらプログラムの解析を行うことができない。例えば、デバッガを使用しながら、GUI へのマウスクリック→ドラッグ→リリースという一連の操作の流れを理解することは困難である。

本研究は、これらの問題を解決するために、GUI プログラムが操作に応じた処理を行うことに注目したプログラム可視化手法を提案する。以下に提案手法の特徴を示す。

1. 操作に応じて実行されたソースコードの関係を可視化
2. 操作および GUI 画面変化と可視化結果の対応を表示

結果として、開発者は GUI への操作を単位として GUI プログラムの構造を理解することが可能となる。本稿では、提案手法、およびその手法を実装したシステムの詳細を述べる。

2 可視化要素の分析

我々は以下に示す 2 つの観点から GUI プログラムの理解作業について分析を行った。

可視化するソースコード情報

GUI プログラムに機能追加を行う場合の理解すべきソースコード中の重要な要素は以下である。

1. 各操作のプログラム全体に対する実装部分

   追加機能に関係のある実装部分を把握するためには、機能に関係する操作に応じた処理がプログラム上のどの部分で実装されているのかを特定する必要がある。開発者は特定したプログラム部分の実装方法を確認した後に、追加機能の実装を行う。

2. 関数呼び出し

   操作によって実行されるプログラムの実装部分は、複数のクラスに点在しており、それぞれが提供する関数を互いに呼び出しあっている。その呼び出しの結びつきを理解するために関数呼び出しを把握する必要がある。

3. クラス階層

GUI プログラムを構成する GUI 部品や描画対象は、継承を使って実装されることが多い。例えば、ドローツールで描画対象である図形オブジェクトを複数種類定義する場合、図形に共通する属性や動作を持つ抽象クラスを作り、このクラスを継承することで図形オブジェクトを定義する。このため、クラス階層を把握することが必要である。

GUI が有する特徴の利用

GUI はマウスやキーボードを用いた操作に応じて処理を行う。それゆえ、一つ一つのマウス操作やキーボード入力に対しては、それぞれ別々のプログラム部分が実行される。そのため、各々の操作に対するプログラム実行部分は一つのプログラム集合とみなすことができる。操作ごとの実行部分として切り分けることで、開発者は操作単位という比較的細かい粒度でプログラムを理解することが可能になる。

4. GUI との同期

操作ごとの理解を行う上では、処理を操作ごとに明確に区別する必要である。例えばマウス操作においては、マウスクリック、ドラッグ、リリースで行われる処理をそれぞれ区別しなければならない。さらに操作に応じた振舞いは、マウスの位置や画面表示に依存して変化するので、操作が行われた場面に応じて区別する必要がある。しかしそれらは操作中に時々刻々と変化するので、GUI への操作を行っている際には操作と処理との対応を取ることが困難である。そのため開発者が操作とプログラム実行部分を対応付けしやすくする工夫が必要となる。

そこで、GUI への操作の入力トリガと更新された GUI 画面を利用して、GUI とプログラム実行部分のソースコード情報との間で同期をとる。

3 可視化手法の設計

以上の分析 1〜4 を満たす可視化を示す。なお、機能追加を目的とする際にプログラムを理解するためにはソースコードを見る必要もある。そのため、提案手法はまずソースコードそのものを表示する。その際、クラス間の関係を明示するため、ソースコードを各クラス定義ごとに区切り、各クラスをクラス名でラベル付けする。その上で、以下のように可視化を行う。

1. 各操作のプログラム全体に対する実装部分

まず、プログラム全体を表現するために、プログラム上で定義されている全てのクラスのソースコードを一画面上に収まるように表示する。さらにそのソースコード上で、操作に対してプログラム実行された行だけを強調表示する。このようにソースコード全体を一画面上に表示することで、開発者は常にプログラム全体を把握しながら、操作に応じて実行された部分を見ることができる。

2. 関数呼び出し

関数呼び出しを示すために、呼び出された関数間のエッジ付けを行う。実行された行の強調表示と併せて閲覧することで、関数中あるいは関数間での実行遷移を把握することができる。図 1 に関数呼び出しの可視化の例を示す。図のように各クラスを表現し、その中で実行されたソースコード行だけを強調表示する (図中では黄色)。さらに関数間の呼び出しが発生した場合には図中の矢印のようにエッジ付けする。この図では Class1 の method() から Class2 の method() が呼び出されている様子を可視化している。

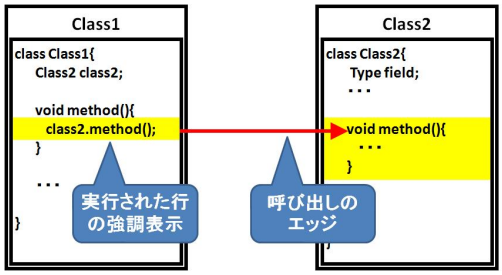

図 1: 関数呼び出しの可視化

3. クラス階層

各クラス表現を、クラスの親子関係を表すツリー構造に基づいて配置する。多重継承のようにツリーで表現しきれない構造は、クラス表現間にエッジを描いて補う。図 2 にツリー構造の表示法を示す。図左のような ClassA を基底クラスとする派生クラス群のツリーは、図右のように表示する。まず、根に表示領域の上半分を割り

当てる。次にその子ツリーに下半分の領域を横方向に等分して割り当てる。後は、割り当てられた子ツリーの表示領域に対して上記の割り当て方法を再帰的に行う。なお、ツリーが複数ある場合には、それぞれのツリーに表示領域をツリー数分だけ横方向に等分して割り当てる。

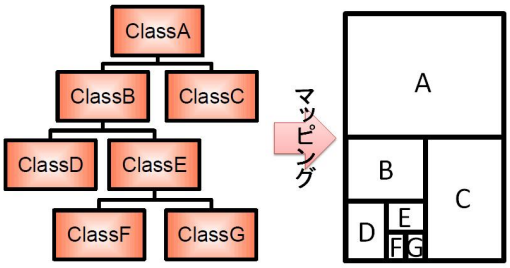

図 2: クラス階層ツリーの可視化

4. GUI との同期

開発者が対象プログラムの GUI を操作する度に、画面遷移およびイベント名をソースコードの可視化結果と併せて表示する。画面遷移は操作前後の画面のサムネイルとする。これによって可視化結果がどの操作を行った時点のものであるのかを想起しやすくする。

また、対象プログラムの GUI への操作に対してインタラクティブに反応して、実行されたプログラムのソースコードと関数呼び出しを順次可視化し、プログラムの実行に応じて徐々に更新する。この結果、操作に応じた処理が終了すると、その操作に対するグラフ表示が完成し、それ以降静的に閲覧することができる。この同期によって、プログラムへの操作と可視化が別々に行われることによる対応のしづらさを解決する。特にマウスなどによる一連の操作に対する動作を理解する際の開発者の負担を減らすことが期待できる。

複数の操作を行うとそれぞれの操作に対応した可視化結果が出力される。それらの可視化結果は後で見返したいことが多いので、履歴として保存し再閲覧可能にする。再閲覧時には、操作前後の画面のサムネイル表示を利用して、可視化結果がどの操作に対応するものであったのかを想起する。サムネイル表示は対象プログラムの起動時の画面から現在の画面までを並べて提示する。

4 GUI への操作に対応した可視化システム

設計した可視化手法に基づいたシステムの実装を行った。システムの概観を図 3 に示す。システムはコントロール部、グラフ表示部、サムネイル表示部によって構成される。

グラフ表示部ではソースコード情報を可視化する (図 3 グラフ表示部参照)。このようにソースコード全体が一画面上に収まるように縮小して表示し、実行されたソースコードは黄色で強調表示している。クラスの配置は図中に示すような形でクラス階層ごとにツリー表示する。赤線は関数呼び出しエッジを示す。ここで図中の緑色のマーカはプログラムの実行開始点を表している。このように遠目から眺めるようにプログラムの実行を見ることで、開発者は各操作のプログラム全体に対する実行部分を知ることができる。

サムネイル表示部には対象プログラムの GUI の画面遷移をサムネイルで並べて表示する (図 3 サムネイル表示部参照)。GUI への操作が行われる度に、その操作前と後での対象プログラムの GUI の画面遷移が可視化情報として保存される。サムネイルを囲っている矢印付きの枠は現在注目している画面遷移であることを示す。グラフ表示部には注目している画面遷移が発生した時点でのプログラムの実行の様子が可視化される。

開発者は本システムへの操作として、対象プログラムの制御、可視化情報の切り替え、ステップトレースを行うことができる (図 3 コントロール部参照)。対象プログラムの制御とは、対象プログラムの起動・再開、一時停止、停止である。可視化情報の切り替えを行うと、閲覧している可視化情報が前後の GUI 操作時のものに切り替わる。ステップトレースは可視化情報ごとの関数呼び出しのエッジをたどるものである。ステップを切り替えるとその時点で行われた関数呼び出しに注目したズーミング表示が行われる。ズーミング表示については次小節で説明する。

図 3 で示されるシステムの概観は図 4 のドローツールを理解対象のプログラムとして起動した場合の表示結果である。開発者は図 4 のドローツールについて理解したい場合には、ドローツールの GUI を直接操作しながら可視化を行う。

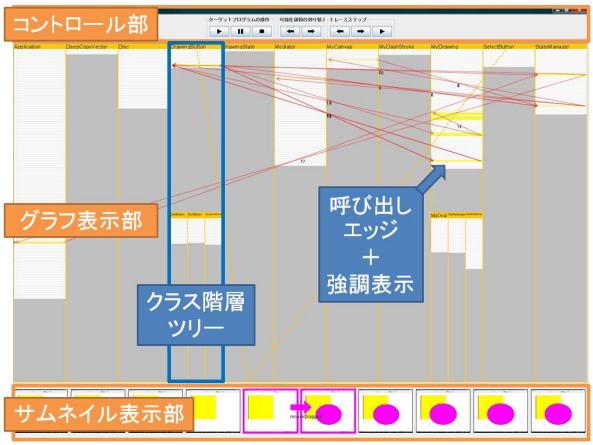

図 3: システムの概観

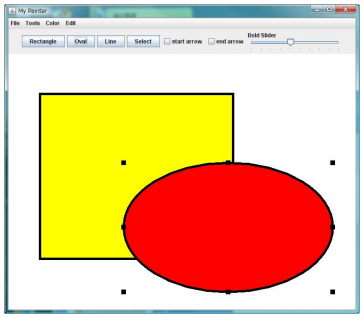

図 4: 対象とする GUI プログラム

ズーミングを用いた詳細表示

操作ごとのプログラム実行を詳細に把握するには、静的グラフを構成するエッジを順に追えばよい。

しかしソースコードが縮小表示されていると、各クラス単位でのプログラムの呼び出しは非常に閲覧しづらい。また、クラス間の関数呼び出しが複雑な場合にはさらには矢印が多くなってしまうため、呼び出し順序関係も理解しづらい。

そこで我々はズーミング機能を用いて静的グラフの注目している部分のソースコードが見えるように拡大しながら、エッジを順に追う詳細表示を実装した。注目部分は見ている時点の実行クラスとその前後のクラス間の関数呼び出しであると考え、そのクラスと関数呼び出しのエッジを拡大表示する。またそれ以外のクラスと呼び出しは注目していない情報とみなし、縮小表示する。エッジはその時点の以前と以後の呼び出しであるかがわかるようにそれぞれ別の色で描画される。

ズーミングに際して、ソースコードの全体表示やクラスの配置を損なうことは、プログラム構造の理解を妨げてしまう可能性がある。そのため、ソースコードの可視化結果の概観を維持しながら、ズーミングを行う必要がある。本システムでは、クラスは親子関係の木構造で表現されているので、木構造に対するズーミング手法の一つである Fisheye 表示を利用する [5]。Fisheye 表示は Furnas によって提案された Focus ＋ Context 手法の一つである。この手法を用いることで概観を維持しながら注目部分を拡大表示することができる。今回用いた Fisheye 表示では、重要度が高い、すなわち注目しているクラスほど、大きな表示領域を確保する。重要度はより新しいエッジに関係するクラスであるほど高いものとした。さらにそれらのクラスに親等が近いクラスにも高い重要度を割り当てるものとした。

また各クラスのソースコード行についてもズーミングを行う。1 つのクラスを表示する領域は限られているので、行数が多いクラスの可読性の低下を防ぐためである。現在は、実行されているソースコード行に近い行ほど、重要度が高いものとみなした Fisheye 表示を行っている。

ズーミング機能を用いた詳細表示を図 5 に示す。今、クラス A の関数からクラス B の関数が呼び出されている。つまりこの場面ではクラス A(図中左上) とクラス B(図中右下) が注目すべきクラスであるため、これらのクラスがズーミング表示されている。さらにクラス B の親クラスであるクラス C(図中右上) も高い重要度を持つためズーミング表示がされている。画面中央の赤線が現在の関数呼び出しエッジである。また青線は前ステップでの呼び出しエッジ、赤の点線は次ステップでの関数呼び出しエッジであることを示す。各ソースコードも注目時に実行された行の周辺のみをズーミング表示している。なおソースコードの赤色での強調表示は関数呼び出しをした行であることを示している。

図 6 は、詳細表示を順次行ったときの例である。それぞれの図中のクラス A、B、C は実行しているクラスや行に応じて、表示領域の大きさが異なっている。例えば、右上、左下の場面では、クラス A はその時点で注目している関数呼び出しには関与していない。そのため、重要度が低いクラスとみなされ、ソースコードが見えないほど縮小して表示されている。このように詳細表示では注目しているクラスや行に応じて拡大部分や拡大率が大きく異なっていることがわかる。

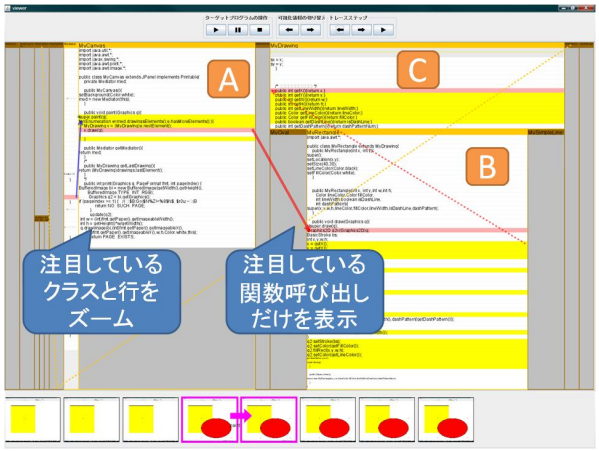

図 5: ズーミング機能を用いた詳細表示

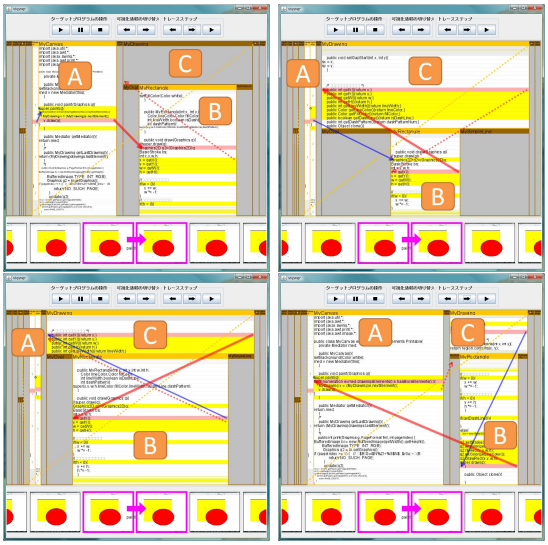

図 6: 連続した詳細表示の利用

5 システムを利用した理解の例

図 4 に示したドローツールを理解の対象としシステムを利用した場合の例を示す。このドローツールはボタンを押すことで描画する図形の種類を選択し、キャンバス内をドラッグすることで図形を描画する機能を有する。

ここでは開発者がドローツールに新たな種類の図形を描画するために描画図形モード選択ボタンとその描画機能を追加する場合を考える。その際には、開発者は既存の図形描画機能である矩形描画機能とそのモード選択ボタンについて理解し、それらの実装方法を模倣して新たな図形描画機能の追加を行えばよい。

開発者はまず対象プログラムが保存されているディレクトリを指定し、本システムにソースファイルを読み込ませる。このときソースファイルは理解が必要なものだけを用意しておけばよい。システムのグラフ表示部には読み込まれたソースファイル内で定義されているクラスが表示される。開発者はまず、このようにして静的に示されたグラフからクラス階層を確認する。

次にシステムからドローツールプログラムを起動し、ドローツールの GUI に対して矩形の描画図形モードの選択と描画操作を行う。それらの操作によって可視化された静的グラフを閲覧することで、矩形描画機能を構成する操作のプログラム全体に対する実装部分を把握する。図 7 は矩形描画機能に関する操作として、キャンバス内でのマウスクリック、ドラッグ、リリース及び矩形モード選択ボタンへのクリックをそれぞれ行った可視化結果を重ね合わせたものである。図中の青枠で囲まれた部分は上記の操作によって呼び出された実行部分である。これらの実行部分が呼び出されたクラスが矩形描画機能に関連のあるクラス (図中に赤円で印す) であることがわかる。

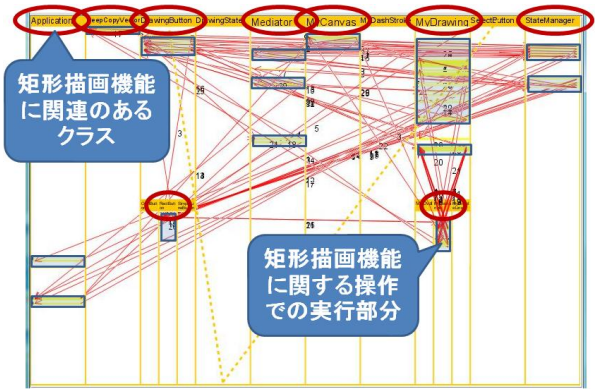

図 7: 矩形描画に関する操作による可視化結果の重ね合わせ

実際には重ねあわせた可視化結果ではなく、一つ一つの操作に対する可視化結果からこれらの情報を得る。そのため、より細かい粒度で操作に対する実行部分について解析することができる。

最後に各操作の可視化結果について詳細に分析するために、詳細表示を行いながら関数呼び出し関係を把握する。詳細表示の様子は図 5,6 に示した通りである。

開発者はこのようにして矩形描画機能の理解を行うことができる。

6 実装

本研究では Java を対象言語にした。Java プログラムの動作を解析するためには各クラスが持つメソッドやフィールド、クラス間の関連といった静的情報、およびプログラム実行中のメソッド呼び出しやフィールドの変化といった動的情報を取得する必要がある。特に動的情報は実行時に取得し、後で参照できるように保存する。動的情報を取得するために JavaVM の実行を明示的に制御し、各クラスの情報を観察することができる JDI(Java Debug Interface)API を使用した。

システムは、まずソースコードから静的なクラス構成を取得してシステムの概観を表示する。その後、JDI を用いて対象となる GUI プログラムを立ち上げる。開発者がプログラムを操作することによりイベントが発生すると、JDI によってその場で動的にプログラムの実行を解析し、実行の情報を表示に反映させる。またグラフ表示部では取得した動的情報を元に実行情報を可視化する。詳細表示のズーミング機能を実装するために Piccolo.Java1.2 を利用した。Piccolo はズーミングインタフェースの構築をサポートするツールキットである [7]。

7 関連研究

ETV はソースコード上のプログラムの実行の様子を紙芝居風に可視化するシステムである [2]。紙芝居のシートの重なり合いから、関数の呼び出しを連続的にたどることを可能にする。関数の呼び出しを可視化するという点では本研究と同じである。なお、このシステムはあらかじめ実行をトレースしておく必要がある。プログラムの実行画面自体は使用しないという点で本研究と異なる。

久永らは GUI を持つ Java プログラムの理解支援ツールを作成した [3]。プログラム上では GUI 部品のインスタンスは木構造状に配置される。その木構造を可視化するために GUI を 3D 表示した上で、関数の呼び出しを見せる。GUI の表示を利用する点、関数の呼び出しを可視化するという点では本研究と同じである。このシステムもあらかじめ実行をトレースしておく必要があるため、プログラムへの操作と可視化が別々に行われる。

SHriMP は Java のプログラム構造とソースコードとドキュメントを閲覧しながらインタラクティブにブラウジングすることができるシステムである [4]。このシステムでは静的な関係として関数呼び出しとクラス階層を見せる。またプログラムの全体構造を表示することも可能である。一方、本研究ではプログラムへの操作に対する動的な実行結果を利用した可視化を行っている。

Seesoft は、大規模なプログラムに関する様々な情報の把握を目的としたシステムである [1]。ユーザはプログラムの概略を把握するとともに、プログラムの履歴情報を獲得することができる。ソースコードの各行を色の付いた一本の線として表現し、一画面上にプログラム全体を表示するという点で本研究に似ている。モジュール間の関連を見せる点、実行時の動的情報を利用している点で本研究とは異なる。

8 まとめと今後の課題

本稿では、GUI プログラムへの操作に対するプログラム実行部分を可視化するシステムを試作した。本システムを用いることで GUI を操作しながら理解作業を行うことができ、ソースコードの実行をたどるわずらわしさを軽減することができると思われる。

我々の提案する手法の有効性を評価し、マルチスレッドプログラムへの対応や変数情報の利用による改良を重ねることで、多くのプログラムに効果的な可視化を行う手法について検討していきたい。

参考文献

[1] Stephen G. Eick, Joseph L. Steffen and Eric E. Sumner Jr. : Seesoft - A Tool For Visualizing Line Oriented Software Statics, IEEE Transactions on Software Engineering, Vol. 18, No. 11, pp. 957–968 (1992).

[2] Minoru Terada : ETV - a Program Trace Player for Students, Proceedings of ACM Special Interest Group on Computer Science Education 2005, pp. 118–122 (2005).

[3] 久永賢司, 柴山悦哉, 高橋伸：GUI プログラムの理解を支援するツールの構築, 日本ソフトウエア科学会第 17 回 (2000 年度) 大会 (2000).

[4] Margaret-Anne D. Storey, Casey Best and Jeff Michaud : SHriMP Views - An Interactive Environment for Exploring Java Programs, Proceedings of IEEE International Workshop on Program Comprehension 2001, pp. 111–112 (2001).

[5] George W. Furnas : Generalized Fisheye Views, Proceedings of ACM Special Interest Group on Computer-Human Interaction'86, pp. 16–23 (1986).

[6] JDI: http://java.sun.com/j2se/1.4/ja/docs/ja/guide/jpda/jdi/index.html/.

[7] Piccolo: http://www.cs.umd.edu/hcil/piccolo/.