データ遷移可視化ツール TFVIS における例外処理への対応

佐藤拓弥 片山徹郎
宮崎大学大学院 工学研究科 工学専攻

1 はじめに

ソフトウェア開発において、デバッグは手間のかかる工程である [1]。効率よくバグの原因を特定するためには、プログラマがプログラムの動的な挙動を把握することが重要である。しかし、プログラムの挙動は一般的に不可視であり、挙動を把握することは困難である。

プログラム実行時の挙動把握には、プログラムの動的な情報の把握が重要である [2]。プログラム実行時の挙動を解析する動的解析には、多くの手法が存在する [3]。しかしながら、これらの手法を用いても、プログラム実行時の挙動把握には手間がかかる。

そこで、我々の研究室で Java プログラムのデータ遷移可視化ツール TFVIS を開発した [4]。TFVIS は、データ遷移可視化と実行フロー可視化によって、プログラム実行時の挙動把握を支援する。TFVIS の可視化により、欠陥を含んだプログラムの実行時の挙動把握を容易にし、プログラムが含む欠陥の特定を支援する。しかし、TFVIS は Java プログラムの特徴の 1 つである例外処理に対応していない。

そこで本研究では、デバッグ支援ツールとしての実用性の向上を目的とした、データ遷移可視化ツール TFVIS における例外処理への対応を行う。具体的には、Java プログラムの特徴の 1 つである、try-catch 文への対応を行い、例外処理を含むプログラムの実行時の挙動を可視化する。

2 TFVIS

TFVIS は、解析部と可視化部から成る。また、解析部は、構造解析部、プローブファイル生成部、動的解析部から成る。

構造解析部では、プログラムの構造の解析を行い、解析結果を構造情報としてファイルに出力する。構造情報は、プローブファイル生成部でのプローブ挿入箇所の判断と、可視化部での図表の作成に用いる。構造解析部によって、ソースコードの各行で起こるイベントを取得する。イベントとは、可視化の基準となる特定の処理であり、各イベントはイベント種別の値を持つ。

プローブファイル生成部では、構造情報を基に、対象ソースコードにプローブを埋め込んだプローブファイルを生成する。プローブは、プログラム実行時の挙動の情報を出力する。また、プローブにはいくつか種類があり、各コードで起きるイベントごとに挿入するプローブが変わる。

動的解析部は、プローブファイルから、実行時の挙動を解析し、解析結果を実行情報として出力する。具体的には、プローブファイルをコンパイルし実行することで、プローブが出力する実行情報を取得する。

可視化部では、解析部が出力する構造情報と実行情報を基に、可視化を行う。

以下に、TFVIS に行った改良と改良後のデータの流れを示す。

2.1 try-catch 文への対応

- 構造解析部の新たなイベント種別の値の定義
  構造解析部において、try-catch 文に対し出力するイベント種別の値を新たに定義する。

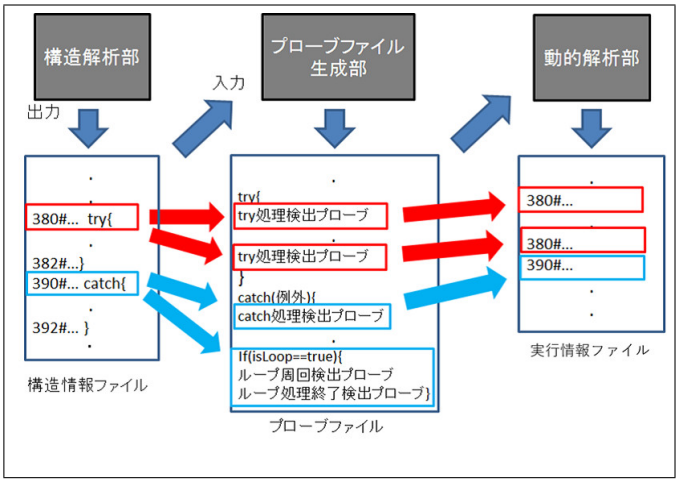

図 1: try-catch 文を含むプログラム適用後のデータの流れ

  新たに定義したイベント種別の値は、try ブロック開始の値 (380)、try ブロック終了の値 (382)、catch 節開始の値 (390)、catch 節終了の値 (392) である。
- プローブファイル生成部の try-catch 文に対する新たなプローブの定義
  プローブファイル生成部において、try-catch 文のイベントに対して挿入するプローブを新たに定義する。try ブロックのイベント用のプローブを、try 処理検出プローブとする。このプローブは、実行時のインスタンスの ID、メソッド ID、メソッド実行番号、行番号を引数とする。そして、可視化で用いる try イベント ID(380)、インスタンス ID、メソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。同様に、catch 節のイベント用のプローブを、catch 処理検出プローブとする。このプローブは、実行時のインスタンスの ID、メソッド ID、メソッド実行番号、行番号を引数とする。そして、可視化で用いる catch イベント ID(390)、インスタンス ID、メソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。
- プローブファイル生成部の try-catch 文に対するプローブの挿入
  プローブファイル生成部において、try-catch 文のイベントに応じて新たに定義したプローブの挿入を行う。try ブロックのイベントに対して、try ブロック開始から try ブロック終了までの全ての行の直前に、try 処理検出プローブを挿入する。このときプローブが得る行番号の値は、直後の行の行番号である。また、catch イベントに対して、直後の行に catch 処理検出プローブを挿入する。そして、ループ処理中の例外処理に対応するため、ループ処理中であるかを示す boolean 型 “isLoop” の定義 (“boolean isLoop=false;”) を挿入する。ループ処理中であれば “isLoop” は true に、それ以外では false になるように、プローブ生成部を変更

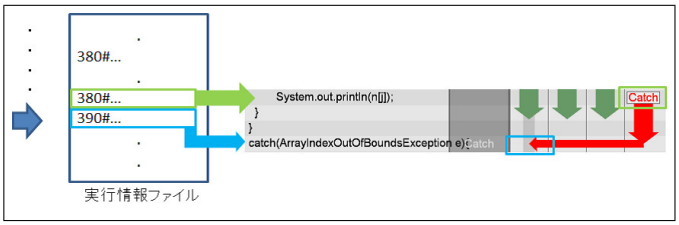

図 2: try-catch 文を含むプログラム適用後の可視化の流れ

する。そして、catch 終了のイベントが発生した際に、“isLoop”が true であれば、直前の行にループ周回検出プローブとループ処理終了検出プローブを実行するようにする。

- 可視化部の try-catch 文のイベントに対する可視化
  例外処理のイベントの発生に対し、データ遷移図の直前の実行の箇所に赤色で “Catch” と記述したボックスを配置する。また、例外処理の発生箇所と実行箇所を結ぶ赤色の矢印を記述する。

2.2 try-catch 文の処理

図 1 に、解析部における、改良後の TFVIS の try-catch 文を含むプログラム適用後のデータの流れを示す。

構造解析部の拡張によって、try ブロックの開始の行で try ブロック開始のイベント、try ブロックの終了の行で try ブロック終了のイベントを取得する。また、catch 節の開始の行で catch 節開始のイベント、catch 節の終了の行で catch 節終了のイベントを取得する。

構造解析部で取得した try-catch 文のイベントに対して、プローブファイル生成部でプローブの挿入を行う。try ブロック開始のイベント種別の値 (380) を持つ行の次の行から、try ブロック終了のイベント種別の値 (382) までのすべての行の直前に、try 処理検出プローブを挿入する。catch 節開始のイベント種別の値 (390) を持つ行の直後に、catch 処理検出プローブを挿入する。catch 節終了のイベント種別の値 (392) を持つ行の直前に、ループ中かどうかを判定するための if 文を挿入する。この if 文の中に、ループ周回検出プローブとループ処理終了検出プローブを挿入する。

プローブファイル生成部が出力するプローブファイルを、動的解析部が実行する。try 処理検出プローブは、可視化で用いる try イベント ID(380)、実行時のインスタンス ID とメソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。また、catch 処理検出プローブは、可視化で用いる case イベント ID(390)、実行時のインスタンス ID とメソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。

図 2 に、try-catch 文を含むプログラム適用後の可視化の流れを示す。可視化部で、構造情報と実行情報を基に、可視化を行う。catch イベント ID を持つ実行情報を実行情報ファイルから読み取る場合、その直前のイベントの実行を参照する。各イベントは、メソッド ID や行番号等の実行時の情報から、データ遷移図上の描画エリアを計算し、保持する。参照する直前のイベントが持つデータ遷移図上の描画エリアに、赤色で “Catch” と記述したボックスを配置する。また、例外処理の発生箇所と実行箇所を結ぶ、赤色の矢印を描画することによって、可視化を行う。

3 適用例

改良した TFVIS が正しく動作することを確認するため、例外処理を含む在庫管理プログラムを適用する。

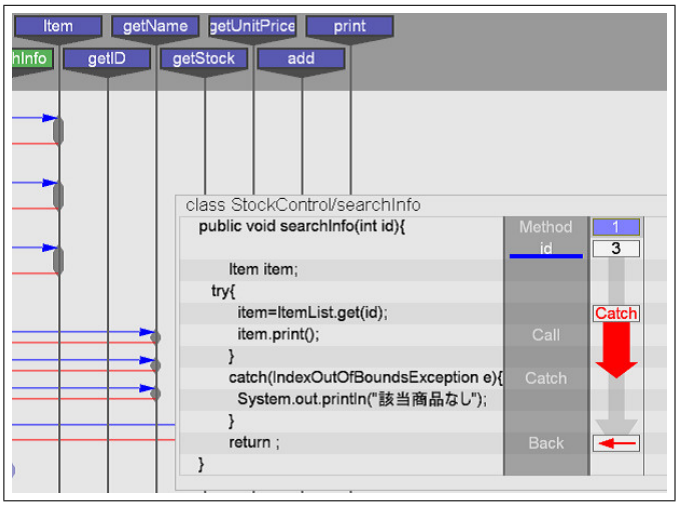

図 3: try-catch 文を含むメソッドの可視化

図 3 に、try-catch 文を含む、SearchInfo メソッドを可視化したデータ遷移図を示す。このメソッドは、「ID」の情報を入力として、その「ID」を持つインスタンスの「ID」と「名前」、「在庫数」、「単価」を表示するメソッドである。このとき、ユーザが存在しない「ID」を参照し、例外が発生するサンプルプログラムを作成した。

図 3 から、リストを参照する行に “Catch” のボックスと、このボックスから、発生した例外処理の行までの赤色の矢印を表示しており、データ遷移図で例外処理の発生を正しく可視化していることがわかる。

4 おわりに

本研究では、デバッグ支援ツールとしての実用性の向上を目的とした、TFVIS における例外処理への対応について述べた。具体的には、Java プログラムの特徴の 1 つである、try-catch 文への対応を行い、例外処理を含むプログラムの実行時の挙動を可視化した。これにより、デバッグ支援ツールとしての TFVIS の実用性が向上したと考えられる。

以下に、今後の課題を挙げる。

- 入力待ち状態の発生を含むプログラムへの対応
- 問い合わせ機能の実装
- マルチスレッドプログラムへの対応

参考文献

[1] Thomas D. LaToza, Gina Venolia, and Robert DeLine: Maintaining mental models: a study of developer work habits, Proceedings of the 28th international conference on Software engineering, pp.492-501 (2006).

[2] Jonathan Sillito, Gail C. Murphy, and Kris De Volder: Asking and Answering Questions During a Programming Change Task, IEEE Transactions on Software Engineering, Vol.34, No.4, pp.434-451 (2008).

[3] 石尾 隆: プログラムの動的解析, コンピュータソフトウェア, Vol.16, No.5, pp.78-83 (1999).

[4] 中村 紘人, その他: Java プログラム実行時のデータ遷移可視化によるデバッグ支援, 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2014, pp.125-130 (2014).