動的トレース解析プラットフォーム Polyspector™

松崎秀則[†1]　東條信明[†1]　黒田亮[†1]　肥塚真由子[†1]

これまでソフトウェアの品質を分析する材料として，詳細な実行トレース情報は十分には活用されてこなかった．クラウド技術の発達によりストレージや計算資源が安価になりつつあり，トレース情報を低コストに蓄積し解析するための環境が整いつつある．そこで我々はソフトウェア実行時のトレース情報を蓄積，解析，可視化するための統合プラットフォーム技術を開発した．本論文では，プラットフォームを利用した実例の一つであるバージョン間差分解析ツールについて紹介し，本プラットフォームの有用性を示す．

1. はじめに

これまでソフトウェアの実行トレース情報は，主に開発担当者がソフトウェアの詳細な挙動を把握し，デバグや性能チューニングを行うための道具として利用されてきた．このような用途においては，特定の動作期間のトレース情報のみが記録され，かつ作業が終了すると消去されてしまう．しかしトレース情報には内部状態遷移，処理時間，メモリアクセスパタンなど，ソフトウェア品質を分析する上で有益な情報が多く含まれており，こういった情報を効率良く抽出する技術があれば，これまでに無かった新しい品質分析が実現できる可能性がある．

そこで我々はソフトウェア実行時のトレース情報を効率良く蓄積，解析，可視化するための統合プラットフォーム Polyspector™を開発した．Polyspector™の設計においては，品質分析における多様なニーズに対応することを念頭に置き，トレース情報のフォーマットや GUI に対して様々な工夫をしている．また複数のトレース情報をまとめて解析するためのアルゴリズムについても検討を行っている．本論文では Polyspector™上で開発した解析ツールの一つであるバージョン間差分解析ツールを紹介し，その有用性とプラットフォームの持つ可能性について示したい．

2. Polyspector™

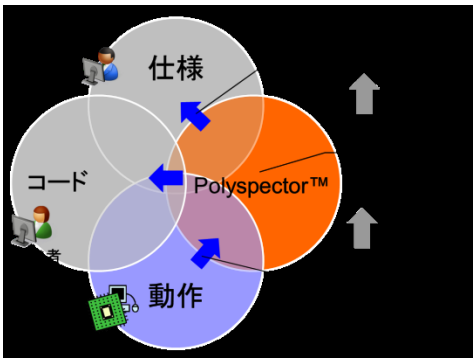

図 1　Polyspector™の概要

図 1 のように，Polyspector™はソフトウェアの実行トレース情報生成，蓄積，解析，可視化までの一連の処理をカバーする統合プラットフォームであり，現在は PC Linux 上で動作する C 言語プログラムをサポートしている．LLVM ベースの独自トレース生成モジュールを提供しており，オプションを与えて対象ソースコードをコンパイル・実行することでトレース情報を生成できる．

本プラットフォームでは，ある 1 つのトレース情報からソフトウェアの詳細な挙動を分析する既存ツール（[1]等）とは異なり，複数のトレース情報を集めてソフトウェアそのものの品質を分析・可視化することに主眼を置いている．たとえば，同一のテストを新旧バージョンのソフトウェアで実行した際のトレース情報から，新バージョンのソフトウェアが旧バージョンに対してどの程度品質が向上(劣化)しているのかを分析する，といった応用を念頭に置いたプラットフォームである．

3. バージョン間差分解析ツール

Polyspector™プラットフォームを利用して，ソフトウェア改善作業によって品質にどのような変化が生じたのかを可視化するためのツールを開発した．本ツールでは，ソフトウェアの静的な階層構造マップを示したうえで，ユーザの目的に応じた解析結果を可視化することができる．

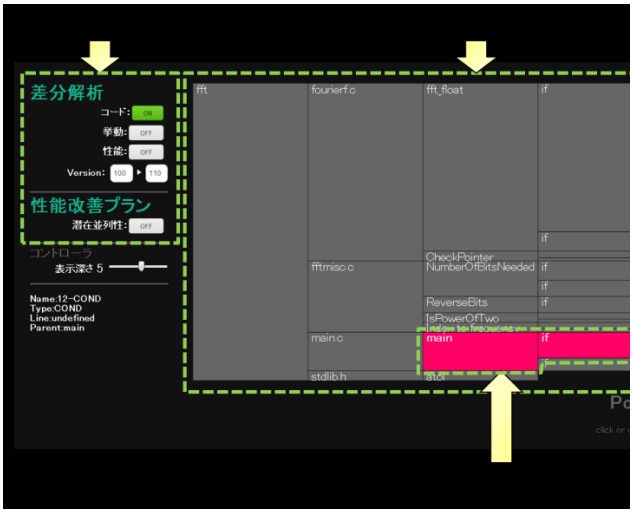

図 2　バージョン間差分解析ツール

3.1 GUI の構造

図 2 に示すように，GUI には大きく分けて 3 つの機能が備わっている．(a) どの解析を行いたいかをユーザが制御するための入力機能，(b) 静的なソフトウェア階層構造を表示する機能，(c) (a) で選択された解析結果に該当する

†1 株式会社　東芝
Toshiba Corporation

ソフトウェア箇所を（b）上に表示する機能である．

3.2 解析の種類

図 3　解析機能の選択

図 3 のように本ツールでは 4 つの解析機能を用意した．ソースコードの改変によって階層上のどこが書き換わったのか(コード)，その結果どこの挙動に変化が生じたのか(挙動)，性能が向上もしくは劣化した箇所はどこか（性能），また性能を改善するための候補として潜在的に並列性を有しているのはどこなのか(潜在並列性)の 4 つである．（コード）は静的な解析を，（挙動）と（性能）は 2 つのバージョンから取得したトレース情報を用いた解析を，(潜在並列性）は改変後のバージョンにおけるトレース情報のみを用いた解析を行っている．

また差分における 2 つのバージョンは任意に選択可能となっており，本例ではバージョン 100 と 110 の間の関係を分析している．

3.3 使用例

図 4　ツールを用いた分析フローの一例

図 4 を用いて本ツールの使用例を示す．まずステップ(a)において（コード）を選択し，ソースコードの改変箇所を表示する．次にステップ（b）においてさらに（挙動）を選択し挙動が変化した箇所を表示すると，改変していない箇所でも挙動が変化していることが分かる．次に（c）で（性能）を選択することで，性能が向上した箇所を青，劣化した箇所を赤で表示する．ソフトウェア全体（階層構造の最左側で表現）は性能が向上しているものの，一部（本例では上の方に配置された）モジュールで劣化していることが分かる．このように様々な視点からソースコードの改変によってソフトウェアがどう変化したのかを分析することができる．

3.4 プラットフォームの有用性に関する考察

3.4.1 同じトレース情報から異なった視点の解析を実現

Polyspector™ではトレース情報として関数コール，分岐，メモリアクセス，分岐間の処理時間などを収集している．これらの情報の組み合わせによって，上記のような様々な解析を実現することが可能である．例えば（性能）は関数コールと分岐と処理時間を，(潜在並列性）は分岐とメモリアクセスを利用した解析である．

3.4.2 複数のトレース情報をもとにした解析を実現

トレース情報を蓄積しておくことで任意のバージョン間での品質傾向分析が可能となっている．これにより直近のバージョン間の分析や，前回のリリースバージョンとの分析など目的に応じた分析が可能となる．さらに 2 バージョン間ではなく，全てのバージョン間のトレンド分析など様々な応用が考えられる．

大量のトレース情報を効率よく管理・解析するための工夫として，Polyspector™では近年ビッグデータ分析を効率化するために発展が続いている NoSQL 型データベースを応用したデータ管理を行っている．

3.4.3 複数の解析結果を共通の GUI で統一化

トレース情報をもとにした様々な解析を統一的な GUI で可視化することで，ツール学習コストの低減と，異種解析の複合を実現した．本 GUI はソフトウェアの静的な階層構造はあまり変化しない点と，ツールユーザが知りたいのはソースコードのどこに問題点や改善点が潜んでいるかであるという点に着目して設計されている．そのため，静的な階層構造の上に，選択された解析結果をマップして表示する形式を採用している．また図 4 (b) のように（コード）と（挙動）といった複数の解析結果をまとめて表示できる工夫もしている．

4. おわりに

ソフトウェア実行時のトレース情報を効率良く蓄積，解析，可視化するための統合プラットフォーム Polyspector™ について，実応用の一つであるバージョン間差分解析ツールをモチーフにその有用性とプラットフォーム応用の可能性について紹介した．

トレース情報にはソフトウェア品質を分析する上で有益な情報が多く含まれており，こういった情報からソフトウェアの品質情報を効率良くフィードバックするための技術を実現したことで，これまでに無かった新しい品質分析を実現できる可能性があると考える．

参考文献

1)　Valgrind’s Tool Suite, http://valgrind.org/info/tools.html