B-020

Javaバイトコード解析を利用した プログラム構造の可視化によるソフトウェア開発の改善

Improvement of software development by visualization of program structure

岡田 太† 福士 陽† 清水 理恵子† 宮崎 肇之†
Futoshi Okada Akira Fukushi Rieko Shimizu Tadashi Miyazaki

1. はじめに

近年の情報システムは，一部の業務を効率化するものだった個々のシステムを統合して企業の業務を全体カバーする様になってきた。これにより大規模化・複雑化が進み，ソフトウェア開発では品質確保と保守作業の効率化が重要視されている。特に大規模なシステムの場合，不良修正時の類似見直しや，仕様変更に伴うプログラム変更時の影響調査が問題となる事が多い。

日立製作所ではソフトウェア開発の改善に関する施策として，プログラム構造を可視化する「依存関係抽出ツール」を開発し，コーディングルールが守られているかどうかのチェックや，プログラム変更時の影響分析の効率化を図ってきた。

本論文では，弊社が行ってきたプログラム構造の可視化によるソフトウェア開発の品質向上と変更管理における改善の取り組み事例を紹介する。

2．依存関係抽出ツールの開発

依存関係抽出ツールは，Java 言語によるソフトウェア開発において，Java の実行形式であるバイトコード(Class ファイル)を解析してプログラム間の依存関係を抽出し，プログラム構造を可視化する事を目的としている。

2.1 バイトコード解析を利用する理由

一般にクロスリファレンスと呼ばれる依存関係の抽出で解析対象とするソースコードは，構文解析処理が必要になるため，大規模システムに適用するには，処理性能の面において実用に耐えないケースが多い。

一方，バイトコードはランタイムでの読み込みに適した構造を持っているためにシンプルであり，ソースコードに比べて解析処理における処理性能が高い。その反面，プログラムの処理内容に関係の無いコメント文の情報が取得できないなど，ソースコードとまったく同じ情報が取得できる訳では無い。

しかしながら，依存関係の抽出に必要な情報としては十分であるため，処理性能の面からバイトコード解析を利用する事とした。

2.2 依存関係抽出ツールの処理概要

依存関係抽出ツールの動作イメージを図 2.2-1 に示す。

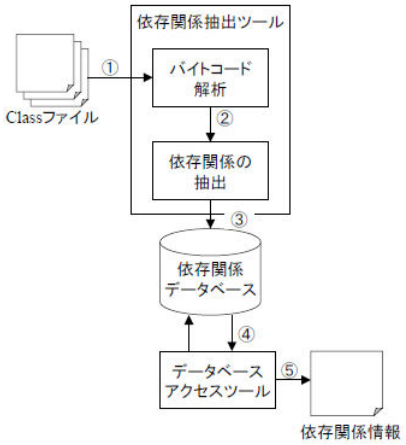

図 2.2-1：依存関係抽出ツールの動作イメージ

依存関係抽出ツールは，バイトコード(Class ファイル)を読み込んで解析し(①)，依存関係の抽出に必要な情報を取得して(②)データベースに登録する(③)。データベースに依存関係を登録した後は，用途に応じて SQL を作成し，データベースアクセスツールを使って依存関係を抽出する(④，⑤)。

なお，「依存関係の抽出に必要な情報」とは，以下に示す様な情報を対象としており，図 2.2-2 に示す様にソースコードの入れ子構造をそのままデータベースの親子関係にマッピングする形で登録する。

- クラス名
- メソッド名
- メンバ変数
- 継承関係
- メソッド呼出処理
- メンバ変数へのアクセス

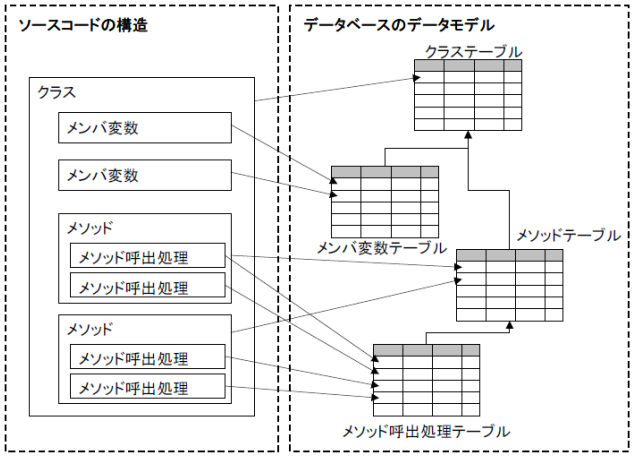

図 2.2-2：クラス構造とデータベースのマッピング

†（株）日立製作所 情報・通信グループ プロジェクトマネジメント統括推進本部, Hitachi, Ltd. Information & Telecommunication Systems. Project & Process Management Division.

ユーザインターフェースにデータベースアクセスツールを利用する事で，依存関係の抽出に使用する SQL の工夫次第で用途毎に適した切り口で依存関係を可視化できる仕組みとした。また，ソースコードの構造をそのままデータベース登録しているため，SQL の作成にあたってバイトコードの知識を必要としないのも特徴である。

3. ソフトウェア開発を改善する取り組み

依存関係抽出ツールを適用したソフトウェア開発の改善に関する取り組みを以下に述べる。

（1）仕様変更おける影響範囲の調査

ソフトウェアの仕様変更によりプログラムの修正を行う場合，変更対象プログラム自体に加えて，変更対象プログラムを呼び出しているプログラムもテスト対象として抽出が必要となる。

通常，テスト対象プログラムの抽出にあたっては，ソースコードまたは設計書から手作業で呼出関係を調べる事になるが，手作業では対象プログラムの規模が大きくなるほど抽出漏れの危険性が高くなる。また，Eclipse などの IDE(統合開発環境)の機能によって機械的に調べる方法もあるが，数メガステップに及ぶ大規模システムでは，1 台の PC 上で処理する上ではメモリが不足してしまうケースがある。

依存関係抽出ツールでは，データベースに依存関係情報を保持している。そのため，一度にメモリを確保する量を DBMS が作業用領域として割り当てたサイズに収まるようにコントロールできた。

事例では，依存関係抽出ツールを適用する事により，対象プログラム 3 メガステップ相当の大規模システムにおいても抽出漏れの危険性を回避し，実用に耐える事ができた。

（2）未使用プログラムの検出

「未使用プログラム」とは，ソフトウェアが完成した時点で，そのソフトウェア内で使用されていない不要なプログラムの事である。

未使用プログラムは，主にソフトウェアの仕様変更に伴うプログラム修正により，プログラム間の呼出関係が変更される際に発生する。不要なプログラムが残存しているとソフトウェアの規模が必要以上に大きくなり，保守作業における効率が悪くなるため，開発時に要否を判断して削除すべきである。しかし，開発中は一時的に呼出処理が実装されていない可能性があるため，要否を判断する事は難しい。

そこで，開発終了時に依存関係抽出ツールを用いてクラスおよびメソッド単位で呼出関係の無いプログラムを抽出する事で未使用プログラムを検出し，保守対象のプログラムを適正化した。

事例では，データベースアクセスプログラム 2001 本中の約 20%にあたる 408 本について未使用プログラムである事が判明し，保守対象から削除する事ができた。

（3）依存関係に関するコーディングルールのチェック

多くのソフトウェア開発では，実装の均一化による品質の向上を目的としてコーディングルールを設けている。コーディングルールには，どのソフトウェアにも共通して適用できるアンチパターンをチェックアウトするための汎用的なルールと，ソフトウェア毎の仕様に特化して実装の均一化を目的とするソフトウェア独自ルールの 2 種類が存在する。

汎用的なルールに関してはチェックするためのツールが数多く流通しているが，ソフトウェア独自ルールについては手作業によるチェックに頼っている傾向にある。

そこで，依存関係抽出ツールを適用して，プログラム間の依存関係に関するルールを対象として，違反している修正対象を抽出した。

なお，ソフトウェア独自ルールのチェック以外にも，依存関係抽出ツールの該当機能の応用事例として，JDK のバージョン間の動作レベルでの差異のある API を呼んでいる箇所の調査にも活用できた。

4. まとめ

依存関係抽出ツールを適用する事により，3 章にて取り上げた以下 3 つの課題を改善する事ができた。

- 仕様変更における影響範囲の調査
- 未使用プログラムの検出
- 依存関係に関するコーディングルールのチェック

また，本ツールを「バイトコード解析技術をソフトウェア開発の改善に活かす手段」として活用できるという事が確認できた。

5. 今後の課題

3 章で述べた取り組みの定着化に向けた課題を以下に述べる。

（1）適用効果の定量的評価

3 章で述べた取り組みの事例では，依存関係抽出ツールの適用効果を評価できていない。

今後の適用事例を通して，それぞれの取り組みにおける工数削減や品質向上の効果について定量的な評価を行っていく。

（2）解析処理のバックグラウンド化

取り組みの定着化には，解析結果が常にソースコードと同期が取れている事が重要であり，そのためには解析処理の自動化が必要である。

ソフトウェア構成管理ツールと連動して，ソースコードの変更を契機に自動的にバックグラウンドで解析処理を行える仕組みを今後検討していきたいと考える。

（3）適用事例のナレッジ化

今後の適用事例についても，本論文で取り上げた適用事例と同様に問題点と解決方法を整理し，ソフトウェア開発に有効に活用できるようナレッジ化する事を検討していく。

以上