The University of Osaka
Institutional Knowledge Archive

Title	オブジェクト指向プログラムの変更作業を支援する影響波及解析システム
Author(s)	横森, 励士; 近藤, 和弘; 大畑, 文明 他
Citation	電子情報通信学会論文誌D-I. 2003, J86-D-I(3), p. 150-158
Version Type	VoR
URL	https://hdl.handle.net/11094/26482
rights	copyright©2003 IEICE
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

論　文

オブジェクト指向プログラムの変更作業を支援する影響波及解析システム

横森　励士[†]　　近藤　和弘[††]　　大畑　文明[†††]　　井上　克郎[††††]

Impact Analysis System for Changes on Object-Oriented Programs

Reishi YOKOMORI[†], Kazuhiro KONDOU[††], Fumiaki OHATA[†††], and Katsuro INOUE[††††]

あらまし　影響波及解析とは，プログラム変更の影響を受ける部分を識別する手法で，回帰テストでのテストケース選択に利用されてきた．我々はプログラム理解，保守といったより広い範囲でも影響波及解析が利用できると考えているが，既存の手法は被影響部分の探索ルールがテストケース選択用に特化されているため，利用目的に応じて探索ルールを定義できる仕組みが必要となっている．また近年のソフトウェア開発環境では，オブジェクト指向言語が多く利用されており，それらに対応した解析手法及びその実装が求められている．本論文では，ユーザの利用目的に応じて様々な影響波及ルールが定義できる影響波及解析手法を提案する．提案手法では，オブジェクト指向言語Java を対象に，クラスメンバ間の関係を表す二つのグラフに基づき解析を行う．また，提案手法をJava 影響波及解析システムとして実装し，その有効性を検証する．

キーワード　影響波及解析，プログラム理解，オブジェクト指向プログラム，Java

1. まえがき

ソフトウェアに加えられた変更による影響を受ける部分を識別し，回帰テストでのテストケースを選択するための手法として，影響波及解析が提案されている．我々はプログラム理解，保守といったより広い範囲でも影響波及解析が利用できると考えているが，従来の影響波及解析手法は，被影響部分の探索ルールなどが回帰テストにおけるテストケース選択用に特化した形で実現されている．そのため，適用時に探索ルールを利用目的に応じて定義できる仕組みが必要となる．

また，現在のソフトウェア開発環境ではオブジェクト指向言語が多く利用されている．オブジェクト指向プログラムでは，従来の手続き型プログラムに比べ，変更の影響が変更箇所以外に及ぶ場合が数多く存在するため，オブジェクト指向言語の概念を考慮した解析手法及びその実装が求められている．

本論文では，Java [6] を対象言語として影響波及解析手法の提案を行う．提案手法では，被影響部分の探索ルールをユーザの目的に応じて選択可能にすることで，回帰テストだけではなく，プログラム理解，保守といった，より広い範囲での影響波及解析の利用を実現している．更に，クラスのメンバ（メソッド，フィールド）間の関係を表現する 2 種類のグラフを利用することで，メソッドのオーバライド，フィールドの隠蔽を考慮した影響波及解析を実現する．また，提案手法をJava 影響波及解析システムとして実装し，手法の有効性を検証する．

以降，**2.** では影響波及解析について紹介し，**3.** では提案する影響波及解析手法について説明する．**4.** でJava 影響波及解析システムについて述べ，**5.** で手法の有効性を検証する．最後に，**6.** でまとめと今後の課題について述べる．

† 大阪大学大学院基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan
†† ソニー・エリクソン・モバイルコミュニケーションズ株式会社，東京都
Sony Ericsson Mobile Communications Japan, Inc., Konan, Minato-ku, Tokyo, 108–0075 Japan
††† （株）東芝，東京都
TOSHIBA CORPORATION, 1–1–1 Shibaura, Minato-ku, Tokyo, 105–8001 Japan
†††† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan

2. オブジェクト指向プログラムに対する影響波及解析

本章では，既存の影響波及解析手法と，本論文の対象となるオブジェクト指向言語に対する影響波及解析について説明する．更に，既存手法の問題点について考察する．

2.1 影響波及解析

ソフトウェア保守工程において，プログラム開発者はソフトウェアに対し多くの変更を行うが，その際，誤って欠陥を作り込んでしまう確率は 50%から 80%にも及ぶことが Hetzel により示されている [12]．要因としては，ソフトウェアに変更を加えたときに，変更していない部分に関しても何らかの影響が及ぶ可能性があることが挙げられる．

ソフトウェアに加えられた変更による影響を受ける部分（被影響部分と呼ぶ）を識別するための手法として，影響波及解析が提案されている．影響波及解析の適用分野の代表例として，変更後のソフトウェアが仕様どおりに動作するかを確認するための，回帰テスト（Regression Testing）[11] への利用が挙げられる．回帰テストは，

Step 1: 被影響部分の識別
Step 2: 修正コンポーネントの再テスト方法の決定
Step 3: 再テストによる補償範囲の認識
Step 4: テストケースの選択，再利用，修正，新規作成

という過程をたどるが，Step 1 において影響波及解析を利用することで，適用すべきテストケースを変更した部分に関係のあるものだけに限定し，必要最小限に抑えることができる．

これまでにオブジェクト指向プログラムに対する影響波及解析手法がいくつか提案されている [1], [5], [8], [13], [14]．ここでは，既存手法を解析の粒度で大きく三つに分類する．

［クラス単位］ クラスを被影響部分の単位とする [13]．メンバ単位・文単位の解析に比べて解析の粒度が大きく，数百クラスにも及ぶような大規模ソフトウェアに対して変更を行う際に有効な手法である．しかし，クラス内のどのメンバが影響を受けているかを特定できないため，影響の予測としては効果が薄く，回帰テストにおいて再テストの必要がないメンバも解析結果に含み得ることが指摘されている [7]．

［メンバ単位］ クラスのメンバ（メソッド，フィールド）を被影響部分の単位とする [8], [14]．オブジェクトの構成要素であるメンバが解析結果となり，直感的に理解しやすく，クラス単位での解析より正確な結果を得ることができる．

［文単位］ 文を被影響部分の単位とする [1], [5]．プログラムスライス（Program Slice）[9] に基づいており，ある文に影響を及ぼす文の集合及び，ある文が影響を及ぼす文の集合を抽出することで被影響部分を特定する．クラス単位及びメンバ単位の解析に比べて正確な結果を得ることができるが，文間の依存関係など，多くの解析が前提となり，解析コストは膨大になる．

本論文では，メソッドの追加，削除，またシグニチャの変更といったメンバに関する変更に着目し，メンバに関する変更による被影響部分の抽出を目的としたメンバ単位での解析を行う．

2.2 既存手法の問題点

従来の影響波及解析手法は回帰テストにおけるテストケース選択用に特化した形で実現されており，被影響部分の探索ルールもテストケース選択用に特化して定義されていた．しかし，プログラム理解，保守などのより広い範囲での影響波及解析の利用を考えた場合，影響の定義は利用目的に応じて異なる．例えば，回帰テストにおけるテストケースの選択 [2] を目的とした場合には，変更または削除されたメソッドをテストするテストケース，及びオーバライド関係の変化したメソッドへの呼出し経路をテストするテストケースの検出が必要となる．一方で，修正箇所の特定を目的としてメソッドオーバライドの変化による影響を求める場合には，オーバライド関係の変化したメソッドを直接呼び出している部分を検出できれば十分である．そのため，ユーザの目的に応じて被影響部分の探索ルールが選択可能な解析手法が必要である．

また，現在のソフトウェア開発環境ではオブジェクト指向言語が多く利用されている．オブジェクト指向プログラムでは，従来の手続き型プログラムに比べ，変更箇所以外に影響を及ぼすような変更が数多く存在する．文献 [4], [10] では，オブジェクト指向プログラムにおける変更は，メソッドのオーバライド（Override），フィールドの隠蔽（Hide）といったオブジェクト指向特有の概念により様々な影響が引き起こされることが述べられている．メンバを解析の単位とした影響波及解析手法では，これらの独自概念は考慮されているが満足な実装がなされていない [8], [14]．そのため，オブジェクト指向言語の独自概念を考慮した解析手法及

びその実装が求められている．

3. MOG，MAG による影響波及解析

本章では，クラスのメンバ間の関係を表現する二つのグラフを利用した影響波及解析手法の提案を行う．提案手法により，メソッドのオーバライド，フィールドの隠蔽を考慮した影響波及解析の実現，及びユーザの様々な目的に対応可能な影響の定義を行うことができる．

3.1 方　　針

影響は，オーバライドや呼出し経路などに変化が生じたメンバから発生し，呼出し経路を通じて波及するものである．回帰テストに利用されてきた既存の影響波及解析は，呼出し経路を逆にたどる，つまり一部の呼出し経路に限定した特殊な波及を考えていたとみなすことができる [2].

我々は，より一般的な影響波及解析の枠組み，具体的には，メソッドのオーバライドやフィールドの隠蔽などによって生じる様々な種類の影響の発生及び波及のパターンを組み合わせてできる枠組みを実現するため，グラフを用いてオーバライド関係及び呼出し関係を表現する．影響の発生はグラフの変化に，影響の波及はグラフ探索にそれぞれ置き換えることができる．

本論文では，これらを実現するために，メンバオーバライドグラフ（Member Override Graph，MOG）及びメンバアクセスグラフ（Member Access Graph，MAG）を利用した手法を提案する．MOG とは，メソッドオーバライド，抽象メソッドの実装，フィールドの隠蔽などのメンバ間のオーバライド関係をグラフで表現したもので，継承により親子関係となるクラスのメンバ間に存在する．従来の手法 [8], [14] では，グラフを構築する際に考慮されているのはクラスの継承までで，メンバ間の関係においては抽出アルゴリズム側で考慮していた．それに対して MOG では，グラフ構築時にクラスの継承に関する情報からあらかじめメンバ間の関係を求め，辺として表現している．

MAG とは，メソッドの呼出し，フィールドの参照などのメンバ間のアクセス関係をグラフで表現したもので．クラスのメンバ間に存在する．

3.2 メンバオーバライドグラフ（MOG）

ここでは，MOG の構成要素である MOG 節点及び MOG 辺の定義，MOG の構築方法について述べる．図 1 (b) は，図 1 (a) のプログラムから構築される MOG である．

MOG 節点は，各クラスの各メソッドに対応した MOG メソッド節点と，各フィールドに対応した MOG フィールド節点の 2 種類からなる．静的初期化子及びコンストラクタはメンバではないが，メンバと同様のアクセス関係をもつことから，これらも MOG 節点として扱う．MOG 節点は後述する MAG 節点と 1 対 1

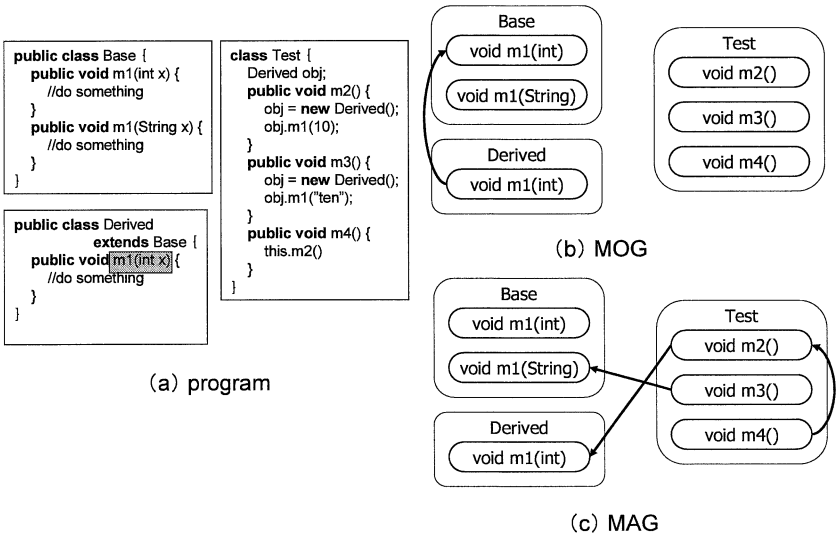

図 1　メンバオーバライドグラフ（MOG）とメンバアクセスグラフ（MAG）
Fig. 1　Member Override Graph (MOG) and Member Access Graph (MAG).

で対応する．

MOG 辺は，二つの MOG 節点間のオーバライド関係を有向辺で表現したものであり，メソッドオーバライドを表す MOG オーバライド辺，抽象メソッドの実装を表現する MOG 実装辺，フィールドの隠蔽を表現する MOG 隠蔽辺の 3 種類からなる．各辺は，オーバライドするメンバからオーバライドされるメンバへ引かれる．図 1 (b) では，MOG メソッド節点`void Derived.m1(int)` から，`void Base.m1(int)` へ MOG オーバライド辺が引かれている．MOG は，各クラスのメソッド，フィールド宣言の解析による MOG 節点の抽出，及びクラスの継承関係の解析による MOG 辺の抽出により構築される．

3.3 メンバアクセスグラフ（MAG）

ここでは，MAG の構成要素である MAG 節点及び MAG 辺の定義，MAG の構築方法について述べる．図 1 (c) は，図 1 (a) のプログラムから構築される MAG である．

MAG 節点は，MOG 節点と同様に，MAG メソッド節点，MAG フィールド節点からなる．MAG 節点は MOG 節点と 1 対 1 で対応する．

MAG 辺は，二つの MAG 節点間のアクセス関係を有向辺で表現したものであり，メソッド呼出しを表す MAG 呼出し辺，フィールドの参照を表す MAG 参照辺の 2 種類からなる．各辺は，アクセスするメンバからアクセスされるメンバへ引かれ，二つの節点間に複数の辺が存在する場合もある．

MAG の構築時には，MAG 節点を抽出した上で，メソッド呼出し式，フィールド参照式を解析し，各フィールドがどのクラスのオブジェクトを参照し得るかをあらかじめ解析した上で，各文において実際に参照し得るメンバの集合を求めることで MAG 辺の抽出を行う．このとき，参照変数が指すインスタンスの型が特定できないことにより，アクセスされるメンバが一意に決まらない場合がある．その際には，その参照変数の型から派生したクラスに存在する，同一シグニチャのすべてのメンバに対して辺を引く．

3.4 MOG，MAG による影響波及解析

変更によって何らかの変化が生じた MOG 節点，MAG 節点を検出し，その節点から MOG 辺，MAG 辺をたどることで，被影響部分の抽出を行う．更に，探索ルールを変更可能とすることにより，ユーザの様々な目的に対応することができる．

3.4.1 被影響部分の分類

提案する影響波及解析により抽出される被影響部分の単位は，プログラム上ではメンバ，MOG，MAG 上では節点となる．本論文では，これらをそれぞれ被影響メンバ（Affected Member），被影響節点（Affected Node）と呼ぶ．

ユーザの目的に応じた被影響メンバの抽出を行うため，被影響部分を表す節点を直接的な影響を表す直接被影響節点（Direct Affected Node）と間接的な影響を表す間接被影響節点（Indirect Affected Node）に分類する．

直接被影響節点は変更が行われた節点に基づいて決定されるもので，表 1 に挙げる 5 種類がある．例えば，メソッドの内容を変更した場合，表 1 の D-E1 から，変更されたメソッドに対応する MAG 及び MOG 節点が直接影響節点に含まれる．また，変更前後の MAG における辺を比較してアクセス関係に変更（追加・削除・更新）があった場合，D-E2〜E5 に対応する MAG 節点が直接影響節点に含まれる．図 2 は，変更が行われた節点が M8 であるときの直接被影響節点の例を示している．

間接被影響節点は表 2 に挙げる 2 種類があり，直接的な変更によって変化した値を参照する節点などの直接被影響節点から推移的に影響を受け得る節点を指す．そのため，変更によって直接影響を受けた節点に関係のあるメンバがすべて含まれることになり，多く

表 1　直接被影響節点
Table 1　Direct affected node.

直接被影響節点	概要
D-E1: 影響の発生元の節点	プログラム変更に対応する節点（発生，消失した節点もこれに該当する）．（図 2 の MOG 節点 {M8} 及び MAG 節点の { M8 }）
D-E2: 辺の発生先の節点	プログラム変更により発生した辺の終節点．（図 2 の MOG 節点 {M7} 及び MAG 節点 {M6，M8}）
D-E3: 辺の発生元の節点	プログラム変更により発生した辺の始節点．（図 2 の MOG 節点 {M8} 及び MAG 節点 {M3，M4}）
D-E4: 辺の消失先の節点	プログラム変更により消失した辺の終節点．（図 2 の MOG 節点 {M6} 及び MAG 節点 {M7，M8}）
D-E5: 辺の消失元の節点	プログラム変更により消失した辺の始節点．（図 2 の MOG 節点 {M8} 及び MAG 節点 {M3，M4}）

表 2　間接被影響節点
Table 2　Indirect affected node.

間接被影響節点	概要
I-E1: 順方向の推移的な影響波及のある節点	直接被影響節点から有向辺に従い到達可能な節点. （図 3 の MAG 節点 {M9, M10, F1, F2}）
I-E2: 逆方向の推移的な影響波及のある節点	直接被影響節点から有向辺の逆向きに従い到達可能な節点. （図 3 の MOG 節点 {M10} 及び MAG 節点 {M1, M3, M4}）

表 3　探索ルールの例
Table 3　The example of search rule.

探索ルール	探索対象となる被影響節点	
	MOG	MAG
R1: アクセス変化メンバ抽出	ϕ	{D-E1, D-E3, D-E5, I-E2}
R2: 関係変化メンバ抽出	{D-E1, D-E2, D-E3, D-E4, D-E5}	{D-E1, D-E2, D-E3, D-E4, D-E5}
R3: 間接アクセスメンバ抽出	ϕ	{D-E1, I-E1, I-E2}

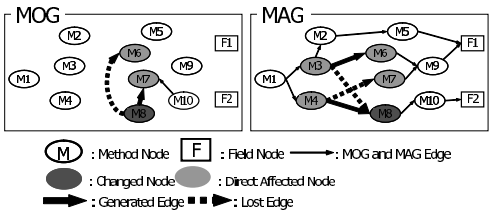

図 2　直接被影響節点の例（表 1 参照）
Fig. 2　A example of the direct affected nodes. (See Table 1)

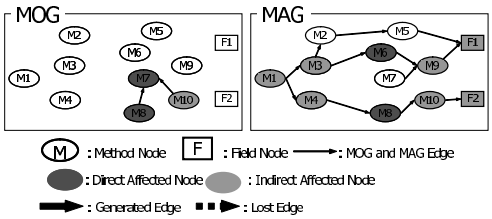

図 3　間接被影響節点の例（表 2 参照）
Fig. 3　A example of the indirect affected nodes. (See Table 2)

のメンバが間接被影響節点として抽出される．間接被影響節点によって，変更箇所で利用されるメンバの限定及び，従来手法のテストケースの限定時において想定されていた，変更により実行結果が変わり得る場所の限定を行うことができる．図 3 は，MOG の直接被影響節点が M7，M8，MAG の直接被影響節点が M6，M8 であるときの間接被影響節点の例をそれぞれ示している．

3.4.2　被影響部分の抽出

提案手法では，前項で定義した各被影響節点の抽出

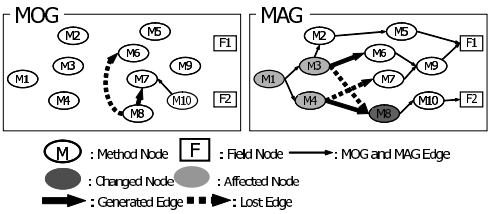

図 4　R1: アクセス発生メンバ抽出
Fig. 4　R1: The extraction of the nodes that changes access relation.

の有無を組み合わせることにより，ユーザの目的に応じた被影響メンバの抽出が可能となる．組合せは数多く存在するが，ここでは代表的な三つの探索ルールについて述べる．なお，各ルールに対応する被影響節点の組合せの一覧を表 3 に示す．

［R1: アクセス発生メンバ抽出］

変更により発生した新たな実行経路上に存在する（すなわち新たにプログラムの実行結果に影響を及ぼす可能性が生じた）部分を抽出する．既存の影響波及解析手法が対象としている，回帰テストでの利用を目的としたものである．テストケース選択に用いる場合は，被影響節点の中でテストケースに対応するものだけを選択すればよい．図 4 に，M8 が変更メンバである場合の抽出例を示す．

［R2: 関係変化メンバ抽出］

オーバライド関係の変化，及びそれに伴うアクセス関係の変化が発生するメンバをすべて抽出する．プログラム変更によるメンバ間の関係の変化を把握し，変更に対応するべき修正箇所を識別するために有効であ

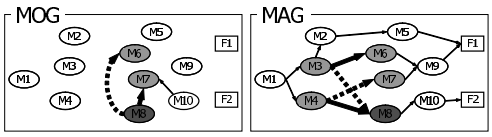

図 5　R2: 関係変化メンバ抽出（図 4 参照）
Fig. 5　R2: The extraction of the nodes that changes override relation. (See Fig. 4)

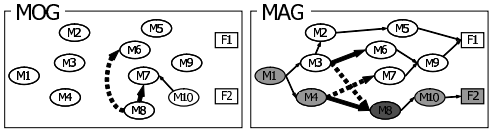

図 6　R3: 間接アクセスメンバ抽出（図 4 参照）
Fig. 6　R3: The extraction of the indirect access nodes. (See Fig. 4)

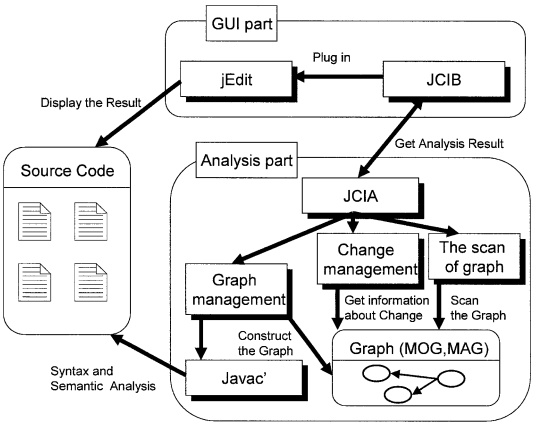

図 7　システム構成
Fig. 7　The structure of the system.

る．図 5 に，M8 が変更メンバである場合の抽出例を示す．

［R3: 間接アクセスメンバ抽出］

変更メンバに直接的及び間接的にアクセスする可能性のあるメンバ，変更メンバが直接的及び間接的にアクセスする可能性のあるメンバをすべて抽出する．メソッド本体やフィールドの初期値などを変更する場合，アクセス関係に変化はないが，それらを使用するメンバの実行結果などが変化する可能性があるため，このルールによる被影響メンバの把握が有効となる．図 6 に，M8 が変更メンバである場合の抽出例を示す．

4. JAVA 影響波及解析システム

本章では，提案手法の実装であるJAVA 影響波及解析システムの構成，及びその機能について述べる．

4.1 概　　要

本システムは，JAVA プログラムにおける，メンバ単位の変更で生じる影響を解析，表示するシステムである．本システムの利用の流れは次のようになる．

Step1:　変更前のプログラムに対するグラフの構築
Step2:　ユーザによるソースファイルへの変更
Step3:　変更後のプログラムに対するグラフの再構築
Step4:　変更前後の差分からの被影響メンバ抽出
Step5:　抽出された被影響メンバの表示

4.2 利用ツール

本システムでは，Java プログラムの字句解析，構文解析，意味解析に javac [15] のコードを流用し，GUI に jEdit [17] を採用した．

javac は JDK [16] 付属の Java コンパイラであり，本システムで流用するにあたって行われたのコード変更は 100 行未満に抑えられており，将来の JDK のバージョンアップにも容易に対応できる．

jEdit は Pestov らにより開発されているオープンソースのテキストエディタで，プラグイン機能の提供により高い拡張性を有している．本システムは，jEdit の 1 プラグインとして動作する．

4.3 システム構成

本システムは，実際に提案手法による影響波及を行う解析部と，ユーザインタフェースとなる GUI 部で構成されている．図 7 はシステムの各構成要素の関連を示したものである．また，以下の説明では，MOG, MAG を単にグラフと呼ぶ．

4.3.1　解析部の説明

解析部は，次の五つの部分から構成されている．

［JCIA］　グラフ管理部，変更管理部，グラフ走査部に指示を出すことにより，影響波及解析を行う．

［グラフ管理部］　指定されたJAVA プログラムのグラフの構築を行う．

［変更管理部］　変更前後のプログラムに対応するグラフを比較し，節点，辺の発生，消失といったグラフの変化を検出する．

［グラフ走査部］　与えられた被影響メンバ探索ルールに従ってグラフを走査し，被影響メンバの抽出を行う．

［javac'］　JAVA プログラムのソースコードに対し，字句解析，構文解析，意味解析を行うとともに，グラフ構築に必要な情報を収集する．

4.3.2 GUI 部の説明

GUI 部は，JCIB と jEdit の二つから構成されている．

［JCIB］ jEdit のプラグインとして動作し，ユーザからの解析要求を解析部に伝え，解析結果を表示する．

［jEdit］ Java プログラムに変更を行う際のエディタとして動作し，JCIB と連携して解析結果を表示する．

5. システムの適用例と有効性

ここでは，実際にソフトウェアに対して実装システムの適用を行い，提案手法の有効性を検証する．

具体的には，ソフトウェアのある基準バージョンのソースコードと，何かしらの変更が行われた後のバージョンのソースコードに関し，システムが検出した被影響メンバと実際のソースコード差分との比較を行う．そして，その結果を考察することで，システムがユーザに対して適切な情報を提供できているか検証する．

［適用対象ソフトウェア］

本システムの適用対象ソフトウェアとして，Java ベースのビルドツールである Ant を選択した．Ant は Jakarta Project で開発されているオープンソースソフトウェアであり，Ant v1.1 は，96 個のクラス（約 20,000 行）から構成されている．

［適用実験］

ここでは，Ant の開発過程において実際に行われた変更に対してシステムの適用を試みる．Ant は v1.1 から v1.2 において，様々な機能追加及び修正が行われているが，その中の一つであるメソッド`Property::init()` の削除に着目し，このメソッドの削除による被影響メンバをシステムを用いて特定する．`Property::init()` の削除によって生じたすべての直接的な影響を調べるために「関係変化メンバ抽出」を探索ルールとして指定し，解析を行った結果，多くの被影響メンバが抽出された．そこで MOG 及び MAG ごとに分けることで,「関係変化メンバ抽出」ルールを「オーバライド関係の変化」と「アクセス先の変化」に分解して再解析したところ，次のようなことを把握できた．

- オーバライド関係の変化

`Property` の親クラス`Task` のメソッド`init()` がオーバライドされなくなった．

- メソッドのアクセス先の変化

`TaskHandler::init()`，`Ant::execute()` 中のメソッド呼出し式における呼出し先が変更された．

［実際の更新作業との比較に基づく考察］

削除されたメソッド`Property::init()` の機能について考えると，その機能が不要な場合は削除され，必要な場合はプログラムの他の部分に移譲されるはずである．後者の場合，`Property::init()` の削除により影響を受けるメンバにおいて，何らかの対応を行う必要がある．そこで，上記の三つの被影響メンバ`Task::init()`，`TaskHandler::init()`，`Ant::execute()` が，v1.1 から v1.2 においてどのように修正されたかを調査した．

- `Task::init()`

v1.1 から v1.2 において全く変更が行われておらず，`Property::init()` 中に存在していた機能は，オーバライドの対象であった`Task::init()` に移譲されたわけではないことがわかった．

- `TaskHandler::init()`

v1.1 から v1.2 においてメソッドに変更が加えられており，各バージョンにおけるソースコードは図 8 のようになっていた．`Property::init()` の削除とは関係のない，機能追加と考えられる部分を除くと，図 8 の網掛部が削除の影響により変更された部分であると思われる．v1.2 では，`Task` 型の参照変数`task` が指し得るオブジェクト（`Task` クラスの子クラスである`Property` クラスのオブジェクトなど）に対して，v1.1 と同様に`init()` の呼出しを行うだけでなく，場合によっては`execute()` の呼出しも行うよう変更されていることがわかる．

- `Ant::execute()`

v1.1 から v1.2 においてメソッドに変更が加えられており，各バージョンにおけるソースコードは図 9 のようになっていた．これにより，`Property::init()` の呼出し式が，`Property::execute()` の呼出し式に置

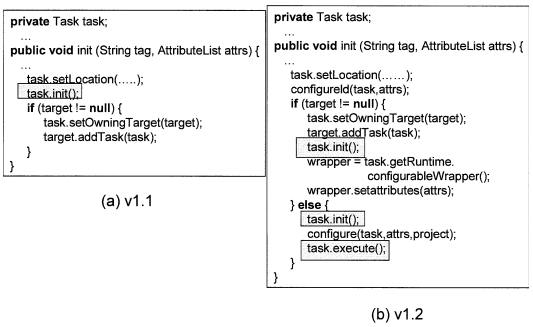

図 8 v1.1 と v1.2 における TaskHandler::init()
Fig. 8 Method TaskHandler::init() in v1.1 and v1.2.

```
public void execute() {
  ...
  Enumeration e = properties.elements();
  while (e.hasMoreElements()){
      Property p = (property) e.nextElement();
      p.init();
  }
 ...
}
```

(a) v1.1

```
public void execute() {
  ...
  Enumeration e = properties.elements();
  while (e.hasMoreElements()){
      Property p = (property) e.nextElement();
      p.execute();
  }
 ...
}
```

(b) v1.2

図 9　v1.1 と v1.2 における Ant::execute()
Fig. 9　Method Ant::execute() in v1.1 and v1.2.

換されていることがわかる．

上記の変更点より，`Property::init()` の機能は，`Property::execute()` に移譲されたと推測できる．実際に，v1.1 における`Property::init()` と v1.2 における`Property::execute()` の間には機能の一致が確認できた．

以上より，システムにより抽出された被影響メンバは，実際の変更作業において確かに修正が行われていたことがわかり，ソフトウェアに対して変更を行う際に本システムによって修正箇所の特定を行うことは有効であると考えられる．今後は，本システムを用いることで変更作業に要する時間が短縮できることを，実際のソフトウェア開発者を被験者とした評価実験を行うことによって検証することが必要である．

6. む す び

本論文では，オブジェクト指向言語であるJava の特性を考慮した影響波及解析手法の提案を行った．本手法では，クラスのメンバ間の関係を表現する 2 種類のグラフを定義し，その上でグラフの変化の種類によって変更の影響を受けるメンバを分類し，それらの探索ルールを選択して適用できる枠組みを実現した．これにより，ユーザの目的に応じた影響の定義，抽出を行うことができる．

また，提案手法をJava 影響波及解析システムとして実装し，適用例を挙げることで有効性を検証した．

今後の課題としては，Rapid Type Analysis [3] の利用による参照変数の指すオブジェクトの型の限定や，システムを利用した評価実験などがある．

文　　献

[1] A. Krishnaswamy, “Program slicing: An application of object-oriented program dependency graphs,” Technical Report TR94-108, Department of Computer Science, Clemson University, 1994.

[2] B.G. Ryder and F. Tip, “Change impact analysis for object-oriented programs,” Proc. Workshop on Program Analysis for Software Tools and Engineering (PASTE 2001), pp.46–53, Snowbird, USA, June 2001.

[3] D.F. Bacon, Fast and Effective Optimization of Statically Typed Object-Oriented Languages, Ph.D. Thesis, Computer Science Division, University of California, Berkeley, Dec. UCB/CSD-98-1017, 1997.

[4] D. Kung, J. Gao, P. Hsia, and F. Wen, “Change impact identification in object oriented software maintenance,” Proc. International Conference on Software Maintenance, pp.202–211, Victoria, Canada, Sept. 1994.

[5] G. Rothermel and M.J. Harrold, “Selecting regression tests for object-oriented software,” Proc. International Conference on Software Maintenance, pp.14–25, Victoria, Canada, Sept. 1994.

[6] J. Gosling, B. Joy, and G. Steele (著), 村上 雅章 (訳), The Java 言語仕様, 1997.

[7] K. Rangarajan, P. Eswar, and T. Ashok, “Retesting C++ classes,” Proc. Ninth International Software Quality Week, San Francisco, USA, May 1996.

[8] L. Li, and A.J. Offutt, “Algorithmic analysis of the impact of changes on object-oriented software,” International Conference on Software Maintenance (ICSM ’96), pp.171–184, Monterey, USA, Nov. 1996.

[9] M. Weiser, “Program slicing,” Proc. 5th International Conference on Software Engineering, pp.439–449, San Diego, USA, March 1981.

[10] S. Eisenbach and C. Sadler, “Changing Java programs,” Proc. International Conference on Software Maintenance (ICSM 2001), pp.479–487, Florence, Italy, Nov. 2001.

[11] T.L. Graves, M.J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel, “An empirical study of regression test selection techniques,” Proc. 20th International Conference on Software Engineering, pp.188–197, Kyoto, Japan, April 1998.

[12] W. Hetzel, “The Complete Guide to Software Testing,” QED Information Sciences, 1984.

[13] X. Chen, W.T. Tsai, and H. Huang, “Omega - An integrated environment for C++ program maintenance,” Proc. International Conference on Software Maintenance, pp.114–123, Monterey, USA, Nov. 1996.

[14] Y.K. Jang, H.S. Chae, Y.R. Kwon, and D.H. Bae, “Change impact analysis for a class hierarchy,” Proc. Asia Pacific Software Engineering Conference (APSEC’98), pp.304–311, Taipei, Taiwan, Dec. 1998.

[15] http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javac.html, “javac - Java programming language compiler.”

[16] http://java.sun.com/j2se/1.3/, “The Java^{TM} 2 Platform, Standard Edition.”

[17] http://www.jedit.org/, “jEdit - Open Source programmer’s text editor.”

（平成 14 年 5 月 31 日受付，9 月 4 日再受付）

横森　励士　（学生員）

平 11 阪大・基礎工・情報卒．平 13 同大大学院修士課程了．現在，同大学院博士課程在学中．プログラム構造解析の研究に従事．

近藤　和弘

平 12 阪大・基礎工・情報卒．平 14 同大大学院修士課程了．同年 SONY（株）入社．現在，ソニー・エリクソン・モバイルコミュニケーションズ（株）に所属．在学中，プログラム構造解析の研究に従事．

大畑　文明

平 14 阪大大学院博士課程了．同年（株）東芝入社．現在，同社研究開発センター所属．工博．プログラム構造解析の研究に従事．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．平 14 阪大大学院情報科学研究科・教授．工博．ソフトウェア工学の研究に従事．