社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

メソッド呼び出しパターンとして現れる横断的関心事の検出

三宅　達也† 　石尾　隆† 　谷口　考治† 　井上　克郎†

† 大阪大学大学院情報科学研究科
〒 560–8531 大阪府豊中市待兼山町 1–3
E-mail: †{t-miyake,ishio,kou-tngt,inoue}@ist.osaka-u.ac.jp

あらまし　アスペクト指向プログラミングは，複数のモジュールに横断して出現する横断的関心事を各々のモジュールから分離し，新しいモジュール単位「アスペクト」として記述する．しかし，横断的関心事はソフトウェアの様々な部分に分散して存在するため，それらをもれなく発見することは困難である．このような問題に対する技術として，アスペクトマイニングが存在する．本研究では，横断的関心事検出のアプローチとしてメソッド呼び出しパターンに注目した．メソッド呼び出しパターンとは，ソースコードの複数箇所に出現する類似した構造をもつコード記述のことであり，横断的関心事の候補と考えられる．メソッド呼び出しパターン抽出ツールを実装し検出したパターンが横断的であるかどうか検討を行った．その結果，メソッド呼び出しパターンとして検出される横断的関心事が存在することを示し，それらの横断的関心事には均一性横断的関心事だけでなく非均一性横断的関心事も含まれていることを確認した．

キーワード　アスペクト指向プログラミング, アスペクトマイニング, 横断的関心事, メソッド呼び出しパターン

Detection of Crosscutting Concerns Using Method Call Patterns

Tatsuya MIYAKE†, Takashi ISHIO†, Koji TANIGUCHI†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1–3 Machikaneyama-cho, Toyonaka, Osaka, 560–8531 Japan
E-mail: †{t-miyake,ishio,kou-tngt,inoue}@ist.osaka-u.ac.jp

Abstract　Aspect-Oriented Programming introduces a new software module unit named aspect for encapsulating crosscutting concerns. However, it is hard to find crosscutting concern code completely since its implementation spreads across many different modules. In this paper, we focus on method call patterns for detecting corsscutting concerns. A method call pattern is a sequence of method calls to imlement a particular kind of concern. A pattern is a candidate of a crosscutting concern since many instances of a pattern spread across the whole system. We applied a sequentiall pattern mining to detect method call patterns as crosscutting concern candidates and inspected the patterns extracted from a Java software. The experiment shows that method call patterns include not only homogeneous but also heterogeneous crosscutting concerns.

Key words　Aspect-Oriented Programming, Aspect Mining, Crosscutting Concern, Method Call Pattern

1. はじめに

近年，プログラミング技法としてオブジェクト指向プログラミングが一般的に普及している．オブジェクト指向プログラミングが現在一般的に利用されている理由の 1 つとして，強力なモジュール化機能によるプログラムの保守性や拡張性の高さがある．しかし，オブジェクト指向言語を用いれば完全なモジュール化が行えるというわけではない．例えば，ロギングや同期処理のように複数のモジュールに横断する単一の処理（横断的関心事）が存在し，プログラムの保守性や拡張性に悪影響を及ぼしている．このような問題を解決するために，より高度なモジュール化機能を提供する技法として，アスペクト指向プログラミング [1] が提案されている．アスペクト指向プログラミングの基本的な考え方は，横断的関心事のような複数のモジュールに分散してしまう処理を,「アスペクト」と呼ばれる単一のモジュールとして一箇所にまとめて記述し，後からコンパイラなどのツールを用いてプログラムを結合するというものである．アスペクトにより横断的関心事を一箇所に記述すること

ができるため，横断的関心事の処理を変更することが容易となる．また各モジュールから横断的関心事にかかわる記述が消えるため，各モジュールの理解も容易になる．

アスペクト指向プログラミングはより高度なモジュール化を可能とするが，既存のプログラムをアスペクト指向言語を用いて書き直すのは非常にコストがかかる．その理由の 1 つは，アスペクトにすべき複数の箇所に分散した処理を，見つけ出すことが容易ではないからである．この問題を解決するため，アスペクトマイニングと呼ばれる，既存のソフトウェアからアスペクトを機械的に抽出するための技術の研究が行われている [2]〜[4]．

本研究では，ソースコードのメソッド呼び出しパターンを抽出することにより，横断的関心事がメソッド呼び出しパターンとして現れているか Java アプリケーションの 1 つ JHotDraw を対象に調査を行い，各メソッド呼び出しパターンの特徴について評価を行なった．メソッド呼び出しパターンとは，ソースコードの複数箇所に出現する類似した構造をもつコード記述のことである．調査の結果，横断的関心事がモジュール化できないためにメソッド呼び出しパターンとしてソースコード中に現れているものが存在し，それらの横断的関心事には均一性横断的関心事だけでなく非均一性横断的関心事も存在していることを確認した．

以降，2 節ではアスペクト指向プログラミングおよびアスペクトマイニングについて説明し，3 節でメソッド呼び出しパターンとその検出方法について述べる．4 節では実験概要と抽出したメソッド呼び出しパターンの評価を行なう．最後に，5 節で本研究のまとめと今後の課題について述べる．

2. アスペクト指向プログラミング

2.1 アスペクト指向の特徴

アスペクト指向プログラミングは，オブジェクト指向プログラミングなどの従来のモジュール機構の弱点を補うプログラミング手法である．アスペクト指向プログラミングは横断的関心事を分離・記述するためのモジュール単位「アスペクト」を導入する．

アスペクト指向プログラミングでは，横断的関心事に関する処理を明示的に呼び出すのではなく，アスペクトが挿入されるポイントをソースプログラム内から選択し，コンパイル時にアスペクトをコード内に織り込む．このような仕組みを実現するためのアスペクト指向特有の概念として以下のようなものが存在する．

ジョインポイント　オブジェクト指向プログラムにおけるアスペクトを挿入することができるソースコード上の位置，あるいは実行時の特定のタイミング．

ポイントカット　実際にアスペクトを挿入するジョインポイントの集合．

アドバイス　ポイントカットによって識別された実行時点の前または後に挿入されるコード，もしくは，ポイントカットによって識別された実行時点において本来実行されるコードのかわりに実行されるコード．

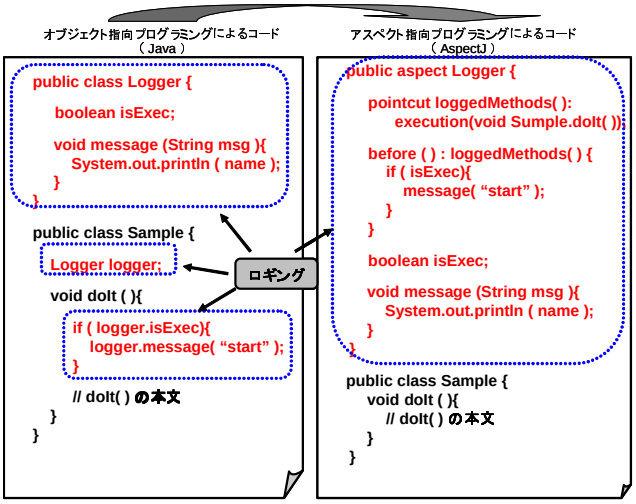

図 1　ポイントカットとアドバイスを用いたコード書き換えの例

オブジェクト指向言語 Java [5] で書かれたコードを上記の概念を用いてアスペクト指向言語 AspectJ [6] のプログラムに書きかえた例を図 1 に示す．この例におけるロギングは Sample クラスの doIt( ) メソッドの本来の処理の実行前に，ロギングを行うかどうかを確かめ，必要であればロギング用メソッドを呼び出し，メソッドの実行を記録している．Java による実現ではロギングに関するクラスを作成し基本的にはそのクラスがロギングの処理を担当しているが，Sample クラス内にもロギングに関するコード（点線枠で囲んだ部分）が分散してしまっている．一方，アスペクトを用いたコードでは，Sample.doIt() の実行時点を示すポイントカット loggedMethods を作成し，実行時点の直前（ジョインポイント）にロギングを織り込むために，before アドバイスとして記述している．Logger アスペクトが Sample クラスを参照してはいるが，ロギングに関するコードはすべて Logger アスペクト内に記述しているため，ロギングの処理を修正・拡張する場合は Logger アスペクト内みを変更すればよい．

2.2 横断的関心事の分類

横断的関心事は均一性と非均一性の 2 つに分類される [7]．

均一性横断的関心事 (Homogeneous crosscutting concern)

出現箇所は横断的であるがそれぞれのコード記述には均一性がある横断的関心事を意味する．例えば，単一のメソッド呼び出しや，同じコードをコピーして実現するような処理はこれに分類される．均一性横断的関心事を管理する場合，複数のジョインポイントで同じアドバイスを動かすことができるためアスペクト化が容易である．

非均一性横断的関心事 (Heterogeneous crosscutting concern)

出現箇所が横断的であるが，出現する場所ごとにコード記述が異なる横断的関心事を意味する．非均一性横断的関心事は，単一のアドバイスでは実装できず，アスペクトでの置き換えは困難である．

2.3 アスペクトマイニング

アスペクトマイニングは，既存のオブジェクト指向プログラムを入力として，その中からアスペクトとすべき候補を発見する手法である．

アスペクトの抽出を行なう際に，最もコストのかかる作業はアスペクトの候補を見つけ出す作業である．キャッシング機能を実現するアスペクトなどのように一部の例外はあるが，基本的にアスペクトの候補として挙げられるものは複数のオブジェクトに散らばっている．しかし，ある機能に関するコードがプログラムの複数の箇所に散らばっているかどうか，またそれがどこに散らばっているのかは詳細な仕様書などがない限り，プログラムの全ての箇所を調べなければわからない．当然，そのような作業には膨大な時間とプログラム全体に対する理解が必要になり容易には行なえない．アスペクトマイニングは，ソースコードの情報からアスペクトの候補を自動的に抽出するプログラム解析手法である．

これまでの研究では，次のようなアプローチが試みられている．

a） コードクローンをアスペクトの候補とする [2]

ある程度のサイズの定型的な処理が複数箇所にわたって現れる場合，それらのコード片はコードクローンとして検出することができる．検出されたクローンのうち，同一クラスやクラス階層に分散したものであれば既存のオブジェクト指向プログラムでのリファクタリングを適用すべきだが，クラス階層に関係なく分散したコードはアスペクトとして抽出すべきである可能性が高い．この手法は，コードクローンの性質上，ある程度まとまったサイズの処理でなければ発見できないという弱点がある．

b） 特定のキーワードを含む文をアスペクトの候補とする [3]

特定のメソッド呼び出しがアスペクトの候補だとわかっている場合，grep などを使って全ての呼び出し箇所を探索する方法が有力である．キーワード検索がどのディレクトリ，ファイルから発見されたかを調べることで，システムの中でどの程度関連したコードが分散しているかを知ることができる．キーワードの検索結果があまりに複数のモジュールに分散しているようであればアスペクトとして一箇所にまとめたほうがよいといえる．字句情報だけでなく意味情報も同時に使用するようにすればかなり精度は高まる．しかし，それでも偶然の一致が生じる可能性はあり，注意する必要がある．

c） メソッドごとの Fan-In をベースに見つける [4]

Fan-In とは，あるメソッドが何箇所から呼ばれているかという情報である．この手法は，メソッドごとにどこから呼び出されているかを列挙していき，呼び出されている数が多いものから順にアスペクトの候補にならないか調べていくアプローチである．List や Arrays などのユーティリティメソッドを除外する必要があるが，どのメソッドがアスペクトになりそうかわからない場合には有効な手段である．

これらのアスペクトマイニング手法では，主に均一性横断的関心事を抽出対象としている．

3. メソッド呼び出しパターン

3.1 メソッド呼び出しパターンの特徴

メソッド呼び出しパターンとはソースコードに頻繁に出現する構造のよく似たコード記述である．具体的には，メソッド呼び出し，条件分岐とループというメソッドに含まれる「特徴」の列として表現される．メソッド呼び出しパターンを閲覧することにより以下の情報を得ることができる [8].

a） 実現したい処理に必要なメソッド群

ソフトウェア開発において，1 つの機能は複数のメソッドの組み合わせからなることが多く，それらを一箇所にまとめて書くことが一般的となっている．このため，メソッド呼び出しパターンには一連の処理に必要なメソッド呼び出しが含まれているために，これを閲覧することで処理に必要なメソッド群を理解することができる．

b） 関連のあるメソッドの呼び出し順

1 つの機能を実現するメソッド群の呼び出し順序にも一定の決まりがある．ある部品において，その部品の初期化作業を行なうメソッドは当然一番初めに呼び出さなければならないし，あるメソッドによって得られる値を参照するメソッドはそのメソッドより後に呼び出さなければならない．メソッド呼び出しパターンは単なるメソッド名の列挙ではなく，条件分岐やループといった制御構造の情報を持ったものであるため，そこに現れるメソッド呼び出しの順序を閲覧することで具体的な呼び出し順を知ることができる．

3.2 メソッド呼び出しパターンの抽出方法

本研究では既存の研究で提案された関数呼び出しパターン抽出手法 [8] を利用して Java 言語を対象としたメソッド呼び出しパターンの抽出を行なう．この手法は大きく分けて 3 つのステップに分類される．まず始めに行なわれるのがソースコードの特徴シーケンスを取得することである．次に，取り出されたソースコードの特徴シーケンスから不要なものを除去する．最後に，生成された特徴シーケンスに対して sequential pattern mining と呼ばれる手法を適用することでメソッド呼び出しパターンを抽出する．以降，各ステップの詳細を述べる．

3.2.1 特徴シーケンスの抽出

ソースコードの各メソッドに対し，ソースコードの特徴シーケンスと呼ばれるものを抽出する．特徴シーケンスとは特徴の利用例を抽象化したものであり，具体的には，1 つのメソッド定義内におけるソースコードの特徴のリストである．ソースコードの特徴とは以下のものである．

- メソッド呼び出し．特徴として抽出されたメソッド呼び出しは，次の 2 通りの方法で同一の特徴であるかどうか判断する．
  - オブジェクト指向言語では，メソッドはクラスごとに定義されるので，メソッドの所属するクラスとメソッド名の両方が一致すれば同一の特徴であると判断する．
  - 横断的関心事に関連するメソッドは，呼び出しだけでなく定義自体も横断的である場合があるので，クラスが違っていてもメソッド名が一致していれば同一の特徴であると判断する．

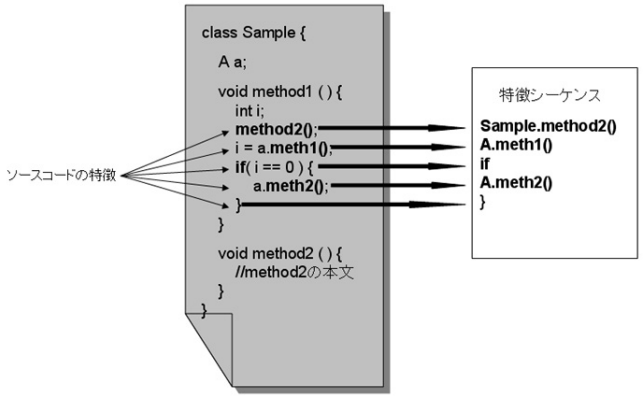

図 2　特徴シーケンスの生成

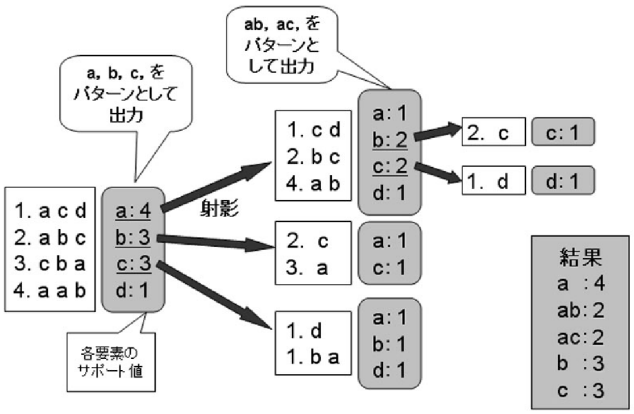

図 3　最小サポート値 2 における PrefixSpan を用いた sequential pattern mining

- 条件文の開始・終了位置.
- 繰り返し文の開始・終了位置.

プログラムの開発はある機能を実装したメソッドを複数組み合わせることで，より大きな機能を実現していく．メソッド呼び出しはそのメソッドがどういったことを実現しようとしているかを知るための重要な手がかりであり，ソースコードの特徴であるといえる．

しかし，メソッド呼び出しを組み合わせて機能を実現する際，単純にメソッド呼び出しを並べるだけで実現できるということは少ない．メソッドの返り値によって処理を変えたり，処理が終わるまで同じメソッド呼び出しを繰り返すといったことがよく行なわれる．ゆえに，条件文の開始・終了位置や繰り返し文の開始・終了位置といったような，プログラムの制御構造を決定するような要素もソースコードの特徴であるといえる．

図 2 は Sample クラスの method1( ) メソッドから生成した特徴シーケンスである．

3.2.2　Sequential patern mining によるパターン抽出

生成した特徴シーケンスを対象にして sequential pattern mining [9] とよばれる手法を適用することで，メソッド呼び出しパターンの抽出を行なう．

sequential pattern mining とは与えられた複数のリストから，ユーザが指定した閾値以上の頻度で共通して現れる部分リストを求める手法である．sequential pattern mining を行なうアルゴリズムは複数あるが，本研究では，一般的に用いられることが多い PrefixSpan [10] を採用した．

PrefixSpan アルゴリズムを使用した sequential pattern mining の流れは次のようになる．

（1）　各シーケンスを構成している全ての要素について，そのサポート値を計算する．サポート値とは要素が出現しているシーケンスの数である．

（2）　サポート値が最小サポート値以上の要素をシーケンシャルパターンとして出力する．

（3）　最小サポート値を超える各要素について射影を行なう．射影とは，全てのシーケンスから特定の要素からの接尾辞を取り出す操作である．

（4）　（3）で射影を行なったシーケンスに対して，再び（1）と同じ作業を行なう．

（5）　サポート値が最小サポート値以上ならば，その要素を 1 つ前に出力したパターンの末尾につけたものをシーケンシャルパターンとして出力する．(3）において新たな接尾辞が取り出されなくなるか，(4）において最小サポート以上の値を持つ要素が存在しなくなるまで（3）〜（5）を繰り返す．

図 3 に PrefixSpan による sequential pattern mining をおこなった場合の様子を示す．ここでは，パターン抽出の閾値である最小サポート値を 2 としてマイニングを行なっている．

4.　実験と評価

本節では，実装したツールを用いて抽出したメソッド呼び出しパターンを閲覧することにより，横断的関心事の中には，うまくモジュール化できないためにメソッド呼び出しパターンとして出現するものがあるかどうかを調べ，そのような横断的関心事に関連するメソッド呼び出しパターンの特徴に対する評価を行なう．

4.1　実 験 概 要

メソッド呼び出しパターンの抽出対象としては JHotDraw [11] を用いた．JHotDraw は図形描画アプリケーションの 1 つで，アスペクトマイニング技術の実験対象として用いられている [4]．ソースコードの総行数は約 18000 行（テストファイルは除外），総メソッド数は約 2900 である．

実際に抽出ツールを用いて，最小サポート値 4，最小パターン長 4 でメソッド呼び出しパターンを抽出したところ，特徴として抽出されたメソッド呼び出しの属するクラスを区別した場合，抽出時間は約 11 秒，抽出パターン数は 38 種類であった．また，クラスを区別せずメソッド名のみで特徴が同一であるかどうか判断した場合，抽出時間は 17 秒，抽出パターン数は 55 種類であった．

4.2　抽出されたメソッド呼び出しパターンと横断的関心事の関連性

メソッド呼び出しの属するクラスを区別して抽出したメソッド呼び出しパターンを手作業で検討した結果，横断的関心事に関連したパターンと関連していないパターンにはそれぞれどのような特徴があったかについて述べる．クラス名を区別せず抽

表 1　抽出されたパターンの関心事の例

パターンが関連する関心事	Support	Class
ループ処理のイディオム	54	31
コマンドの取消	14	14
コマンド取消用の情報保存	12	12
図形の選択解除	10	10
図形の選択	9	9
ループ処理のイディオム	8	8
コマンド実行の影響チェック	6	6
マウス操作	6	6
ポリゴン処理	6	1
イメージの描画	6	1

出した場合の差異については 4.3 節で述べる．

表 1 に抽出されたメソッド呼び出しパターンがどのような関心事に関連していたかの例を示す．表の各項目のうち，“ パターンの関心事 ”はそのパターンが関連している関心事を，“ Support ”はそのパターンの出現回数を，“ Class ”はそのパターンが出現するクラスの数を示す．“ Support ”の値が高くても“ Class ”の値が 1 であれば，そのパターンが関連する関心事は 1 つのクラスに集約されているため横断的でない．

4.2.1　横断的関心事に関連したパターン

抽出されたメソッド呼び出しパターンは全部で 38 種類であり，そのうち 22 種類が横断的関心事に関連するパターンであった．横断的関心事に関連したメソッド呼び出しパターンの多くが次のような特徴を持っていた．

- パターンの要素であるメソッド呼び出しのメソッド名に，特定のトークンが多数存在していた．例えば，Undo 機能に関連したメソッド呼び出しパターンの場合，要素に getUndoManager()，pushUndo() などを持ち，パターンの要素であるメソッド呼び出しのメソッド名に undo というトークンが多数存在していた．
- パターンの実現する機能がパターンが出現するメソッドの機能と一致している場合，パターンが出現するメソッドの名前に特定のトークンが多数存在していた．例えば，パターンの出現するメソッドの名前が mouseDown，mouseUp などであるパターンの場合，パターンの出現するメソッドの名前に mouse というトークンが多数出現しており，このことからマウス操作に関連するメソッドが複数のクラスで宣言されていることがわかる．

4.2.2　横断的関心事に関連していないパターン

抽出したメソッド呼び出しパターンの中には，複数のクラスに出現しているが，横断的関心事とは関連がないパターンも存在した．図 4 に例を示す．

図 4 のメソッド呼び出しパターンは 57 箇所で出現しており，非常に出現頻度の高いパターンである．しかし，このパターンに含まれる要素は繰り返し処理の制御を行なうメソッド呼び出しのみであり，実際に繰り返し行なわれる処理に関連するメソッド呼び出しは含まれていない．このため，このパターンに関連するコードが実現する機能は，繰り返し処理の過程で呼び出されるメソッドに応じて様々であり，一定ではない．

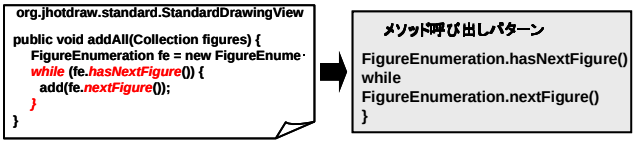

図 4　横断的関心事に関連していないパターンの例

このように，メソッド呼び出しパターンの中には，何らかの関心事を実現するための機能ではなく，それらを実現するためのより小さな機能群のうちの 1 つに関連したパターンが存在した．このようなパターンが実現する機能は，機能単位が小さすぎて特定の横断的関心事に関連したパターンであるとはいえない．

4.3　クラスの区別による影響

4.3.1　特定のクラス階層に出現するパターン

特定のクラス階層に現れるメソッド呼び出しパターンの中には，クラス階層内の全てのクラスに出現するパターンと，クラス階層内の一部のクラスのみに出現するパターンが存在した．例えば，Undo 機能に関するパターンは AbstractCommand のほとんどのサブクラスに出現しているが，CopyCommand などの一部のコマンドには出現しない．

このようなパターンが抽出された場合，パターンが出現するクラスのみを要素とした新たなクラス階層の提案を行なうことができる．例えば AbstractCommand のサブクラスに UndoableCommand などを作成し，Undo 機能に関連したパターンが出現しているクラスは UndoableCommand のサブクラスとする．こうすることにより横断的関心事である Undo 機能が AbstractCommand のどのサブクラスに散らばっているかをクラス階層の情報として明示することができる．

また，同じクラス階層内のパターンが抽出されなかったクラスは，実際に横断的関心事に関連していないのか，横断的関心事の実装をし忘れているのかの確認の対象になりえる．

4.3.2　メソッド名のみの一致で抽出したパターン

メソッド呼び出しパターンの抽出過程において，クラス名の区別を行わず，メソッド名が同じであれば同一の特徴と判断してメソッド呼び出しパターンを抽出した結果，クラス名を区別した場合には得られないメソッド呼び出しパターンが抽出された．例を図 5 に示す．

図 5 のメソッド呼び出しパターンの要素である setUndoActivity()，getUndoActivity() は AbstractCommand，AbstractHandle，AbstarctTool の 3 つの抽象クラスでそれぞれ定義されており，createUndoActivity() は 3 つの抽象クラスのサブクラスごとに定義されている．これらは同一の横断的関心事 Undo に関連したメソッドであり，AbstractCommand のサブクラスで 11 箇所，AbstractHandle のサブクラスで 6 箇所，AbstarctTool のサブクラスで 9 箇所，計 26 箇所で出現する．クラス名とメソッド名の両方を考慮した場合，それぞれのメソッドは異なる特徴として判断される．メソッド名のみで判断した場合，これらのメソッドが所属するクラスが異なっていても同一のメソッドと判断され，図 5 のメソッド呼び出しパター

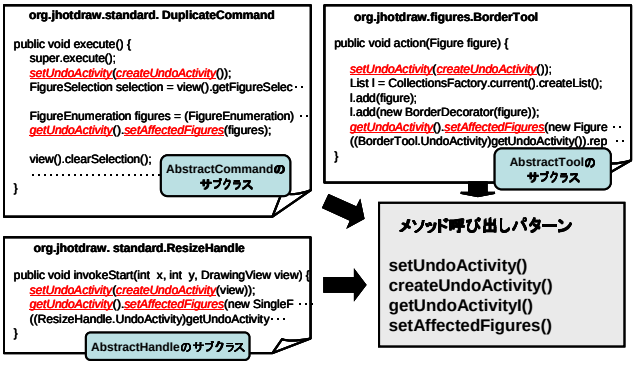

図 5　メソッド名のみの一致で抽出したパターン

ンを確認できた．

4.4 考　　察

既存の手法の多くは均一性横断的関心事の抽出に優れているが，非均一性横断的関心事の抽出には向いていない．例えば，コードクローンを利用した手法 [2] では，図 5 のように，機能が同じであってもコードに均一性がない場合，クローンであると判断されないため抽出することができない．また，非均一性横断的関心事は複数のメソッド呼び出しで実現されるため，単一のメソッドに対する Fan-In を利用した方法 [4] では，非均一性横断的関心事の要素であるメソッドを別々に抽出することはできても，それらの関連性を知ることはできない．

メソッド呼び出しパターンを用いると，コードが完全に一致していなくても，メソッド呼び出しの順序が一定であれば抽出することができるため，非均一性横断的関心事もある程度抽出することができる．非均一性横断的関心事はアスペクトを用いてモジュール化が難しいため [12]，アスペクトマイニングでは抽出対象としない場合もあるが，実装コードの出現位置を把握することは有益である．

5. む　す　び

アスペクト指向プログラミングは，複数のモジュールを横断した関心事をアスペクトという単位でモジュール化することができる．しかし，横断的関心事はソフトウェア内に分散して存在している．本研究は，メソッド呼び出しパターンの抽出手法を用いて，横断的関心事検出することを目的としている．

この目的を達成するため，Java プログラムに対してメソッド呼び出しパターンを抽出するツールの実装を行い，抽出したメソッド呼び出しパターンがアスペクトとして記述すべき横断的関心事に関連するかどうか調査した．その結果，メソッド呼び出しパターンの中には横断的関心事に関連するパターンが存在し，それらの中には非均一性横断的関心事も含まれていることを確認した．また，横断的関心事には，メソッドの呼び出しだけでなく，定義も横断的であるものが存在することを考慮し，あえてメソッドの属するクラスを識別せず，メソッド名が同じであれば同一のメソッド呼び出しであると判断することにより，さらに広範囲から横断的関心事を検出することに成功した．

これらのメソッド呼び出しパターンから得られる情報は，横断的関心事の検出だけでなく，性質の分類や横断的関心事の実装を忘れているクラスの提示，新たなクラス階層の提案などにも利用できると考えられる．

今後の課題としては，アスペクトとして記述すべき横断的関心事に関連するメソッド呼び出しパターンの特徴を利用した，アスペクトマイニング手法の提案および実装が挙げられる．また，本研究で抽出したメソッド呼び出しパターンには横断的関心事に関連していないものも存在する．このようなパターンの特徴を発見し，除去する手法を確立する必要がある．

文　　献

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier and J. Irwin: Aspect Oriented Programming. In *Proceedings of Europian Conference on Object-Oriented Programing(ECOOP 97)*, vol.1241 of LNCS, pp.220-242, 1997

[2] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwe :On the Use of Clone Detection for Identifying Crosscutting Concern Code. *IEEE Transactions on Software Engineering, Vol.31, No.10, pp.804-818, October 2005.*

[3] W.G. Griswold., Y. Kato and J.J. Yuan :Aspect browser: Tool support for Managing Dispersed Aspects. Technical Report CS99-0640, Department of computer Science and Engineering, University of California, San Diego. 1999.

[4] M. Marin, A. van Deursen, and L. Moonen :Identifying Aspects using Fan-In Analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering(WCRE 2004)*, pp.132-141, 2004.

[5] Java Technology. http://java.sun.com/

[6] The AspectJ Project. http://aspectj.org/

[7] A. Colyer and A. Clement: Large-Scale AOSD for Middleware. In *Proceedings of the 3rd International Conference on Aspect-Oriented Software Development(AOSD 2004)* , pp.56-65,Lancaster, UK, March 2004.

[8] 中山崇，松下誠，井上克郎，:ソースコードの差分を用いた関数呼び出しパターン抽出法の提案. 情報処理学会研究報告，Vol.2006, No.35，2006-SE-151，pp.49–56，2006

[9] R. Agrawal and R. Srikant: Mining Sequential Patterns. In *Proseedings of the 11th International Conference on Data Engineering(ICDE 95),pp.3-14*, Washington,DC,USA,March 1995. IEEE Computer Society.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu: PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In *Proceedings of the 17th International Conference on Data Engineering(ICDE 2001),pp.215-224*, Washington,DC,USA,April 2001. IEEE Computer Society.

[11] JHotDraw as Open-Source Project. http://www.jhotdraw.org/

[12] S. Apel, T. Leich, G. Saake: Aspectual Mixin Layers: Aspects and Features in Concert. In *Proceeding of the 28th International Conference on Software Engineering(ICSE 2006), pp.122-131*, Shanghai, China, May 2006