特別研究報告

題目

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特徴評価

指導教員
井上 克郎 教授

報告者
三宅達也

平成 19 年 2 月 20 日

大阪大学 基礎工学部 情報科学科

平成 18 年度 特別研究報告

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特徴評価

三宅達也

内容梗概

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」として記述することで，保守性や再利用性，拡張性を向上させる．

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．アスペクトマイニングとは既存のオブジェクト指向プログラムからアスペクトとして記述することが効果的だと推測される一連のコードを自動的に抽出ための手法である．

本研究では，アスペクトマイニング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る．類似したコード記述が複数箇所に出現するということは同種の機能が複数箇所に記述されていることを意味する．これは特定の関心事に関する記述が複数のモジュールに出現するという横断的関心事の特徴に該当している．このことから，メソッド呼び出しパターン抽出手法をアスペクトマイニングに応用できると考えた．

この考えの妥当性を確認するため，オブジェクト指向言語 Java を対象に，既存のアルゴリズムを用いてメソッド呼び出しパターン抽出ツールを実装し，抽出したメソッド呼び出しパターンに，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あるかどうか，評価を行った．その結果として，メソッド呼び出しパターンとして出現する横断的関心事の存在が確認され，それらのメソッド呼び出しパターンのパターン長が長いものほど横断的関心事に関連していることがわかった．

主な用語

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

目 次

1 まえがき **4**

2 アスペクト指向プログラミング **5**

2.1 アスペクト指向の特徴 5

2.2 アスペクトマイニング 9

3 メソッド呼び出しパターン **11**

3.1 メソッド呼び出しパターンの特徴 11

3.2 メソッド呼び出しパターンの抽出方法 11

3.2.1 ソースコードの特徴抽出 12

3.2.2 特徴シーケンスの生成 14

3.2.3 Sequential patern mining によるパターン抽出 14

3.2.4 抽出されたパターンのフィルタリング 15

4 実装方針 **18**

4.1 コード情報抽出部 18

4.2 メソッド呼び出しパターン抽出部 20

4.2.1 特徴抽出部 20

4.2.2 特徴シーケンス生成部 20

4.2.3 sequential pattern mining 実行 21

4.3 GUI 21

4.3.1 メソッド呼び出しパターン抽出設定変更部 22

4.3.2 メソッド呼び出しパターン情報表示テーブル 22

4.3.3 ソースビューワー 23

4.3.4 クラス階層ツリー 23

5 抽出したメソッド呼び出しパターンの評価基準 **25**

5.1 パターン自体の評価基準 25

5.1.1 パターン長 25

5.1.2 サポート値 25

5.1.3 パターンが出現するクラス数 27

5.2 パターンの要素であるメソッドに関する評価基準 27

6　評価 28

6.1　メソッド呼び出しパターンの評価 29

6.1.1　パターン長に注目した評価 29

6.1.2　サポート値に注目した評価 30

6.1.3　メソッド呼び出しパターンとその関心事 31

6.2　パターンの要素であるメソッドに関する評価 34

6.3　メソッド呼び出しパターンとアスペクトの関連性 35

6.4　アスペクトマイニングツールとして利用するには 36

6.5　既存手法との比較 36

7　まとめ 39

謝辞 40

参考文献 41

1　まえがき

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，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「アスペクトマイニング [2]」の研究が行われている [3, 4, 5, 6]．この研究では，アスペクトと判断できる処理には，何らかのパターンに従ってコーディングされているという考え方があり，異なるパターンに着目した研究が行われている．

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．その結果，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，中でもメソッド呼び出しパターンの長さが長いものに対して，アスペクトを用いた書き換えを適用しやすいということが判明した．

以降，2 節ではアスペクト指向プログラミングに関してよ詳しく説明し，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 節ではメソッド呼び出しパターンの評価基準について説明し，5 節ではメソッド呼び出しパターンの検出方法の実装について述べる．6 節では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，7 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．オブジェクト指向プログラミングでは，システムに必要な機能を個々のオブジェクトに分割することにより，ソフトウェアの保守性や再利用性を高めている．

しかし，オブジェクトという単位でシステムの機能を分担する場合，図 1 のロギングや例外処理などのように担当するオブジェクトを 1 つに決めることができず複数のオブジェクトに横断してしまう機能が存在する．このようにオブジェクト指向プログラミングにおいて複数のオブジェクトを横断してしまう処理は横断的関心事と呼ばれ，この横断的関心事が存在するソフトウェアには次のような問題が発生する．

- 横断的関心事の仕様を変更する際，複数の場所に散らばっているコードの全てを探し出して変更を加えなければならない．
- オブジェクトを再利用する際，そのオブジェクトに横断的関心事が混ざっていると，横断的関心事に関連するコードを除去するか，横断的関心事が関連しているオブジェクトを全て再利用する必要がある．
- 横断的関心事のみをオブジェクトから独立して再利用することができない．

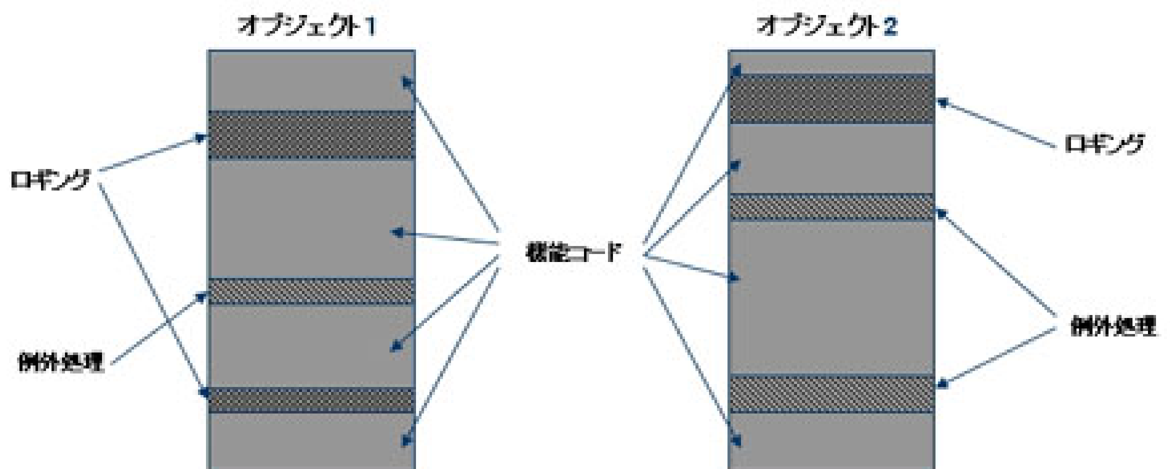

図 1: オブジェクト指向プログラミングにおける横断的関心事

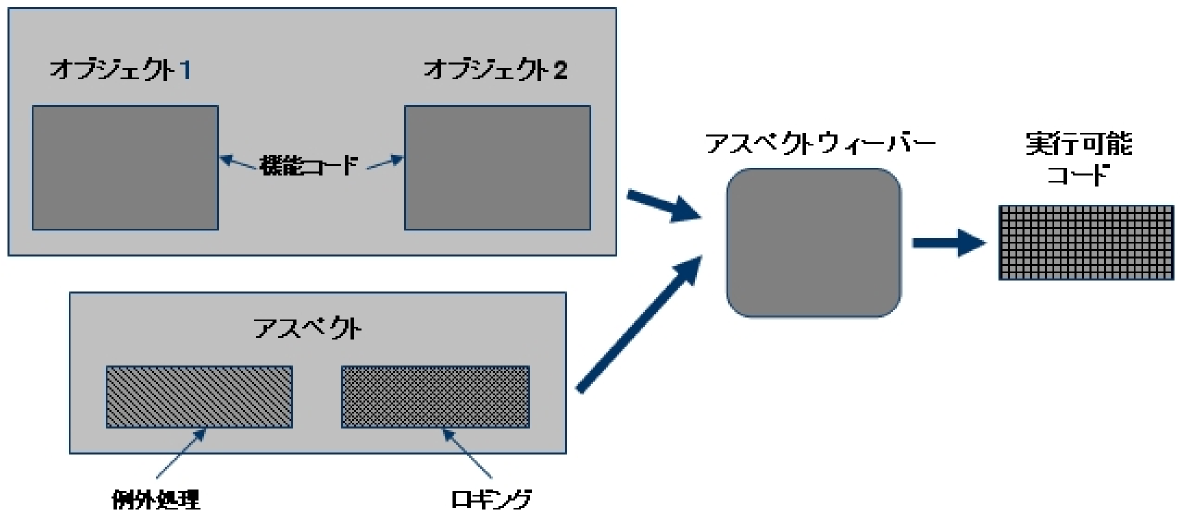

図 2: アスペクト指向言語プログラミングの流れ

このような問題に対する答えの1つがアスペクト指向プログラミングであり，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．図1で表されたオブジェクト指向プログラムはアスペクト指向プログラミングを利用すれば図2のように書き換えることができる．

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，図2のように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．言語処理系によって違いはあるが，たとえば次のようなプログラムの実行時点がジョインポイントとして定義される．

- オブジェクトに対するメソッド呼び出し
- オブジェクトのメソッドの実行
- オブジェクトの持つフィールドへのアクセス
- 例外の発生

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．

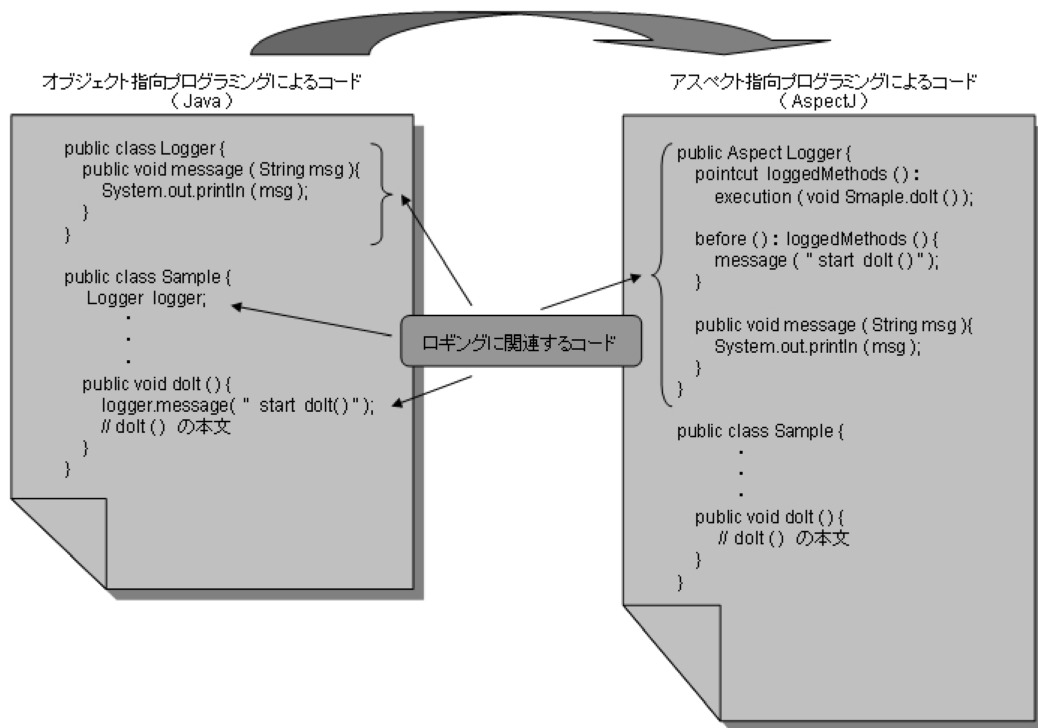

図 3: ポイントカットとアドバイスを用いたコード書き換えの例

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．

オブジェクト指向言語 Java [7] 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 [8] のプログラムに書きかえた例を図 3 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 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，ロギングの処理を変更する場合は Logger アスペクト内みを変更すればよい．

また，Aspect プログラミングでは他のクラスやインターフェイスのメンバを，アスペクトの中で宣言することもできる．これをインタータイプ・メンバ宣言という．

図 4 はインタータイプ・メンバ宣言を用いたコード書き換えの例である．本来は Sample

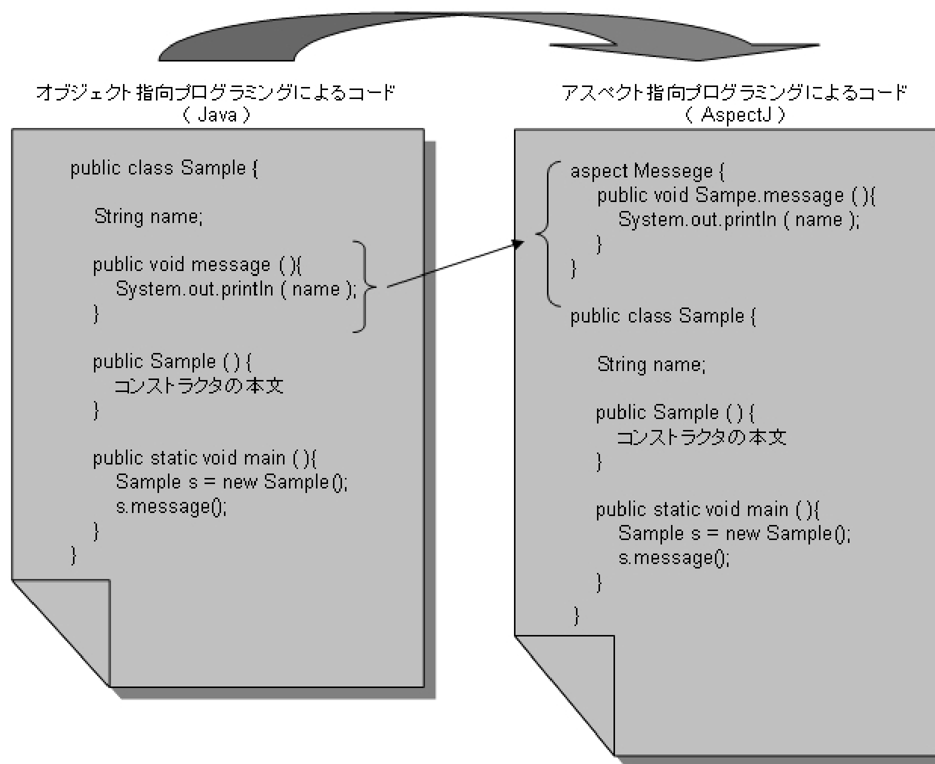

図 4: インタータイプ・メンバ宣言を用いたコード書き換えの例

クラスのメンバである message メソッドが，Message アスペクトとして定義がされているので，この書き換えの前後の 2 つの Sample クラスは同じ機能を有する．

図 3，図 4 のコードからもわかるようにアスペクト指向プログラミングでは横断的関心事をアスペクトとして一箇所にまとめて記述することができる．これには次のような利点があり，横断的関心事がもたらす問題を解決する．

- 横断的関心事の仕様が変更されても，関連するコードは全て単一のアスペクトに記述されているため，変更漏れや変更の波及も認識しやすい．
- オブジェクトから横断的関心事を取り除くことで，オブジェクトにはその本質的な機能だけを記述することができるようになる．これによって保守性や可読性，再利用性が向上する．
- オブジェクト指向プログラミングでは横断的関心事の再利用は困難であったが，アスペクト指向プログラミングではポイントカットのみを変更することによって，他の場所でも再利用することができる．

アスペクトの用途については，これまでに様々な研究が行なわれており，例外処理やデザインパターンの置き換え，計算結果の再利用やオブジェクト生成の管理などの利用法が知られている [9].

2.2 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．

コードクローンをアスペクトの候補とする [3] 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．

特定のキーワードを含む文をアスペクトの候補とする [4] 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．

メソッドごとの Fan-In をベースに見つける [5] 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，メソッドごとに誰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，どのクラスか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．

メソッドの一番最初または最後に呼ばれているメソッドを調べる [6] この手法は，メソッド一番最初または最後に呼びれているメソッドに注目し，多数のメソッドの先頭で呼ばれているメソッドほどアスペクトである可能性が高いと判断している．

アスペクトマイニングで得られたアスペクト候補は開発者がアスペクトへと書き換える．そのための手順としてアスペクト指向リファクタリングが提案されており [10]，その自動化も研究されている [11]．

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閲覧することにより以下の情報を得ることができる [12].

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 つの機能は複数の関数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関数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関数群を理解することができる．

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関数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関数は当然一番初めに呼び出さなければならないし，ある関数によって得られる値を参照する関数はその関数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．

メソッドの引数や返り値の扱い方 メソッド呼び出しパターンは，ソフトウェア部品の典型的な利用法を抽象化して取り出したものであるため，メソッド呼び出しパターンに対応したソースコードを閲覧することで，引数の指定の仕方や返り値の扱い方といった詳細なコーディングの仕方を学習することができる．

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メソッド呼び出しパターン抽出手法 [12] 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から特徴シーケンスと本研究で呼んでいるものを生成する．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．

以下では，各ステップの詳細な説明を行なう．

3.2.1　ソースコードの特徴抽出

まず始めにソースコードからソースコードの特徴と呼ばれるものを取得する．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
- 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．よって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．

ソースコードの特徴抽出ステップでは，上記の定義した特徴をその出現順に抽出していく．この過程において注意すべき状況とそれに対する処理を以下に示す．

条件式におけるメソッド呼び出し 条件式で呼び出されるメソッドは，図 5 のように条件式の前にその戻り値を変数に格納することにより条件式外に出すことができる．よって特徴の抽出順は条件式内のメソッド呼び出しが制御文の開始要素よりも先になる．

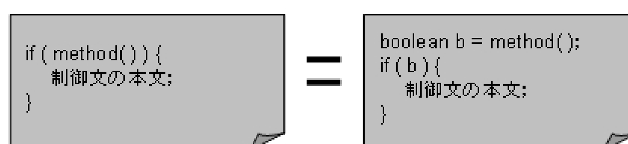

図 5: 条件式におけるメソッド呼び出しのコード例

メソッド呼び出しのネスト 引数として呼び出されたメソッドは，図 6 のようにその戻り値を事前に変数に格納し，引数には変数を指定してやることができる．ゆえに，特徴の抽出順は引数として呼び出されるメソッドのほうが，そのメソッドを引数にとるメソッドよりも先になる．

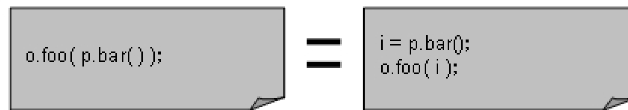

図 6: メソッド呼び出しのネストのコード例

式中に複数のメソッド呼び出しが存在する場合 式中に呼び出されるメソッドは，言語仕様が順序を定めていないので，図 7 のようにどちらのメソッドが先に呼ばれてもかまわない．本研究では，クラス名とメソッド名からなるハッシュ値の比較によって順序を一意に定め，同一の式が複数回出現した場合に呼び出し順序が一致するようにしている．

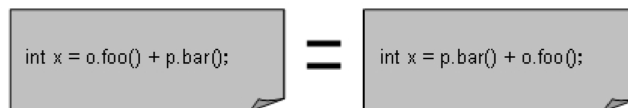

図 7: 式中に複数のメソッド呼び出しが存在する場合のコード例

オーバーライド オーバーライドされたメソッドはオーバーライドされるメソッドと比べて，コード記述には違いがあるが，実現しようとする機能は基本的に同じである．ゆえに図 8 に示している 3 つのメソッドの呼び出しは全て同一の特徴（スーパークラスのメソッド）として抽出する．

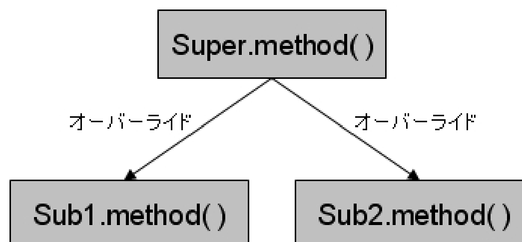

図 8: オーバーライド

オーバーロード オーバーロードされたメソッドは引数や返り値が異なり，利用方法が変わってくるので，異なる特徴として抽出する．

3.2.2　特徴シーケンスの生成

取得したソースコードの特徴を特徴シーケンスという単位にまとめる．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．ソースコード内の各メソッドは，それに対応する特徴シーケンスを持つ．

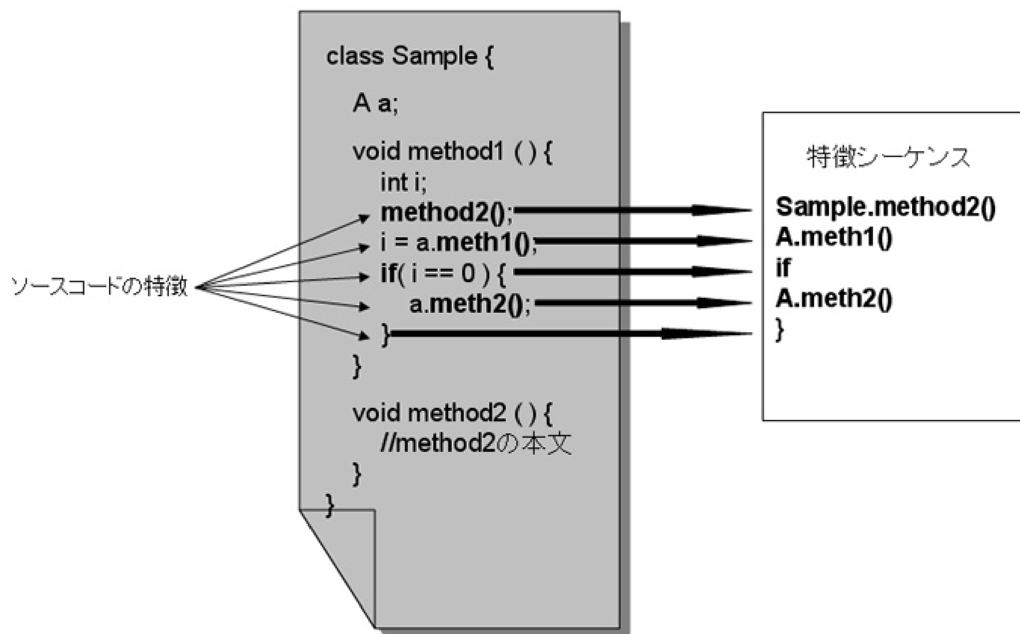

図 9: 特徴シーケンスの生成

図 9 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．

次に，生成された特徴シーケンスから不要なシーケンスを除去する．ここでいう不要なシーケンスとはメソッド呼び出しが一度しか行なわれない，もしくはメソッド呼び出しがまったく行なわれない特徴シーケンスのことである．メソッド呼び出しが 2 つ以上なければ実現したい処理に関連するメソッド群やその使用順序などの情報を知ることはできないので不要なシーケンスであるといえる．

3.2.3　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 [13] 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 [14] を採用した．

PrefixSpan は射影と呼ばれる操作を繰り返し行なうことでシーケンシャルパターンを抽出するアルゴリズムである．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．図 10 は要素 a についての射影を行なう様子を表している．

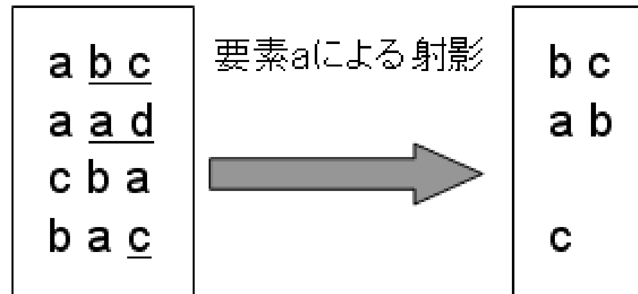

図 10: 要素 a についての射影

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．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．

(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．

(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．

(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．

(6) （3）に戻る.（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）～（6）を繰り返す.

図 11 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．

3.2.4　抽出されたパターンのフィルタリング

3.2.3 節のアルゴリズムでメソッド呼び出しパターンを抽出することはできるが，このままではメソッド呼び出しパターンとして価値のないものまで抽出してしまう．具体的には次

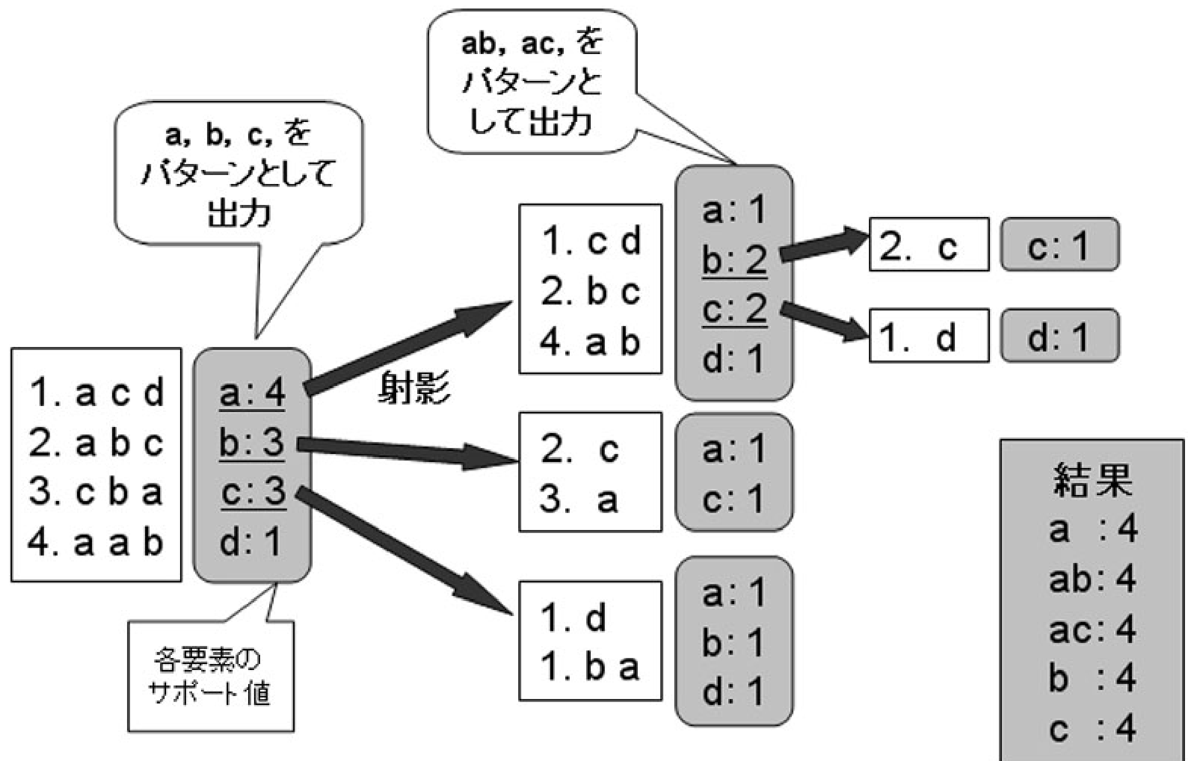

図 11: 最小サポート値 2 における PrefixSpan を用いた sequential pattern mining

のようなパターンがメソッド呼び出しパターンとして価値がないといえる．

制御文要素の対応が取れていないパターン パターンの要素を前から順に見ていったときに，制御文の開始要素しか存在せずそれ以降のパターン内のメソッドがすべて制御文内で呼び出されているようにみえたり，逆に制御文の開始要素がないのに突然，終了要素が出現するようなパターンが含まれる．このようなパターンは利用価値が低く，抽出しても意味がない．

パターン内のメソッド呼び出しと関連性のない制御文要素を持つパターン 制御文要素の閉める割合が多くなると，パターン内のメソッド呼び出しと関連性のない制御文要素を持つパターンが含まれ，パターン全体の利用価値が低くなる．

メソッド呼び出し間の関係にかかわる情報を持っていないパターン パターンとして抽出はされても同じ処理に関連しているメソッド群やメソッドの呼び出し順序などの，メソッド呼び出しパターンに期待する情報を持っていないもが存在する．このようなものは抽出しても特に意味がない．

このようなメソッド呼び出しパターンはパターンとしての価値が低いだけでなく，パターン抽出にかかる資源を浪費させたり，アスペクトマイニングを行なう際にメソッド呼び出しパターンから取ってくる情報の質を低下させる．こうした不要なメソッド呼び出しパターンを次のようにして除去している．

対応する制御文要素が存在しないパターンを除去 制御文要素を特徴として抽出する際，制御文要素の開始・終了の対応関係も取得しておき，3.2.3 で説明したアルゴリズムのステップ (5) の後で，次のいずれかに該当するパターンを除去する．

- パターン内のある制御文終了要素について，それに対応する制御文開始要素が含まれていない場合
- パターン内のある制御文開始要素について，それに対応する制御文終了要素が含まれておらず，かつその終了要素よりも後に続く要素がパターンに含まれている場合

このフィルタリングによってある制御文の終了要素以降に登場する要素がその制御文の内部にあるかのようにパターンとして抽出されることを防いでいる．

制御要素が連続するパターンを抽出しない 制御文の開始要素のすぐ後に，制御文要素の終了要素が抽出されるようなパターンは，その制御文内で呼び出されているメソッドがパターンの要素に含まれていない．このような制御文要素はパターン内に制御文要素と関連のあるメソッドが存在しないので，sequential pattern mining 実行時に随時除去する．

同じメソッド呼び出し要素が連続するパターンを除去 一般に，同じメソッド呼び出し要素が連続する場合，そのメソッド呼び出し要素はそれ単体で役割をなしていることが多く，同じパターン内で出現している他のメソッド呼び出し要素との関連性は低い．ゆえに，sequential pattern mining 実行時に随時除去する．

制御文要素が 3 分の 2 を超えるものを除去 パターンが抽出された際，パターン内の制御文要素の割合を調べ，それが 3 分の 2 を超えているものは除去する．

メソッド呼び出しが 1 個以下のパターンを除去 パターンが抽出された際，メソッド呼び出し要素が 1 個以下であれば，複数のメソッドの順序関係などの情報を持たないパターンなので除去する．

4　実装方針

本節では，3 節で説明したメソッド呼び出しパターンの抽出法を実装したシステムに関して説明する．図 12 は本システムの概要を表している．本システム l は大まかに次の 4 つに

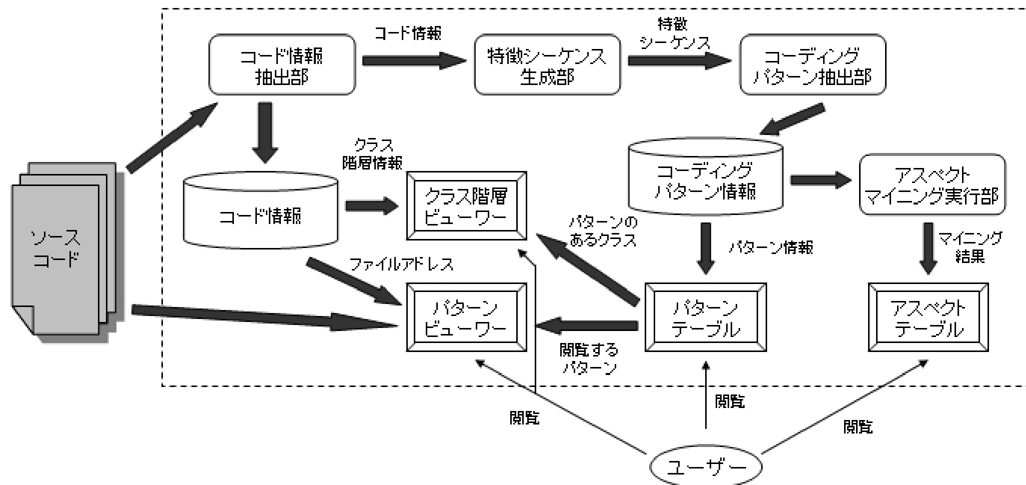

図 12: システムの概要

分けられる．

- パッケージやクラス，メソッドなどの構成情報を抽出してくる部分
- メソッド呼び出しパターンを抽出する部分
- アスペクトマイニングを実行する部分
- メソッド呼び出しパターンの抽出設定の変更や，抽出されたメソッド呼び出しパターンやアスペクト候補の表示を行なうための GUI

以下では，システムの各部分に関して詳しく説明する．なお，本システムは，JDK 5.0, GUI ライブラリである SWT [15]，Java ソースコードを解析するためのコンパイラコンパイラ JavaCC [16] を用いて開発した．

4.1　コード情報抽出部

コード情報抽出部ではソースコードから入力となるプログラムのパッケージやクラス，メソッドなどの情報を抽出する．具体的にどのような情報を抽出するかは表 1 に示す．

また，ここで抽出したクラス情報を用いてクラス階層情報も取得し，クラス階層ツリーとして表示する．

表 1: 抽出されるコードの情報

パッケージ情報	クラス情報
パッケージ名 親パッケージの情報 パッケージ内に存在するパッケージの情報の集合 パッケージ内に存在するクラスの情報の集合	クラス名 ファイルの情報 所属パッケージの情報の集合 スーパークラスの情報 宣言されているフィールドの情報の集合 宣言されているクラスの情報の集合 宣言されているメソッドの情報の集合
メソッド情報	フィールド情報
メソッド名 このメソッドが所属するクラスの情報 戻り値の型（クラス情報） 宣言されているクラスの情報の集合 宣言されているフィールドの情報の集合	フィールド名 型の情報（クラス情報）

4.2 メソッド呼び出しパターン抽出部

メソッド呼び出しパターンは，まずソースコードの特徴を抽出し，抽出した特徴を要素とする特徴シーケンスを生成，その後取得した特徴シーケンスに対して sequential pattern mining を行なうことによって抽出する．以下に各ステップについて説明する．

4.2.1 特徴抽出部

特徴抽出部ではメソッド呼び出しや制御文の開始・終了要素をソースコードの特徴として抽出する．このとき表 1 で示している情報をもとに，以下の情報を取得し特徴がもつ情報として管理する．

特徴番号 特徴がシーケンス内の何番目の要素であるか示す.

識別コード 特徴がメソッド呼び出しのときはクラス名とメソッド名をつなげた文字列のハッシュコード，制御文要素のときはそのトークンのハッシュコード．sequential pattern mining において要素の識別に使用する．ハッシュコードにすることにより文字列比較よりも高速な比較が行なえる．

要素名 特徴がメソッド呼び出しの場合はクラス名とメソッド名，制御文要素のときはそのトークン．

行番号 特徴が出現する行番号．特徴シーケンスの出現位置と合わせることにより特徴のソースコード内での位置を知ることができる．

対応する要素の特徴番号 特徴が制御文要素のとき対応する制御文要素の番号を示す．この情報を利用して sequential pattern mining 実行時に対応する制御文要素がパターン内に含まれているかどうか調べる．

4.2.2 特徴シーケンス生成部

特徴シーケンスは各メソッド定義内で抽出された特徴の情報を配列で出現順に管理する．特徴シーケンスが持つ情報を以下に示す.

特徴シーケンス番号 sequential pattern mining において射影行なうとき，新たなシーケンスがどの特徴シーケンスの接尾辞であるかを示す.

出現位置情報 特徴シーケンスが存在しているファイル，クラス，メソッドなどの情報.

メソッド呼び出し要素数　シーケンス内でのメソッド呼び出し回数を示す．この情報により，メソッド呼び出し回数が 1 以下のような，不要な特徴シーケンスを除去することができる．

特徴の配列 抽出した特徴を出現順に格納してある．sequential pattern mining の実行対象となる．

上記の情報以外に，各特徴がどの特徴シーケンスの何番目に出現しているのかと出現回数を示す，特徴のマップ情報も同時に取得する．このマップ情報を利用することにより PrefixSpan による sequential pattern mining を大幅に高速化することができる．

4.2.3 sequential pattern mining 実行

sequential pattern mining 実行部では 3.2.3 節で述べた処理を実現している．

sequential pattern mining アルゴリズムは PrefixSpan を採用している．このアルゴリズムを実装したものを用いて，特徴シーケンス生成部で生成した全特徴シーケンスに対して，4.3.1 節で説明するメソッド呼び出しパターン抽出設定変更部で指定した設定に基づいてメソッド呼び出しパターンを抽出する．各特長が同じメソッド呼び出しもしくは同じ制御文要素であるかどうかは，4.2.1 節で説明した特徴の持つ情報の内,「識別コード」を比較することによって判断する．また，ある特徴 A についての射影を行なうとき，特徴 A の出現位置を知るため全特徴シーケンスを探索すると非常に時間がかかるので，特徴シーケンス生成部で作成した特徴のマップ情報を利用することにより，特徴 A が出現する特徴シーケンスのみ探索するようにしている．

抽出された全メソッド呼び出しパターンの情報は 4.3.2 節で説明するメソッド呼び出しパターン情報表示テーブルで閲覧することができる．

4.3 GUI

GUI は以下の 4 つの部品で構成される．

- メソッド呼び出しパターン抽出設定変更部
- メソッド呼び出しパターン情報表示テーブル
- ソースビューワー
- クラス階層ツリー

以下で各部の説明を行なう．

4.3.1　メソッド呼び出しパターン抽出設定変更部

できるだけ多くの実験結果を得るため，メソッド呼び出しパターンの抽出設定を変更することができる．具体的には，次の項目が変更可能である．

- 最小サポート値
- 最小パターン長
- スーパークラスのメソッドと継承されたメソッドを区別するか否か
- 最大制御文含有率
- 最小メソッド呼び出し回数
- 除去するパターンの選択

メソッド呼び出しパターン抽出設定変更部のスクリーンショットを図 13 に示す．

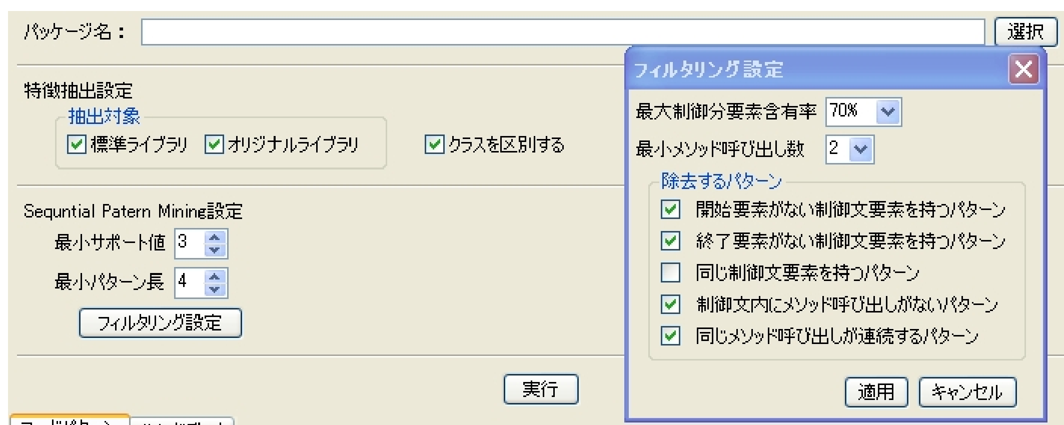

図 13: メソッド呼び出しパターン抽出設定変更部のスクリーンショット

4.3.2　メソッド呼び出しパターン情報表示テーブル

検出されたメソッド呼び出しパターンの情報を表示する．具体的には次の情報が表示される．

- コートパターン番号
- サポート値
- パターン長

- パターンが出現するクラス数
- 各パターンが存在するクラス名とメソッド名
- パターンの要素とその要素がソースコード内で出現する行番号

メソッド呼び出しパターン情報表示テーブルのスクリーンショットを図 14 に示す．

パターン	サポート値	パターン長	クラス数	メソッド名	要素	行番号
⊞パターン12	3	5	3			
⊟パターン13	3	6	3			
⊞org.jhotdraw.standard.StandardDrawing			3	handles		
⊞org.jhotdraw.samples.pert.PertFigure			3	handles		
⊞org.jhotdraw.figures.GroupFigure			3	handles		
⊞パターン14	3	5	3			
⊞パターン15	3	4	2			
⊟パターン16	3	6	3			
⊟org.jhotdraw.standard.CutCommand.Un...			3	rememberSelect...		
					createList	146
					while	147
					hasNextFigure	147
					nextFigure	148
					add	148
					}	149
⊞org.jhotdraw.standard.CompositeFigure			3	figures		
⊞org.jhotdraw.util.UndoableAdapter			3	rememberFigures		
⊞パターン17	3	5	2			
⊞パターン18	3	5	3			
⊞パターン19	3	4	2			
⊞パターン20	6	4	2			
⊞パターン21	3	6	3			

図 14: メソッド呼び出しパターン情報表示テーブルのスクリーンショット

4.3.3　ソースビューワー

メソッド呼び出しパターン情報表示テーブルで選択されたパターンが出現するソースコードを表示する．表示されたソースコードはパターンの要素が赤色でハイライトされる．

図 15 はソースビューワーのスクリーンショットである．

4.3.4　クラス階層ツリー

コード情報抽出部で抽出したクラス階層情報をクラス階層ツリーにして表示する．また，メソッド呼び出しパターン情報表示テーブルで選択されたパターンが出現するクラスをハイライトすることができるので，メソッド呼び出しパターンがどのクラス階層に散らばっているか簡単にわかる．

図 16 はクラス階層ツリーのスクリーンショットである．

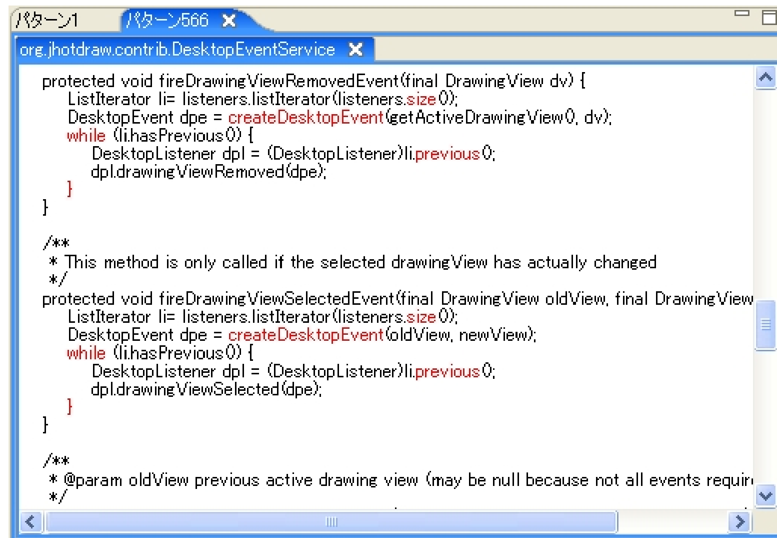

図 15: ソースビューワーのスクリーンショット

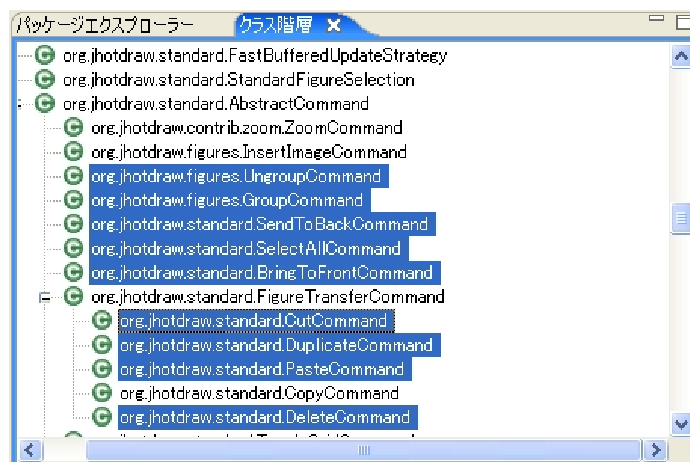

図 16: クラス階層ツリーのスクリーンショット

5　抽出したメソッド呼び出しパターンの評価基準

この節では，抽出したメソッド呼び出しパターンからどのような情報を得ることができるのか，また，それらの情報がアスペクトマインニングにどのように役立つかについて説明する．

5.1　パターン自体の評価基準

5.1.1　パターン長

メソッド呼び出しパターンのパターン長とは，そのメソッド呼び出しパターンを構成する特徴の数である．ゆえに，この値が大きいほどこのパターンに対応するコード記述は長くなる．

ソフトウェアは複数の機能によって構成され，これらの機能もまた，より小さい機能の集合で構成されている．オブジェクト指向プログラムでは，これらの機能のうち一定の大きさ以上の機能ごとにクラスを割り当てコード記述を一箇所に集めている．しかし，本来クラスを割り当てるべき大きさの機能であったとしても，それが横断的であるためクラスが割り当てられないものも存在する．機能の大きさはコード記述の長さにある程度比例するので，パターン長が長いメソッド呼び出しパターンが実現しようとする機能はクラスを割り当てるべき大きさの機能である可能性が高い．このようなメソッド呼び出しパターンに関連するコード記述にはアスペクトを割り当て記述を一箇所に集めるべき候補に挙げることができる．

また，図 17 のメソッド 1，メソッド 3 のように，定義内におけるメソッド呼び出しパターンＡの占める割合が大きいとき，メソッド呼び出しパターンＡに関連する処理が実現しようとする機能とメソッド 1，メソッド 3 が実現する機能が一致する可能性が高い．つまり，メソッド 1 とメソッド 3 は同一の関心事を実現しようとしている可能性が高い．このような場合，メソッド 1，メソッド 3 の宣言そのものがインタータイプ・メンバ宣言によるアスペクト指向リファクタリングの対象となりうる．一方，図 17 のメソッド 2，メソッド 4 には，メソッド呼び出しパターンＢが存在しているが，パターンＢが占める割合は小さい．このような場合，メソッド呼び出しパターンＢに関連する処理は，メソッド 2，メソッド 4 が実現しようとする機能を構成するより小さな機能群のうちの 1 つに過ぎず，メソッド 2 とメソッド 4 が同一の関心事を実現しようとしている可能性はほとんどない．

5.1.2　サポート値

メソッド呼び出しパターンのサポート値とは，ソースコード内でそのメソッド呼び出しパターンが何回出現するかを示す．図 18 ではどのパターンも 3 箇所で出現しているので，ど

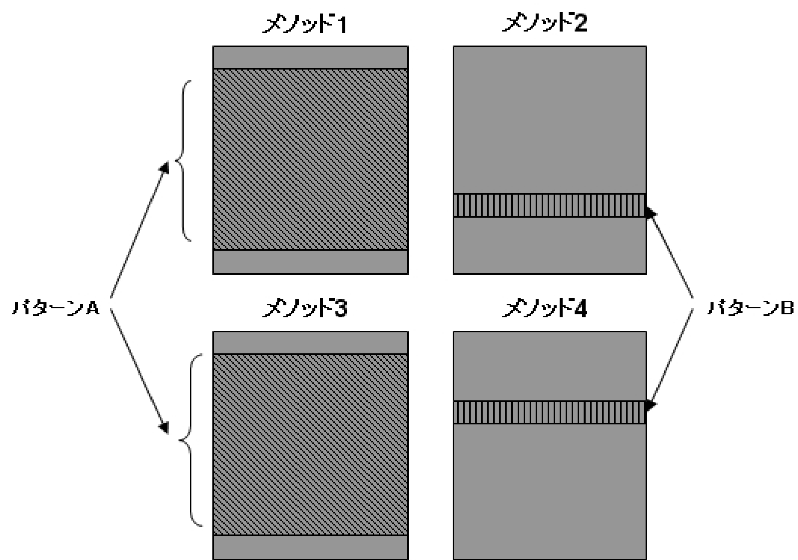

図 17: メソッド呼び出しパターンの占める割合

のパターンもサポート値は 3 である．

サポート値が高いほど，そのメソッド呼び出しパターンに関連する部分の拡張や保守を行なうときに修正しなければならない箇所が増え，より高いコストがかかる．このようなサポート値の高いメソッド呼び出しパターンが実現する処理をアスペクトとして一箇所に集めることができれば，保守性の向上が期待できる．

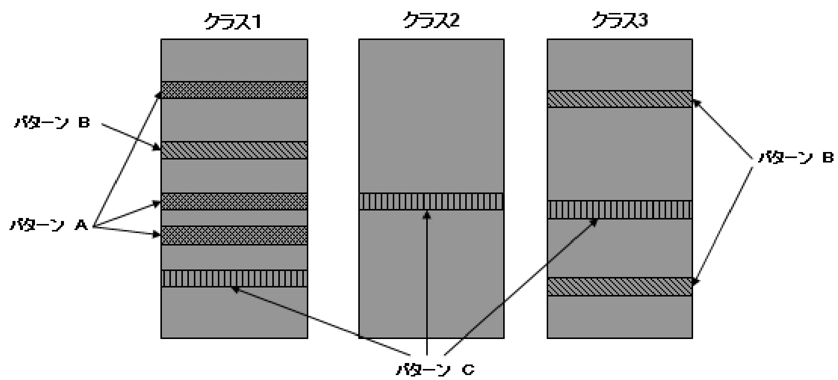

図 18: パターンの散らばり方

5.1.3 パターンが出現するクラス数

この値は抽出したパターンが出現するクラスの数であり，値が大きいほどコードが多数のクラスを横断していることを表す．

この値が 1 である場合は，たとえサポート値が高くても，横断的関心事に関連したパターンであるとはいえない．例えば，図 18 においてパターン A，パターン C はどちらも 3 箇所で出現しているのでサポート値は 3 であるが，パターン A はクラス 1 のみで出現しており横断的ではない．また，パターン A のパターン長が長かったとしても，横断的でない以上無理にアスペクトにする必要はない．オブジェクト指向リファクタリングが適用できるようであれば，適用すべきである．一方パターン C はクラス 1，クラス 2，クラス 3 にまたがって存在しており横断的であるのでアスペクトの候補としてあげることができる．

このように，パターン長やサポート値からのみの情報では明らかに横断的ではない関心事に関連したメソッド呼び出しパターンが混ざってしまうが，抽出したパターンが出現するクラス数を参照することにより，そのようなメソッド呼び出しパターンをフィルタリングすることができる．

5.2 パターンの要素であるメソッドに関する評価基準

メソッド呼び出しパターンの構成要素であるメソッド呼び出しのうち，パターン内でしか呼び出されないメソッドは，そのメソッドが属するメソッド呼び出しパターンが関連する関心事に属している可能性が高い．

一方，メソッド呼び出しパターン内だけでなく，パターン外でも何度も呼び出されるメソッドは，メソッド呼び出しパターンが関連する関心事とパターン外で実現される関心事を横断していると判断できる．このようなメソッドはアスペクトの候補となり得る．

6　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．

評価に用いた環境は次の通りである．

- CPU：Pentium4 3.20GHz
- RAM：1GB
- OS：FreeBSD6.0 STABLE

メソッド呼び出しパターンの抽出対象としては JHotDorw [17] 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 [5, 6, 18]．ソースコードの概要は以下の通りである．

- バージョン ： 5.4b1
- 総行数 ： 約 18000 行（テストファイルは除外）
- 総メソッド数 ： 約 29000

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，抽出時間は約 11 秒となった．

抽出されたメソッド呼び出しパターンはアルゴリズムの都合上，同種のパターンをいくつも抽出してしまう．例えば，”abcde”といったようなパターン長 5 のパターンが抽出されるとき，”abcd”や”abce”などのようなパターン長 4 のパターンも同時に抽出される．これらのパターンは同種のものであるといえるので，このようなパターン群を 1 つのカテゴリにまとめると，抽出結果のカテゴリ数は 49 であった．

これらのカテゴリに属するパターンの内，サポート値が最大のパターン，パターン長が最長のパターン，サポート値とパターン長がバランスよく高いパターンを中心に，全てのカテゴリに対してメソッド呼び出しパターンと関連するコード記述の調査を行なった．また，それらのメソッド呼び出しパターンのうち次のような特徴をもつものを，横断的関心事でありアスペクトの候補となり得ると判断した．

- 出現箇所が多く，かつ複数のクラスに分散している．
- メソッド呼び出しパターンが特定の関心事に関連している．

6.1　メソッド呼び出しパターンの評価

6.1.1　パターン長に注目した評価

パターンの長さが長いメソッド呼び出しパターンで最も目立ったメソッド呼び出しパターンは図 19 に示したパターンである．図 19 には SendToBackCommand クラスと PasteCommand クラスのコードサンプルしか示していないが，他に 2 つのクラスで抽出されており，このメソッド呼び出しパターンは計 4 箇所で出現している．

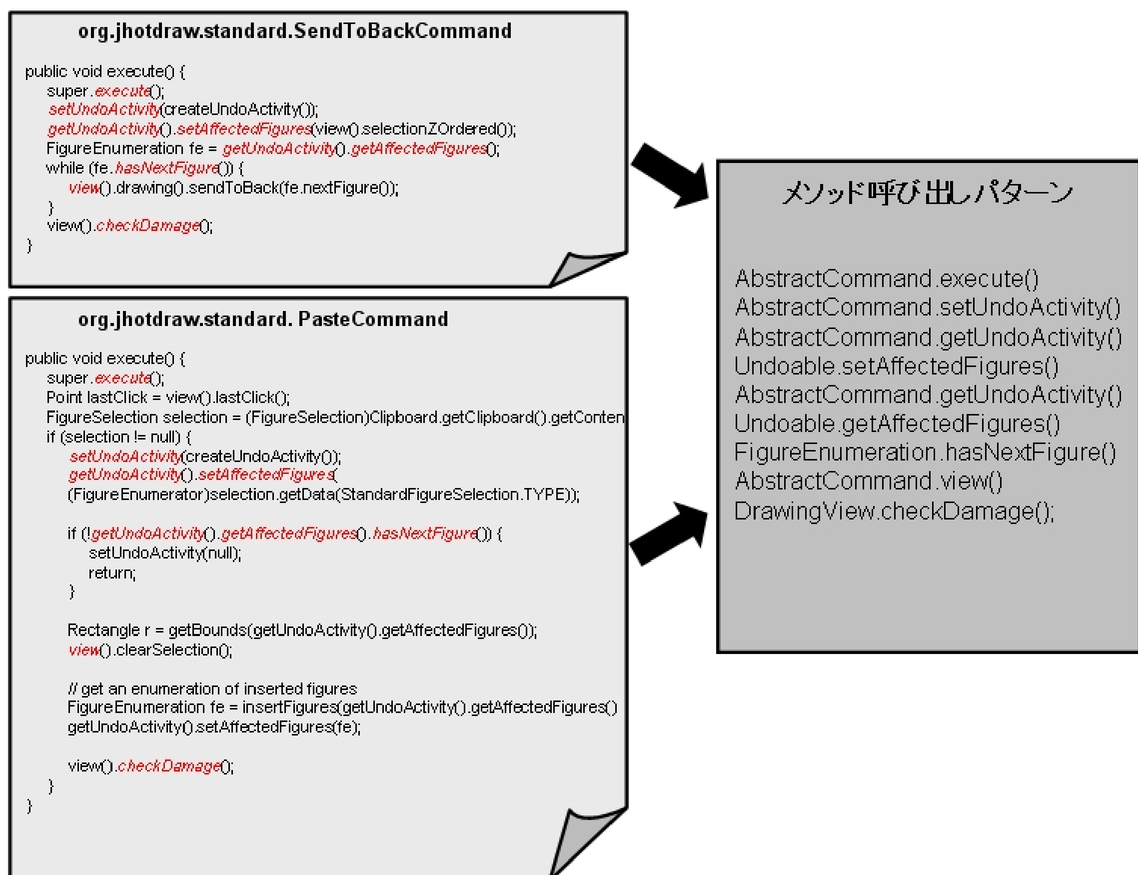

図 19: パターン長が長いメソッド呼び出しパターンの例

図 19 で抽出されたメソッド呼び出しパターンの構成要素のうち，AbstractCommand.setUndoActivity()，AbstractCommand.getUndoActivity()，Undoable.setAffectedFigures()，AbstractCommand.getUndoActivity()，Undoable.getAffectedFigures() は Undo 用の情報を保存する処理に関するパターンであり，関心事 Undo に属する．AbstractCommand.view()，DrawingView.checkDamage() はコマンドの実行終了を View に通知するためのパターンで通

知関連の関心事に属する．図 19 のパターンから Undoable.getAffectedFigures()，FigureEnumeration.hasNextFigure() を除いたパターン長 7 のパターンは計 11 箇所（全て execute メソッド内）で出現している．

execute メソッドは何らかのコマンドを実行するという関心事を実現するメソッドであり，ここで抽出されたパターンはコマンド実行に関連するクラス群の中で，横断的に出現している．

6.1.2 サポート値に注目した評価

抽出したメソッド呼び出しパターンの中で最もサポート値が大きかったメソッド呼び出しパターンは図 19 に示したパターンである．図 20 には CompositeFigure クラスと StandardDrawingView クラスのコードサンプルしか示していないが，他に 55 箇所で抽出されており，非常に出現頻度の高いパターンである．

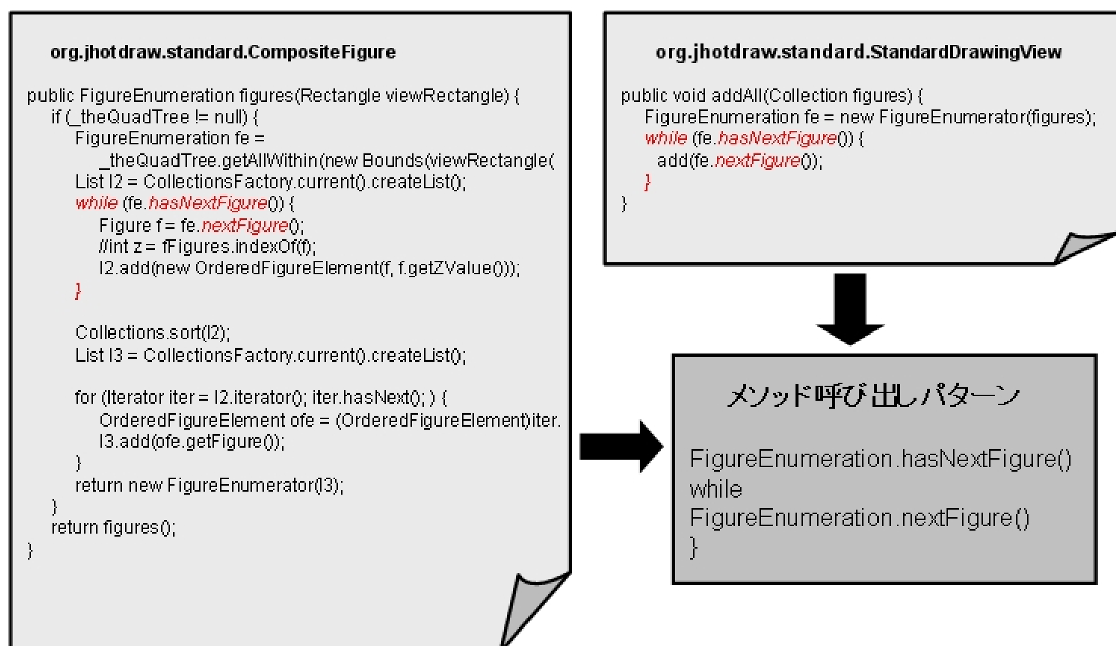

図 20: サポート値が大きいメソッド呼び出しパターンの例

このメソッド呼び出しパターンが実現する機能は，特定の図形集合に対する繰り返し処理である．ただし，このメソッド呼び出し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．さらに，

CompositeFigure.figures() メソッドの定義を見てもわかるようにこの処理は，何らかの機能を実現するためのより小さな機能群のうちの 1 つであり，アスペクトとするには機能単位が小さすぎる．ゆえにこのメソッドーパターン呼び出しパターンはアスペクトにすべき特定の横断的関心事に関連しているとはいい難い．

図 20 で示したメソッド呼び出しパターン以外でサポート値の高いメソッド呼び出しパターンとして，execute メソッドに出現したパターンや関心事 undo に関連したパターンが挙げられる．execute メソッドに出現したパターンは 6.1.1 節で示したようにアスペクトリファクタリングの対象となりえる．関心事 undo に関しての具体的な説明は省略するが，アスペクトリファクタリングの候補といえる．

6.1.3　メソッド呼び出しパターンとその関心事

ここでは，カテゴリごとに分類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を示す．

表 2，3 は各カテゴリの情報とそのカテゴリが関連する関心事を，パターン長の降順で示したものである．表の各項目のを説明する．

No. カテゴリの番号を示している項目．

メソッドの関心事 メソッド呼び出しパターンが現れるメソッドが実現する関心事を示す項目．パターンが出現するメソッドの関心事が一定ではないものは空白としている．

パターンの関心事 パターンが関連している関心事を示す項目．パターンが実現する処理が 20 で説明したような明らかにアスペクトの候補でないと判断できるようなパターンはパターンの関心事を空白にし，各パターン長にそのようなパターンがいくつあるかわかるようにしてある．

Len 各カテゴリに属するメソッド呼び出しパターンのパターン長の最大値を示す項目．

Sup 各カテゴリに属するメソッド呼び出しパターンのサポート値の最大値を示す項目．

Cls メソッド呼び出しパターンが出現するクラスの数を示す項目

AOR アスペクト指向リファクタリングの実行する際に用いる手段を示す項目．”PA”はポイントカットやアドバイスを，”ITD”はインタータイプ・メンバ宣言を用いるのが望ましく，空白はアスペクト指向リファクタイングの対象外であることを意味する．

表 2: 抽出されたパターンの関心事一覧 1

No.	メソッドの関心事	パターンの関心事	Len	Sup	Cls	AOR
1	コマンドの実行	コマンドの取り消し コマンドの実行終了通知 選択図形に関する処理	9	12	12	PA
2	コマンドの取り消し	コマンドの取り消し	9	12	12	ITD
3	図形ハンドルの生成	図形ハンドラの生成	8	5	5	ITD
4	パネルサイズに関する処理	パネルサイズに関する処理	8	4	1	
5	コマンドの取り消し コマンドのやり直し	コマンドの取り消し コマンドのやり直し 選択図形に関する処理	7	7	6	ITD ITD PA
6	指定された図形の探索	指定された図形の探索	7	6	4	ITD
7	ツールバー作成	ツールバー作成	7	4	4	ITD
8	図形の描画処理	図形の描画処理	7	6	1	
9	視点の拡大縮小	視点の拡大縮小	7	5	1	
10,11			7			
12	ビューリスナに関する処理	ビューリスナに関する処理	6	6	2	ITD
13-15			6			

表 3: 抽出されたパターンの関心事一覧 2

No.	メソッドの関心事	パターンの関心事	Len	Sup	Cls	AOR
16	図形ハンドルの呼び出し	図形ハンドルの呼び出し	5	8	8	ITD
17	図形ハンドルの呼び出し	コマンドの取り消し	5	5	5	PA
18	図形の移動処理	図形の移動処理	5	5	5	ITD
19	図形の選択解除	図形の選択解除	5	5	5	ITD
20	図形の描画処理	図形の描画処理	5	5	5	ITD
21	マウスイベントに対する処理	コマンドの取り消し ツール実行終了通知	5	4	4	PA
22	マウス操作に対する処理	マウス操作に対する処理	5	4	3	ITD
23-29			5			
30	マウス操作に対する処理	マウス操作に対する処理	4	5	5	ITD
31	マウス操作に対する処理	マウス操作に対する処理	4	4	4	ITD
32	マウス操作に対する処理	指定された図形の探索	4	4	4	PA
33	テキストの書き込み	テキストの書き込み	4	4	4	ITD
34	テキストの読み込み	テキストの読み込み	4	4	4	ITD
35	コマンドのやり直し	コマンドのやり直し	4	4	4	ITD
36		選択図形の変更通知	4	4	4	PA
37	図形の削除	図形の削除	4	4	4	ITD
38	指定された図形の探索	指定された図形の探索	4	4	2	ITD
39-49			4			

メソッドの関心事と，そのメソッドに出現するパターンが関連している関心事が別のものはメソッドの関心事に関する記述にパターンが関連する関心事が混ざり合っている．このようなパターンはジョインポイントやアドバイスを用いたアスペクト指向リファクタリングの候補である．

メソッドの関心事と，そのメソッドに出現するパターンが関連する関心事が同じもでかつパターンの存在するクラス数が大きいものはインタータイプ・メンバ宣言を用いたアスペクト指向リファクタリングの候補である．

また，アスペクトにはなりえないメソッド呼び出しパターンのカテゴリは，パターン長9または8のものには0，パターン長7のものには2，パターン長6のものには3，パターン長5のものには7，パターン長4のものには11存在しており，パターン長が長いものほとアスペクトにすべきの関心事と関連しているパターンである可能性が高いことは明らかである．ただし，パターンが存在するクラス数が極端に低いものは横断的ではないので，アスペクトの候補から除外する．

6.2　パターンの要素であるメソッドに関する評価

5.2では特定のパターン内でのみ呼び出されるメソッドはアスペクトである可能性が低いと主張した．これは逆に，様々なパターン内で呼び出されるメソッドはアスペクトの可能性が高いことを意味する．この主張を確認するために，抽出されたメソッド呼び出しパターン外で呼び出されているものはサポート値1のパターン内で呼び出されていると考え，各メソッドに対して「パターン外での呼び出し回数 ＋ 抽出されたパターンの内このメソッドを含むパターンの数」を示す数値をとり，単純な呼び出し回数が多いものと，この数値が大きいものを比較した．

図21は左が呼び出し回数が多い順にソートした結果，右が各メソッドが呼び出されるパターンの数でソートした結果である．選択されているメソッドはアスペクトである．

hasNextFigure，nextFigure，currentなどはアスペクトでないにもかかわらず呼び出し回数が非常に多いが，特定のメソッド呼び出しパターンでのみ呼び出されており，これらのメソッドを呼び出すパターン数は少ないためアスペクトの候補とから除外することができた．しかし，同時にchangeやclearSelectionなどのようにアスペクトであるものにも，呼び出し回数は多いが，これらを呼び出すパターン数は少ないものが存在しアスペクトの可能性が高いものから除外されてしまった．

以上のことから，特定のパターン内でのみ呼び出されるメソッドはアスペクトである可能性が低いとは限らず，アスペクトの中にも特定のメソッド呼び出しパターンでのみ呼び出されるものが存在することがわかった．

メソッド名	Fan-In
org.jhotdraw.framework.FigureEnumeration.hasNextFigure	106
org.jhotdraw.framework.FigureEnumeration.nextFigure	101
org.jhotdraw.util.CollectionsFactory.current	59
org.jhotdraw.util.CollectionsFactory.createList	57
org.jhotdraw.framework.Figure.displayBox	54
org.jhotdraw.framework.DrawingView.drawing	49
org.jhotdraw.standard.AbstractFigure.displayBox	45
org.jhotdraw.standard.AbstractCommand.view	42
org.jhotdraw.standard.AbstractHandle.owner	35
org.jhotdraw.standard.AbstractTool.view	35
org.jhotdraw.standard.AbstractFigure.changed	34
org.jhotdraw.framework.DrawingView.clearSelection	34
org.jhotdraw.standard.AbstractFigure.willChange	26
org.jhotdraw.standard.AbstractTool.editor	25
org.jhotdraw.util.UndoableAdapter.undo	25
org.jhotdraw.framework.DrawingView.checkDamage	23
org.jhotdraw.util.StorableInput.readInt	22
org.jhotdraw.util.UndoableAdapter.isRedoable	21
org.jhotdraw.util.StorableOutput.writeInt	20
org.jhotdraw.util.StorableInput.readStorable	20
org.jhotdraw.standard.AbstractTool.mouseDown	20
org.jhotdraw.standard.AbstractFigure.listener	18
org.jhotdraw.standard.AbstractTool.drawing	18
org.jhotdraw.standard.AbstractFigure.write	17
org.jhotdraw.standard.StandardDrawingView.drawing	17
org.jhotdraw.standard.AbstractFigure.read	17
org.jhotdraw.util.StorableOutput.writeStorable	17
org.jhotdraw.framework.Figure.moveBy	17
org.jhotdraw.standard.AbstractCommand.execute	17
org.jhotdraw.standard.AbstractFigure.figures	17

メソッド名	パターン数
org.jhotdraw.standard.AbstractFigure.displayBox	41
org.jhotdraw.standard.AbstractFigure.changed	26
org.jhotdraw.standard.AbstractFigure.willChange	18
org.jhotdraw.framework.Figure.displayBox	17
org.jhotdraw.standard.AbstractHandle.owner	14
org.jhotdraw.figures.PolyLineFigure.pointAt	14
org.jhotdraw.standard.AbstractTool.view	13
org.jhotdraw.standard.AbstractTool.editor	13
org.jhotdraw.util.StorableInput.readInt	13
org.jhotdraw.util.StorableOutput.writeInt	13
org.jhotdraw.util.StorableInput.readStorable	13
org.jhotdraw.standard.AbstractFigure.write	13
org.jhotdraw.standard.StandardDrawingView.drawing	13
org.jhotdraw.standard.AbstractFigure.read	13
org.jhotdraw.util.UndoableAdapter.isRedoable	12
org.jhotdraw.util.StorableOutput.writeStorable	11
org.jhotdraw.framework.Figure.moveBy	11
org.jhotdraw.standard.AbstractTool.deactivate	11
org.jhotdraw.framework.DrawingView.selectionCount	11
org.jhotdraw.framework.DrawingView.checkDamage	10
org.jhotdraw.util.UndoableAdapter.undo	9
org.jhotdraw.application.DrawApplication.open	9
org.jhotdraw.applet.DrawApplet.view	9
org.jhotdraw.framework.Figure.addFigureChangeListener	8
org.jhotdraw.standard.AbstractTool.activate	8
org.jhotdraw.framework.DrawingView.add	8
org.jhotdraw.util.StorableOutput.space	8
org.jhotdraw.standard.AbstractCommand.view	7
org.jhotdraw.util.Command.execute	7
org.jhotdraw.util.Geom.center	7

図 21: メソッドの呼び出し回数（左）とメソッドが呼び出されるパターンの数（右）

6.3　メソッド呼び出しパターンとアスペクトの関連性

6.1 節で紹介したメソッド呼び出しパターンに関しても調査を行なった結果，メソッド呼び出しパターンとアスペクトには以下のような関連性があった．

- 6.1.3 節で示したようにメソッド呼び出しパターンのパターン長が長いものはアスペクトの候補である可能性が高い．これには次のような理由が考えられる．

  パターン長が小さいものには 6.1.2 節で示した繰り返し文の制御に関連したパターンなどのように，特定の関心事に関連しているというには機能単位が小さすぎるものが多くなる．一方，パターン長が長いものは機能単位が大きくなるのでそのパターンが出現しているメソッドの関心事と一致するものが増える．このようなもののうち，パターンが複数のクラスで出現するものは同じ関心事を実現しているメソッドが複数のクラスで定義していることを示しているので，このようなメソッドはインタータイプ・メンバ宣言を用いて一箇所に集めるべき対象といえる．また，パターン長は短いがアスペクトに関連しているメソッド呼び出しパターンのうち，パターンが実現する関心事が他の関心事と連携する必要がある場合，関心事ごとに出現順序が決まってくる．例えば 6.1.1 節で示したパターンは，まず実行コマンドの取り消しに関連するコードでコマンド実行前の情報を保存し，その後コマンドを実行し，実行終了通知を行なう，という順序は守られなければならない．その結果，アスペクトに関連する 2 つの短い

パターン（6.1.1 節のパターンの場合は，実行コマンドの取り消しに関連するパターンと実行終了通知に関連するパターン）が連結されて 1 つの長いパターンとして現れる．

- 6.1.2 節で示したようにサポート値が大きさはアスペクトである可能性とあまり関係がない．6.1.2 節では，アスペクトにすべきである execute メソッドに出現したパターンや，undo に関連するメソッド呼び出しパターンもサポート値が大きいという結果が得られているが，これらのパターンは，同時にパターン長も長くアスペクトの可能性が高い理由はサポート値よりもパターン長に依存しているといえる．
- サポート値の大きさはアスペクトリファクタリングの効果と関連している．サポート値が高いほど多くの場所に分散してパターンが存在しているので，この値が高いものをアスペクトリファクタリングしたときほどより多くのコードが一箇所に集約され，リファクタリングの効果は大きくなる．

6.4 アスペクトマイニングツールとして利用するには

メソッド呼び出しパターンをアスペクトマイニングツールとして応用するには，6.3 節の内容からもわかるようにまず第一にメソッド呼び出しパターンのパターン長の長いものから順にアスペクトの可能性が高いと判断すべきである．

しかし，メソッドの定義内にもともと特徴となる要素が少ししかない場合，そのメソッド内に出現するパターンの長さは必然的に短くなる．このような場合パターン用は短くてもそのパターンがメソッドの定義を占める割合は高くなる．実際，抽出結果にはこのようなパターンが含まれており，そのメソッドはアスペクトの候補と判断することができた．ゆえに，パターン長の次に重要視する値としてメソッド定義を占めるパターンの割合にも注目する必要がる．

サポート値はアスペクトリファクタリングの効果を示す指針として利用することができる．

6.5 既存手法との比較

ここでは，既存のアスペクトマイニング手法とメソッド呼び出しパターンを用いたアスペクトマイニングの違いを示す．

2.2 節で既存のアスペクトマイニングのアプローチを 4 つ紹介した．本研究で行なっているメソッド呼び出しパターンを用いた方法は，この 4 つのうちコードクローンをベースにまとめる手法に似ている．しかし， コードクローンをベースにまとめる方法 [3] は，クローン内に別のコードが挿入されているものの抽出に弱いため，横断的関心事のようなコード間にしばしば他の関心事のコードが混ざり合うものの抽出にはあまり向いていない [19]．一方，コード間に他の関心事のコードが混ざっていても，特定の関心事を実現するために必要なメ

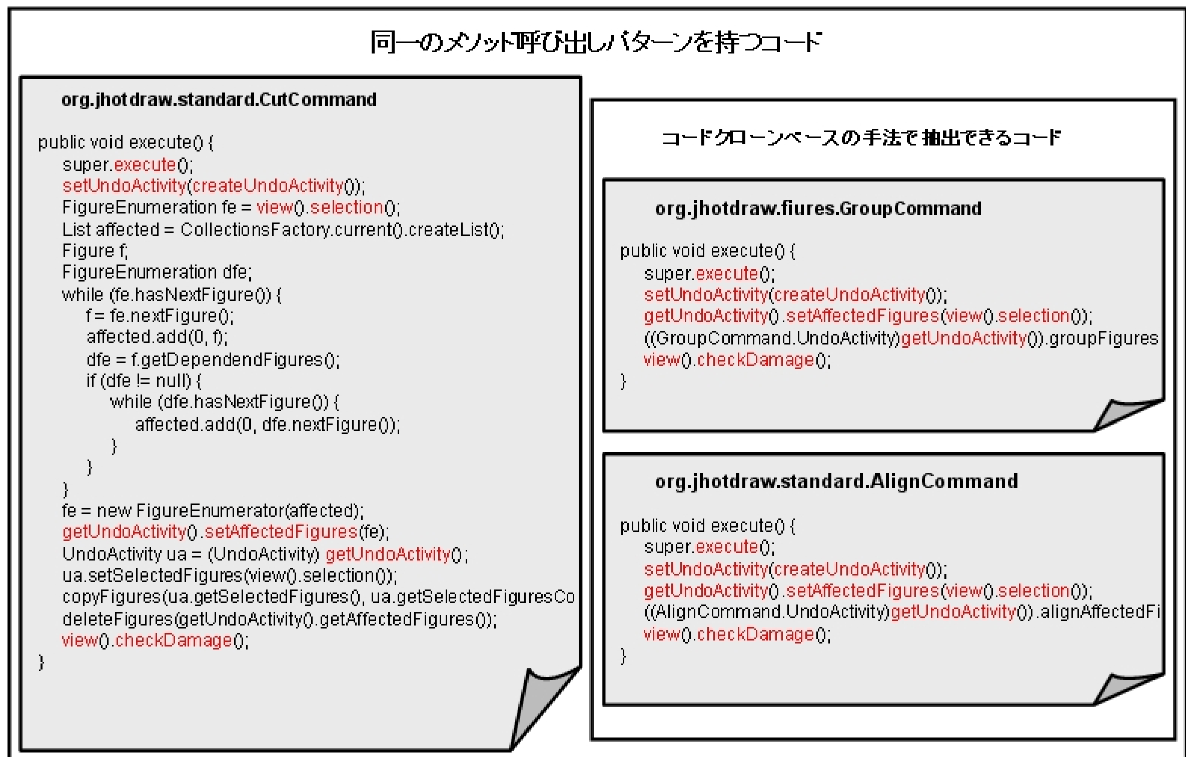

図 22: コードクローンベースで抽出できるものと，メソッド呼び出しパターンを用いた方法で抽出できるものの違い

ソッドの呼び出し順は変わらないので，メソッド呼び出しパターンを用いた手法ならこのようなものも同一の関心事に関連しているものとして抽出することができる．

図 22 に例を示す．CutCommand クラス，GroupCommand クラス，AlignCommand クラスは同じメソッド呼び出しパターンを持っており，これらのメソッド呼び出しパターンはすべてコマンドの実行に関連している．これらのうち，GroupCommand クラス，AlignCommand クラスにおけるコードはほぼ同一のコード記述であるのでコードクローンをもとにした手法でも抽出できる．しかし，CutCommnd クラスに記述されているコードは 1 つのメソッドに複数の役割が入り込んでいる（interleaving [19]）．このような，関心事が他のコードと混ざり合っているものをコードクローンをベースとした手法で抽出するのは難しい．

本来，アスペクト指向プログラミングの目的は同じコード記述を一箇所に集めることではなく，同じ関心事に関連するコードを一箇所に集めることにある．コードクローンがアスペクトリファクタリングの対象となりえるのは，コード記述が同じだからではなく，同じ関心事を実現しようとするコードは記述は似通ってくるという性質があるからである．ゆえに，コード記述レベルで同一のものを抽出するコードクローンベースの手法よりも，機能レベル

で同一のものを抽出するメソッド呼び出しパターンを用いた手法のほうが，よりアスペクト指向プログラミングの目的に適っているといえる．

7 まとめ

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，このような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どうかという疑問に答える事を目的としている．

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アスペクトとすべき関心事に関連するパターンが存在することを確認した．また，パターン長が横断的関心事と何らかの関連を持っており，一方でサポート値は横断的関心事との関連が弱いという特徴が見られた．

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，本研究でのアスペクト抽出はインタータイプ・メンバ宣言の利用が主となっている．しかし，メソッド呼び出しパターンの要素として出現するメソッド呼び出しの中にもポイントカットとアドバイスを用いたアスペクトの抽出が適切な例も見られた．このような，どちらの抽出方法が適しているかどうかを判定する手法の確立も今後の課題として挙げられる．また，本研究の評価対象は JHotDraw のみであり，本研究から得られた結論はまだ一般的であるとは言い難い．今後様々な対象に対して実験および評価を行なう必要がある．

謝辞

本研究において，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究において，適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 助教授に深く感謝いたします.

本研究において，逐次適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 氏に深く感謝いたします.

本研究において，様々な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 谷口 考治 氏に深く感謝いたします.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier and J. Irwin: "Aspect Oriented Programming", Proceedings of ECOOP, vol.1241 of LNCS, pp.220-242(1997).

[2] J. Hannemann and G. Kiczales. Overcoming the prevalent decomposition of legacy code. In Proc. Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering, pages 220–242, 2001.

[3] 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.*

[4] 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.

[5] M. Marin, A. van Deursen, and L. Moonen. Identifying Aspects using Fan-In Analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering(WCRE2004)*, pp.132-141, 2004.

[6] J. Krinke, Mining Control Flow Graphs for Crosscutting Concerns, wcre, pp. 334-342, 13th Working Conference on Reverse Engineering (WCRE 2006), 2006.

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

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

[9] I. Kiseleve. "Aspect-Oriented Programming with AspectJ". Sams Publishing(2002).

[10] R. Laddad, Aspect-Oriented Refactoring Series. http://www.theserverside.com/tt/articles/ article.tss?l=AspectOrientedRefactoringPart1

[11] D. Binkley, M. Ceccato, M. Harman, F. Ricca, P. Tonella, "Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects", IEEE Transactions on Software Engineering, vol. 32, no. 9, pp. 698-717, Sept., 2006.

[12] 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

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

[14] 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 *ICDE'01: Proceedings of the 17th International Conference on Data Engineering,pp.215-224*, Washington,DC,USA,April 2001. IEEE Computer Society.

[15] SWT: The Standard Widget Toolkit. http://www.eclipse.org/swt/

[16] JavaCC - The Java Parser Generator. Berkeley Software Distribution (BSD) License. https://javacc.dev.java.net/

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

[18] SERG AMR Fan-in Analysis Results. Concerns and Fan-In Values for JHotDraw. http://swerl.tudelft.nl/bin/view/AMR/FanInAnalysisResults

[19] R. Ettinger and M. Verbaere. Untangling: a slice extraction refactoring. In AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development, pages 93-101. ACM Press, 2004.