B-012

デザインパターン検出のための識別要素の設計
Design features for design pattern detection

鵜飼 公平† 酒井 三四郎†
Kohei Ukai Sanshiro Sakai

1. はじめに

ソフトウェア開発の保守工程では新しい機能の追加や，発生した問題の修正が行われる．ソフトウェアを理解するには少なくない時間を要し，それでも完全に理解するのは困難である．そのような保守作業では設計書や仕様書とソースコードの食い違いが往々にして起こる．食い違いがあると，さらに理解が難しくなり時間がかかるという悪循環が発生する．ソースコードから意味を汲み取るアプローチとしてデザインパターン[1]の検出が挙げられる．

ソースコードからデザインパターンを検出することは，つまりソースコードから設計意図を抽出することに繋がる．設計意図を抽出することでソースコードの意図を理解し，設計書とソースコードの食い違いを埋めることに役立てることができる．また，難解なソースコードの理解を早めることにもなる．設計者だけの認識ではなく，デザインパターンという共通の認識を用いることができるので，検出箇所に適用されたデザインパターンの設計意図を考慮に入れた上で修正，拡張することが可能となる．

しかし，デザインパターンをソースコードから検出することは容易なことではない．なぜなら，デザインパターンは抽象的な設計であり，様々な変形を加えられたデザインパターンが適用されている．そのため柔軟な検出が研究されているが，検出箇所がパターンとして使われていない部分も検出してしまっている場合もある．本研究では，実際にデザインパターンとして使える箇所を検出することを目的とする．

2. 識別要素の設計

2.1 関連研究

デザインパターンを検出する研究はこれまでも様々な形で行われてきている[2][3]．これらの研究ではデザインパターンを検出する方法として行列のテンプレートマッチングが導入されている．行列の成分はデザインパターンを区別する特徴を値に置き換えている．さらに完全一致する部分だけでなく，類似性の高い箇所も検出できるような検出基準を採用している．これはデザインパターンの様々な適用方法に対応するためである．

[3]の研究では Web ページ[4]で検出結果と検出ツールを公表している．検出結果に該当するソースコードを読むと，確かに検出したパターンの特徴が見受けられる．しかし，特徴が見られるだけであり，そのパターンが必ずしも適用されているわけではないことがわかる．例えば，State パターンならば，State オブジェクトを変更するための仕組みが存在するはずである．しかし，検出結果が指す該当箇所にはそのような仕組みは見られない場合が多い．

本研究では，上述したようなパターンの特徴を持っていても実際には用いられていない結果を排除し，実際に State パターンとして使える部分を検出する．本論文では検出するパターンを State パターンに絞って検出を試みる．上述したように State パターンは State オブジェクトを変更するための仕組みがなくては状態遷移できない．状態オブジェクトを変更するためにはメソッド内で代入文が必要である．よって State パターンを区別する特徴に State オブジェクトへの代入文を加えて検出を試みる．

2.2 検出方法

デザインパターンの設計にはいくつかの特徴がある．特徴とはクラス間の関係やクラスの構成やメソッド呼び出しなどの振る舞いである．これらの特徴の組み合わせからデザインパターンを識別する試みがある[2]．対象システムからデザインパターンを検出することは，特徴の組み合わせに一致するクラス群を見つけ出すことに等しい．そのためには，まず対象システムを構成するクラスのソースコードを Abstract Syntax Tree(AST)で表現する．AST で表現することで特徴を抽出しやすくするためである．抽出した特徴の組み合わせを比較する方法は，関連研究[3]でも採用されているテンプレートマッチングを用いる．テンプレートマッチングでは行列を比較対象とする．各行と列がクラスを示し，成分が特徴を示している．例えば，クラス i とクラス j の間になんらかの関係が存在するならば，成分 E_{ij} と関係を示す値の論理和をとる．各特徴を示す値は，あらかじめ 2 の累乗数を定めておく．これは論理演算によって比較するためである．デザインパターンのソースコードも同様に，AST で表現し，抽出した特徴から行列を生成し，テンプレートとする．このようにして生成した対象システムとデザインパターンの行列をテンプレートマッチングする．

2.3 パターン識別要素

デザインパターンを区別するための設計上の特徴をパターン識別要素とし，検出に用いる各識別要素の名前と説明を表 1 に示す．

表 1: パターン識別要素一覧

識別要素	説明
抽象性	クラスが抽象クラス，またはインターフェースである
継承	継承したクラス，または実装したインターフェース
集約	フィールドで宣言された配列などの複数個のデータを持たないオブジェクトの型
N 集約	フィールドで宣言された配列などの複数個のデータを持つオブジェクトの型

†静岡大学情報学部, Faculty of Infomatics, Shizuoka University

引数	メソッドで定義された引数の型
返り値	メソッドで定義された返り値の型
生成	生成文で生成されるインスタンスの型
Template &Hook	フィールドで宣言されたオブジェクトから呼び出されたメソッドが抽象メソッドである場合のオブジェクトの型 ※メタパターン[5]を構成するテンプレートクラスとフッククラスの関係
代入	フィールドで宣言されたオブジェクトに対して代入文がある場合のオブジェクトの型

3. 実験

実際に State パターンとして使われている箇所を検出する実験を行う．上述の State パターンを検出するために代入の識別要素を導入する．つまり，State オブジェクトに対して代入文の有無を調べる．そして，代入の識別要素を導入した場合と導入しなかった場合を比較する．

3.1 実験方法

オープンソースのソースコードから 2 種類の State パターンテンプレートを使い State パターンを検出する．代入の識別要素を導入するテンプレートと，導入しないテンプレートである．代入の識別要素を導入して検出することを「代入あり方式」，導入しないで検出することを「代入なし方式」とする．

検出数，実用数，検出時間を記録する．検出数とは Context と State の同じ組み合わせを持つ ConcreteState が 2 種類以上ある検出箇所の数とする．検出結果に該当するソースコードを調べ，実際に State パターンとして使われているか否かを判断する．判断基準は状態遷移する仕組みが存在するか否かとする．この調査結果の数を実用数とする．

代入あり方式の検出数と実用数を比較し，実際に State パターンとして使われている箇所が検出できているかを確かめる．代入あり方式の実用箇所と代入なし方式の実用箇所を比較し，代入なし方式で実際に State パターンとして使われていると判断した箇所が検出できていることを確かめる．検出時間を比較して，代入の有無による検出時間の差が検出方式によらないことを確認する．

実験対象のシステムを表 2 に示す．

表 2: 実験対象のシステム

システム	Version	URL
OpenProj	1.4	http://openproj.org/
JEdit	4.2	http://www.jedit.org/
Smack	3.1.0	http://www.igniterealtime.org/
JFtp	1.52	http://j-ftp.sourceforge.net/
Netty	3.2.0	http://www.jboss.org/netty.html

3.2 実験結果

実験結果を表 3 に示す．代入あり方式の検出数と実用数が等しいので，実際に State パターンとして使われている箇所が検出できていることが確認できた．代入あり方式の実用箇所と代入なし方式の実用箇所を比較すると，代入あり方式の全ての実用箇所は代入なし方式で検出した箇所に含まれていた．しかし，OpenProj と JEdit では代入なし方式で実際に State パターンとして使われていると判断した箇所が代入あり方式では全て検出できていない．

また，代入の識別要素を導入したことで不一致箇所をより早く取り除くことができるようになり検出時間が約 1~3%短縮された．

表 3: 代入を導入しない場合とした場合の検出結果

	代入なし方式		代入あり方式	
システム	検出数	実用数	検出数	実用数
OpenProj	12	10	3	3
JEdit	12	6	5	5
Smack	17	2	2	2
JFtp	3	1	1	1
Netty	39	14	14	14

4. おわりに

実験結果から代入文の識別要素が実際に使われている State パターンの検出に有効であることがわかった．しかし，状態遷移を示す代入文の識別要素には問題が残されている．

- 代入文はあるが，null が代入されるだけでオブジェクトが代入されず状態遷移が起こらない
- 代入文のあるメソッドがコンストラクタからのみ呼び出されているため，動的な代入が不可能である
- 特定のクラスのオブジェクトが代入されるのみで他の状態に遷移していない

また，状態遷移を示す識別要素は Context 内の代入文の有無だけでは不十分である．それは状態を保持する State オブジェクトの可視性が private でない場合，他クラスからの直接アクセスやサブクラスにおける代入文が考えられるためである．そのため，代入なし方式では実際に State パターンとして使われていると判断した箇所が代入あり実験では検出できなかった．

これらの問題を解決することで，実際に State パターンとして使われている箇所が検出できると考えられる．また，他のパターンにおける実用箇所の検出においても，現在の識別要素を使った検出結果から実際にパターンが使われている箇所の特徴から共通の特徴を割り出し，新たに識別要素に加えることで，State パターンと同様に実際に使われているパターンが検出できると考えられる．

参考文献

[1]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides 著, 本位田 真一, 吉田 和樹 監訳: オブジェクト指向における再利用のためのデザインパターン 改訂版, ソフトバンク パブリッシング(1999).

[2]Jing Dong, Yongtao Sun, Yajing Zhao: "Design pattern detection by template matching", Proceedings of the 2008 ACM symposium on Applied computing pp. 765-769 (2008).

[3]N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis: "Design Pattern Detection Using Similarity Scoring", IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 896-909 (2006).

[4]Nikolaos Tsantalis: Design Pattern Detection, java.uom(online), available from <http://java.uom.gr/~nikos/pattern-detection.html> (accessed 2010-06-22).

[5]W.プリー, 佐藤啓太, 金澤典子,:デザインパターンプログラミング（補訂版）, 株式会社トッパン(1998)