B-020

デザインパターン検出結果を基にクラスを配置したプログラム設計理解のためのクラス図生成
Class Diagram Generation for Understanding Program Design Based on the Design Pattern Detection Result

鵜飼公平 † 酒井三四郎 † 松澤芳昭 †
Kohei Ukai Sanshiro Sakai Yoshiaki Matuzawa

1. はじめに

プログラムの設計を理解する方法の 1 つにデザインパターン検出がある。ソースコードから構造情報を抽出し、クラス図を生成するリバースエンジニアリングという方法もあるが、構造情報から設計意図を読み取ることは容易でない。デザインパターン検出ならばリバースエンジニアリングによって抽出される構造情報に設計意図を補足することができる。

デザインパターンには、それぞれに設計意図を持たせて作られている。デザインパターンは GoF[3] によって度々直面する設計問題とその解法を分類し一般化された。デザインパターンを構成するクラスには、設計問題を解決するための役割と協調関係が与えられているため、設計意図を読み取ることができる。

本論文では、デザインパターン情報をクラス図のクラス配置に利用することで、プログラムの設計理解に役立てることができることを確認した。デザインパターンが適用されているプログラムからクラス図を生成し、デザインパターン情報に基づいてクラスを配置した。そのクラス図を用いてプログラムを解析し、設計を理解することで、どのように役に立ったかを評価した。

2. 関連研究

本システムで使用するデザインパターン情報は P-MARt[2] を採用した。P-MARt から得られるデザインパターン情報は手作業でプログラムソースの中から探索されており、その精度は高いことが期待できる。精度の高いデザインパターン情報を用いるため、その情報を基にクラスが配置されたクラス図からは、正しい設計意図を読み取ることができる。

本システムのユーザインタフェースは UDoc[1] というリバースエンジニアリングツールを参考にしている。UDoc はユーザが指定したクラスをクラス図に配置すると、配置したクラスと継承・関連・集約の関係のあるクラスを四方に展開できるようになる。展開したクラスから再び四方に関係のあるクラスを展開でき、この作業を繰り返すことでユーザが指定したクラスと関係のあるクラス間の構造が把握できる。本システムでも、ユーザが指定したクラスと関連の関係を持つクラスを展開できる。

本システムに入力するデザインパターン情報はデザインパターン検出器で検出された結果であることを想定している。デザインパターン検出研究の調査が行われた論文 [4] では、検出結果の可視化はほとんど考慮されていないという結果が得られている。そのため、本論文の試みは新規性が高い。

†静岡大学大学院情報学研究科, Shizuoka University Graduate School of Informatics

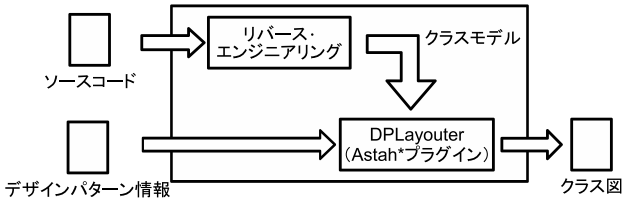

図 1: DPLayouter のシステム構成図

3. 提案システム：DPLayouter

3.1. 概要

本研究では、デザインパターンを利用した、理解容易性の高い UML クラス図の自動生成を目的としたソフトウェア「DPLayouter」を提案する。DPLayouter は、リバースエンジニアリングによって作成されたクラス図に対して、デザインパターンに基づく配置を行う。デザインパターンには設計意図が込められており、配置の慣習といった設計者の共通知識も含まれているため、クラス図の理解容易性の向上が期待できる、というのが我々の仮説である。

DPLayouter のシステム構成を図 1 に示す。DPLayouter は UML モデリングツールである Astah*のプラグインとして設計されている。DPLayouter が組み込まれた Astah*への入力データは、ソースコードと「デザインパターン情報」である。ソースコードのリバースエンジニアリングによってクラスモデルを生成する部分は Astah*の機能を利用する。DPLayouter は生成されたクラスモデルとデザインパターン情報を入力として、デザインパターンに基づく配置が適用されたクラス図を生成する。

入力される「デザインパターン情報」とは、入力対象のソースコードに適用されたパターン（パターンインスタンス）が形式的に記述されたものである。現在、本システムは P-MARt リポジトリが採用している XML 形式のデザインパターン情報の入力に対応している。将来的には、デザインパターン検出器を利用して検出した情報が入力されることが期待される。

3.2. 機能とユーザインタフェース

本システムのユーザインタフェースを図 2 に示す。本システムにはユーザが選択したクラスを起点に選択したクラスと関連の関係にあるクラスを配置する「関連クラス展開形式」でクラスを配置する。起点となるクラスは構造ツリーか表示されているクラス図の中か

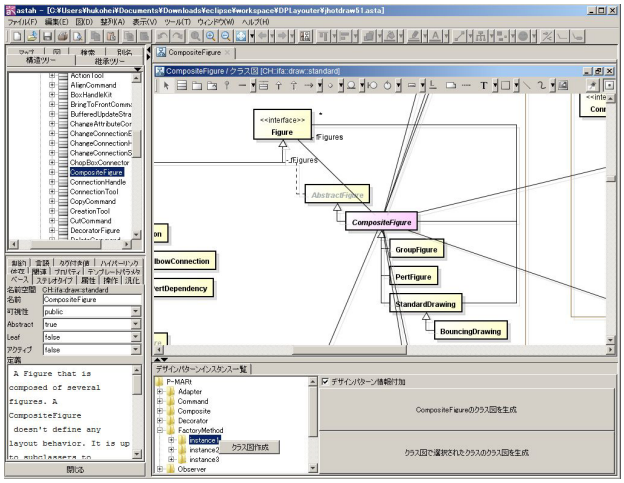

図 2: DPLayouter のインタフェース

ら選択する。選択したいクラスを含むパターンが分かるならば、デザインパターンインスタンス一覧からパターンインスタンスを選択し、起点クラスを選ぶこともできる。

本システムが想定している使用プロセスについて説明する。ユーザは最初に構造ツリーから起点となるクラスを選択し、クラス図を表示させる。表示されたクラス図から、さらに関係を追跡したいクラスを選択し、クラスを表示させる。このように、ユーザは起点となるクラスを移り変えていくことで、クラス構造を探索的に理解する。

3.3. クラス配置のアルゴリズム

3.3.1. クラス図の基本アルゴリズム

ユーザが選択したクラスを「起点クラス」とし、起点クラスが関連先となるクラスを「関連元クラス」とし、起点クラスが関連元となるクラスを「関連先クラス」とする。起点クラスは中央に、関連元クラスは左側に、関連先クラスは右側に配置される。関連元クラスは Astah*のリバースエンジニアリングにより読み込まれた順で縦一列に配置される。配置された関連元クラスは起点クラスと関連線で結ばれる。関連先クラスも関連元クラスと同様に配置される。

実際に配置されたなクラス配置を図 3 に示す。ViewerModel が起点クラスとなっている。SourceCodePanel、AttributeSubMenu、ASTPanel、EvaluationResultPanel、XPathPanel、ASTNode-PopupMenu と、MainFrame が ViewerModel に対して関連を持っているので、関連元クラスとなり、ViewerModel の左側に縦一列に配置されている。ViewerModel は SimpleNode、ViewerModelListener に対し関連を持っているので、それらのクラスは関連先クラスとして ViewerModel の右側に縦一列で配置されている。

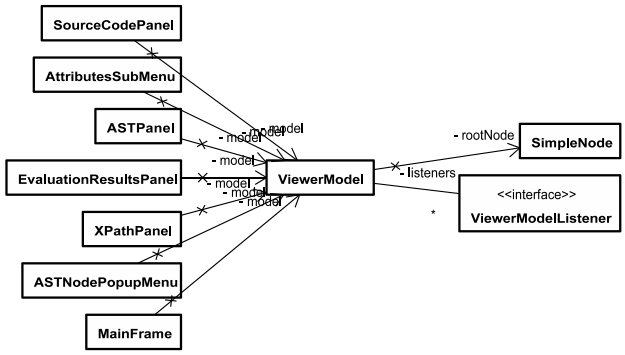

図 3: 基本的なクラス配置（具体例）

3.3.2. デザインパターン情報の付加

本システムでは「関連クラス展開形式」にデザインパターン情報を付加したクラス図を生成する。その具体例を図 4 に示す。デザインパターン情報付加の仕様は、起点クラス、関連元クラス、関連先クラスがいずれかのパターンインスタンスに含まれている場合、そのパターンインスタンスに含まれる全てのクラスが配置される、というものである。

デザインパターン情報を付加した関連クラス展開形式の具体例として、図 4 に示した例のケースを利用して述べる。起点クラスの ViewerModel が Observer パターンインスタンスに含まれているので Observer パターンインスタンスが配置される。関連元クラスの AttributeSubMenu と ASTNodePopupMenu はいずれのパターンインスタンスに含まれないので、起点クラスの左側に縦一列に配置される。それ以外の ASTPanel、EvaluationResultPanel、MainFrame、SourceCodePanel と、XPathPanel は ViewerModel が含まれた Observer パターンインスタンスに含まれているので、左側には配置されない。関連先クラスである ViewerModelListener も ViewerModel が含まれた Observer パターンインスタンスに含まれるので、右側には配置されない。それ以外の関連先クラスである SimpleNode は Visitor パターンインスタンスに含まれるので、右側に Visitor パターンインスタンスが配置される。

3.3.3. デザインパターン種類毎の配置アルゴリズム

パターンインスタンスに含まれるクラスは、Gamma らによるデザインパターンの原典 [3] と同様の構造を持つように配置される。Observer パターンの例をソースコード 1 に示す。13 行目では ConcreteSubject に該当するクラスを、Subject に該当するクラスが親クラスとなる階層構造になるよう配置している。階層構造の配置等の共通処理をライブラリ化することで、クラスの配置を十数行で定義できる。

4. 評価実験

4.1. 評価方法

DPLayouter を JHotDraw5.1 の読解問題に適用することで、DPLayouter の有用性を検証した。筆頭著者が

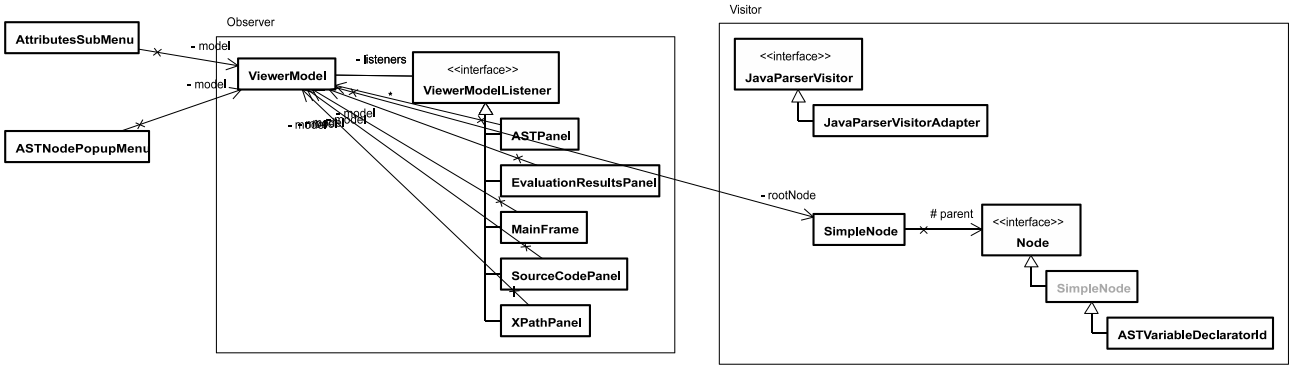

図 4: パターンインスタンスが含まれるクラス配置

ソースコード 1: Observer パターンの定義

```
1  public class ObserverLayout extends DesignPatternLayout {
2
3    public ObserverLayout(IDiagram diagram, PatternInstance
         patternInstance) {
4      super(diagram, patternInstance);
5    }
6
7    @Override
8    protected void doLayout() throws Exception {
9      // Subject を配置
10     SingleLayout subjectLayout = layoutSingle("Subject");
11
12     // ConcreteSubject の階層構造を Subject の右下に配置
13     InheritanceHierarchyLayout subjectHierarchyLayout =
           layoutHierarchically(
14         "ConcreteSubject", subjectLayout);
15
16     // Subject と ConcreteSubject を継承で繋ぐ
17     connectHierarchy(subjectLayout, subjectHierarchyLayout);
18
19     // Observer を Subject の右に配置
20     SingleLayout observerLayout = layoutSingle("Observer");
21     moveRightOfHierarchy(observerLayout, subjectLayout,
22         subjectHierarchyLayout);
23
24     // ConcreteObserver の階層構造を Observer の右下に配置
25     InheritanceHierarchyLayout observerHierarchyLayout =
           layoutHierarchically(
26         "ConcreteObserver", observerLayout);
27     connectHierarchy(observerLayout, observerHierarchyLayout);
28
29     // Subject と Observer を関連で繋ぐ
30     connectByAssociation(subjectLayout, observerLayout);
31   }
32 }
```

DPLayouter を利用して JHotDraw5.1 の読解を行う過程について記録し、その過程の質的分析を行った。

結果で示されているクラス図は、用紙の関係上本システムで生成されたクラス図をトリミングし、一部クラスを省略されている。

4.2. 結果

4.2.1. 図形の定義の設計

被験者は、まず、フレームワークに付属しているサンプル JavaDrawApp を実行することで、JHotDraw がどのような Draw 系ソフトウェアを開発するフレームワークであるのかを理解している。次に、JavaDrawApp のソースコードを参照することで、createTools() でツールバーにツールを登録していることを理解している。createTool() より、サンプルで表示できる図形がそれぞれ RectangleFigure、RoundRectangleFigure、EllipseFigure、LineFigure で定義されていることを理解している。被験者は、ここで構造ツリーから起点クラスとして JavaDrawApp を選択し、クラス図（図 5）を生成し、JavaDrawApp は Composite パターンのインスタンスであることを確認している。Composite パターンインスタンスにおいて、RectangleFigure などが Leaf に該当することが分かり、Composite パターンの設計意図から図形は Figure のサブクラスの複合オブジェクトで定義できることを理解している。

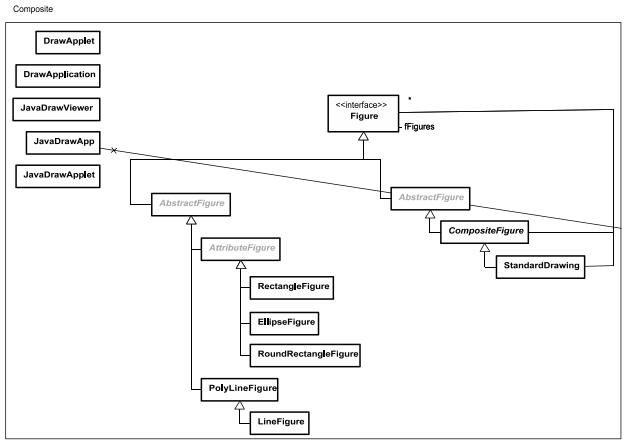

図 5: JavaDrawApp を起点クラスとしたクラス図

4.2.2. 編集領域に図形を出現させる設計

被験者は、JavaDrawApp を起点クラスとしたクラス図から Figure を起点クラスとしたクラス図 (図 6) を生成し、Figure が Prototype パターンインスタンスであることを確認している。Prototype パターンインスタンスにおいて、CreationTool が Client に該当し、Prototype パターンの設計意図からダイアグラムエディタの編集領域上に出現している図形は CreationTool に登録した図形のクローンを用いる設計だということを理解している。

4.2.3. 選択したツールを保持する設計

被験者は、Figure を起点クラスとしたクラス図から CreationTool を起点クラスとしたクラス図（図 7）を生

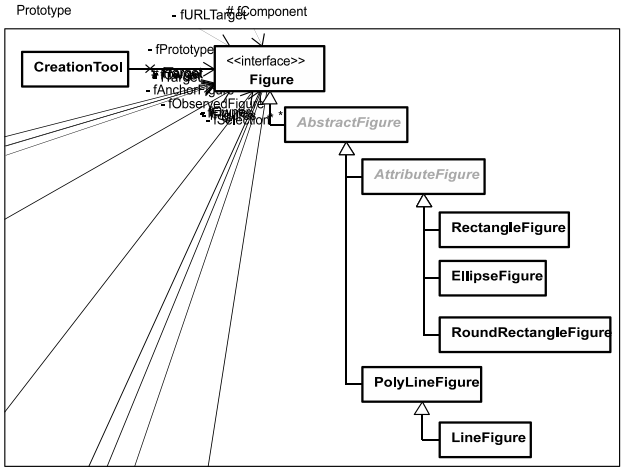

図 6: Figure を起点クラスとしたクラス図

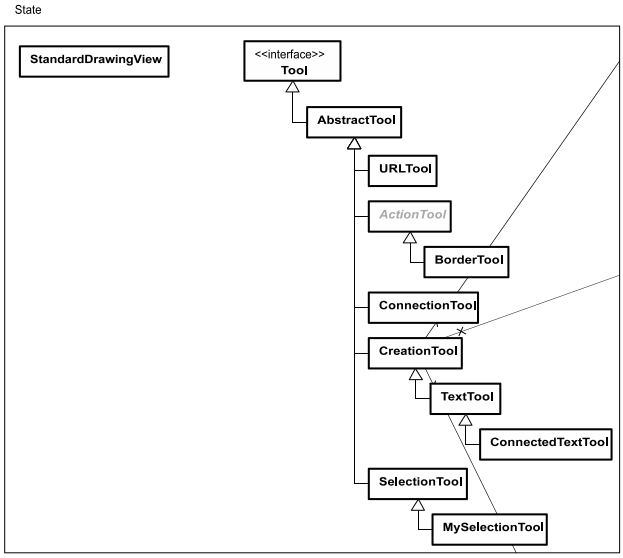

図 7: CreationTool を起点クラスとしたクラス図

成し、CreationTool が State パターンインスタンスにも含まれることを確認している。StandardDrawingView が Context、Tool が State に該当することから、どのツールが選択状態なのかを StandardDrawingView がわかる設計になっていることを推測していた。

4.2.4. 図形の更新設計

被験者は、CreationTool を起点クラスとしたクラス図から StandardDrawingView を起点クラスとしたクラス図 (図 8) を生成し、StandardDrawingView が Observer パターンインスタンスに含まれることを確認している。その Observer パターンインスタンスにおいて、StandardDrawingView は Drawing から DrawingChange イベントを受け取ることを理解している。また、Figure が関連先クラスであり、別の Observer パターンインスタンスに含まれていることを確認している。その Observer パターンインスタンスにおいて、Figure は FigureChange イベントを Drawing に通知していることを確認している。これら 2 つの Observer パターンインスタンスから図形の更新は StandardDrawing を介して StandardDrawingView に通知する設計になっていることを推測していた。実際に StandardDrawing のソースコードを参照し、推測が正しいことを確認している。

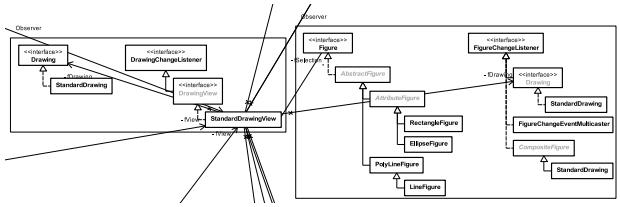

図 8: StandardDrawingView を起点クラスとしたクラス図

4.3. 結果のまとめ

結果より、本システムを設計理解や構造探索に役立てていることを述べる。Figure が Composite パターンに含まれていることから図形の定義方法を理解することから、デザインパターン情報を設計理解に役立てている。JavaDrawApp を起点クラスとするクラス図から関連の関係にない Figure を起点クラスにすることから、デザインパターンインスタンスに含まれるクラスを配置することは、関連の関係のあるクラス以外にも探索経路をユーザに与えている。

5. おわりに

プログラム設計理解を目的とした、デザインパターン情報に基づきクラスを配置するシステムを開発した。実際に JHotDraw5.1 の解析に使用し、設計理解に有用であることを確認した。デザインパターン情報に基づきクラスを配置したことで、位置関係から設計意図を捉えることが容易になった。

参考文献

[1] Christopher Deckers, UDoc, <http://udoc.sourceforge.net/main/index.html> (accessed 2013-06-26)

[2] Yann-Gal Guhneuc, “PMARt: Pattern-like Micro Architecture Repository”, Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories, July 2007. Note: 3 pages

[3] 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)

[4] Ghulam Rasool, Detlef Streitfdert, “A Survey on Design Pattern Recovery Techniques”, International Journal of Computer Science Issues Vol.8 Issue 6, pp.51-260, Nov 2011