デザインパターンの適合性確認方法に関する研究

2003MT012 後藤 寛 2003MT058 水野 佳範

指導教員 青山 幹雄

1. はじめに

ソフトウェア設計に対して繰り返し現れる問題と解決策を明示的に整理したものがデザインパターンである．デザインパターンの適用は再利用性の高いソフトウェア設計を可能にする．しかし，デザインパターンの適用は様々なスキルが必要であるため，デザインパターンの意図からの逸脱が起こる．本研究では意図からの逸脱を防ぐために適合性の評価方法を提案する．

2. デザインパターン適用における問題

2.1. 意図からの逸脱

パターン適用のスキル，知識が不足するとパターンの意図からの逸脱が発生する．意図からの逸脱が起こることにより，課題が解決されないだけでなく，パターンを適用したソフトウェアの生産性や品質の低下を起こしてしまい，再利用性を阻害する新たな課題を生じる原因となる．

2.2. 逸脱の発生時

逸脱の発生時は次の 3 つである．

(1) パターンの選択時

非機能要求を意識し，パターンを選択するとき．

(2) パターンの展開時

構造や協調関係を正しくリネーム，マッピングするとき．

(3) ソフトウェアの変更時，拡張時

パターンを適用したソフトウェアのパターンの構造や協調関係の変更をするとき．

2.3. 逸脱の内容

パターンカタログ[1]において，パターンの特徴を特に示しているのは構造，協調関係，目的の 3 つの項目である．この 3 つの項目に対する逸脱の内容の例を示す(表 1)．

表 1 逸脱の内容の例

項目	逸脱の内容
目的	•適用するデザインパターンがその意図とは違う目的で利用
協調関係	•パターンを構成しているメソッドの不正な呼び出し •パターンを構成しているクラスを不正に生成
構造	•クラス関係（継承・委譲）が不正 •不正なメソッドの再定義 •必要（不必要）メソッドの不定義（定義） •要素（クラス）個数の不正

3. デザインパターン適合性評価方法

3.1. 適合性の基準

パターンの意図からの逸脱を防ぐために，適合性の評価方法を提案する．適合性の評価はパターンを用いて設計したソフトウェアが，パターンの意図したものとなっているかを確認するものである．適合性評価の視点として以下の 3 つを挙げる．

(1) 構造

パターン適用ソフトウェアがパターンの意図した構造になっているか．

(2) 協調関係

パターン適用ソフトウェアがパターンの意図した協調関係になっているか．

(3) 目的

パターンがソフトウェア設計者の設計課題に対し，パターンの目的に従って適切に選択されているか．

3.2. 扱う視点

本研究では構造の視点から適合性を評価し，協調関係，目的の視点は扱わない．構造の視点からは，パターン適用後のソフトウェアの構造が複雑化しても，基盤となる構造は変化しないため，カタログの記述と比較が可能である(図1)．

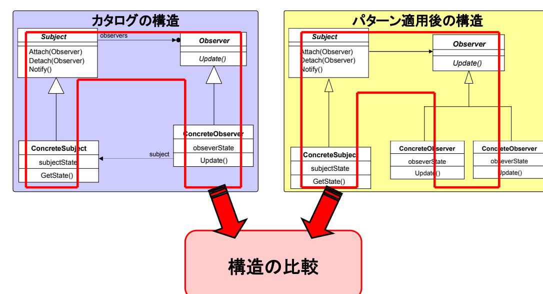

図 1 構造の視点からの適合性評価

一方，協調関係の視点からは，複数のパターンを適用したソフトウェアの複雑な実行トレースをカタログの抽象化された協調関係と比較することは困難である．

目的の視点からは，選択段階において設計者の課題や非機能要求を把握することが必要になるため，適合性を評価できない．

3.3. 構造適合性評価の要素

本研究ではパターンの構造適合性評価の要素を以下のように定義する.

(1) ロール

パターンを構成するクラス.

(2) クラス間の関係

ロールと他のロールとの継承関係.

(3) メソッド

ロールが持つべきメソッド.

(4) フィールド

ロールが持つべきフィールド.

3.4. Observer パターンの構造適合性評価の要素

例として, 図 2 の Observer パターンのクラス図を用いて構造適合性評価の要素を以下に示す.

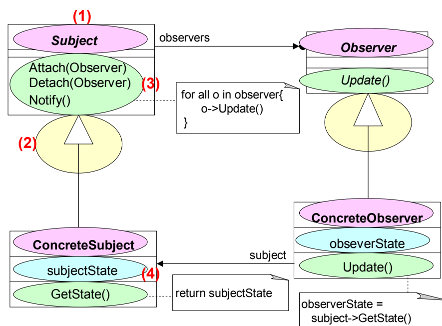

図 2 Observer パターンの構造

(1) クラスのロール

Subject, Observer, ConcreteSubject, ConcreteObserver

(2) クラス間の関係

ConcreteSubject は Subject を継承している.

ConcreteObserver は Observer を実装している.

(3) メソッド

Subject はメソッド Attach, Detach, Notify を備える.

Observer はメソッド Update を備える.

ConcreteSubject はメソッド GetState を備える.

ConcreteObserver はメソッド Update を備える.

(4) フィールド

ConcreteSubject はフィールド subjectState を備える.

ConcreteObserver はフィールド observerState を備える.

4. 構造適合性評価の方法

4.1. 適合性評価方法

デザインパターンを適用したソフトウェアの構造に対して適合性を評価する方法を図 3 に示す.

個々のパターンの構造を定義し, パターンの展開時, パターン適用ソフトウェアの変更, 拡張時に展開や変更を行ったパターンが, 構造の定義に従っているかを確認する.

ソースコード内にパターンのロール, ロールが持つメソッド, フィールドといったパターンの構成要素が存在するか, また, 継承などのクラス間の関係が存在するかを確認する.

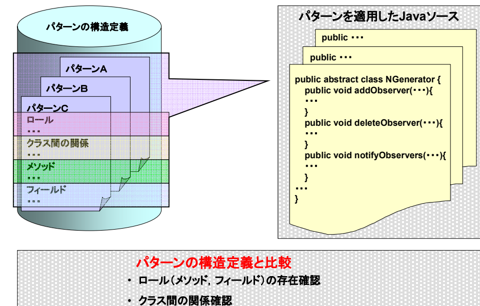

図 3 適合性評価方法

4.2. アノテーションによるロール識別

ソフトウェア設計者がパターンを適用するにはカタログに記載されているクラス名, メソッド名などを設計者の課題に合わせてリネームするため, 設計者が独自に付けた名前に対しロールを確認するのは困難である. そこで設計者が独自に付けた名前に対し, アノテーションを用いることでロールを識別する方法を提案する.

4.3. アノテーション付加

アノテーションを用いてパターンのロールを定義する情報を付加する. アノテーションは, パターンの適用を行うソフトウェア設計者がソフトウェアを実装するときソースコードに付加するものとする.

図 4 ではアノテーションの付加方法を示している. あるクラスがパターンのロールである場合, クラス宣言の前に @DPType を付加する. また, このロールがメソッドを持つ場合, これらに該当するメソッドの前に@DPMethod を付加する.

@DPType をクラス宣言の前に付加する場合, name の値には適用しているパターン名, role の値にはそのパターンのロール名, inst の値にはそのパターンを示すインスタンス名を記述する. @DPMethod をメソッドに付加する場合には, role の値にそのパターンのロールが持つメソッド名を記述する. @DPField の場合も同様に role にフィールド名を記述する.

```
@DPType(name = "Observer", role = "Subject",inst = "Observer1")
public abstract class NumberGenerator {
    @DPMethod(role = "Attach")
    public void addObserver(NGObserver observer){
    ・・・
    @DFMethod(role = "Detach")
    public void deleteObserver(NGObserver observer){
    ・・・
    @DPMethod(role = "Notify")
    public void notifyObservers(){
    ・・・
```

図 4 アノテーションを付加したソースコード

4.4. アノテーション付加によるメリット

パターンを適用するソースコードにアノテーションを付加するメリットは次の 2 つである.

(1) 可読性の向上

複雑なソフトウェアになるにつれて, ソフトウェアのソースコードは繁雑になる. アノテーション付加により, プログラムに適用されたパターンが認識でき可読性が向上する. ロール名の付加により, あるクラスがパターンのどのロールの役割を担っているかなどの認識, 理解に役立つ.

(2) ツール化の容易さ

アノテーションはコンパイルされたプログラム中に情報として意味づけされたオブジェクトを残す. 実行時にアノテーションにアクセスすることができる方法としてリフレクション API が J2SE 5.0 によってサポートされているため, アノテーションを処理するツールの実現が容易である.

5. 適合性評価支援環境

5.1. 支援環境の機能

本研究ではパターンを適用しているソフトウェアの実装中に, パターンのロールを担うクラスの欠如やロールに必要なメソッドの未定義, フィールドの不足, ロールを担うクラス間の関係の不整合などの情報を視覚的に表示することによって適合性を評価する.

5.2. ツールの開発

本研究では Eclipse プラグインである CheckStyle[4]にパターンの構造の適合性を確認するモジュールを追加することにより適合性を評価する.

CheckStyle とは Java コードの記述形式がコーディング規約に準じているかをチェックするオープンソースの解析ツールである. Eclipse と連携し, Java ソースの保存後, コンパイルと同時に実行され, チェック結果を Eclipse ビューに表示する. CheckStyle の各チェック機能は Java のクラス単位で扱われている.

ツールは Java により実装し, 各パターンの構造のチェック機能をクラス単位で実装した. ツールのプログラム規模を表 2 に示す.

表 2 プログラム規模

クラス数	28個
総コード行数	873行

パターン構造定義ファイル数	23個
パターン構造定義ファイルの平均行数	18行

5.2.1. パターン構造定義ファイルの作成

アノテーションの比較に用いるパターンの構造定義を XML によって記述する(図 5). 各クラスの役割を role タグと表現し, role タグの属性として, 各クラスの名前, クラス間の関係等を埋め込んでいる. Method タグにはそのロールに必要なメソッドの情報を, Field タグには必要なフィールドの情報を name という属性を用いて表現する.

```
<?xml version="1.0" encoding="Shift_JIS"?>
<DP xmlns="http://www.dp.com/namespaces/dp">
    <Pattern name="Observer">
        <role name="Subject" implements="" extends="">
            <Method name="Attach"></Method>
            <Method name="Detach"></Method>
            <Method name="Notify"></Method>
            <Field name=""></Field>
        </role>
        <role name="ConcreteSubject" implements="" extends="Subject">
            <Method name="GetState"></Method>
            <Field name="subjectState"></Field>
        </role>
        <role name="Observer" implements="" extends="">
            <Method name="Update"></Method>
            <Field name=""></Field>
        </role>
        <role name="ConcreteObserver" implements="Observer" extends="">
            <Method name="Update"></Method>
            <Field name="observerState"></Field>
        </role>
    </Pattern>
</DP>
```

図 5 パターン構造定義ファイル

5.2.2. 処理のプロセス

実装するモジュールの処理のプロセスを図 6 に示す.

(1) class ファイルの読み込み

リフレクションでアノテーションを読み込むために, Eclipse のワークスペース内の現在のプロジェクトから.classpath ファイル(XML)を読み込み, class ファイルの位置を特定する.

(2) パターン構造定義ファイルの読み込み

パターン構造定義ファイルを読み込むために, 環境変数として指定された"DPPATH"からパターン構造定義ファイルの位置を特定する. また, パターン構造定義ファイルを解析する際にはそのファイルがスキーマに従っているか検証する.

(3) 構造の比較結果を出力

パターン構造定義ファイルと class ファイルのアノテーションを比較し, コンパイル時に適用しているパターンに不足しているロール等の情報を Eclipse ビューに出力する.

図 6 処理のプロセス

5.2.3. 実行結果

パターンを適用したプログラムに対するツールの実行結果を示す．Observer パターンを適用途中のクラス図を図 7 に示す．この時点での実行結果を図 8 に示す．図 7 では Observer パターンのロールである Subject クラスを実装していないため，図 8 のように Eclipse ビューに不足しているロール Subject とその関係が表示される．また Observer パターンのロールである ConcreteSubject と ConcreteObserver に必要なフィールドが定義されていないので Eclipse ビューに，ロールに不足しているフィールドが表示される．

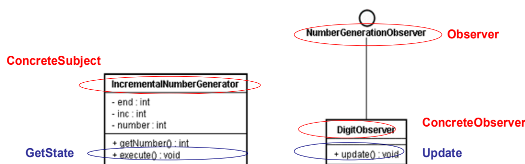

図 7 Observer パターン適用途中のクラス図

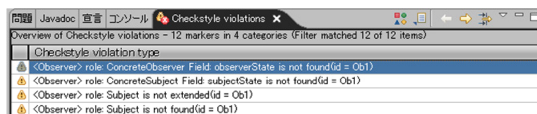

図 8 実行結果

6. 考察

6.1. ツールの効果

本研究で提案した方法とツールを用いてパターンの適合性を評価することにより，期待できる効果を以下に示す．

(1) 適用状況の把握による逸脱の防止

ある時点で不足しているパターンのロール，またロール間の関係の不整合などを表示することにより，パターンの適用状況を把握することが容易になり構造からの逸脱を防ぐことができる．

(2) ソフトウェアの変更，拡張時の逸脱の防止

パターンを適用したプログラムの変更を行う際，パターンのロールとなるクラスを新たに加える場合に，不足しているパターンのロール，またロール間の関係の不整合などを表示することにより適用されているパターンの構造を正しく把握でき，構造からの逸脱を防ぐことができる．

6.2. ツールの実装

提案するツールのパターン構造定義ファイルは 23 個の GoF のパターンを定義している．あるパターンが GoF カタログと異なったバリエーションでもそのパターンと定義される場合がある．本研究で作成したツールはパターン構造定義ファイルを XML 形式で記述しており，構造定義を変更するにはパターン構造定義ファイルを書き換えるだけで，構造を変更できる．

6.3. 関連研究

文献[2]では構造，協調関係に関する適合性を，パターンの協調関係にあるクラス間の関係のメッセージの送受信として適合性の検証を提案している．しかし，適合性の検証をツールで支援していない．適合性の確認を人手で行うのは困難であり，ツールを用いた適合性の確認は欠かせないと考えられる．

文献[3]ではパターンの適用で起こりうる逸脱，それに対する解決策や防止策を明文化することを提案している．逸脱を予め予想しているが，逸脱の確認方法は提案していない．

7. 今後の課題

(1) メソッドの処理内容の抽出

メソッドのロール確認は行ったが処理内容までは確認できていない．メソッドの処理内容の確認方法を考える．

(2) ツールの拡張性

GoF 以外のパターンの構造をこのツールで適合性を確認するには，パターンの構造定義ファイルを作成するとともに新たなモジュールを追加する必要がある．構造定義ファイル，モジュールの再利用性の向上を考える．

(3) 他の視点からの適合性の評価

構造の視点だけですべての逸脱が防げるとは限らない．他の 2 つの視点を含めた総合的な評価が必要である．

8. まとめ

本研究ではデザインパターンの意図からの逸脱を問題とした．アプローチとして逸脱を防ぐために適合性の評価を提案し，構造の視点からの適合性評価を扱った．構造を評価する方法としてソースコードにアノテーション付加を行った．人手での確認が困難なため適合性評価のためのツールを作成し，作成したツールの有効性を評価した．

参考文献

[1] E. Gamma, et al., 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，ソフトバンクパブリッシング, 1999.

[2] 畑口 剛之, 他, デザインパターンへの適合性確認手法について，情報処理学会ソフトウェア工学研究会, No. 121-20, Nov. 1998, pp. 155-162.

[3] 鹿糠 秀行, 他, ソフトウェアパターンの適用における逸脱パターンの提案，情報処理学会ソフトウェア工学研究会, No.137-5, May 2002, pp. 33-40.

[4] CheckStyle Home Page, http://checkstyle.sourceforge.net/.