ソフトウェア工学 136－20
（2002．3．8）

デザインパターンを用いたシステム開発支援に関する提案

追立 正司† 藤尾 光彦‡

† 九州工業大学大学院 情報工学研究科
‡ 九州工業大学 情報工学部
†‡ 〒820－8502 福岡県飯塚市川津 680－4 Tel. 0948－29－7740
E-mail: †syo@rieman.ces.kyutech.ac.jp , ‡fujio@ces.kyutech.ac.jp

概要

直面している問題を解く時に、プログラマは経験によってパターンを選択している。それは、パターンを利用しているプログラマだけでなく、パターンの作者であってもパターンの応用の可能性を完全に理解しているわけではないからである。本研究では、設計中のユースケースに対して、最適な設計クラスをデザインパターンという形で提示する方法の提案を行う。パターンの導出には意思決定の一つである AHP(階層分析法)を用いる。

Proposal concerning the software development support that used Design Patterns

Syouji Oitate† and Mitsuhiko Fujio‡

†Graduate School of Computer Science and Systems Engineering,
Kyusyu Institute of Technology
‡Faculty of Computer Science and Systems Engineering,
Kyusyu Institute of Technology
††Kawazu 680-4, Iizuka, Fukuoka 820-8502, Japan Tel. 0948－29－7740
E-mail: †syo@rieman.ces.kyutech.ac.jp , ‡fujio@ces.kyutech.ac.jp

Abstract

Programmer selects suitable Design Pattern to solve problems by his experiences. Because, not only programmers using Design Patterns but also Designer do not understand the whole possibility of application of Design Patterns. In this article, we propose a method that presents an optimal design class for the usecase under designing.We use Analytic Hierarchy Process(AHP) that is one of Decision making Method to derive a Design Pattern.

1 はじめに

経験豊かなプログラマは、経験不足なプログラマよりずっと生産的である。それは、プログラマが経験によっていろいろな知恵を身に付けたからである。そして、経験を積むに従い、プログラマは新しい問題と以前に解いた問題との類似性に気づく。プログラマは繰り返される事柄についての知恵を得ることで、パターンを適用する状況を認識でき、そこから可能な解法を見つけることができるのである。しかし、経験の少ないプログラマやパターンを知らないプログラマが、パターンを適用する状況を正しく認識することは難しい。本研究では、Erich Gamma, Richard Helm, John Vlissides, Ralph Johnson ら

によって出版された「デザインパターン」という本(Gof本)に記されているパターンが既存のJavaアプリケーション内のどのモジュールにおいて利用されているかを検索するシステムを構築する。そして、得られた値を利用して意思決定の一つであるAHP(階層分析法)を用いて状況に適用したパターンを導出する方法を提案する。

2 デザインパターン

2.1 デザインパターンのカタログ

デザインパターンとは「ソフトウェア開発において繰り返し発生する問題に対する、再利用可能な解法を言葉で記述したもの」である。Gof本では、生成、構造、振舞いという3つの目的によって以下のようにカタログが分類されている。

- 生成に関するパターン
  - AbstractFactory
  - Builder
  - FactoryMethod
  - Prototype
  - Singleton
- 構造に関するパターン
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- 振舞いに関するパターン
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Medhiator
  - Memento
  - Observer
  - State
  - Strategy
  - TemplateMethod
  - Visitor

2.2 各デザインパターンの仕様

Gof本では、パターンをC++によって論じている。しかし、現在のオブジェクト指向言語の主流はJavaである。本研究では、Javaベースのデザインパターンを検索するための仕様を考え、それを本システムでは実装している。ここでは、AbstractFactory、Adapterを検索するための仕様について簡単に述べる。

2.2.1 AbstractFactory

図2にAbstractFactoryパターンのクラス図を示す。

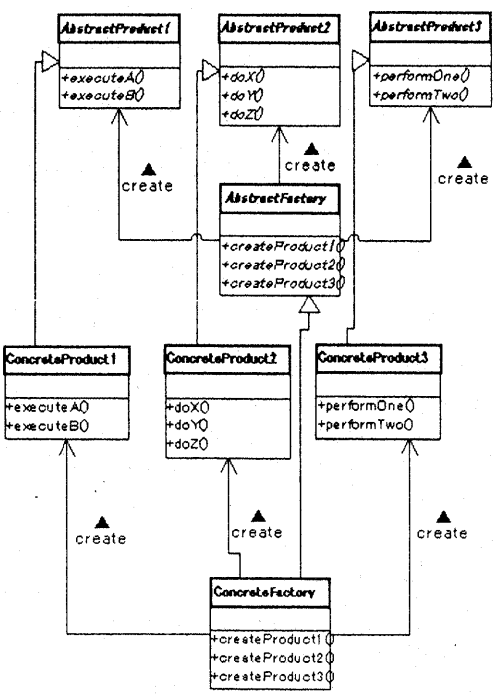

図1: Abstract Factoryパターンのクラス図

- **AbstractProduct(抽象的な製品)**
  - 抽象クラスまたはインタフェース。
  - インタフェース API を必ず 1 つ以上持つ。
- **AbstractFactory(抽象的な工場)**
  - 抽象クラスまたはインタフェース。
  - AbstractProduct のインスタンスを生成するためのインタフェース API を AbstractProduct の数だけ持つ。また、このインタフェース API の返り値はそれぞれの AbstractProduct 型である。
- **ConcreteProduct(具体的な製品)**
  - AbstractProduct を extends または implements する。
  - AbstractProduct のインタフェース API を実装する。
- **ConcreteFactory(具体的な工場)**
  - AbstractFactory を extends または implements する。
  - AbstractFactory のインタフェース API を実装する。

2.2.2 Adapter(継承)

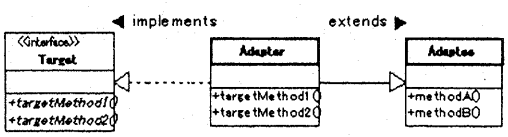

図 2: Adapter(継承) パターンのクラス図

- **Target(対象)**
  - インタフェース。Adaptee を Adapter が継承するため、Target は抽象クラスになることは出来ない。
  - Client が必要としているインタフェース API を持つ。
- **Adaptee(適合されるもの)**
  - Target および Adapter が必要としている具象メソッドを持っている。
- **Adapter(適合させるもの)**
  - Target を implements する。
  - Adaptee を extends する。
  - Target のインタフェース API を実装する。また、このインタフェース API の実装は Adaptee の具象メソッドを利用する。

3 AHP(階層分析法)

AHP は、1971 年に Thomas L. Saaty により提唱された不確定な状況や多用な標準元準における意思決定手法で、問題分析において主観的判断とシステムアプローチをうまくミックスした問題解決型意思決定手法である。AHP の特徴としては以下のようなものがある。

- 意思決定者の主観的判断を意思決定法に取り入れることによって、勘や経験などを生かした意思決定が可能となる。
- 評価元準となる要素が複数があり、互いに共通の尺度がない問題に対して有効である。
- ペア比較値を利用することにより、ペア比較をする際の意思決定者の負担を軽減できる。
- 人間が思考過程において少しずつ問題を解決していくことと同様のアプローチが、問題を階層化することによって得られる。

3.1 代替案導出までの手順

AHP による代替案の導出までの手順は以下の通りである。

1. **問題に対する要素を抽出し、階層構造に分解する**
   例えば、「新車の選定」という問題の代替案を AHP により導き出す時、図 3 のような階層構造になる。
2. **各レベル (階層) の要素間のペア比較を行う**
   各レベル毎に以下の表 1 に元づいてペア比較を行う。ペア比較とは、行の項目と列の項目の重要度を意思決定者が主観で比較することである。

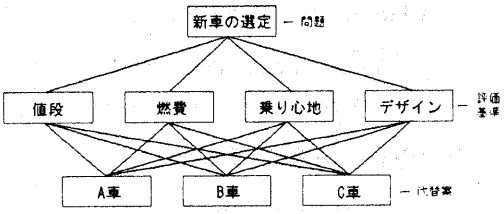

図 3: 「新車の選定」の階層的構造

ペア比較値	意味
1	両方の項目が同じぐらい重要
3	行の項目の方が列よりやや重要
5	行の項目の方が列の方より重要
7	行の項目の方が列ろの方よりかなり重要
9	行の項目の方が列ろの方より絶対的に重要
2, 4, 6, 8	補間的に用いる
上の数値の逆数	列の項目から行の項目を見た場合に用いる

表 1: ペア比較値

例えば、図 3 の問題の場合、「値段安さ」と「燃費」に重点に置くと表 2 のようになる。

	値段	燃費	乗り心地	デザイン
値段	1	3	5	7
燃費	1/3	1	5	7
乗り心地	1/5	1/5	1	3
デザイン	1/7	1/7	1/3	1

表 2: 「値段安さ」と「燃費」に重点を置いたペア比較表

表 2 を行列で表すと式 (1) のように表記される。

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 1/3 & 1 & 5 & 7 \\ 1/5 & 1/5 & 1 & 3 \\ 1/7 & 1/7 & 1/3 & 1 \end{pmatrix} \tag{1}$$

3. **各レベルの要素間のウエイトを計算する**

要素の個数を n, 各要素のウエイトを $w_1, w_2, \cdots, w_n$ とすると式 (2) のように表される。

$$A \cdot w = n \cdot w \tag{2}$$

また、式 (2) は固有値問題

$$(A - n \cdot I) \cdot w = 0 \tag{3}$$

に変形できる。この時、A の最大固有値 λ_{max} と固有ベクトル $v = (v_1, v_2, \cdots, v_n)$ を求めることによって、ウエイトベクトル $w = (w_1, w_2, \cdots, w_n)$ は式 (4) のように与えられる。

$$w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} v_1/\lambda_{max} \\ v_2/\lambda_{max} \\ \vdots \\ v_n/\lambda_{max} \end{pmatrix} \tag{4}$$

例えば、式 (1) のペア比較行列 A の w を求めると式 (5) のようになる。

$$w = \begin{pmatrix} 0.544 \\ 0.311 \\ 0.098 \\ 0.048 \end{pmatrix} \tag{5}$$

4. **$C.I.$ の計算を行う**

意思決定者がペア比較において首尾一貫である A を決定することは大変難しい。そこで以下のような $C.I.$(consistency index) を AHP では定義している。$C.I.$ は式 (6) で与えられる。

$$C.I. = \frac{\lambda'_{max} - n}{n - 1} \tag{6}$$

例えば、式 (5) の $C.I.$ は式 (14) のようになる。

$$C.I. = \frac{4.228 - 4}{4 - 1} = 0.096 \tag{7}$$

5. **階層全体のウエイトの計算し、代替案を決定する**

まず、評価元準のそれぞれ要素に対する各代替案のペア比較を行う。そして、各レベルで求めたウエイトベクトルを用いて、代替案を決定する。例えば、図 3 の問題を解決するためのそれぞれの要素 (値段、燃費、乗り心地、デザイン) に対する各代替案のペア比較表はそれぞれ表 3, 表 4, 表 5, 表 6 のようになる。

表 3, 表 4, 表 5, 表 6 のペア比較表のペア比較行列 A のウエイトベクトル w, $C.I.$ は、それぞれ式 (8), 式 (9), 式 (10), 式 (11) のようになる。

	A 車	B 車	C 車
A 車	1	2	3
B 車	1/2	1	2
C 車	1/3	1/2	1

表 3: 「値段」に関する各代替案のペア比較表

	A 車	B 車	C 車
A 車	1	1/5	1/2
B 車	5	1	7
C 車	2	1/7	1

表 4: 「燃費」に関する各代替案のペア比較表

$$w = \begin{pmatrix} 0.540 \\ 0.297 \\ 0.163 \end{pmatrix}, \quad C.I. = 0.046 \qquad (8)$$

$$w = \begin{pmatrix} 0.106 \\ 0.744 \\ 0.150 \end{pmatrix}, \quad C.I. = 0.059 \qquad (9)$$

$$w = \begin{pmatrix} 0.540 \\ 0.163 \\ 0.297 \end{pmatrix}, \quad C.I. = 0.005 \qquad (10)$$

$$w = \begin{pmatrix} 0.200 \\ 0.400 \\ 0.400 \end{pmatrix}, \quad C.I. = 0.000 \qquad (11)$$

図 3 の問題を解決するためのそれぞれの要素 (値段、燃費、乗り心地、デザイン) に対する各代替案のペア比較を行ったウエイトベクトルを一つの表にまとめると表 7 のようになる。

表 7 の値に値段、燃費、乗り心地、デザインのウエイトを列方向に乗算する。さらに行方向にそれらの値を加算すると表 8 のようになる。表より、B 車、A 車、C 車の順に好ましいことがわかる。

	A 車	B 車	C 車
A 車	1	3	2
B 車	1/3	1	1/2
C 車	1/2	2	1

表 5: 「乗り心地」に関する各代替案のペア比較表

	A 車	B 車	C 車
A 車	1	1/2	1/2
B 車	2	1	1
C 車	2	1	1

表 6: 「デザイン」に関する各代替案のペア比較表

3.2 不完全ペア比較行列

AHP によって与えられた問題の代替案を導出する際に、ペア比較表のすべてマスを埋めることが出来ない場合がある。つまり、行の項目と列の項目を比較できない場合である。例えば、表 2 の不完全ペア比較表は表 9 のようなものがある。

表 9 のような不完全ペア比較表からでも固有値問題が適用できる。不完全ペア比較行列の固有値問題は式 (12) のように表現できる。

$$\begin{pmatrix} 3 & 0 & 5 & 0 \\ 0 & 2 & 5 & 7 \\ 1/5 & 1/5 & 2 & 0 \\ 0 & 1/7 & 0 & 3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \lambda \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} \qquad (12)$$

式 (12) によって示される固有値問題を解くことによって、ウエイトベクトル w, 最大固有値 λ_{max} は式 (13) のように与えられる。

$$w = \begin{pmatrix} 0.427 \\ 0.427 \\ 0.085 \\ 0.061 \end{pmatrix}, \quad \lambda_{max} = 4.000 \qquad (13)$$

	値段 (0.544)	燃費 (0.311)	乗り心地 (0.098)	デザイン (0.048)
A 車	0.540	0.106	0.540	0.200
B 車	0.297	0.744	0.163	0.400
C 車	0.163	0.150	0.297	0.400

表 7: 集計表

	値段	燃費	乗り心地	デザイン	評価
A 車	0.294	0.033	0.053	0.010	**0.390**
B 車	0.162	0.231	0.016	0.019	**0.428**
C 車	0.089	0.047	0.030	0.019	**0.185**

表 8: 代替案の決定

4 本システム開発の概要

4.1 XMI Toolkit

本システム開発では、IBM の alphaWorks によってリリースされている XMI Toolkit version 1.15 により得られた XMI フォーマットを利用してデザインパターンの検索を行う。なお、XMI Toolkit は下記のアドレスよりダウンロードが可能である。

http://alphaworks.ibm.com/tech/xmitoolkit/

XMI Toolkit とは、Rational Software 社の UML ツール「Rational Rose」のモデル情報と Java によって実装されたソースをお互いに変換可能にするツールである。XMI Toolkit による相互変換の流れを表すと図 4 のようになる。

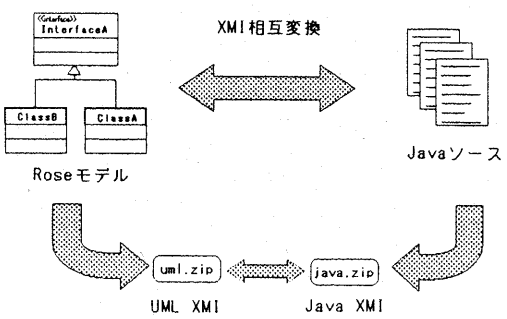

図 4: XMI Toolkit による Rose モデルと Java ソースの相互変換

	値段	燃費	乗り心地	デザイン
値段	1		5	
燃費		1	5	7
乗り心地	1/5	1/5	1	
デザイン		1/7		1

表 9: 表 2 の不完全ペア比較表

4.2 XMI

XMI(XML Metadata Interchange) とは、UML モデルを異なるツール間で相互運用するための統一フォーマット仕様である。1998 年 6 月に XMI の仕様が発表され、その後、1999 年 1 月には正式な OMG 標準としてリリースされた。

例えば、以下のような Java のソースがある時、そのクラス図は図 5 のように表記され、XMI は次のページのように記述される。なお、XMI の記述はここでは簡略化して紹介している。

Member クラスのソース

```
package jp.ac.kyutech.ces.flab;

public class Member{

  private String name = "syo";

  public void work(){
    System.out.println(this.name + "が仕事をしています。");
  }
}
```

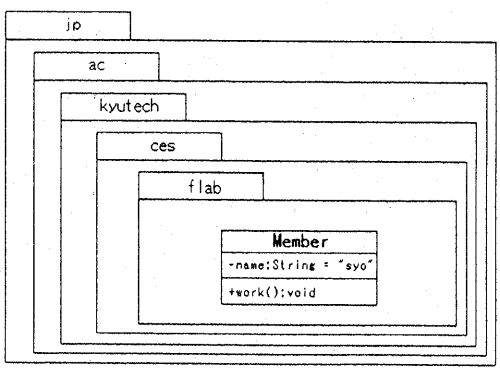

図 5: Member クラスのクラス図

Member クラスの XMI フォーマット

```
<?xml version="1.0" encoding="UTF-8" ?>
<XMI.content>
  <Class xmi.id="_1">
    <Element.name>
      jp.ac.kyutech.ces.flab.Member
    </Element.name>
    <Class.isPublic xmi.value="true" />
    <Class.isInterface xmi.value="false" />
    <Attribute xmi.id="_1.4">
     <Element.name>name</Element.name>
     <Attribute.visibility xmi.value="private"/>
      <Attribute.type>
      <Class xmi.uuid="String"href="String.xml"/>
      </Attribute.type>
      <Element.contains>
        <Source>
          <Element.name>
            field-init
          </Element.name>
          <Source.Code>"syo"</Source.Code>
        </Source>
      </Element.contains>
    </Attribute>
    <Method xmi.id="_1.5">
      <Element.name>work</Element.name>
      <Method.visibility xmi.value="public" />
      <Method.returnType>
        <Class xmi.uuid="void" href="void.xml" />
      </Method.returnType>
      <Element.contains>
       <Source>
          <Element.name>
            method-body
          </Element.name>
          <Source.Code>
            {System.out.println(this.name + "が
仕事をしています。"); }
          </Source.Code>
        </Source>
      </Element.contains>
    </Method>
  </Class>
</XMI.content>
```

4.3 デザインパターンの検索結果

本システムでは発見されたデザインパターンのクラス間の情報を XML フォーマットで出力している。例えば、AbstractFactory パターンを検索した場合、以下のような XML フォーマットのファイルが出力される。

検索された情報を表す XML フォーマット

```
<?xml version='1.0' encoding='Shift_JIS'?>
<designPatterns>
  <abstractFactoryPattern>
     <abstractFactory>
       <class-name>AbstractFactory</class-name>
     </abstractFactory>
     <abstractProduct>
       <class-name>AbstractProductA</class-name>
       <class-name>AbstractProductB</class-name>
       <class-name>AbstractProductC</class-name>
     </abstractProduct>
     <concreteFactory>
        <class-name>ConcreteFactory</class-name>
     </concreteFactory>
     <concreteProduct>
        <class-name>ConcreteProductA</class-name>
        <class-name>ConcreteProductB</class-name>
        <class-name>ConcreteProductC</class-name>
     </concreteProduct>
  </abstractFactoryPattern>
</designPatterns>
```

4.4 AHP の適用

本システムでは発見されたパターンのクラス間の情報を元に、モジュール毎にパターンが使われていた回数を以下のような XML フォーマットで管理している。例えば、モジュール 1 に対するパターンの回数は以下のような XML フォーマットで記述される。

パターンの使用回数を表す XML フォーマット

```
<?xml version='1.0' encoding='Shift_JIS'?>
<designPatternsVolume>
    <module>
        <module-Name>モジュール 1</module-Name>
        <iterator>30</iterator>
        <adapter1>100</adapter1>
        <adapter2>20</adapter2>
        <templateMethod>20</templateMethod>
        <factoryMethod>40</factoryMethod>
        <singleton>60</singleton>
        <prototype>70</prototype>
        <builder>30</builder>
        <abstractFactory>10</abstractFactory>
        <bridge>40</bridge>
        <strategy>30</strategy>
        <composite>20</composite>
        <decorator>30</decorator>
        <visitor>50</visitor>
        <chainofResponsibility>20
        </chainofResponsibility>
        <facade>30</facade>
        <mediator>50</mediator>
        <observer>30</observer>
        <memento>20</memento>
        <state>50</state>
        <flyweight>60</flyweight>
        <proxy>60</proxy>
        <command>20</command>
        <interpreter>10</interpreter>
    </module>
</designPatternsVolume>
```

	偏差値		偏差値
Iterator	49.273	Decorator	49.273
Adapter(継承)	56.056	Visitor	51.211
Adapter(委譲)	48.304	Chainof Responsibility	48.304
TemplateMethod	48.304	Facade	49.273
FactoryMethod	50.242	Mediator	51.211
Singleton	52.180	Observer	49.273
Prototype	53.149	Memento	48.304
Builder	49.273	State	51.211
AbstractFactory	47.335	Flyweight	52.180
Bridge	50.242	Proxy	52.180
Strategy	49.273	Command	48.304
Composite	48.304	Interpreter	47.335

表 10: モジュール 1 におけるパターンの偏差値

表 10 の数値を元に、モジュールに対するパターン同士のペア比較行列 A を作成し、ウエイトベクトル w を導出する。

また、モジュール同士のペア比較行列 A については、意思決定者が図 6 に示される本システムの GUI を利用してペア比較行列 A を作成する。スライダーを用いることによって、意思決定者が数字をあまり気にせずにペア比較を行うことが出来る。なお、スライダーを動かすと、ペアの成分のスライダーも連動して動くようになっている。また、ペア比較を行えない場合は「比較不可」のラジオボタンを押すことによって不完全ペア比較行列を得る。

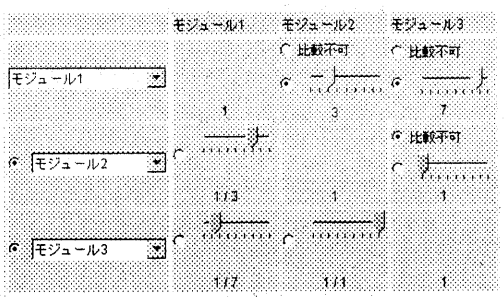

図 6: ペア比較行列を決定する本システムの GUI

図 6 の入力の場合、式 (14) のような不完全ペア比較行列を得る。

$$A = \begin{pmatrix} 1 & 3 & 7 \\ 1/3 & 2 & 0 \\ 1/7 & 0 & 2 \end{pmatrix} \tag{14}$$

階層全体のウエイトを計算した後に、代替案として選択されたデザインパターンを GUI に出力する。

5 まとめ

パターンの利用回数のデータ用いて、経験の少ないプログラマやパターンを知らないプログラマでも、直面しているユースケースの設計に最適なパターンを導出する方法を提案した。そのために、Java アプリケーションにおいて、GoF 本で論じられているデザインパターンを検索することのできるシステムを開発した。

参考文献

[1] Hans-Erik Eriksson, Magnus Penker 著: 杉本 宣男, 落合 修, 武田 多美子 訳: "UML ガイドブック", トッパン, 1999.

[2] Mark Grand 著: 宮本 道夫, 瀬尾 明志 訳: "UML を使った Java デザインパターン-再利用可能なプログラミング設計集-", カットシステム, 2000.

[3] 木下 栄蔵 著: "わかりやすい意思決定論入門", 近代科学社, 1996.

[4] 丸山 宏, 田村 健人, 浦本 直彦 著: "XML と Java による Web アプリケーション開発", ピアソン・エデュケーション, 2000.