6ZB-04

エージェントを用いたパターン指向リファクタリングツールの考案*

山腰 哲　　青柳 哲　　木村 耕
電気通信大学 情報工学科

1 はじめに

デザインパターンは近年，ソフトウェアの再利用性を向上させるアプローチとして注目を集め，CASEツールによるデザインパターン適用支援が要求されている [1].

一方，人工知能の分野では，知的で柔軟なシステム構築を可能にする技術として，エージェントが期待されている.

本研究では，エージェントを用いて既存のコードの解析およびアンチパターン [2] の検出を行い，デザインパターンを適用するリファクタリングツールを考案する．これにより，エージェントによる問題解決に適したデザインパターンの選択支援の一手法を提案する.

2 方針

ツールの構築にあたり，以下の点について留意する.

- Gammaのデザインパターン [3] を利用する.
- 問題点の検出に開発のアンチパターン [2] を利用する.
- 入力データは既存のソースコードとし，本研究ではJava言語で記述されているものとする.
- 共生·寄生エージェントモデル [4] を採用し，エージェントの状態遷移はFIPA† 98に準拠する.

3 ツールの構想

本ツールは図1に示すようなマルチエージェントシステムとして構成される．以下に，各エージェントの機能を示す.

Master Refactor サーバに常駐するエージェント．デザインパターンやアンチパターンを知識として持つ．クライアントの要求に応じ，複製(Slave)を生成しクライアントに派遣する．戻って来たSlave Refactorの結果報告を受け，自らの知識を更新する.

Slave Refactor 実際にクライアントへ移動し，リファクタリングを行うエージェント．Parserを獲得し部品として使用する．Wrapper Agentを介してソースコードを取得し，解析する．判定材料が不足していれば，Interface Agentを通して開発者の意図を抽出する．サーバに戻るとMaster Refactorに結果を報告した後消滅する.

Parser Slave Refactorのコンポーネントエージェント．ソースコードを解析する.

Wrapper Agent ソースコードやカスタマイズ情報を管理する．Slave Refactorの要求に応じ，リポジトリに蓄えられているコードの入出力を行う.

Interface Agent ユーザとの対話を行うエージェント.

3.1 リファクタリング

Refactorは図2に示す手順でリファクタリングを行う．図の実線の部分が本研究で実現する部分である.

3.2 アンチパターンの知識化

Refactorの知識として，アンチパターンの「症状と結果」の項目を利用する．リスト1にアンチパターン肥満児の症状の一つ,「単一のクラスに大量の属性や操作がある」をPrologで表現した例を示す.

```
hasmanyfields(C):-hasfields(C,N),geq(N,60).
hasmanymethods(C):-hasmethods(C,N),geq(N,60).
theblob(C):-hasmanyfields(C),hasmanymethods(C).
```

リスト 1: 肥満児の症状

---

* An idea of Pattern Oriented Refactoring tool using an agent by YAMAKOSHI Akira, AOYAGI Satoshi, KIMURA Koh, Department of Computer Science and Information Mathematics, The University of Electro-Communications

† Foundation for Intelligent Physical Agents, http://www.fipa.org/

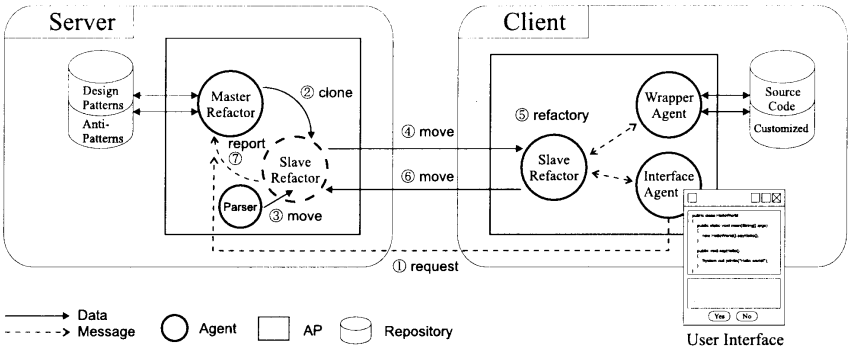

図 1: 本ツールの構成

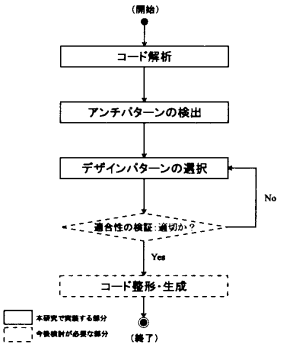

図 2: リファクタリングの流れ

3.3 適用可能性の知識化

アンチパターン肥満児は，オブジェクトの責任分割に失敗し，単一の巨大なクラスが処理を独占する，という設計に見られる．このような設計は GUI クラスによく見られ，Observer デザインパターンを適用することで解決できる．そこで，本研究では適用可能性をリスト 2 のように定義する．

```
guiclass(X):-guiclass(Y),superclass(Y,X).
applicable(observer,X):-theblob(X),guiclass(X).
```

リスト 2: Observer の適用可能性

3.4 デザインパターンの選択手法

知識化された症状と適用可能性を利用して Prolog インタプリタにより推論を行い，デザインパターンの選択する．

この例では，リスト 1，リスト 2 から Observer パターンが適用可能であると導くことができる．なお，式中の `hasfields`，`hasmethods` などの事実は，コードの解析によって与えられる．

4 考察

現状では，適用可能性の記述が直感的である．より適したデザインパターンの選択を可能にするために，デザインパターンの動機や適用可能性を基にした知識化，および目的への適合性の検証が必要である．

5 おわりに

現在，今回紹介した知識を基に推論を行うプロトタイプの実装は完了している．今後，適用可能性の知識化を強化し，適合性の検証やコードの再構築を実現する．

参考文献

[1] 山本純一，松本一教：CASE ツールによるデザインパターン適用支援，情報処理学会研究報告，96-SE-111-6, Vol. 96, No. 84, pp. 41–48 (1996).

[2] Brown, W. J., Malveau, R. C., McCormick III, H. W. S. and Mowbray, T. J.: *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons (1998), 岩谷宏訳，アンチパターンソフトウェア危篤患者の救出，ソフトバンク (1999).

[3] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley (1994), 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，ソフトバンク (1995).

[4] 飯島正，山本喜一，土居範久：拡張可能なエージェントのための共生・寄生モデル，電子情報通信学会技術報告，KBSE97-33, Vol. 97, No. 503, pp. 25–31 (1998).