社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－SE－144　(10)
2004／3／18

デザインパターン利用促進のための モデリング支援ツールの開発

木梨 充高　　小飼 敬　　上田 賀一

茨城大学工学部情報工学科

〒 316-8511 茨城県日立市中成沢町 4-12-1

近年, ソフトウェア開発において繰り返し遭遇する一般的な問題とその解決策をまとめたデザインパターンが注目されている. 本研究では, 要求分析記述を解析し, 適用できそうなデザインパターンを判断/提示するモデリング支援ツールを開発した. 各パターンの説明を特徴付けている単語の組合せによってデザインパターンを判断するルールを準備し, 要求分析記述を解析して得られた単語とマッチングすることによってパターンを判断する. 要求分析はユースケース図/記述を対象とし, それらとクラス図およびシーケンス図を作成する作図環境を用意した. 本支援ツールによって, 分析から設計に移行する段階で優れた設計の採用が容易になると考えられる.

A Modeling Support Tool to Advance Use of Design Patterns

Michitaka KINASHI, Kei KOGAI, and Yoshikazu UEDA

Ibaraki University

4-12-1 Naka-Narusawa, Hitachi, Ibaraki, 316-8511 Japan

Recently, it pays attention to the design pattern that summarizes the general problem to appear often and that solution in software development. In this study, we developed the modeling support tool which judges and presents the design pattern to be applied by analyzing requirement description. We prepare the rule that a design pattern is judged by the combination of the word which characteristics put the explanation of each pattern on. This tool judges the design patterns by matching it with the words obtained by the requirement description analysis. This support will make easy the adoption of the excellent design at a stage to shift from the analysis to the design.

1　はじめに

近年, ソフトウェアの再利用性, 拡張性の向上のために, オブジェクト指向分析/設計技法を用いたオブジェクト指向開発が普及してきている. E.Gammaらは, オブジェクト指向開発において繰り返し遭遇する一般的な問題に対する解決策をデザインパターンとしてまとめあげた. 各パターンには, パターンを適用する目的, 動機, 適用可能性, および具体的な設計例などが文章とダイアグラムによって記述されている. デザインパターンを利用することで, 再利用に優れた設計を効率よく構築することが可能となる.

一方, ソフトウェア開発の初期段階では, 対象システムの要求分析を, システムが提供するサービス(機能)や利用者を決定するためにダイアグラムや自然言語を用いてそれらを定め, 設計方針やソフトウェアアーキテクチャを決定していく. 次の詳細な分析/設計段階において, デザインパターン適用の可否が検討される.

従来の設計ツールによる, デザインパターン利用の支援は, 設計者がデザインパターンを選択し, クラス名などの必要なパラメータを入力することでデザインパターンを適用したクラス図等を作成す

る方式を採っている. パターンを適用すべきか, どのパターンを適用するかといった判断は開発者に委ねられ, 開発者がデザインパターンを熟知していることを前提としており, パターン利用の支援としては不十分である. そこで, 本研究では, 要求分析記述を解析し, 適用できそうなデザインパターンを判断/提示する設計支援ツールを作成した.

2 章でデザインパターンについて説明する. 3 章で本設計支援ツールの概要について述べる. 4 章では, パターンを判断するためのルールを記述したパターン判断規則の概念について, 5 章では, 要求分析記述の解析方法について述べる. 6 章で本設計支援ツールの実行例について, 7 章で関連研究について述べ, 最後に 8 章でまとめと課題を述べる.

2 デザインパターン

オブジェクト指向開発で繰り返し遭遇する一般的な問題とその解決策をまとめて文書化したデザインパターンが E.Gamma らによって考案された [2]. デザインパターンを使用することで, 優れた設計やアーキテクチャの再利用が容易になり, 開発の効率やオブジェクト指向設計の理解度向上に繋がる.

デザインパターンは, クラス図やシーケンス図による図形的表現と, 設計中の決定事項や代替案, トレードオフ, および使用例等を説明する文章で構成される. この一貫した記述形式はパターンテンプレートと呼ばれ, E.Gamma らのパターンテンプレートを構成する項目には,

·パターン名と分類	·目的	·別名
·動機	·適用可能性	·構造
·構成要素	·協調関係	·結果
·実装	·サンプルコード	·使用例
·関連するパターン		

の 13 項目がある [2].

本研究では,“目的”,“動機”,“適用可能性” の 3 つの項目に着目し, 適用できそうなデザインパターンを判断するためのルールを作成した.

3 設計支援ツール

本章では, どのような設計支援を行うのか, また, それを実現するためのツール構成について順に説明する.

3.1 設計支援の方針

デザインパターンの支援機能を持つ従来の設計ツール (Borland 社 TogetherControlCenter[13] など) では, 設計者が使用したいデザインパターンを指定し, そのパターンを構成するクラスの名前などのパラメータを適宜入力することで, クラス図を作成する. すなわち, 設計者がデザインパターンを充分に理解していることを前提とした支援であり, 設計者は対象システムの要求分析記述を参考に適切なデザインパターンを判断する必要がある.

そこで本支援ツールでは, 対象システムの要求分析記述から適用できそうなデザインパターンを判断/提示することで, 設計者の負担の減少および開発効率向上を目指す. 適用できそうなデザインパターンが提示されることで, 一般に理解が困難であるとされるデザインパターンの利用促進も期待できる.

3.2 ツールの構成

本支援ツールは, UML が事実上の標準に成りつつある現状を踏まえ, UML におけるユースケース図, ユースケース記述を用いた要求分析, クラス図を用いた設計を行うソフトウェア開発を想定している.

要求分析記述 (ユースケース記述) には設計方針を決定するための情報が含まれていると考えられる. まず, 要求分析記述中の自然言語を形態素解析/構文解析し, 単語および単語同士の関係を取得する. その解析結果と, デザインパターンに精通した者があらかじめ作成しておいたパターン判断規則とのマッチングを行い, 適用できそうなパターンを判断/提示する. 本設計支援ツールの構成図を図 1 に示す.

4 パターン判断規則

本研究では, 適用できそうなデザインパターンを判断するためのルールを定義し, パターン判断規則としてまとめあげた. パターン判断規則は何をもと

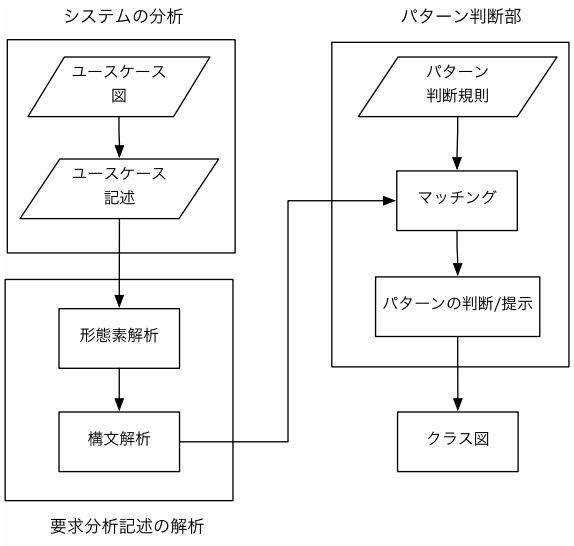

図 1: 本ツールのシステム構成図

・目的
あるクラスに対してインスタンスが 1 つしか存在しないことを保証し, それにアクセスするためのグローバルな方法を提供する.

・動機
システムには多数のプリンタは接続できるが, プリンタスプーラは 1 つ存在すれば良い. ファイルシステムやウィンドウマネージャも 1 つでなければならない. このようなときに, あるクラスのインスタンスは 1 つしか作成できないようにし, そのインスタンスを取得する方法を提供することはできないだろうか？クラス自体がその唯一のインスタンスを管理する責任を持つことで解決できる.

・適用可能性
クラスに対してインスタンスが 1 つしか存在してはならず, また, クライアントがそのインスタンスを取得する方法を提供しなければならないとき

図 2: Singleton パターンの目的, 動機, 適用可能性

に作成するのか, どのように記述するのかを順に説明する.

4.1 パターン判断規則の抽出

デザインパターンを一貫した記述で定義するパターンテンプレートの項目 “目的”,“動機”,“適用可能性” に着目した. これらの項目は, そのパターンを適用すべきかどうか判断するために使用されるため, 各パターンに特有な特徴を示す単語が含まれている可能性が高い. Singleton パターンにおける例を図 2 に示す.

Singleton の適用可能性を示す特徴的な単語としては,「インスタンス」「1 つだけ」「作成する」「存在する」などが挙げられる. Singleton は比較的簡単なパターンであるため, 目的, 動機, 適用可能性が似通った内容になっている. また, 動機は我々が遭遇しそうな具体的な問題が挙げられることが多いが, 上の例で登場した「プリンタスプーラ」「ファイルシステム」は有意義な例であるとは思えない. そこで, 有意義な具体例が示されている例として Mediator パターンの動機を図 3 に示す.

図 3 の例には, ダイアログなどのユーザインタフェース部分において利用することで, 再利用性のある GUI 部品が作成できることが示されている. このように, パターンテンプレートの動機からはパターンが適用されうる場面の具体例から単語を抽出できる. Mediator の例では, Mediator パターンの適用可能性を示す単語として「GUI」「部品」「連携」などが挙げられる.

4.2 記述形式

前章では, パターンテンプレートの項目 “目的”,“動機”,“適用可能性” からそのパターン特有の単語を抽出した. その単語を組み合わせることでパターン判断規則を記述する.「オブジェクト」「1 つだけ」「作成する」から Singleton パターンを提示するパターン判断規則は以下のように記述する.

$$(\text{オブジェクト}), (1\text{つだけ}), (\text{作成する}) \Rightarrow Singleton$$

「オブジェクト」の代わりに「インスタンス」,「1 つだけ」の代わりに「唯一」のように類似した意味を持つ異なる単語によって, 同様の意味を持つパターン判断規則が考えられる. これらの単語の組合せの数だけパターン判断規則を記述するのは冗長なので, 以下のように記述できるようにした.

$$(\text{オブジェクト}, \text{インスタンス}), (1\text{つだけ}, 1\text{個だけ}, \text{唯一}), (\text{作成する}, \text{生成する}) \Rightarrow Singleton$$

1 つの括弧は代替可能な単語の集合を表し, その中に含まれる単語は論理和として解釈され, 各集合同士は論理積として解釈される. 上の例では,「オブジェクト」または「インスタンス」という単語が

・動機
1つのシステムを多くのオブジェクトに分割することは, 一般に再利用性を高めるが, オブジェクト間の関連を増やすことがせっかく高めた再利用性を再び低下させる.
すなわち, 多くのオブジェクトがお互いに関連を持つことで, あるオブジェクトは他のオブジェクトがないと働かない, すなわち, システムがあたかも一枚岩のように振舞うということが起こり得る.
GUI(Graphical User Interface) において, ボタンや入力フィールドが多く表示され, ある入力フィールドに文字列が入力された時に, あるボタンが押せるようになる, といった連携をしなければならない場合がある. 各オブジェクトが他のオブジェクトの状態を調べなければならない. このボタンはほかのダイアログで使うために単独で移植することができない.
そこで, 全てのオブジェクトのメッセージを受け取り, 調停を担当するオブジェクトを用意することで各オブジェクト同士の依存度を削減することができる.

図 3: Mediator パターンの動機

存在し, かつ「1つだけ」または「1個だけ」または「唯一」という単語が存在し, かつ「作成する」または「生成する」が存在するときに Singleton パターンを提示する, という意味になる.

パターン判断規則に記述する単語の品詞は, 名詞, 動詞, および一部の助詞に限定する. 助詞については,「オブジェクトが」のような文節中の「が」や「を」といった助詞は意味がないので記述しないことにする.「1つだけ」の「だけ」も助詞ではあるが他の単語と結び付くことで意味を持つので, パターン判断規則中に記述してもよい. これらの助詞は区別可能である.

また, パターン判断規則において, 動詞は終止形で登録する必要がある. 要求分析記述において動詞は様々な形に活用されており, これをパターン判断記述で列挙するのは不可能である. そこで, 自然言語の解析後, 終止形でない動詞は終止形に変換している. 例えば, 要求分析記述を解析して得られた動詞「保存しておく」はそれの終止形「保存する」に変換される.

4.3 アクタの取り扱い

ユースケース図の構成要素であり, ユースケース記述の項目の1つに挙げられるアクタがある. アクタはシステムのサービスの利用者, もしくはサービスを利用する外部のシステムであり, アクタ名は一般に名詞である (名詞抽出法). ユースケース記述中の文章にはアクタが名詞として登場する. あるユースケースに「自動車工場」というアクタが関わっているとき, そのユースケース記述において「自動車工場がタイヤを作成する」と記述されるかもしれない. しかし,「自動車工場」という単語は分析/設計を行うシステム特有の単語であるためパターン判断規則で用意することは困難であり, また用意するべきでもない. そこで「#アクタ」という単語を用いて以下のようにパターン判断規則を記述できるようにした.

$$(\#\text{アクタ}), (\text{作成する}) \Rightarrow FactoryMethod$$

パターン判断規則中の単語に「#アクタ」が現れた場合, 解析中のユースケース記述の項目 “アクタ” に記述されているアクタをそこに展開する (置き換える). 今回の例では, パターン判断規則中の「#アクタ」が「自動車工場」に展開され, Factory Method パターンを提示することになる.

本研究において, これまでに作成したパターン判断規則の一部を図 9 に示す.

5 要求分析記述の解析

本設計支援ツールでは, ユースケース記述を使用した要求分析記述を想定しているが, ユースケース記述の記述形式が厳格に定められているわけではないので, 本研究におけるユースケース記述の項目を定義する必要がある. その項目を以下に示す.

- ユースケース名
- アクタ
  主アクタ, サブアクタを区別せずに, 当該ユースケースと関係のある全てのアクタを列挙する
- 事前/事後条件
- 基本/代替系列

パターン判断規則において, 主アクタ, サブアクタは区別されないので, 当該ユースケースと関係のある全てのアクタを列挙する項目としてまとめた. また, 各項目を解析する方法は同じであるため, ここでの項目としてはそれぞれをひとつにまとめた.

5.1 CaboChaによる構文解析

ユースケース記述中の日本語を対象とした自然言語解析が必要である. 自然言語解析は形態素解析, 構文解析の順で行われる. 構文解析ツールの 1つに, 奈良先端科学技術大学松本研究室において開発された CaboCha[6] がある. 日本語の構文解析ツールとしては最も高い精度 (89.29%) を実現し, C/C++等から利用できる CaboCha ライブラリが用意されている. また, CaboCha は 1 文ずつ解析を行うという制約がある. しかし, あるパターンを判断するための単語が異なる文に存在している場合, 各単語は全く別の意図で使われている可能性があるため, その制約が問題になるとは考えにくい. 以上の理由から, 本支援ツールにおける要求分析記述の解析は CaboCha によって行うことにした.

5.2 解析結果の変換

パターン判断規則を構成する単語は名詞, 動詞 (終止形) 等を基準にしているので, 解析結果を変換する必要がある. 本研究では次の変換を行っている.

- 助詞の削除
  (例)「書類が」⇒「書類」
- 接尾用の名詞は直前の名詞と連結する
  (例)「図書館」「員」⇒「図書館員」
- 動詞は終止形に変換し, 直前の単語が名詞のとき連結する
  (例 1)「記録」,「する」⇒「記録する」
  (例 2)「覚えておく」⇒「覚える」

5.3 単語のマッチング

要求分析記述中の文章「商品の予約を受けたとき, 予約券を渡し, 商品の販売は後で行う」を解析することで, 単語「商品」「予約」「受ける」「予約券」「渡す」「販売」「後で」「行う」が取得される. これらの単語群は以下のパターン判断規則を満たす.

(代わりに, 代理で, 後で), (作成する, 生成する, 行う)

$$\Rightarrow Proxy$$

最初に, パターン判断規則の 1 番目の単語集合 (「代わりに」「代理で」「後で」) 中の「後で」が解析結果に含まれているため, この集合は真となる. 次に 2 番目の単語集合 (「作成する」「生成する」「行う」) 中の「行う」が解析結果に含まれているので, この集合も真となる. パターン判断規則中の全ての単語集合が真であるとき, そのパターンが適用されうると判断する.

6 実行例

本設計支援ツールは, ユースケース図, クラス図, シーケンス図の作図環境を備えている. ユースケース記述は, ユースケース図中のユースケースをダブルクリックすることで表示される, ユースケース記述入力ダイアログを使用して記述する. ユースケース図およびユースケース記述を本設計支援ツールで作成した画面をそれぞれ図 4, 図 5 に示す.

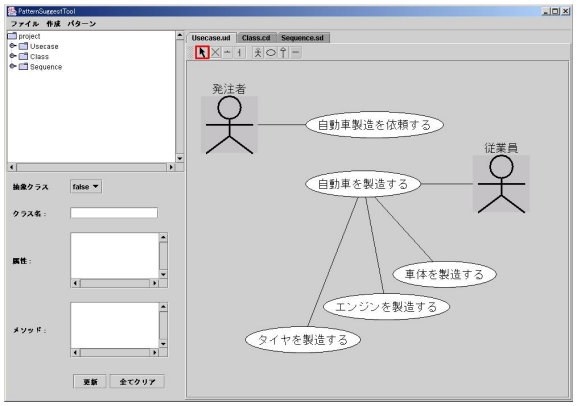

図 4: 本支援ツールで作成したユースケース図

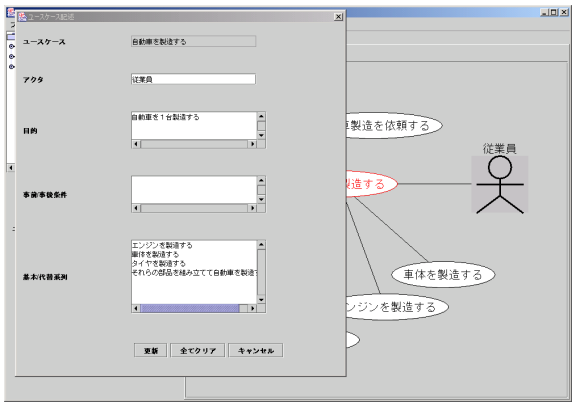

図 5: 本支援ツールで作成したユースケース記述

適用できそうなパターンの候補とその説明ダイアログを提示している画面を図 6 に示す. 左側のダイアログで適用可能性のあるパターンとして Singleton パターンと Factory Method パターンが提示されている (パターン判断規則の適合数も表示され, 適合数が 0 のパターンに対応するボタンは表示が反転していて押すことができない). 右側のダイアログは, Factory Method パターンに対応するボタンをクリックすることで表示された, パターンを説明するヘルプ画面である. ヘルプ画面はパターンの説明を記述した HTML, クラス図, シーケンス図の 3 つの画面がタブで切替え可能になっている. ユーザはヘルプ画面を参考に, 提示されたパターンを適用すべきかどうか検討する. パターンを適用する場合はクラスを作成する作業に移る.

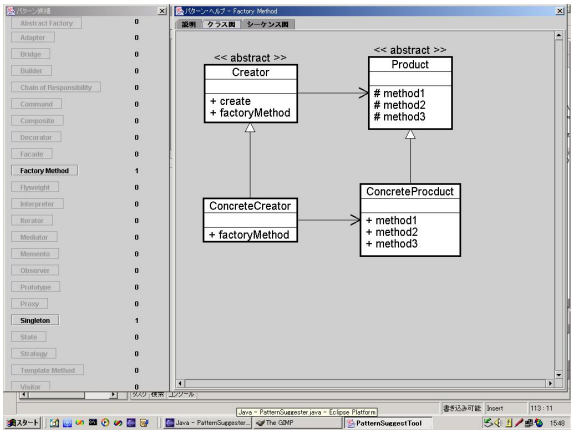

図 6: 本支援ツールがパターンを提示している画面

今回の例におけるパターンの判断処理は以下のようになる.

- ユースケース「自動車製造を依頼する」(図 7) 中の事前条件「依頼先の工場は 1 つだけ存在すれば良い」を解析/変換することで, 単語「依頼先」「工場」「1 つだけ」「存在する」「良い」が抽出され, これはパターン判断規則「(1 つだけ), (存在する) $\Rightarrow$ *Singleton*」を満たす.
- ユースケース「自動車を製造する」(図 8) 中の基本系列の 1 文「それらの部品を組み立てて自動車を製造する」を解析/変換することで, 単語「それら」「部品」「組み立てる」「自動車」「製造する」が抽出され, これはパターン判断規則「(部品), (組み立てる) $\Rightarrow$ *FactoryMethod*」を満たす.

ユースケース	自動車製造を依頼する
アクタ	発注者
目的	自動車製造を工場に依頼する
事前条件	依頼先の工場は 1 つだけ存在すれば良い
基本系列	自動車製造を自動車工場に依頼する

図 7: 「自動車製造を依頼する」ユースケース記述

ユースケース	自動車を製造する
アクタ	従業員
目的	自動車を 1 台製造する
事前条件	
基本系列	1. エンジンを製造する 2. 車体を製造する 3. タイヤを製造する 4. それらの部品を組み立てて自動車を製造する

図 8: 「自動車を製造する」ユースケース記述

7 関連研究

デザインパターン適用支援に関する研究としては, 山本らの研究 [9], および永山らの研究 [10] がある. 山本らの研究では, 登録済みのデザインパターンのクラス図と設計者が作成したクラス図との対応関係を入力していくことによってパターン適用を支援している. Borland 社の開発支援ツール TogetherControlCenter[13] におけるデザインパターン適用支援も同様の手法である. また, 永山らの研究では, クラス名の類似度に着目して適用できそうなデザインパターン判断している.

いずれの研究でも, 各パターンを説明する文章および要求分析記述に着目した支援は行われていない. 本研究では, 要求分析記述を解析することで適用できそうなパターンを判断した. 要求分析記述に設計方針を関わる単語が含まれていることは十分に考えられるため, パターン判断規則の作成が容易な本研究の方式は有意義であると考えられる.

8 まとめと今後の課題

本研究では, 要求分析記述の自然言語処理とパターン判断規則中の単語のマッチングによるパターン判断を行なったが, 要求分析記述の解析における問題として, 動詞を終止形に変形することによる意味の取り違えが挙げられる. 例えば, 「作成した書類」は本来は名詞であるが, 解析後には「作成する」「書類」に分割されるため, 意図しないデザインパターンが提示されることがある. このような問題は自然言語が持つ意味を解釈することの困難さから発生するものであり, 本研究では取り扱わなかったが, 考慮する必要がある.

今後の課題としては, パターン判断規則の充実および洗練が挙げられる. パターン判断規則中に「オブジェクト」「インスタンス」といった, 実装に近い単語が登場する. 詳細な要求分析記述であっても, そのような単語が現れることはあまり期待できないので, 実装段階に偏りすぎないパターン判断規則を作成する必要がある. 他には, 包含関係にあるパターン判断規則を登録できないようにすることや, 作図環境の UML への対応を進めることなどが挙げられる.

また, 要求分析記述を構文解析しているが, 単語間の関係を考慮したマッチングは行っていないので考慮する必要がある. 逆に, 形態素解析のみで充分であるということもありえる. その場合には,CaboCha が内部で呼び出している形態素解析ツール ChaSen[7] と同等の解析精度でありながら 3,4 倍の解析速度を誇る MeCab[8] の採用を検討する.

参考文献

[1] 結城 浩,“Java 言語で学ぶデザインパターン入門”

[2] Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides, 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・パブリッシング株式会社,1995

[3] Perdita Stevens,Rob Pooley, 児玉 公信監訳,“オブジェクト指向とコンポーネントによるソフトウェア工学”, 株式会社ピアソン・エデュケーション,2000

[4] 河合照男,“UML がわかる” 株式会社技術評論社,2002

[5] 青山幹雄, 中谷多哉子,“オブジェクト指向に強くなる” 株式会社技術評論社,2003

[6] URL:http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/,“奈良先端科学技術大学松本研究室 CaboCha”

[7] URL:http://chasen.aist-nara.ac.jp/,“奈 良 先端科学技術大学松本研究室 ChaSen”

[8] URL:http://cl.aist-nara.ac.jp/~taku-ku/software/mecab/,“奈良先端科学技術大学松本研究室 MeCab”

[9] 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ソフトウェア工学 No1t11-6,1996

[10] 永山 英嗣, 原田 実,“デザインパターン適用における設計図融合と最適パターン探索の支援系 OOPAS”, オブジェクト指向シンポジウム,2000

[11] 畑口 剛之, 池田 健次郎, 岸 知二,“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 No121-20,1998

[12] 追立 正司, 藤尾 光彦,“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”, 情報処理学会ソフトウェア工学 No136-20,2002

[13] URL:http://www.borland.co.jp/together/controlcenter/index.html,“Borland 社 TogetherControlCenter”

```
(1つだけ, 1個だけ),(存在する) -> Singleton
(互換性, バージョンアップ),(変換する, 適応する, 適用する) -> Adapter
(スーパークラス, 親クラス, 基本クラス, 抽象クラス),(アルゴリズム, 雛型),(作成する) -> Template Method
(#アクタ), (作成する) -> FactoryMethod
(インスタンス, オブジェクト),(生成する, 作成する) -> Factory Method
(部品),(組み立てる) -> Factory Method
(オブジェクト, インスタンス),(コピー, 複製),(作成する, 生成する) -> Prototype
(機能),(実装),(分断する, 分離する, 分割する, 分ける) -> Bridge
(アルゴリズム, 戦略),(選択する, 切り替える) -> Strategy
(入れ子, 再帰),(要素),(同一),(扱う) -> Composite
(入れ子, 再帰),(要素),(同一視する) -> Composite
(ファイル, ディレクトリ) -> Composite
(入れ子, 再帰),(要素),(同一),(扱う) -> Decorator
(入れ子, 再帰),(要素),(同一視する) -> Decorator
(ファイル, ディレクトリ) -> Decorator
(データ, 要素),(処理),(分断する, 分離する, 分割する, 分ける) -> Visitor
(ウィンドウシステム),(メッセージ),(分散する) -> Chain of Responsibility
(処理, 責任),(分散する) -> Chain of Responsibility
(処理),(制御する, 分割する, 分ける, 管理する) -> Facada
(状態, 変化, メッセージ, イベント),(通知する, 伝達する, 伝える, 送信する) -> Mediator
(状態, 変化, メッセージ, イベント),(通知する, 伝達する, 伝える, 送信する) -> Obsever
(アンドゥ) -> Memento
(オブジェクト, インスタンス, 状態),(保存する, 記憶する, 記録する, 復元する) -> Memento
(状態遷移) -> State
(状態),(遷移する, 変化する) -> State
(オブジェクト, インスタンス),(共有する) -> Flyweight
(代理, 後で),(処理する) -> Proxy
(遅延, 遅延処理) -> Proxy
(プロキシ) -> Proxy
(アクセス制限) -> Proxy
(オブジェクト, インスタンス),(代わりに, 代理で, 後で),(作成する, 生成する) -> Proxy
(オブジェクト作成, インスタンス作成, オブジェクト生成, インスタンス生成),
(後で),(行う) -> Proxy
(オブジェクト作成, インスタンス作成, オブジェクト生成, インスタンス生成),
(遅延する) -> Proxy
(オブジェクト, インスタンス),(後で),(作成する, 生成する) -> Proxy
(コマンド, 命令),(保存する, 記憶する, 覚える) -> Command
(描画, 作図),(履歴, 操作),(保存する, 記憶する, 覚える) -> Command
(GUI),(操作, 処理, 操作手順, 処理手順),(保存する, 記憶する, 覚える) -> Command
(イベント),(保存する, 記憶する, 覚える) -> Command
```

図 9: パターン判断規則 (抜粋)