Vol.2010-SE-167 No.24
2010/3/19

振舞と目的による設計パターンの体系化とパターン適用ツールの実現

奥村 和恵[†]　金澤 典子[†]　塚本 享治[†]

簡単に変更できて保守性が高いシステムを作るために、ソフトウェアパターンが使われている。しかし多数のパターンが提案されてはいるが、未整理の状態である。だからどれを使えばよいか判断が難しい。一方でパターンを設計に適用するのも煩わしい。誰もが設計過程でパターンを簡単に選択し、設計図に適用できるツールの開発を目指している。
本稿は約 75 のパターンについて、目的・効果と実装時の処理構造に注目して分類した。そしてパターンをクラス図に適用するツールを作成し、適用実験を行った。

A proposal for classifying software patterns and making a tool fitting patterns to UML diagrams

Kazue Okumura[†] Noriko Kanazawa[†] Michiharu Tsukamoto[†]

The software patterns have been proposed for designing systems better. But there are too many patterns to select. And patterns were not be classified enough. Additionally, an expert designer annoys by applying patterns to diagrams. So we are developing an tool for everyone can select and use the patterns easy.
To this purpose, we analyzed more than 75 patterns, classified these from pattern's objective, and methods structure. Following the analysis, we developed a tool which helps us to fit patterns to UML diagrams.

1. はじめに

システム設計の問題を解決する手段のひとつにソフトウェアパターンがある。オブジェクト指向世界のクラスを効果的に設計する戦略を集めたものである。いわばソースコードのアルゴリズムに対応する技術にあたる。このパターンを設計に適用すれば、システムの保守や変更が簡単になり、コードの再利用性が高まり、かつ品質を一定に保つ効果が期待できる。

しかし分野・種類・粒度の様々なパターンが提案されている。これらを体系立てて紹介する解説本も多く出版されているが、設計者が意図に合うものを選択するのは難しい。処理時間や再利用性などの適用効果について、アルゴリズムのように明確に体系化されていない点も問題である。またクラスや処理などある程度決まっているパターンを、設計図に適用するのも煩わしい。

そこで本稿では設計者がパターンを選ぶのを補助するために、提案されている多くのパターンを目的と振舞に着目して体系化する。そしてこれの利用を促進するために、既存の設計図へパターンを適用する設計支援ツールを開発する。

2. アプローチ

設計者がパターンを選ぶ時は、ある程度パターン名を予想してから解説書を探していることが多い。パターン全体を俯瞰して把握できない点がひとつの原因である。この方法では設計者が多くのパターンの知識を持つ必要がある。だが設計問題に合わせて適切なものを選べれば、幅広い知識がなくても利用できる。そこでパターン全体から目的に合わせて選べる方法を考えた。

そのためにまず下記種類の中から約 75 個のパターンを分析した。

- GoF デザインパターン[1]…アルゴリズムとデータ構造をひとつにまとめた。抽象度が高い汎用設計手法
- PLoPD パターン[2]…汎用設計パターンや、分散・並行化設計、工程・組織化など多様な目的のパターン集。[2]はこの中からマイクロ設計レベルを集めた
- Software Architecture Pattern (POSA)パターン[3]…ソフトウェアアーキテクチャを中心に、設計問題の解決方法を構造からパターン化した
- J2EE パターン[4]…並行処理設計に適す
- EJB パターン[5]…分散コンポーネント用。通信やデータアクセスに関係

次にパターンを選ぶ基準を考えた。設計者は通常、適用できるサブシステムの状況・環境（＝アーキテクチャ）や、適用できる設計の段階、パターンの粒度などを基準に選んでいる。この中で「設計の問題とその解決方法」に沿って選ぶには、パターンの目的と効果が重要になる。そこで研究が進んでいるアーキテクチャではなく、適用目的・効果を分類基準の横軸に選んだ。

一方で多くの先行研究がパターンのクラス構成から類似度を導出している。にもか

[†] 東京工科大学大学院バイオ・情報メディア研究科メディアサイエンス専攻
Tokyo University of Technology, Graduate School of Bionics, Computer and Media Science, Media Science Program

かわらず振舞という処理動作の面から分類した先例は少ない。だが本稿はパターンの類似関係には振舞も関係していると考えた。またパターンの目的・効果と振舞には関係があるかもしれない。このような理由から振舞を分類基準の縦軸にした。

解説本にある 75 パターンを、以上 2 点の基準から分析し分類した。この際にパターン全体を俯瞰して把握するために、種類や言語で隔てず分類し体系化した。その体系化にもとづき、パターンを UML 図に適用するツールを構築する。

3. 先行研究

(1) ソフトウェアパターンの体系化の先行研究

[3]など多くの解説本がパターンをカテゴリ・アーキテクチャ・目的・レイヤー・パターンの粒度などから分類している。しかし本稿はパターンを用いる際に設計問題とその解法から選べるように体系化した。

[6]らはソフトウェアパターンを開文部脈・結果文脈の類似度などから自動体系化した。本稿は人が体系化を行い、パターンの利用状況（＝文脈）よりもその問題の解法に注目した。また[7][8][9]は静的なクラス図の構成要素から分類していた。しかし動的な処理もパターン間の類似度に関係するはずなので、振舞を中心に体系化する。

(2) ソフトウェアパターン適用支援ツールの先行研究

[10][11]らは独自の CASE ツール上で、パターンの適用支援ツールを開発した。CASE ツールとはオブジェクト指向開発を支援するツールである。これらがクラスの類似度により関連付けるのに対し、本稿は振舞の共通点により関連付けた。また作成するツールの汎用性・利便性を高めるために、市販のモデリングツール[12]を利用した。

[13][14][15]らはソースコードのクラスへパターンを追加した。これらの適用範囲はコードを設計する工程に限られる。だが本稿は分析工程～概要設計工程でもパターンを適用でき、かつ実装言語に依存しない。

4. パターンの分類

4.1 振舞による分類

4.1.1 処理の分類

多くのパターン解説本は振舞をシーケンス図に表している。図では多数の処理（＝メッセージ）を時系列に沿って描くことで、パターンが行う処理の相互作用を表している。このメッセージが振舞の最小構成要素である。よって振舞を分類する前に、メッセージをメソッドの種類や操作内容にから分類した。基本的には以下のような種類があることが分かった。

- new …生成する。クラスのインスタンス化を示す操作。
- set, update …値やオブジェクトを設定する。主に setter メソッドを使う。
  - 更新操作(update)には「更新しろ」という命令と、setter メソッドを使って実際に更新するとい 2 つの意味がある。このうち後者は set と同義であるとみなす。
- get …値やオブジェクトを取得する。getter メソッドなどを使う。
- do, execute, call, access, send, accept, invoke …実行する、呼び出す操作。特定のメソッドを実行する操作。
  - do, call, execute…主な引数なしで呼び出し
  - send…引数にメッセージを送ってサービスを呼び出す。
  - accept…メッセージ以外の引数がある場合
  - invoke…非同期の呼び出し
- オリジナルの操作 …以上に述べた以外で、各パターンに特有の操作。例えば以下がある。
  - loopkup …オブジェクトがあるか調査・確認する操作。データベース関係でよく使われる。
  - select, dispatch …サービスや戦略、オブジェクトなどの選択・割振

基本操作の他にもパターン特有の操作が存在している。例えば「メッセージを適合させる」などは、通信に関するパターンに見られる。これらはパターンの共通構造に直接は関係しない。そこで「その他」として処理の分類対象から外し、かつ振舞を分類する際に無視することにした。

4.1.2 共通する構造による分類

シーケンス図を分析した結果、振舞が完全に一致するパターンはほとんど存在しなかった。しかし大きな範囲で共通する、次の 2 種類の構造が見つかった。そこで各パターンがそれぞれの構造に当てはまるか分析してみた。

(1) 複数のメンバーへアクセスする「まとめ役」がある

全体の約 2/3 のパターンが図 1 のように、複数あるメンバーへ代表して接続するオブジェクトを持っていた。本稿はこれを「まとめ役」と呼ぶ。例に挙げた Application Service (J2EE)の他にも Façade (GoF)や Business Delegate (J2EE)などのパターンが当てはまる。

集約されるメンバーは、サービスかデータかという基準で分類できた。さらに、Proxy パターンのように、集約メンバーを特定するかという基準でも分かれた。

また Factory Method (GoF)などの生成に関するパターンは、クライアントが生成クラス（Factory クラス）を呼び出す。すると生成クラスは単体でオブジェクトを生成する。つまり生成クラスは、生成されるオブジェクトへの生成を代表しているとも言える。よって生成クラスを「生成されるオブジェクトのまとめ役」とみなすことにした。

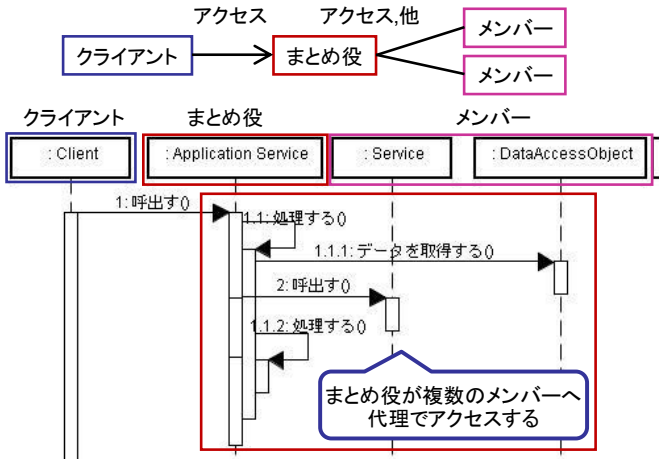

図 1 まとめ役を持つ振舞　例:Application Service パターン

まとめ役を持つパターンの中には、まとめ役とは別に仲介役を持つパターンもあった。そこでこの 2 役が同一か、それとも別かどうかを判断基準にした。さらにこの中でも、まとめ役がメンバーを生成する場合と、単に呼び出すだけの場合の 2 パターンに分けた。

(2) データや振舞をひとつのオブジェクトで保持する「一時オブジェクト」を持つ

Data Transfer Object パターン(J2EE)のように、データを外部に独立させるパターンも多く見つかった。オブジェクトを細分化するのは、情報をカプセル化するためである。変更時に修正する範囲を狭めるという効果が期待できる。この型はオブジェクト指向設計を実現するパターンと判った。

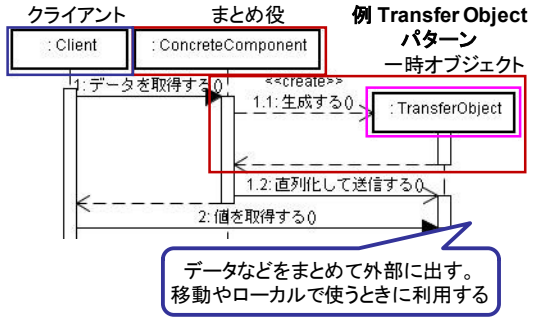

図 2 データを一時オブジェクトに出す振舞　例: Transfer Object パターン

Data Access Object (J2EE)パターンは、クライアントが代理人を介して一時オブジェクトを使う。一方で Data Transfer Object パターンのように、クライアントが直接アクセスする場合もあった。これらから一時オブジェクトのある振舞については、代理人を通すか否かを第 2 の基準にした。

(3) 振舞の共通構造とクラス構成の関係

(1)(2)の共通する振舞は、クラス構成からも同様の基準が推測できることに気づいた。図 3 は図 2 の Transfer Object パターンのクラス図である。クラス図でも振舞と同様の構造がみられた。すなわちクライアントがまとめ役クラスを介して、一時オブジェクトクラスへアクセスしている。

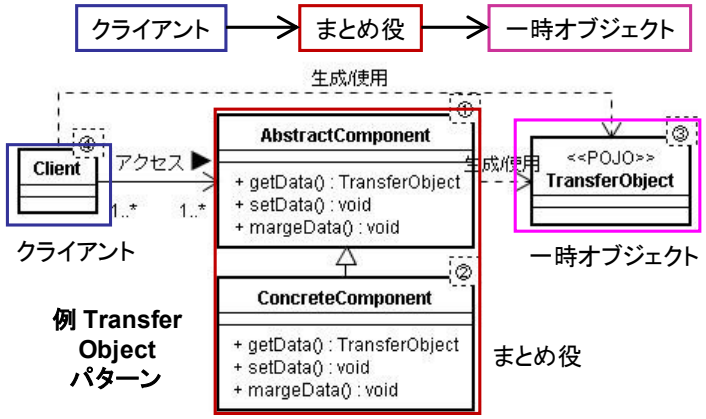

図 3 振舞とクラス構成が関連する　例:Transfer Object パターン

4.1.3 個別なパターンの振舞の構造による分類

前節とは別にパターンの振舞について分類してみた。本節の共通する振舞により分類する点は、前節と同じである。だがより細かいレベルで分析・分類を行った。そしてメッセージの構造が共通するものをまとめた。

メッセージの構造を分析して最初に目についたのが、Chain of Responsibility パターン(GoF)のような特殊な構造であった。図 4（左図）のようにオブジェクトからオブジェクトへ処理（責任）が移動する振舞である。あるいは右図の Visitor パターン(GoF)のように、自分自身のクラスを引数として与え、他のクラスから自分自身のメソッドを起動してもらう構造などがある。このように特徴ある振舞を最初に定義した。

Vol.2010-SE-167 No.24
2010/3/19

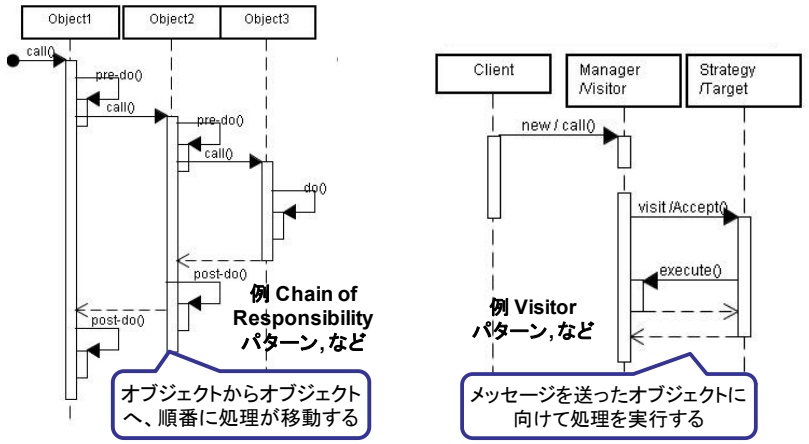

図 4 特殊な振舞の構造 Chain of Responsibility(右)と Visitor(左)

次に複数のパターンに共通して存在する構造を分類した。例えば Proxy パターンは、付録 C の D 図のように Proxy クラスが遠隔実行するサービスを生成（new）して実行する。これは Service Adapter(J2EE)や View Handler(POSA)などのパターンに共通して見られた。Dispatcher（サービスの分配者）クラスも目的や分野は違うが、この Proxy クラスと似た役割をするとも考えられる。

他にも例えば Command パターン(GoF)には、処理を固めて持つ Command クラスがある。この Command クラスは他のパターンでも、同じ様な目的で利用されていた。このように複数のパターン間で共通する振舞の構造を抽出した。すると主に付録 C の 14 種類の型に分類できた。

4.2 目的と効果による分類

言語やアーキテクチャに拘らず目的を分析したところ、種類を超えて目的や効用の似たパターンが多くあることに気づいた。結果、目的は主に付録 A の 3 傾向にまとめられ、より細かい分類基準を設けた。なお複数の目的を持つパターンもあったので、ひとつパターンを複数の目的へ分類した。

さらにパターンを適用する区分を以下の 3 種類に分けて、目的の分類基準に加えた。

①データが対象、②振舞が対象、③データ・振舞の両方が対象

4.3 振舞と目的による体系化

目的（効用）と振舞が関連したパターンを、種類に関わらず把握できることは重要である。そこで前項までの分類結果を総合して、付録 B にまとめた。表中には重複を含んだ約 86 個のパターンがあり、このうち 42 個のパターンへ背景色が付いた。つまり約 50%のパターンは、目的・効果と振舞が関連していることが分かる。

5. パターン適用ツールの実現

5.1 XML 形式によるパターンの定義

クラス図にパターンを追加する時、クラスと関連だけでなく、属性や振舞・多重度なども追加する必要があり面倒である。そこで構成と振舞の両方を示せるクラス図へ、パターンのクラスを追加するツールを作成した。

まずクラスの情報を機械が扱える形式にしなければならない。そこでこれを表現する専用の XML 形式を図 5、図 6 のように定義した。そして約 20 のパターンをこの XML 形式で記述した。そしてツールが適用できるパターンのライブラリを用意した。

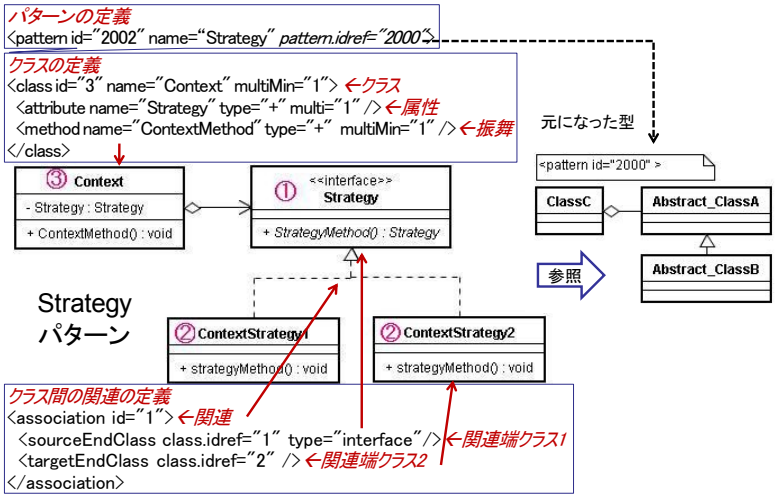

図 5 パターンのクラス図と XML の対応

```
<?xml version="1.0" encoding="Shift_JIS"?> ←XML 宣言
<pattern id="2002" name="Strategy" pattern.idref="2000" > ←パターン
   <classes>←ここからクラスを定義
     <class id="3" name="Context" multiMin="1" pattern.idref="2000" class.idref="3" >←クラス
       <attribute name="Strategy" type="+" multi="1" /> ←属性
       <method name="ContextMethod" type="+" multiMin="1" /> ←振舞
     </class>・・・クラスが続く・・・
   </classes>
```

```
  <associations> ←ここから関連を定義
    <association id="1" pattern.idref="2000" association.idref="1"> ←関連
      <sourceEndClass class.idref="1" type="interface" enable="1" /> ←関連端クラス 1
      <targetEndClass class.idref="2" /> ←関連端クラス 2
    </association>・・・関連が続く・・・
  </associations>
</pattern>
```

図 6 クラスを定義した XML

5.2 パターン適用ツールの試作

利用者の手間を掛けずに、パターンが図に適用できるのが望ましい。そこで図 7 のようなツールを試作した。このツールは UML モデリングツール[12]で作成されたクラス図を操作して、パターンを追加するものである。

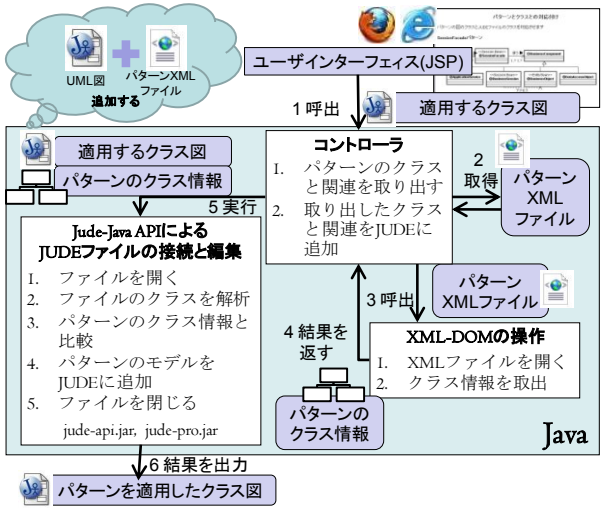

図 7 パターン適用ツールの構成と処理

まず利用者は適用したいパターンを指定する。次に利用者は追加するパターンのクラス図に合わせて、クラス図のクラスに識別番号を追加する。そして追加対象のクラス図があるファイルを指定して、ツールを実行する。すると Java のコントローラが入力情報を得る。そして指定パターンの XML ファイルを、前節で用意したライブラリから読み込む。次に与えられたファイルへ、パターンのクラス・関連などを追加する。この際に[12]を操作する API[16]を利用した。最終的にパターンのクラスが追加されたファイルが出力される。

6. 実験と考察

6.1 パターン適用ツールの実験

パターン適用ツールにより、指定されたパターンをクラス図に追加する実験を行った。事前に利用者は編集するクラス図と、適用するパターンを決めておく。今回は図 5 にある Strategy(戦略)パターン(GoF)を追加した。そしてパターンの適用実験を行った。

実験の結果、図 8 の左図のようにツールはクラス図の指定した箇所へパターンを追加することができた。この際にパターンの属性やメソッドなどは、XML ファイルと同じ名前で機械的に追加された。その後、図 8 の右図のように人が新しく追加された要素を図に合わせた。具体的にはクラス名や属性名、メソッド名、関連名などをクラスに合わせて変更した。

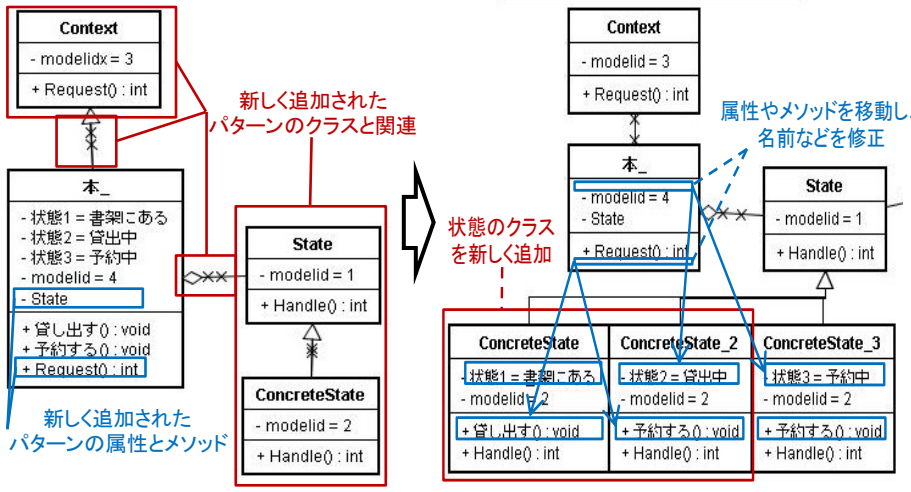

図 8 パターンを適用したクラス図(図書館の例。左がパターン追加後、右が人修正後)

6.2 パターンの体系化の考察

振舞と目的（効果）という 2 つの基準によりパターンを分類した。そして体系化した結果を付録 B の直交表にまとめた。その結果、同じ解決手法に分類されたパターンのうち 50%が、振舞と目的・効果に関連があると判明した。ただし目的の分類は振舞の分類と同時に行ったので、振舞の分類項目に引きずられた可能性がある。本稿ではこの分類項目の適切さを判断することはできなかった。よって今後は特に目的(効果)の分類項目を見直す必要がある。

本稿は目的と振舞の分類を手で行った。このため類似関係が分からず、整理しきれないパターンもあった。今後はこれらの類似度を自動的に計測したい。機械が分類を行えば、人による分類ミスを防ぎ、かつ類似度を数値で明確に示すことができる。

6.3 パターン適用ツールの考察

パターン適用ツールを用いた実験の手順と結果について考察する。

利用するパターンを決めた後、追加するクラス図のクラスとパターンのクラスを対応させた。それには利用者がパターンのクラスについての知らなければならない。この時に本ツールへパターンを指定すると、そのクラス図が表示できた。この機能により解説書からパターンのクラス図を探す時間が短縮できた。

次に適用対象のクラス図へ、パターンのクラスを合致させる。具体的にはどのクラスがパターンのどのクラスと対応するかを id で指定する。これは設計者にとって非常に難しい作業なので、時間が掛かった。しかし実際に対応箇所を図に登録する作業は、「modelid= n」という属性をクラスに追加するだけで終わった。

そしてツールを実行すると、クラス図の指定した箇所にパターンが機械的に追加されていた。そしてこれらの要素を人がクラス図に適合させた。追加できた要素は、クラス、その属性とメソッド、クラス間の関連、その名前と種類である。これによりパターンのクラスを定義した XML 形式は、有効だったと言える。

本ツールの適用プロセスを総合して評価すると、人の果たす役割が大きいことが分かった。今後はパターンを適用する箇所を自動で判断させるなど、適用の自動化を進めてゆく必要がある。そうすればより楽にパターンを利用できるようになるだろう。さらにクラス間の文脈を考えて適用できれば、人の手を借りることなく利用できるツールになる。

7. おわりに

ソフトウェアパターンは完全には整理し切れていない。本稿では今まであまり研究されていなかった振舞と、パターンの目的・効果に着目して体系化を試みた。その結果約 50%のパターンでこの 2 つの基準間に関連が見られた。しかし各基準の項目の正確さは検証できなかった。それでも問題の解決方法とパターンの処理動作から体系化する提案になるだろう。

次に既存のクラス図へパターンを追加するツールを作成した。この際にツールがパターンを利用するために、クラスを XML 形式で定義しデータ化した。そして適用実験を行うと、クラス図の指定した個所へパターンのクラス情報が機械的に追加できるようになった。今後は振舞の類似度を自動的に測定できれば、より明確に整理できるだろう。

参考文献

1. **Erich=Gamma, ほか:** デザインパターン改訂版. : SoftBank Creative, 1999.
2. **PLoPD_Editors, 細谷竜一, 中山裕子:** プログラムデザインのためのパターン言語. : SoftBank Publishing, 2001.
3. **Frank=Buschmann, ほか:** ソフトウェアアーキテクチャ. : 近代科学社, 2000.
4. **Deepak=Alur, John=Crupi, Dan=Malks.** J2EE パターン第2版. : 日経BP, 2005.
5. **Floyd=Marinescu.** EJB デザインパターン. : 日経BP 社, 2003.
6. **久保淳人, ほか:** 文書間類似度によるソフトウェアパターン間関連分析と複合関連の導出. : 情報処理学会ソフトウェア工学研究報告 SE-154-9 Vol.2006 No.125 pp.65-72, 2006.
7. **大木幹雄** クラス構成演算によるデザインパターン導出実験. : 情報処理学会 オブジェクト指向シンポジウム L3499A-2003 pp.145-148, 2003.
8. **平山洋介, 大木幹雄** 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支援CASE ツールに関する研究. : 日本工業大学 ソフトウェア工学研究室 2000 年度卒業研究, 2000.
9. **Salukz=Koudituwakkz=R, Peter=Bertok.** Pattern Categories: A Mathematical Approach for Organizing Design Patterns. : Proceedings of the 2002 conference on Pattern languages of programs - Volume 13, pp.63-73, 2002.
10. **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. : 情報処理学会ソフトウェア工学研究報告 SE-111-6 Vol.1996 No.84 pp.41-48, 1996.
11. **永山英嗣, 原田実** Design Patetm Applying Support OOPAS by Design Diagram Merging. : IEICE TRANS. INF. & SYST. VolE83-D No.6, 2000.
12. **(株)チェンジビジョン.** Jude-Professional. (オンライン) http://jude.change-vision.com/jude-web/index.html.
13. **讃井崇喜, ほか:** デザインパターンのモデル化と適用支援ツール. : 電子情報通信学会ソフトウェアサイエンス研究報告 SS-100(63) pp.1-8, 2000.
14. **大月美佳, ほか:** デザインパターンのSGML による構造化文書化とその閲覧. : 情報処理学会論文誌 Vol.39 No.3 pp.636-645, 1998.
15. **瀬川淳一, ほか:** デザインパターン利用支援システムのソースコード生成支援. : 情報処理学会第57回全国大会 5J-8 第1分冊 pp.225-226, 1998.
16. **(株)チェンジビジョン.** JUDE-API. (オンライン) http://jude.change-vision.com/javadoc/jude-api/5_5/api/ja/doc/javadoc/index.html.

付録A　パターンの目的の分類基準

傾向	分類基準	詳細な分類基準	区分	パターン
構造の整理	まとめ役を付ける	複雑なサービスの簡単な入口と内部隠ぺい ※集約される側が多様な場合	・・・振舞・データ・両方の3区分・・・	・・・それぞれ該当するパターン・・・
		ひとつの共通の入口 専門の入口 ※集約者を限定する		
		状態や Client により振舞やデータを変える=違う振舞を一クラスに見せる		
		インターフェイスの仲介役		
		共通部分をまとめる		
		処理や Filter をまとめる		
	構造を集約・独立する ※まとめ役はなし（構造をまとめるアプローチ）	データの集約		
		振舞の集約		
		データと振舞の一元化		
		その他、不明		
	構造の分離・分割 （構造を細分化するアプロー	クラスと状態の分離		
		オブジェクトと永続化データの分離		

	チ)	作業の構造(振舞)を組織化		
		クラスと実装の分離		
		データと振舞の分離		
		MVC(GUI)の分離		
通信・分散並行設計	一時保存データをカプセル化しキャッシュへ保存 ※通信の一部	転送・共有データのカプセル化と一時保存(TO)		
		データアクセスのコマンドを独立		
		キャッシュに DB へのアクセスデータを保持		
	通信・並行・並列処理設計	通信・メッセージングのパターン		
		分散処理と並行性制御設計のパターン		
		並列処理のパターン		

データアクセスとデータ操作	データ・メソッドの操作	情報の共有		
		生成のパターン		
	データ（アクセス）関連	検索のパターン		
		更新のパターン		
		復旧のパターン		
		データアクセス（Object にデータを保持）		
		データアクセス（Object にデータと接続ロジックを保持）		
その他	その他	その他		

付録B　パターンの振舞と目的・効果による体系化（一部抜粋）　※背景色付きは、目的・振舞の 1 カテゴリに付き、2 つ以上のパターンがある箇所

パターン	目的と	まとめ役を付ける							構造の集約・独立 *まとめ役なし(構造をまとめるアプローチ)		構造の分離・分割（構造を細分化するアプローチ）						一時保存.(TO系) データをカプセル化してキャッシュへ保存			データ・メソッドの操作				データ(アクセス)関連		
	対象範囲	複雑なサービスの簡単な入口と内部隠ぺい *集約される側が 多様な場合			ひとつの共通の入口 専門の入口 *集約者を限定する			振舞の集約	データと振舞の一元化		クラスと状態の分離	オブジェクトと永続化データの分離	作業の構造(振舞)を組織化	クラスと実装の分離	データと振舞の分離	MVC(GUI)の分離	通信用の TO 転送・共有デー	データアクセスのコマンドを独	キャッシュに DB への アクセスデータを	情報の共有	生成			データアクセス（Objectにデータを保持 DAO系）		データアクセス（オブジェクトにデータと 接続ロ
振舞の構造		データ	振舞	データと振舞の両方	データ	振舞	データと振舞の両方	振舞	オブジェクトと振舞の両方		オブジェクト	オブジェクト	振舞	振舞	オブジェクトと振舞の両方	オブジェクトと振舞の両方	データ	振舞	データと振舞の両方	データ	データ	振舞	データと振舞の両方	オブジェクト	振舞	データと振舞の両方
特殊な形	Chain Of Responsibiligy型	Business Object,					Decorator,																			
	Intercepting Filter型		Application Service,	Proxy,									Master-Slave, Whole-Part,													
	State, Visitor型								State,		State,				Visitor,											
まとめ役型	Proxy型			Proxy, 反応体,		View Handler,										View Handler,	View Handler,						Essence の生成,			
	Memento型																Memento,									
	Business Delegate Factory型																					Business Delegate Factory,	Essence の生成, Builder, Prototype,			
通信型	Session Facade型			Facade,		Front Controller,																				
Command型	Command型			Facade,	オブジェクト同期体,	Command, EJB Command,								Command, EJB Command,	Command, EJB Command,		オブジェクト同期体,									
TO型	TO Assembler型		Service Activator-Aggregator 戦略,			Front Controller, Front Controller-Command and Controller戦略,		Generic Attribute Access,										Data Access Command Bean,			TO Assembler,			Data Access Command Bean,		
	Strategy型																Data Transfer HashMap,				DTO Factory,					
	Transfer Object型		Business Delegate,		Bodyguard,				Application Controller,								Transfer Object,			Flyweight,						
	Domain Store型		Service Locator,									Domain Store,							Domain Store,							Domain Store,
DAO型	DAO型1		Business Delegate,				Data Access Object,		Data Access Object,										Data Access Object,					Data Access Object,		Data Access Object,
	DAO型2						DAO-Read Only Lowset戦略,		DAO-Read Only Lowset戦略,								Value List Handler,		DAO-Read Only Lowset戦略, Value List Handler,					DAO-Read Only Lowset戦略,		DAO-Read Only Lowset戦略,
	DAO型3											Domain							Domain Store,							Domain Store,
特殊	オリジナル						Singleton,									MVC,	Cache,									

付録C　振舞によるパターンの分類

	特殊な構造			まとめ役型			通信型		Command型	TO型				DAO型			特殊な振舞			他		
振舞の説明	責任が連鎖する	まとめ役が複数のメンバーを呼び出す	呼び出したオブジェクトに対して処理を実行する	サービスの代理実行する	まとめ役を介してTOを生成し、Set/Updateをして、取得する	Clientがまとめ役を生成し、後はMementoタイプ(左)と同じ	送信(呼出)し、まとめ役がターゲットを呼出す	2つのTOを生成する。かつメッセージを送って通信する	まとめ役がターゲットを生成し、ターゲットを実行する	TOのシンプルな型1	TOのシンプルな型2	典型的なTOの型3	「典型的なTOの型3」の中で、TOが2つ生成される型	DAOのシンプルな型 1	「DAOのシンプルな型 1」の中で、DAOが2つある型	「DAOのシンプルな型 1」の中で、ResultSetとクエリを利用する型	どれとも似ていない振舞	複数の型を併せもつ	シンプルな型	呼出(call)と実行(execute)の組合せ	とりあえず似てる1	とりあえず似てる2
パターン	Business Object, Chain Of Responsibility, Decorator,	Application Service, Intercepting Filter, Master-Slave, Proxy, Whole-Part,	Visitor, State,	Essenceの生成, Proxy, Service Adapter, View Handler, 反応体,	Memento,	Builder, Business Delegate Factory, Essenceの生成, Prototype,	Front Controller, Session Facade, Web Service Broker,	Web Service Broker−Custom XML Message戦略,	Command, Command Processor, Message Facade, オブジェクト復旧体, オブジェクト同期体,	Data Access Command Bean, Front Controller−Command and Controller戦略, Generic Attribute Access, Service Activator−Aggregator戦略, Service Adapter, TO Assembler,	Strategy, DTO Factory, Data Transfer HashMap,	Action Adapter, Application Controller, Bodyguard, Business Delegate, Flyweight, Lazy Load, Transfer Object, Work Adapter,	Domain Store, Service Locator,	Business Delegate, Data Access Object,	DAO−Read Only Lowest戦略, Value List Handler,	Domain Store,	Blackboard, Cache, Competing Consumers, MVC, Singleton,	EJB Service Locator, Essenceの生成, Optimistic Concurrency, Pessimic Concurrency,	空オブジェクト	Application Service −Command戦略, Client −Dispatcher Server, Composite Entity, EJB Command, Forworder −Reciever, Front Controller −Dispatcher in Controller戦略, Transaction Context, 受取人−接続人, 先行体, 半同期/半非同期,	Point to Point Distribution, 非同期完了型トークン,	Mediator, Observer,
図	A	B	C	D	E	F	G	H	I	J	K	L	M	N			—	—	—	—	—	—