デザインパターンにおける表現技法

三　河　佳　紀*

Expression technique in Design Patterns

Yoshinori MIKAWA

要　旨

ソフトウェアパターンを用いることにより、優れた過去の経験をソフトウェア開発に再利用することが可能である．オブジェクト指向ソフトウェア設計においては、ＧｏＦのデザインパターンを用いることにより、効率の良い設計が可能となっている．このデザインパターンの構造に関する表現方法を、より具体的に表現する補助的技法について報告する．

Abstract

The experience of the excellent past by using a software pattern is reusable for development of software. As for the object oriented software design, the good design of the efficiency is possible by using the design pattern of Gang-of-Four (GoF).

It is reported about the supplementary technique of expressing the way of expressing it about the structure of this design pattern more concretely.

1．はじめに

過去の経験を基に、繰り返し利用可能なソフトウェアの構築は，オブジェクト指向の概念が出現する以前より試みられて来た．いわゆる再利用可能な部品としてのソフトウェアのパターン化である．プログラミングで考えるならば例示的なものが多く，基本パターンでの作法をまとめたもの，あるいは基本的なアルゴリズムを集約したカタログ集としてのパターン化である．これらは個々の要素をそのまま利用することは可能であったが，形を変えての利用（再利用）は困難であり、新たに開発し直す場合が多く，類似した部品が数多く増える状況にあった．そのため円滑なソフトウェア部品としての再利用までには至らない以場合が多かった．しかし，オブジェクト指向の概念が出現したことにより，再利用が現実化することになった．これらは従来のプログラミングレベルに留まっていた再利用に対し，分析・設計レベルにおいても再利用を考慮するようになったためである．

＊　講　師　　情報工学科

プログラミングにおいても，カプセル化・継承・ポリモルフィズムなどを行なうオブジェクト指向プログラミングの出現により，実装レベルにおいても再利用がいっそう現実のものになってきた．オブジェクト指向ソフトウェア設計においては，GoFのデザインパターン[1) 2) 4) 5)]を用いるのが有効である．本報では，GoFのデザインパターンにおける解法に関する構造の表現方法を記述レベルだけではなく，実行レベルを考慮した図形的表現を用いることにより，視覚的に明確に捉えることを試みたので報告する．

2．デザインパターンについて

オブジェクト指向に基づきソフトウェアを設計するのは簡単ではない．その際，優れた経験に基づき作成されたカタログ等があれば，ある程度円滑に設計が進められる．E.Gamma，R.Helm，R.Jhonson，J.Vissidesら４人は，オブジェクト指向ソフトウェアを設計する際の指針として，23個のデザインパターンをカタログ化[1)]した．これはGoF（Gang-of-Four）のデザインパターンと呼ばれ，オブジェクト指向ソフトウェア等を設

計する際に広く用いられている．これらのカタログは，その目的により生成，構造，振る舞いの3つに分類されており，カタログ内でのデザインパターンの記述については，個々のパターンごとに次の13項目に形式化され統一されている．

1）パターン名，分類
2）目的（原理・意図・扱う範囲）
3）別名
4）動機（問題を解くシナリオ）
5）適用可能性（適用状況・粗悪設計例）
6）構造（図形的表現）
7）構成要素（クラス・オブジェクト）
8）協調関係
9）結果
10）実装（技法・言語依存問題）
11）サンプルコード（C++，smalltalk）
12）使用例（2つ以上提示）
13）関連するパターン(関連デザインパターン)

このように統一表現を行なうことにより，過去の優れた設計経験を比較し目的に応じて参照でき，効率良く利用することが可能となっている．この効果として，求める設計例を選択し再利用することにより，設計に関わる時間の大幅な短縮や，従来のシステムをこのカタログに基づき整理し文書化することにより保守の向上にもつながることなどが上げられている．

GoFのデザインパターンの後にもいくつかのデザインパターンが提案されており，現在では先程の目的による3分類に，平行処理の競合制御に関する設計パターンとしての「並列処理」を加え4分類に分類されている（表1）．今後もこれらの新しいパターンは増えることが予想される．

本報ではGoFのデザインパターンについてのみ扱うことにした．

表1　デザインパターンの分類とパターン数

分　類	GoF のデザインパターン	その他のデザインパターン
生　成	5パターン	1パターン
構　造	7パターン	
振る舞い	11パターン	3パターン
並列処理		4パターン

3．構造の記述

1つのデザインパターンは13項目に形式化されて記述されるが、それらの中で視覚的に記述されるのが構造に関する部分である．構造はデザインパターンのクラス間の構造をOMT手法[3）8）9）]に基づき図形的（視覚的）に表現されたものである．図1に示すのは，オブジェクトの生成に関するパターンであるAbstractFactoryの構造である．

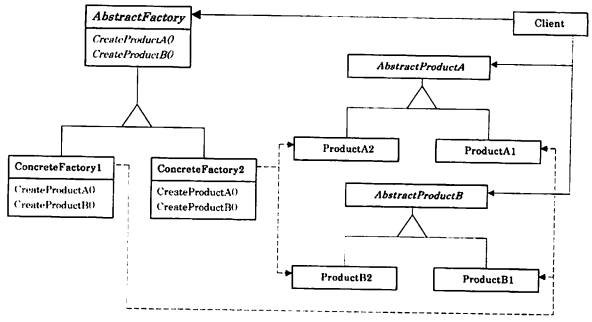

図1　AbstractFactoryの構造

このAbstractFactoryは，互いに関連したり，依存し合うオブジェクト群をその具象クラスを明確にせず生成するためのインターフェースを提供することを目的としたパターンである．このパターンの適用可能性については何点か上げられており，例えばその中において，システム構築の際に複数存在する部品集合の中から1つのみ選択する場合や，システムを独立（部品生成・組み合わせ・表現から）にした方が良い場合などにこのパターンを用いると有効であるとされている．

図1の構造から次のことが読み取れる．Clientクラスについては2つのクラス(AbstractFactory，AbstractProduct）で宣言された実装を持たないオペレーションの集合であるインタフェースのみを使用する形であること．AbstractFactoryクラスについては，インタフェースを宣言していること．これはオブジェクト（AbstractProduct）生成へのオペレーションに対するインタフェースである．ConcreteFactoryクラスは，オペレーションの実装を行なっていること．これはオブジェクト（ConcreteProduct）を生成するオペレーションの実装である．ConcreteFactoryオブジェクトで生成される部品オブジェクトの定義については，ConcreteProductクラスにより行ない、またここではAbstractProductクラスのインタフェースの実装を行なっていること．AbstractProductクラ

スにおいては，各部品ごとにインタフェースを宣言していることなどである．

このように構造に関しては図形的表現を用い記述され，視覚的にもこの記述は重要であるがデザインパターン全体を考えた場合これだけでは不十分であるためGoFのデザインパターンでは13項目に形式化され記述されているのである．

4．本研究で用いた表現方法

デザインパターンの利用は，オブジェクト指向ソフトウェアの設計に際し有効である．そのパターンにおいて特に図形的表現を用いて記述される構造の形式は，設計者にとってデザインパターンのカタログを紐解く場合，特に印象に残る部分でもある．しかし，クラス間の関係を表したクラスダイアグラムだけでは不十分な(理解しずらい)場合もある．そこでこの構造の記述形式の適用を手助けする表現方法があると便利である．すなわち記述レベルだけではなく，実行レベルでのインスタンスの関係（何を実行するのか）について図形的に表現することにより構造の適用を手助けすることを本研究で試みることにした．

通常のOMT手法に基づくクラスの表記法では，クラス名を上段に，その下にオペレーション，さらにその下にはクラスが定義するデータを表示す

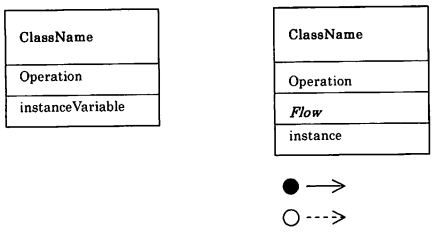

図2　OMT表記法と今回用いた表記法

るのが一般的である．図2は左側に通常のクラスの表記法と，右側に本研究で用いた表記法を示したものである．通常の表記法では項目は3分割であるが、本表記法ではメソッドの流れを表す部分を設けているため4分割にしている．また，●と実線の矢印は記述レベルでの到達点と流れを表す場合に用い，○と破線は実行レベルでの到達点と流れを表す場合に用いる．これらの表記を用いることにより，記述レベルでの表現と実行レベルでの表現を明確にすることが出来る．

5．具体的な表現の試み

5．1　記述レベルでの表現

図1のAbstractFactoryの構造図を本研究での表現方法を用い，記述レベルで表したのが図3である．Clientのメソッドは AbstractFactoryをcallする．ここではAbstractProductのオブジェクト生成へのオペレーションであり，この場合ProductAとProductBの2つのインタフェースがある．その為，黒丸には複数を表す＊を付与している．AbstractFactoryのデザインパターンでは2つの例を示してあるが，これは当然2つ以上であっても同様に＊を付与することで表現できる．次にAbstractProductへの生成である．ここでも各部品ごとのインタフェースを宣言することになるので＊を付与している．以上が図1の構造に対する記述レベルにおける表現である．

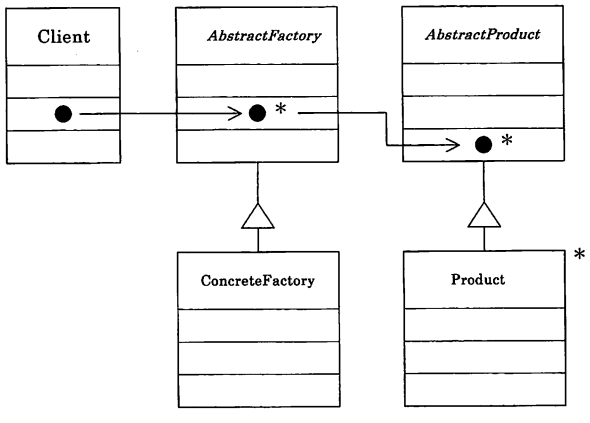

図3　記述レベル（AbstractFactory）

このように記述レベルで表現することにより，オペレーションが複数存在する場合の表現も簡略化出来，クラス間の関係が明確になりデザインパターンにおける構造図の解釈が容易になる．

5．2　実行レベルでの表現

実行レベルでの表現については，具体的な実行レベルでのインスタンスの関係について表現することが目的である．記述レベルと同様に図1のAbstractFactoryの構造図について本研究の表現方法により表したのが図4である．基本的な表現方法は記述レベルと同様である．

ClientはAbstractFactoryクラスのサブクラスであるConcreteFactoryをcallする．このクラス（ConcreteFactory）のインスタンスは実行時に生成される．またインスタンス化される際にアプ

リケーション内では1度だけ仕事を行なうが，異なる部品の場合はClientは別のConcreteFactoryを生成する必要がある．このためここでは1つの場合についてのみ記述すれば良いことになる．

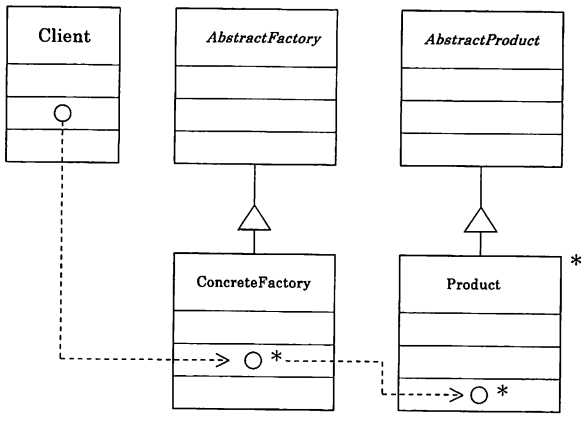

図4　実行レベル（AbstractFactory）

ConcreteFactoryは次に特定の実装を持つProductを生成する流れに移行する．すなわち，AbstractProductのサブクラスであるProductへの生成である．この場合，各部品ごとに複数生成されることが考えられる．このように実行レベルでの流れを表現することにより、何を実行するのかというインスタンスの関係も把握することが可能となる．

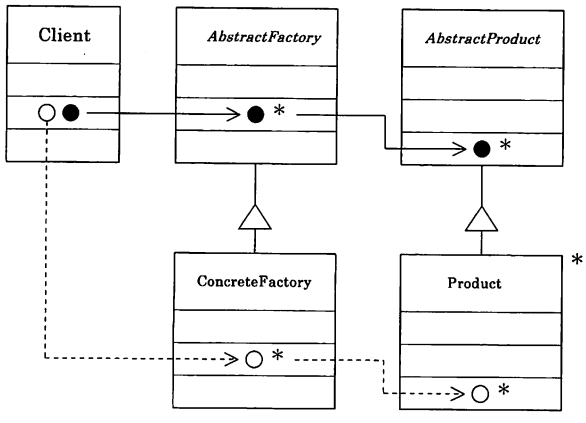

図5　記述＋実行レベル（AbstractFactory）

記述レベル，実行レベルについて具体的な表現方法を試みたが、これら2つの図式表現を重ね合わせることにより更に解釈のし易い形式に移行できる．実際に重ね合わせたものを図5に示す．黒丸と矢印は記述レベルであり、白丸と矢印は実行レベルのものである．重ね合わせたことにより記述レベルと実行レベルについての流れが明確になっている．

6．他のデザインパターンへの適用

何点かのデザインパターンに対して，本表現技法を適用してみる．ここで用いるデザインパターンは，クラス生成に関わるFactoryMethodパターン，オブジェクト・振る舞いに関わるChain of Responsibilityパターン及び，Iteratorパターンであり実際に本表現技法を適用してみる．

6．1　FactoryMethodへの適用

FactoryMethodパターン（図6）は，クラス，生成に関するデザインパターンであり，その目的は次のとおりである．

- オブジェクト生成の時のインタフェースのみを規定
- クラスのインスタンス化はサブクラスが決定

である．すなわちインスタンス化はサブクラスに任せるというものである．

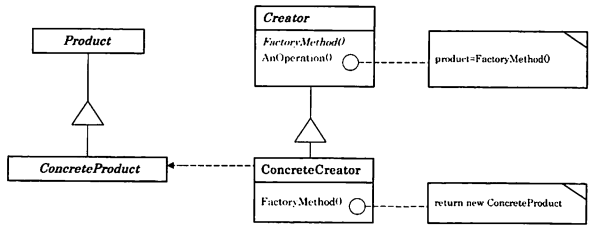

図6 FactoryMethodの構造

図6の構造図は次のように読み取れる．Creatorクラスにおいてはは FactoryMethodを宣言しており，これはオブジェクト（product型）を返すものである．ConcreteCreatorクラスにおいては FactoryMethodをオーバーライドし，ConcreteProductクラスのインスタンスを返す形態である．Productクラスにおいては，インターフェースを定義している，これはFactoryMethodの生成するオブジェクトのインタフェースである．ConcreteProductクラスにおいてはProductクラスのインタフェースの実装を行なうということである．

デザインパターンFactoryMethodに対して，本表現技法を適用してみると図7のように表すことが出来る．記述レベルではCreatorクラスにおいて，product型のオブジェクトを返す形態であるので表記中その記述を● → ●で表している．

その後Productへの生成という流れになる．実行レベルにおいてはCreatorのサブクラスであるConcreteCreatorにより FactoryMethodをオーバーライドし，ConcreteProductクラスのインスタンスを返す形態である．表記中その形態の記述は○ → ○で表している．次にProductクラスのインタフェースの実装を持つConcreteProductへの生成へと移行する，このようにFactoryMethodについても本表現技法での表記を適用することが可能である．

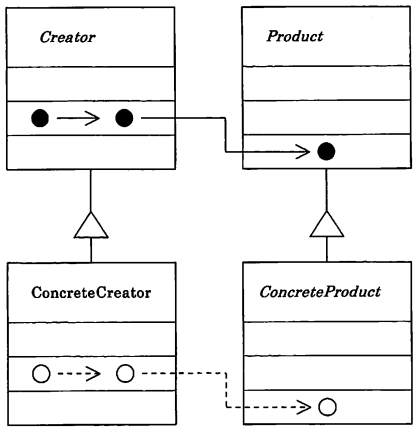

図7　FactoryMethodへの適用後

6．2　Chain of Responsibilityへの適用

Chain of Responsibilityパターン（図8）は，オブジェクト，振る舞いに関するデザインパターンであり，その目的は次のとおりである．

- 1つ以上のオブジェクトに要求を処理する機会を与え，要求送信オブジェクトと受信オブジェクトの結合を避ける．
- 受信する複数のオブジェクトをチェーン状につなぎ，あるオブジェクトがその要求を処理するまでそのチェーンに沿って要求を渡す．

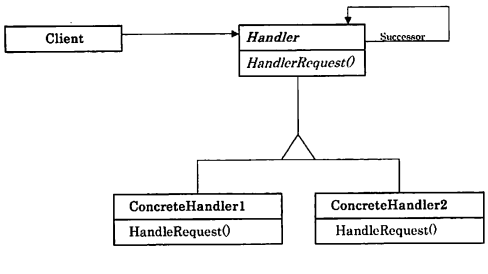

図8　Chain of Responsibility の構造

図8の構造図は次のようになっている．

Handlerクラスは，要求処理のインタフェースの定義であり，successorへのリンクを実装している．ConcreteHandlerクラスは要求の中の処理可能なものについては処理を行い，処理不可能であれば要求をsuccessorに転送する．Clientクラスについてはチェーン中のConcreteHandlerオブジェクトに要求を出すものである．

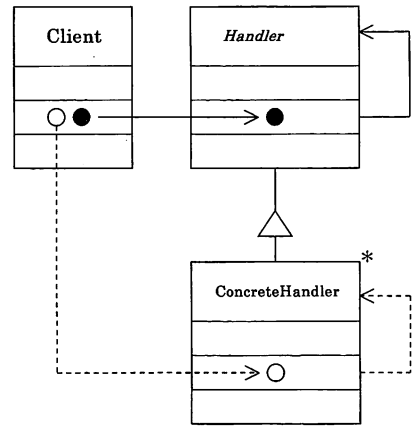

図9　Chain of Responsibility適用後

デザインパターンChain of Responsibilityに対して，本表現技法を適用してみる．図9が適用後の状態である．ここでは今までとは異なる新たなチェーン状の表現を考慮する必要がある．記述レベルではClientがHandlerをcallして要求が処理するまで次々と要求を渡す形となる．実行レベルにおいては，clientからの要求はHandlerのサブクラスであるConcreteHandlerへ流れる．これも記述レベルと同様に要求を処理するまでチェーン状に繰り返す形である．チェーン状のつながりについては記述・実行レベル共に矢印の折り返しという形で表記を行なった．このようにチェーン状の表現を持つChain of Responsibilityにおいても本表現技法での表記を適用することが出来た．

6．3　Iteratorへの適用

Iteratorパターン（図10）はChain of Responsibilityパターンと同様に，オブジェクト，振る舞いに関するデザインパターンの1つである．その目的は次のようになっている．

- 集約オブジェクトが基にある内部表現を公開せずに，その要素に順にアクセスする方法を提供．

図10の構造図は次のようになっている．Clientクラスが破線で囲まれているが，これはこのパターンが client を構成要素として含まない

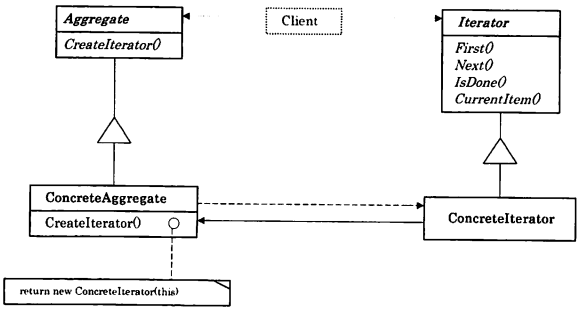

図10 Iterator の構造

ことを表している．Aggregateクラスについてはインタフェースを定義している．これはIteratorオブジェクトを生成するためのものである．ConcreteAggregateクラスはAggregateクラスのインタフェースに対しConcreteIteratorクラスのインスタンスを生成し返す形態である．ConcreteIteratorクラスについてはインタフェースの実装である．これはIteratorクラスで定義されたインタフェースについてである．Iteratorクラスについてはインタフェースの定義であり，これは要素にアクセス，あるいは操作のためのインタフェースである．

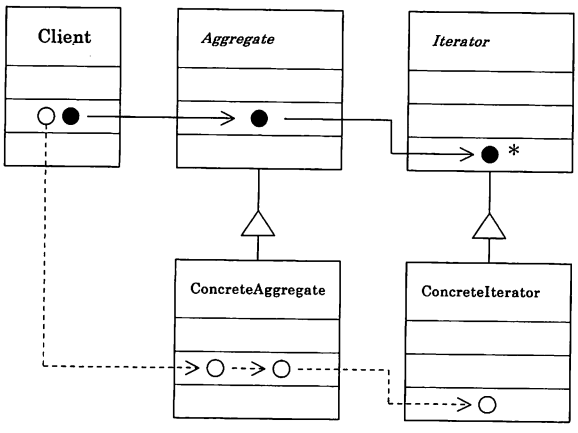

図11 Iterator 適用後

デザインパターンIteratorに対して本表現技法を適用したのが図11である．実行レベルにおいてConcreteAggregateクラスではAggregateクラスのインタフェースに対し，インスタンスを生成し返す形態であるため，6.1のFactoryMethodと同様の表現を用いている．このようにIteratorに対しても本表現技法を適用することが出来た．

7. おわりに

本研究ではオブジェクト指向ソフトウェア設計の際に，問題に対する解法として有効なデザインパターンの構造について，その記述形式の適用を助けることを目的に新たに補助的な表現方法を考案し，何点かのデザインパターンに対して実際に適用することを試みた．本表現技法を用いることにより，記述レベルでのクラス間の関係の明確化，あるいは表現の簡略化がなされ，実行レベルにおいてはインスタンスの関係等が視覚的に得られることが確認できた．しかし本表現技法はあくまでもデザインパターンの構造に対する補助的な表現であり，単独使用ではなくデザインパターンの構造図と併用することにより，そのデザインパターンにおける構造図の解釈が容易になると考える．今後の課題としては本表現技法のカタログ化，表現方法の細分化，UML[6)7)]での表現などが上げられる．

参考文献

1) E.Gamma,R.Helm,R.Johnson,J.Vissides: 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，1999
2) W.pree：デザインパターンプログラミング（補訂版），株式会社トッパン，1998
3) J.ランボー，M.ブラハ，W.プレメラニ，F.エディ，W.ローレンセン ：オブジェクト指向方法論OMT，株式会社トッパン，1996
4) R.Johnson,中村宏明,中山裕子,吉田和樹：パターンとフレームワーク，共立出版，1999
5) 佐原伸：デザインパターン オブジェクト指向分析・設計技法，株式会社ソフト・リサーチ・センター，1999
6) Martin Fowler・Kendall Scott：UMLモデリングのエッセンス，アジソン・ウェスレイ・パブリッシャーズ・ジャパン，1998
7) C.Larman：実践UML パターンによるオブジェクト指向開発ガイド,株式会社ピアソン・エデュケーション，1998
8) P.Coad，E.Yourdon：オブジェクト指向分析（OOA），株式会社トッパン，1996
9) S.Shlaer，S.J.Mellor：オブジェクト指向システム分析，近代科学社，1995

（平成12年11月29日受理）