デザインパターンの形式化とコード生成支援

Formalizing Design Patterns and A Source Code Generation Support System

佐藤 大輔 †
Daisuke Sato

† 公立はこだて未来大学大学院 システム情報科学研究科
Graduate School of FUTURE UNIVERSITY-HAKODATE
g2104019@fun.ac.jp

伊藤 恵 ‡
Kei Ito

‡ 公立はこだて未来大学
FUTURE UNIVERSITY-HAKODATE

近年のソフトウェアはあらゆる分野で重要な役割を担っており, その需要も今後ますます拡大することは間違いない．それにつれ, 開発期間の短縮や効率の改善, コストの削減など, ソフトウェア開発の現場に対する要求もますます厳しくなっている． ソフトウェアパターンとは, ソフトウェア開発の各局面で繰り返し現れる問題に対する解法・指針である. その中でも, デザインパターンは, プログラム・クラス設計の工程で, 優れた設計を導出する課程をパターン化したものであり, 開発者の大きな手助けとなり得る． 本研究ではまず, 非形式的に与えられている GoF のデザインパターンを形式化することにより, 電子カタログ化や機械的処理を容易にする意味的構造を表現できる, 構造化記述形式の枠組みを設計する．次いで, プログラムのコーディングを行う際に, デザインパターンをより容易に扱えるよう, コード生成の機能を持ったツールを開発し, 利用者を支援する仕組みを提案する．

1 はじめに

ソフトウェアパターンは, ソフトウェア開発の各プロセスにおいて繰り返し現れる問題に対する解法・指針である．ソフトウェアパターンを活用した場合の効果として，以下の 3 点が挙げられる．

1. 思考過程と成果を再利用できる．
2. 意志疎通を促進できる．
3. 優れたソフトウェアの解読を促進できる．

これらの利点から，オブジェクト指向技術が多用されるようになるにつれ，ソフトウェアパターンは非常に重要な存在となっている．

プログラム・クラス設計段階に用いるソフトウェアパターンの一つとして，GoF のデザインパターン [1] がある．これは Gamma らが，オブジェクト指向ソフトウェアの構造や機能について，典型的に現れる特徴的な類型を抽出して解析し，23 個のパターンとして非形式的な文書でカタログ化したものである．GoF のデザインパターン[1]では，パターンの構造を表現するためのモデル，振る舞いを示すための疑似コードなどが平文による説明を付加して用いられている．これは，開発者自身が文書を読み解き，パターンの意味的な構造を理解し，パターンを選択，適用する際の助けとなる．開発者はパターンの持つ性質，構造とその構成要素間の協調動作を学び，この構造とその構成要素間の協調動作を実現する関係を壊さないように，開発環境に合った具体化を行う (図 1)．

開発規模が増大し，使用するパターンの数や具体化される箇所が増える場合，開発者がそれらのすべてを考慮し，設計，把握，管理することは困難になる．その結果，パターンの意味的構造を破壊し，先に挙げた 3 点の効果が得られなくなる．

本研究では，パターンの持つ意味的構造を破壊することなく，且つ，開発者がパターンを使用する際の負担を軽減することを目標に，デザインパターンの XML による形式化を行い，パターンリポジトリ

[1]以下，単に「パターン」と表記する場合，これを指す．

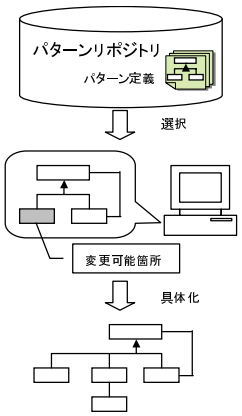

図 1: デザインパターンを用いた開発の流れ

として蓄積する．さらに，パターンリポジトリより，使用するパターンを選択しソースコードの枠組みを与え，開発者を支援する仕組みを提案する．本システムの概要を図 2 に示す．

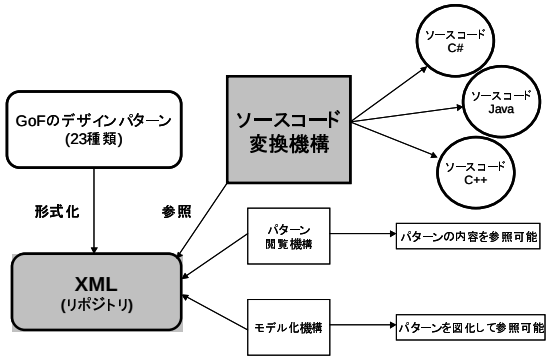

図 2: 本研究で提案するシステムの概要

2 関連研究

GoF のデザインパターンに形式的な記述を与え，検索・閲覧を容易にする試みとして大月らによる [3] がある．[3] では，[1] に示されたそれぞれのパターンの内容を，SGML を用いて形式化を行い，Web ブラウザにてパターンの検索を可能にし，構造や用途を使用者に表示する仕組みを提案している．大月らは [3] のほか，ソフトウェアの共同開発支援を目指して，オブジェクト指向ソフトウェア部品の統合的且つ分散的なリポジトリの構築を研究しており，本稿で提案する方式を研究するにあたり参考になる点が多い．しかし，構造の記述においては，ほぼ [1] に示された形のまま SGML 化しているため，コード生成支援という観点では記述法の見直しが必要である．

また [2] では，GoF デザインパターンを Java 言語を用いて形式化を行い，ツールを用いてパターンのモデルを提示することで，開発者がパターンを使用する際の負担を軽減する手法を提案している．形式化の方法については，構造をよく表現しているため，コード生成についても有用かと考えられるが，[2] での目標は，パターンが使用時において複数の組み合わされた状態であっても，これをモデルとして視覚化し，開発者に注意を促すことにあり，コード生成までの支援は目指しておらず，形式も Java 言語での記述の為，使用可能な環境が限られてしまう．

3 デザインパターン

デザインパターンの利用は，元々は建築設計の分野で提唱された手法であり，経験的に得られた繰り返し現れる重要且つ有用な設計を，それぞれ体系的に名前付けし，説明を加え，評価したものである．ソフトウェア工学はしばしば建築工学などとも類似する部分があると言われるが，デザインパターンもソフトウェア工学に取り入れられ，ソフトウェアの設計におけるノウハウや，システムの構造を部品として再利用するのを促進するものとして注目されている．デザインパターンは基本的に以下の 4 種類の記述から構成される．

1. パターン名
2. 問題 ••• どのような場合にパターンを適用すべきか
3. 解法 ••• 設計の要素，それらの関連および責任，協調関係
4. 結果 ••• パターンを適用する際の結果やトレードオフ

デザインパターンをカタログ文書としてまとめた著名な文献 [1] では，これらの 4 項目をさらに細かく分類し，以下のような構成をとっている．

1. 名称 Name
   別名 As Known As
2. 目的 Intent
   動機 Motivation
   適用可能性 Applicability

3. 構造　Structure

   構成要素　Participants

   協調関係　Collaborations

   実装　Inplementation

   サンプルコード　Sample Code

4. 結果　Consequences

その他　使用例，関連パターン

これらは平文での説明がなされており，構造や振る舞いを表す OMT 図[2]や疑似コードも使われている．本研究での記述の書式も基本的にこれを踏襲する．

4 XML によるパターン記述形式

4.1 XML の利点

本研究ではパターンに形式的な記述を与える言語として XML を用いる．XML は文書中の文字列に意味付けができるデータ構造の表現が可能な言語として，近年広い分野で使われており，XML で記述されたデータの汎用性や互換性といった面からも，パターンを定義するのに最適であると考えた．

XML ではデータの要素をタグで囲み，文書を構造化する．各タグは属性を持つことができ，これにより様々な意味構造を表現することが可能である．

XML データとして記述する枠組みが決まっていれば，定義したいパターンが増えた場合も記述形式に則って新たなパターン定義をリポジトリに蓄積することで，パターンの利用は容易になる．

また，本研究で開発するシステム以外でも，パターンリポジトリに蓄積されたデータは参照可能であり，本研究で目的とするコード生成以外の用途として，パターンの閲覧や，モデル化といった，関連研究に挙げたような目的にも転用可能である．

4.2 パターン記述形式

デザインパターンに意味的構造を持った形式的記述を与えるため，まず，文献 [1] の内容を XML で記述すべく，パターンの説明文より各々の項目に対応するタグを定義する．ソースコード生成に使用する structure タグ以外の部分については，大月らが論文 [3] で定義したタグが適当であると考えるため，これを参考にする．各項目とそれに対応するタグの名称を図 3 に示す．パターンリポジトリに蓄積されたパターンより，それらの説明を参照できるようにするため，コード生成に必要な構造記述以外の平文による説明は，[1] の記述内容をタグの要素としてそのまま埋め込む．以下に State パターンの例を示す．

[2]UML の創始者の 1 人であるランボーが，UML 以前に提唱していたモデリング手法である．Object Modeling Technique.

名称	<pattern>のname属性
別名	<pattern>のalias属性
目的	<intent>
動機	<motivation>
適用可能性	<applicability>
構造	<structure>
構成要素	
協調関係	
実装	<implementation>
サンプルコード	<sample_code>
結果	<consequence>
使用例	<known_uses>
関連パターン	<related_patterns>

図 3: タグの定義

```
<GoFDesignPattern>
  <Pattern name=“ State ” alias=
“ Object for States ”>
    <intent>
      オブジェクトの内部状態が変化したときに …
    </intent>
    <motivation>
      ネットワーク網を表す TCPConnection クラス …
    </motivation>
    <applicability>
      次に示すいずれかの場合に，State パターン …
    </applicability>
    <structure>
      <note>
        *ここには，構成要素，協調関係の説明文が入る*
      </note>
      *コード生成に必要な情報が入る*
    </structure>
    <implementation>
      State パターンでは実装上の様々な問題を …
    </implementation>
    <sample_code>
      次の例は,「動機」の節で述べた TCP 接続の例に …
    </sample_code>
    <consequence>
      State パターンは次のような結果を …
    </consequence>
    <known_uses>
      Johnson と Zweig[JZ91] は，State パターン …
    </known_uses>
    <related_patterns>
      Flyweight パターン:いつどのように …
    </related_patterns>
  </Pattern>
</GoFDesignPattern>
```

この例では，パターン名を Pattern タグの属性として，別名とともに与え，Pattern タグ内にそれ以下の項目を要素として入れ子にしている．structure タグ内の note タグには，平文で与えられた構造，構成要素，協調関係の説明を記述する．structure タグ内

のコード生成に必要な情報については次節にて説明する．

4.3 コード生成に必要な記述形式

パターンに形式的な記述を与える際に，開発者がパターンの持つ意味的構造を破壊するのを防ぐため，ホットスポット (パターンを具体化する際に変更しなければならない部分) とフローズンスポット (意味的構造を壊すため変更すべきでない部分) を考慮する必要がある．これらの説明のため前節で用いた State パターンを例にとる．State パターンの OMT 図は図 4 のように表される．

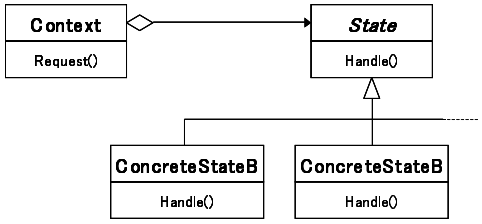

図 4: State パターンの OMT 図

Context クラスはクライアントに必要なインタフェースを定義し，状態を表す ConcreteState クラスのインスタンスを保持している．State クラスは Context クラスの個々の状態に関する振る舞いをカプセル化するためのインタフェースを定義している．ConcreteState クラスは Context クラスの 1 つの状態に関する振る舞いが実装される．

これらのクラスの関係において，Context クラスと State クラスの集約関係を，ConcreteStateA クラスとの集約関係に変更すると，ConcreteStateB の実装している振る舞いを使用することができず，State パターンの意味的構造が崩れてしまう．そのため Context クラスと State クラスの集約関係はフローズンスポットである．また，ConcreteState クラスを具体化する際，A，B，･･･ と振る舞いに応じて増やしていかなければならないため，ホットスポットであると言える．

つまり，State パターンを形式化するには以下の点に注意する必要がある．

1. Context クラスと State クラスの集約関係．(フローズンスポット)
2. State クラスと ConcreteState クラスの継承関係．(フローズンスポット)
3. ConcreteState クラスは複数存在することがある．(ホットスポット)
4. Context クラスは ConcreteState クラスのインスタンスを保持しなければならない．
5. ConcreteState クラスは State クラスを具象化するメソッドを実装しなければならない．

これらの点を考慮し，以下に State パターンの structure タグ内における形式的な記述例を示す．

```
<role name=“ state ”
      attribute=“ interface ”
      implements=“ ”
      integrate=“ ”
      *以下，他クラスとの関連を記述*
      ···>
  <ban>
    *ここに禁止操作を記述*
  </ban>
</role>
<role name=“ ConcreteState ”
      attribute=“ class ”
      implements=“ State ”
      integrate=“ ”
      *以下，他クラスとの関連を記述*
      ···>
  <ban>
    *ここに禁止操作を記述*
  </ban>
</role>
<role name=“ Context ”
      attribute=“ class ”
      implements=“ ”
      integrate=“ State ”
      *以下，他クラスとの関連を記述*
      ···>
  <ban>
    *ここに禁止操作を記述*
  </ban>
</role>
```

各クラスの役割を role タグと表現し，role タグの属性として，各クラスの情報を埋め込んでいる．ban タグには開発者がパターンを利用する際，禁止しなければならない点について注意を促すため，必要な情報を埋め込む．

5 パターン利用支援システム

デザインパターンを用いたシステム設計では，使用するパターンを一つ，もしくは複数選択し，パターンの各要素を開発対象のシステムに対応した構成要素として具体化し，既存部品などの組み込みを行い，言語依存情報を補填してコーディングを行う．本研究では，開発者が行うこれらの作業を機械的に自動

支援するためのシステムを提案する．最終的にはパターン構造を反映した自動コード生成を目指すが，生成するのは完全に動作可能なソースコードではなく，ソースコードの枠組みである．パターンを利用する開発者が，パターンの誤った適用を行うのを防ぐため，コーディングの一段階手前まで誘導する．システムの基本的な流れを図 5 に示す．パターンの選択，

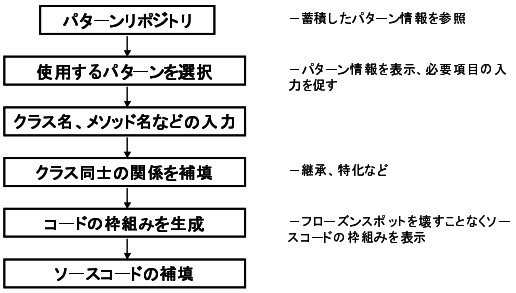

図 5: パターン利用支援の流れ

具体化を行う際，各段階でそれぞれのパターンの具体化に最低限必要な情報の入力をシステムの利用者に求めることで，意味的構造の破壊や，利用者の混乱を防止する．生成されたコードを直接利用者が変更してしまうと，パターンの意味的構造が崩れてしまうおそれがあるが，これは，禁止操作をコメントとしてソースコード内に表示し，注意を促す．

6 まとめと今後の課題

本研究では，デザインパターンを機械的に扱うため，XML を用いて形式的な記述を行った．論文 [3] では SGML を用いていたが，この場合より，現在普及している XML はライブラリの充実などから扱いやすく，コード生成に必要なデータの記述に優れていると考える．またパターン利用支援システムについては，2005 年 7 月現在，提案するシステムを開発中である．パターンの XML 記述形式も，システムの使用時に必要な情報と補完し合いながら，相互に突き詰めていく必要がある．

参考文献

[1] E.Gamma, R.Helm, R.Johnson, and J.Vlissides, Design Pattern: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995; 本位田他訳, 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.

[2] 小林 隆志, 佐伯 元司, デザインパターンのオブジェクト指向モデル化と支援ツールへの応用. コンピュータソフトウェア, Vol. 21, No. 1 (2004), pp. 60–75.

[3] 大月 美佳, 瀬川 淳一, 吉田 紀彦, 牧之内 顕文, デザインパターンの SGML に基づく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.

[4] M.Otsuki, N.Yoshida, A.Makinouchi, A Source Code Generation Support System Using Design Pattern Documents Based on SGML, IEEE, 1999

[5] 鷲崎 弘宜, 深澤 良彰, ソフトウェアパターン研究の現在と未来. 情処研報, Vol.2003, No.55 (SE-141)