ソフトウェアパターンの分析に基づくパターン記述言語の提案

奥村 和恵† 金澤 典子† 塚本 享治†
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†

1. はじめに

簡単に変更でき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提唱された。しかし慣れない設計者がパターンを利用するにはその数が多く、どれを使えばよいか判断が難しい。一方慣れた設計者はパターンを設計図に記入する手間が煩わしい。誰でもパターンを簡単に利用できるようにしたい。

そこで本稿は約 80 パターンについて、目的と実装時の構造を分析した。次に目的、クラス構造、振舞の 3 種毎に分類した。そして分類したパターンを記述できる言語構文を考案した。

2. ソフトウェアパターンの分析と分類

ソフトウェア設計に使われるパターンは、レイヤーやアーキテクチャごとに数多く存在する。GoF デザインパターン[1]、POSA[2]、J2EE デザインパターン[3][4]、EJB デザインパターン[5]、その他[6]などである。

設計過程でパターンを利用するには、これらパターンを総括的に扱う必要がある。そこで既存のパターンを分析し、新たな分類基準を導入することにした。用いた分類基準は、パターンの目的、クラスの構造、振舞の 3 種である。

2.1. パターンの目的による分類

第一に、ソフトウェアパターンを利用する目的別に各パターンを分類した。目的を分類することにより、設計過程で用いるパターンを選択しやすくするためである。分類結果の概要は以下である。

- サービスの内部隠蔽と簡単な窓口-Façade, Broker, Service Locator, Command, 他
- 作業構造を組織化-Command Processor, 他
- エンティティの集約-Composite Entity, 他
- エンティティの共有-Flyweight
- クラスと実装の分類-Bridge, Visitor, 他
- 仲介者(共通インターフェイス)-Adapter, 他
- オブジェクト構造-型オブジェクト, 他
- データアクセス-DAO, Domain Store, 他
- データの一時保存-DTO, Composite TO, 他
- CRUD-Factory, Builder, Observer, 他
- 分散処理と並行設計、並列処理-Broker, 他
- MVC(GUI)-MVC, PAC
- メッセージング- Control Bus, 他
- その他-Interpreter, OR マッピング, 他

以上の各項目について、さらに詳しく利用目的を整理した。その一部を図 1に示す。

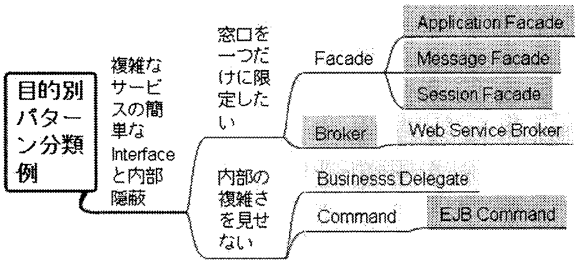

図 1 目的による分類（例）

2.2. パターンのクラス構造による分類

第二に、パターンの静的な構造である、クラス図のクラス構造により分類した。

クラス構造による分類は主に、汎化型（インターフェイス化）、移譲型（集約・コンポジション）、その他の 3 種に分かれた。

- 汎化型（インターフェイス化）
  - ➢ 単純な汎化
    - ✧ 標準的汎化=TO, Proxy, Layer, 他
    - ✧ 親クラスの集約=Command, State, 他
    - ✧ 親クラスのコンポジット=Command, 他
    - ✧ 子クラスの集約=Composite, 他
  - ➢ 子クラスの多重汎化=Adapter
  - ➢ 汎化の階層化
    - ✧ 標準的汎化の階層化=Service Façade
    - ✧ 親クラスの集約=Decorator, 他
  - ➢ 汎化の二重構造
    - ✧ 標準的汎化の二重構造=Iterator, 他
      - ● 生成パターン=Factory Method, 他
    - ✧ 親クラスを集約=Observer, Bridge
    - ✧ 親クラスをコンポジット=Visitor
- 移譲型（集約・コンポジット）
  - ➢ 単純な集約=Memento, Master-Slave, 他
  - ➢ 自分自身をコンポジット=Business Object, 他
- その他
  - ➢ 複雑な複合体=Value List Handler, 他
  - ➢ 汎化も移譲も無し=DAO, 他

A proposal for pattern definition language based on analyzing patterns
Kazue Okumra†, Noriko Kanazawa†, Michiharu Tsukamoto†
†Tokyo University of Technology, Graduate Course of Bio-Informational Media, Major of Media Science

2.3. パターンの振舞による分類

第三に、パターンの動的な構造である振舞により分類した。振舞とは具体的に言うと、シーケンス図におけるメッセージ群のことである。振舞による分類結果の一例を図 2に示す。

目的が同じなら、振舞も似た構造となるものが多いと予想していた。中には生成やデータアクセスなど、目的が同じ故に振舞も似るパターンもあった。しかし振舞が似ていても、目的が異なるパターンも多く存在した。

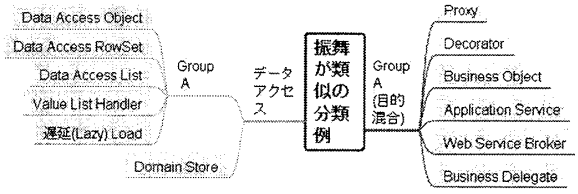

図 2 振舞の類似による分類（例）

3. ソフトウェアパターン定義言語の提案

2 章で分類したクラス構造や振舞を記述するために、パターンを定義する言語を考案した。

1 パターンにつき UML の図を 2 つ、XML 形式の言語で定義する。2 つの図とはクラス構造を表すクラス図と、振舞を表すシーケンス図である。

リスト 1, 2 がパターン定義言語の特徴である。

リスト 1　クラス図を定義する言語の特徴

```
<Diagram id="SimpleInterface" type="Class"
 target="SimpleInterface" domain="Prototype, Proxy,..."
 name="ClassDiagram_SimpleInterface">
  <Pattern id="SimpleInterfaceBase" target="SimpleInterface"
  domain="Prototype,... maxOccurs="unbounded">
    <GroupClass id="SimpleInterface" type=""
    domain="Proxy" multiplicity=""
    maxOccurs="unbounded">
      <Class id="Abstract" type="" domain="Proxy"
       name="Abstract" multiplicity="1"
      maxOccurs="unbounded"/>                    クラス
    </GroupClass>*
    <GroupAssociation id="SimpleInterface" type=""
     domain=" Proxy " multiplicity=""
    maxOccurs="unbounded">
      <Association id="Interface" type="Interface"
       domain=" Proxy" name="AssociationA"        関連性
      maxOccurs="unbounded">
        <Association.sourceEndClass Class.idref="Abstract"
         parent.idref="SimpleInterfaceBase" rrowhead=""
        multiplicity="1"/>
        <Association.targetEndClass
         Class.idref="SimpleInterface.Concrete"
         parent.idref="SimpleInterfaceBase" arrowhead="1"
        multiplicity="*"/>
      </Association>
    </GroupAssociation>
  </Pattern>
</Diagram>
```

リスト 2　シーケンス図を定義する言語の特徴

```
<Diagram id="TO" type="Sequence" target="TO"
domain="DTO,..." name="SequenceDiagram_TO">
  <Pattern id="TOBase" target="TO"
domain="DTO, Flyweight, ..." maxOccurs="unbounded">
    <GroupLifeline id="TOBase" Pattern.idref="Others..."
    parent.idref="OtherPattern" domain="DTO.."
    maxOccurs="unbounded">
      <Lifeline id="Client" Class.idref="Client"
      maxOccurs="unbounded"/>                    ライフライン
    </GroupLifeline>
    <GroupMessage no="1" id="Create" type="Create"
    target="Create" domain="" multiplicity="1"
    maxOccurs="unbounded">
      <Message no="1" id="ClientRequest" type="Create"
       domain=" DTO..." name="get Att" multiplicity="1"
      maxOccurs="unbounded">
        <Message.sender Lifeline.idref="Client"
        parent.idref=""/>
        <Message.receiver Lifeline.idref="Factory"
        parent.idref=""/>
        <Message.parameter id="Parameter" type="Attribute"
        domain="" name="" multiplicity="*"
        Class.idref="" maxOccurs="unbounded"/>
      </Message>                                 メッセージ
    </GroupMessage>
    <Fragment id="IfCashed" type="Alt" domain="a" name="
キャッシュされていない場合" maxOccurs="unbounded">
      <Message no="1" id="ClientRequest" ...
       maxOccurs="unbounded""/>                  条件分岐, 繰返, 他
    </Fragment>
  </Pattern>
</Diagram>
```

4. パターン定義言語の記述実験

3 章で定義した言語で、ソフトウェアパターンをどれくらい定義できるか実験した。GoF デザインパターン、POSA, J2EE パターン、EJB パターン、その他から約 80 パターンを集めた。そして全体の 60%にあたる、クラス図 40 ヶとシーケンス図 48 ヶを定義言語で記述した。残り 40%は未定義であり、定義言語に改善の余地が残った。

5. おわりに

本稿ではソフトウェアパターンを設計に利用しやすくするために、パターンの分析と分類を行った。分類はパターンの目的、クラス構造、振舞の 3 種毎である。

次にパターンのクラス構造や振舞を定義する言語を考案した。そしてパターン定義言語で 48 パターンを記述した。これによりパターン専用処理系で利用できる、48 ヶのデータが用意できた。

参考文献

[1] デザインパターン改訂版, Erick Gamma 他, 1999
[2] ソフトウェアアーキテクチャ, Frank Buschmann 他
[3] J2EE パターン第 2 版, Deepak Alur 他, 日経 BP, 2005
[4] J2EE デザインパターン, William Crawford, 2004
[5] EJB デザインパターン, Floyd Marinescu, 2003
[6] プログラムデザインのためのパターン言語, PLoPD Editors, 2001