1N－8

オブジェクト指向開発アプローチ Crossover (1) - 設計モデル -

岸 知二, 前川佳春, 池田健次郎, 野田夏子
NEC マイコンソフトウェア開発研究所

1 はじめに

各種のオブジェクト指向分析 / 設計技法が提案されているが、そこでは分析 / 設計モデルをレビュー等で確認しながら徐々にコード化するため、ソフトウェアを動作させるまで性能問題等が見過ごされることが多い。一方オブジェクト指向プログラミングはプロトタイプ開発に適しているが、基本的にボトムアップなアプローチであり、全体構造設計などの上位からの視点は不十分である。

2 Crossover アプローチ

我々は部品を用いて全体構造を早期にプロトタイピングしながら、徐々に詳細へとコード化していく開発アプローチ Crossover の検討を進めている。Crossover では以下のサイクルを繰り返しながら開発が進められる。

1. 部品を用いて設計モデルを構築する。
2. 設計モデルからコード生成をする。
3. 実行確認をする。

本アプローチはボトムアップなプロトタイピングとは違い、全体の構造定義から実際のコードを生成してしまい、その枠組みの中で徐々に定義を詳細化していくため、大きな問題が開発終了間際になるまで残る危険性を減らすことができる。

なお本アプローチはコード生成の機能 [2] や生成されたコードを実行確認する機能 [3] などを前提としたものとなっている。

"Object-Oriented Development Approach: Crossover (1) - Design Model -", Tomoji KISHI, Yoshiharu MAEGAWA, Kenjiro IKEDA and Natsuko NODA, NEC Corporation

3 設計モデル

3.1 要件

Crossover アプローチを実現するためには、以下の要件を満たす設計モデルが必要である。

- 再利用可能であること: ソフトウェアは OS やプラットフォームなどの既存資産の上に構築されるため、既存部分を前提として階層的にモデルを構築できなければならない。
- コード生成可能であること: 構築された設計モデルからは実行可能なコードが生成できなければならない。本設計モデルの実行意味は対象とする環境によって規定されることになる。
- 混在記述可能であること: 実行確認上重要な部分から優先してコードレベルまで詳細化する必要があるため、設計モデルとコードとの混在が可能でなければならない。

3.2 基本要素とインタフェース記述言語

設計モデルはクラス、モジュール、インスタンス間関係を基本要素として持つ。クラス定義、モジュール定義はインタフェースと実装から構成される。混在記述を認めるためクラスの実装は状態モデルのような抽象度の高い記述でも、コードでの記述でもよい。インピーダンスミスマッチを解決するため、クラスやモジュールのインタフェースはインタフェース記述言語で定義される。モジュールの実装はモジュールに含まれるクラス定義とモジュール定義の集合である [1]。

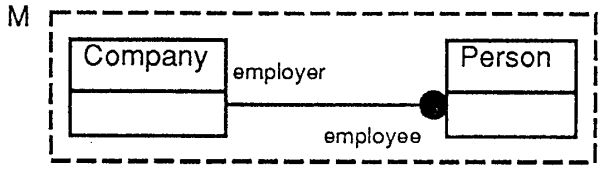

図 1: 設計モデルの例 (図式でのイメージ)

図 1のインタフェース記述例を以下に示す。

```
Module M {
    export Class Company ;
    export Class Person ;
  }
Class Company {
    relation Assoc<Person> [many] employer
    inverse Person::employee ;
  }
Class Person {
    relation Assoc<Company> [1] employee
    inverse Company::employer ;
  }
```

4 部品を用いた設計モデル

基本要素に対応して、クラス、モジュール、インスタンス間関係を部品化することができる。

4.1 クラス部品とモジュール部品

既存のクラス定義やモジュール定義を部品として参照できる。例えばクラスライブラリやフレームワークはモジュール部品に対応し、そのモジュールをインポートしてその中の抽象クラスを参照したり、具象クラスに対してアクセスすることで利用できる。

4.2 関係部品

関係自身は実装を持たず、その実現はクラスやモジュールの定義中に分散される。関係部品中ではその実現に必要なクラス部品やモジュール部品と、コード変換に必要な情報が管理される [2]。

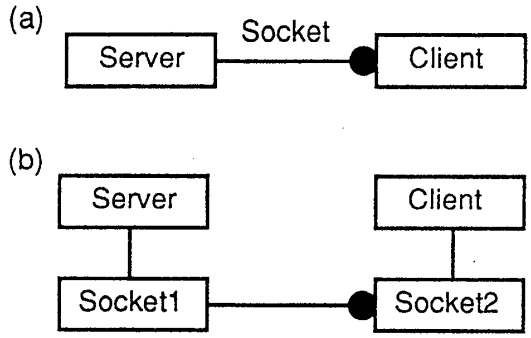

図 2: 関係部品の利用例

図 2の (a) では、 Server と Cleinet をソケットで対応づけることを関係部品 Socket を用いて示している。ソケットの存在を陽に示すと (b) のような表現になるが、プロセス間の 1: 多の関係を明示したい場合には (a) の表現の方が自然である。

(a) の実装例としては、相手側との関係はポート番号を用いて代替的に管理し、ソケット機能を利用して相手の識別や通信などを行う方法が考えられる。その場合にはポート番号を管理するクラス部品やソケット機能に対応するモジュール部品が必要となる。

5 おわりに

現在インタフェース記述言語を C++ にバインドしてコード生成機能などの試作を進めている。今後の設計モデルや各種部品の検討に反映させたい。

参考文献

[1] 岸知二, 他: オブジェクト指向設計における制御アーキテクチャの定義. ソフトウェア工学研究会 Vol.94, No.55, (1994).

[2] 野田夏子, 他: オブジェクト指向開発アプローチ Crossover (2) - 生成系 -. 情報処理学会第 49 回全国大会 (1994).

[3] 前川佳春, 他: オブジェクト指向開発アプローチ Crossover (3) - 実行確認系 -. 情報処理学会第 49 回全国大会 (1994).