オブジェクト指向分散環境 OZ のクラスの設計

1K－9　中川 祐 (富士ゼロックス情報システム)　　塚本 享治 (電子技術総合研究所)

1　はじめに

オブジェクト指向分散環境 OZ は分散環境におけるソフトウェアの開発と利用を容易にすることを目的としたオブジェクトの共有と交換に基づくシステムである。

このシステムは先に開発したオブジェクト指向分散環境 OZ++[1] の次世代型である。OZ++ 開発の経験を生かし、軽量で操作性のよいシステムを作ることを狙っている。

2　OZ++ のクラス構造

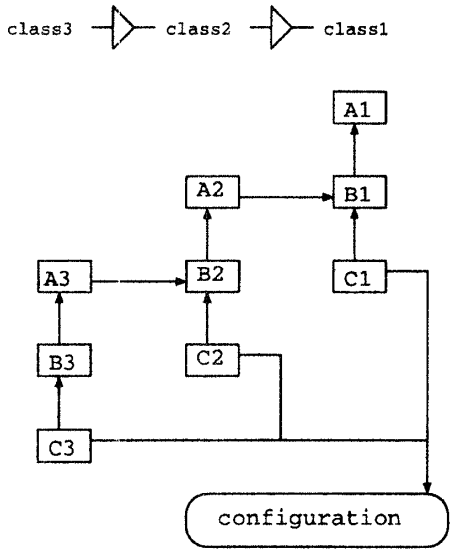

図 1: OZ++ のクラス構造

OZ++ の１つのクラスを構成するデータとその関連を図1に示す。このクラス、Class3 はスーパークラスに Class2、さらにそのスーパークラスに Class1 が存在する。

OZ++ の１つのクラスは３つのパートから成立する。A パートはパブリックパートとも呼ばれ、クラスの公開インタフェースを提供する。B パートはプロテクティドパートとも呼ばれ、サブクラスに対するインタフェースを提供する。C パートは実装パートとも呼ばれ、クラスの実行コードを提供する。A、B、C の順で A の側を上位、C の側を下位と呼ぶ。上位パートが修正された場合、下位パートも修正が必要である。つまり再コンパイルの必要がある。

各スーパークラスもそれぞれこの３つのパートが必要である。サブクラスに対するインタフェースは B パートであるため、Class3 の A パートはスーパークラスである Class2 の B パートに依存して作成される。同様に Class2 の A パートはそのスーパークラスである Class1 の B パートに依存して作成される。スーパークラスの B パート以上の修正はサブクラスの A パート以下の修正を意味する。

OZ++ でオブジェクトを生成するにはそのオブジェクトのクラスとすべての祖先クラスの C パートを決定しなければならない。また OZ++ ではパートのバージョン管理を行っているため、それぞれのクラスの C パートは複数存在しうる。従って Class3 のオブジェクトを生成するにはクラスの継承階層に従って、C1、C2、C3 のそれぞれを指定しなければならない。指定された C パートの集合をコンフィグレーションと呼ぶ。集合を決定してコンフィグレーションを作成する作業をコンフィギュアと呼ぶ。

3　OZ のクラス構造

3.1　設計

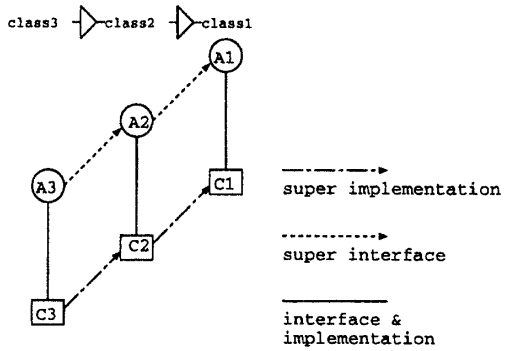

図 2: OZ のクラス構造

新たに OZ を設計するにあたりこの設計を変更することを考えた。これはまずデータの個数が膨大になり、メモリ使用量が大きくなること。さらに実際に開発作業を行ってみると操作しづらいという問題があり、より構造を簡略化する方向で検討を行った。

そこで B パートを廃止することを思いついた。従来の設計では確かにスーパークラスの C パートを変更してもサブクラスの再コンパイルが不要であった。ところ

Design of Class of OZ
An Object-Oriented Distributed Systems Environment
Yu Nakagawa(Fuji Xerox Information Systems, Co., Ltd.),
and Michiharu Tsukamoto(Electrotechnical Laboratory)

がこれはコンフィグレーションの変更を伴うため再コンフィギュアが必要である。従って開発者はBパートの存在の恩恵をあまり感じない。またスーパークラスというものがサブクラスにとってホワイトボックスであるべきという考え方を採用するならば、Bパートという存在自体に意味がない。さらにこの構造は複雑すぎてプログラマがデータ構造を空間的にイメージするのが困難である。

そこで図2に示す設計に至った。Class3のAパートはスーパークラスであるClass2のAパートから派生して作成する。Class3のCパートはClass2のCパートから派生する。Cパートを生成する際にすべての祖先クラスのCパートを決定するため、コンパイル時にクラスの実行コードすべてが決定することになる。そこでコンフィギュアという作業も必要なくなった。

3.2 Javaとの比較

この新しい設計をJavaのものと比較してみる。JavaではClassとInterfaceが独立し、それぞれが継承を行うことができる[2]。顧客から見たオブジェクトの型はClassかもしれないし、Interfaceであるかもしれない。この意味でClassというのは実行コードを持つことができる特別なInterfaceであるかのように思える。ところがInterfaceのスーパーインタフェースはInterfaceでなければならず、Classの下位型としてInterfaceを作ることはできない。これは不自然であり、かつプログラマにとって制約になると思われる。プログラマは型をInterfaceとして作るかClassにするのかを十分に考えて作成しなければならない。Classとして作成したものをInterfaceとして用いたい場合があるかもしれないが、それは不可能である。

JavaとOZのクラス設計を比較するとまずJavaではClassとInterfaceがその機能において重複する部分があるのに対して、OZのAパートとCパートは全く独立したデータである。またJavaのInterfaceは実行コードを持つことができないのに対して、OZのAパートは必ずCパートを持たなければならない。

ここでAパートは必ず対応するCパートを持つという規則は実は必要でなかったということに気が付いた。これはOZ++の悪い点を継承している。

3.3 再設計

そこで先ほどの設計を修正することとした。Aパートは必ずしも対応するCパートを必要とはしない。

するとClass3のCパートを作成するには様々な可能性が考えられる。まず従来通りスーパークラスであるClass2のCパートから派生することができる。Class2ではなくそのさらにスーパークラスであるClass1から派生させることもできる。全く新規に作るというのも可能である。さらに既に存在するClass3の別のCパートから作成するということも可能である。

AパートではなくCパートの継承というのは既存のコードを利用してその差分をコーディングするという意味しかない。利用するコードは自クラスまたはスーパークラスのインタフェースに対して作成されたものであるため、意味論的には不条理はない。

この方式ではJavaのようにClassとInterfaceという区別はない。1つのAパートをCパートを付け加えることによりClassとして利用することも、サービスの抽象であるInterfaceとして利用することもプログラマの自由である。

AパートとCパートに関する規則をまとめてみる。

- Aパートはクラスのインタフェースを提供する。
- Aパートは既存の1つ以上のAパートから派生して作成することができる。派生Aパートは派生元Aパートの下位型となる。
- Cパートは厳密に1つのAパートに対して作成され、Aパートが示すクラスの実行コードを提供する。
- Cパートは既存の1つのCパートから派生して作成することができる。それは以下の条件を満たさねばならない。

  派生元のCパートの上位パートであるAパートをA1、元のCパートの上位パートをA2とすると、A1はA2と一致するかA2がA1から派生したパートでなければならない。

4 まとめ

OZ++の設計を簡略化しJavaの設計を発展させたClass構造を設計した。現在これを実装中であり平成10年春の完成を目指している。

本研究は、情報処理振興事業協会(IPA)の「創造的ソフトウェア育成事業」の一環として行われたものである。

参考文献

[1] 塚本：「オブジェクト指向分散環境(OZ++)の研究開発」,開放型基盤ソフトウェア研究開発評価事業報告書,IPA,'96

[2] Arnold & Gosling:「The Java Programming Language」,
Addison-Wesley Publishing Company,'96