Ⅳ－64　交通流シミュレーションモデルにおけるクラス概念の導入について

北海道大学大学院　学生員　田頭　直樹
北海道大学工学部　正　員　中辻　隆
北海道大学工学部　正　員　藤原　隆

1．はじめに

近年交通量の増加に伴い、都市部における交通渋滞、大気汚染などが大きな問題となっており、それらに対する解決策として、交通流シミュレーションを用いて円滑な交通流を実現することが有効であると考えられている。

現在様々な機関で独自にモデルの構築が行われており、市販・公開されている交通流シミュレーションプログラムもある。それに伴い、交通流を表現するロジックも多種多様に存在している。しかし、それぞれが独立しているので、有効なモデル，ロジックの比較や過去のリソースの再利用といったことが困難となっている。

そこで、本稿では既存の交通流シミュレーションモデルが抱えている問題を指摘し、それらを改善するため、システム開発の新しい手法であるオブジェクト指向設計を可能にするC++言語のクラス概念を用いたモデル開発について考察する。

2．交通流シミュレーションモデル

(1) モデル化

現在新たに交通流シミュレーションモデルを構築しようとするならば、大きく分けて以下の3通りの方法が考えられる。

・自己開発
　Fortran，Basic，C言語などを用いて独自のアルゴリズムを構築する。
・既存プログラムの使用
　市販・公開されているプログラムに必要な初期値を代入しモデルを完成させる。
　「NETSIM」（米国連邦道路庁）など。
・汎用シミュレーション言語による開発
　シミュレーションシステムの構築を支援するツールを用いてモデルを作成する。
　「Witness」（AT&T　ISTEL）など。

(2) 問題点

さらにそれらを用いてモデルを作成した場合に生じる問題点とそれぞれに共通する問題点を以下に示す。

・自己開発
　－多大な労力と時間を要する。
・既存プログラムの使用
　－ロジックの内容がわからない。
　－仕様変更が出来ない。
・汎用シミュレーション言語による開発
　－交通流専門でないため、交通流特性の表現が困難である。
・共通の問題点
　－各プログラムの互換性がないため、ロジックの比較が出来ない。
　－開発者以外はプログラムを改良することが困難なため、プロジェクトとして成り立たない。

(3) 現状と解決策

手軽に、交通流をモデル化したいのであれば、既存プログラムを使用するか、汎用シミュレーション言語を用いたモデルの作成を試みればよい。しかし多くの場合、それらの方法で作成されたモデルがユーザーの要求をすべて満たすことは期待できず、ユーザーは自己開発に頼らざるを得ないが、(2) で述べたような問題は依然として残ったままである。

このような問題は、交通流シミュレーションに関する特有な問題ではなく、多くのシステム開発において

Object Oriented Design on Traffic Simulation Model
By Naoki Tagashira, Takashi Nakatsuji, Takashi Fujiwara

問題とされている。その解決策としてオブジェクト指向設計という手法があり、今現在最もオブジェクト指向設計を実現できるのはC++言語とされている。

3．オブジェクト指向設計

オブジェクト指向は、１９８０年代終わりに誕生した。８０年代は構造化プログラミングに端を発した構造化アプローチが大流行した時代であったが、従来のアプローチでは解決できない問題点もいくつかあり、これらを解決するために、いわゆるオブジェクト指向という新しい手法が開発された。

オブジェクト指向の４つの基本概念とそれぞれがもたらす効果を示す。

（1）抽象化

オブジェクト指向アプローチはデータ（属性）と手続き（メソッド）をひとまとめにした、モノ（オブジェクト）によって構成されている。問題の重要な側面もしくは注目したい問題の側面のみを強調してモデル化する。このことを「抽象化」という。また、現実世界のモノをオブジェクトとして抽象化するという意味の他に、同じ性質を持つオブジェクト群をさらに「クラス」として抽象化し、そしてクラス群をもう一段上のさらに抽象的なクラスとして抽象化する、という意味もある。

モノの構造は、もっとも安定した側面であるので、局所的な変更が可能となる場合が多く、モノ中心に思考単位を変換することによりシステム自体が実世界と類似することになり、設計が容易になる。さらに、分析の焦点もより明確化される。

（2）カプセル化

オブジェクト指向アプローチは、データと手続きを一体化したオブジェクトを単位としてシステムを開発する。この際、オブジェクトが保有するデータに外部からアクセスする場合は、手続きを通してのみアクセスできるようにする。これを「カプセル化」という。

カプセル化により、オブジェクトの利用者はオブジェクトに定義された手続きの方法さえ知っていれば、オブジェクト内部のデータ構造あるいは手続きの実装を知らなくてもよい。逆に、オブジェクトの仕様と実装の分離は、オブジェクトの提供者がオブジェクト内部の変更、例えば内部データ構造や手続きの実装の変更を行ってもその影響が利用者に波及しないので、変更が容易になる。

（3）継承

従来の言語では、個々の構成要素（モジュール）をそのままの形式で再利用することはできたが、その一部をわずかに変えたり拡張すれば、再利用は難しかった。そのために、既存のものとわずかに異なるだけでも、一から作りなおさなければならなく、重複開発が行われ類似した機能を持つシステムが数多く作られる結果となった。これらの問題を解決するために、オブジェクト指向アプローチでは、上位クラスの概念を下位クラスが引き継ぐと共に下位クラスでは上位クラスに存在しない新たな性質が追加できるようなメカニズムを用いている。このことを、「継承」という。

継承というメカニズムを取り入れることにより、クラスを体系化でき、新たなクラスの定義が容易に行える。また、既存のシステムを拡張する際、既存のコードと拡張した部分のコードを分離できる。具体的にいえば、拡張部分を記述した下位クラスを作り試してみる。もし問題があれば、拡張部分をさらに修正するかもしくは、拡張部分と関連してくるオリジナルの上位のクラスを修正する、などの試行錯誤が容易に行える。

（4）状態

オブジェクト指向アプローチでは、オブジェクトに定義されている手続き群を利用できるかどうかは、特定の内部データにより判断される「状態」によって決まる。さらに、手続きの実行による状態の遷移を表現することで、提供者は、利用者に対して確実に正しい利用手順を示すことができる。

提供者：取り決め以外の場合のオブジェクトの正しい手続きを保証しなくてもよい。
利用者：取り決めに従った場合のオブジェクトの正しい手続きが保証される。

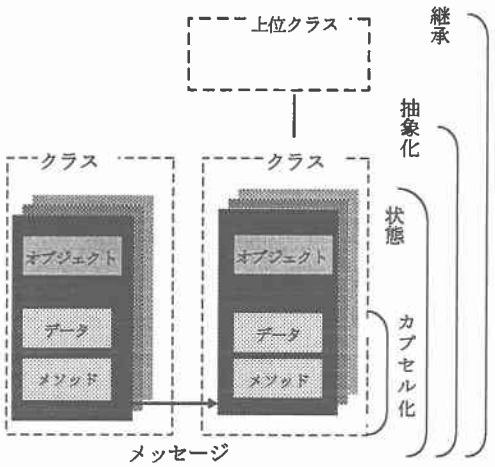

図1　オブジェクト指向の基本構成

4．クラス概念

3．においてオブジェクト指向設計の4つの基本概念を説明したが、それらを用いて実際にプログラミングするのであれば、図1に示されているような「クラス」と言われるものを定義しなければならない。ここでは、C++で用いられるクラスを説明することにする。

（1）オブジェクトとクラス

オブジェクト指向プログラミングでは、オブジェクトと呼ばれる相互作用する部分により構成され、同一の構造を持つ場合がある。そのような場合、一度オブジェクトの構造をクラスで決定し、後に必要な数だけ複製する。このように、クラスは特定のオブジェクトを作成するひな形と考えることが出来る。クラスにより生成されたオブジェクトを、そのクラスのインスタンスと呼ぶ。

（2）クラスの構成

クラスは、属性を表すデータメンバとそれを操作する手続きを定義したメンバ関数とを一体にした型になっている。一般にデータメンバを外部に対して非公開にして、メンバ関数を公開する。こうすることにより、外部からはメンバ関数を介さないとデータメンバを参照することは出来ず、勝手なデータメンバの変更，使用を防ぐことが出来る。

（3）交通流クラスの定義

ここで、実際にクラスを交通流に対して定義した場合の具体例を示す。交通流シミュレーションで必要となってくる要素を、車両，信号機，道路などといった具合に『モノ』に着目し分類し、それぞれに属性と手続きを定義する。さらにそこで定義した属性と手続きの変更や追加が必要であれば、そのクラスから派生した下位クラスを作成する。

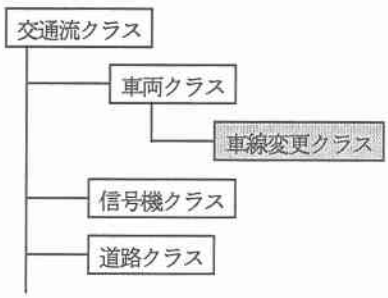

図2　交通流クラス

図2の車両クラスのデータメンバ（属性）とメンバ関数（手続き）を次のように宣言する。

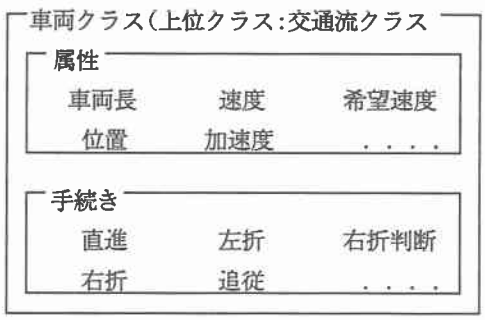

図3　車両クラスの宣言

ここで、車両が車線変更する手続きを定義したい場合、車両クラスから派生した車線変更クラスを宣言する。この場合、車線変更クラスで宣言されたオブジェクト（インスタンス）は、上位クラスである交通流クラスの属性，手続きをすべて継承しているので、新たに追加するメンバのみを記述すれば良い。また、上位クラスの実装部分のプログラムの内容を知らなくても良い。

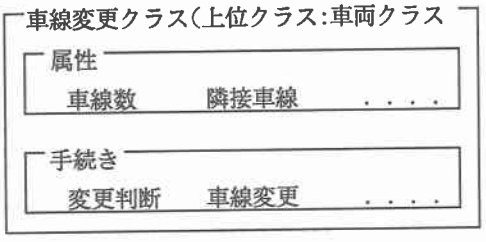

図４　車線変更クラス

（4）クラスの使用例

```
//ファイル vehicle.h
//CVehicle クラスのヘッダファイル
class CVehicle{
    double x, y;          ┐ データメンバの宣言
    ..............        ┘ (非公開部分)
    public:
    void LTurn();         ┐ メンバ関数の宣言
    ..............        ┘ (公開部分)
};
//ファイル vehicle.cpp
//CVehicle クラスのソースファイル
#include "Vehicle.h"
void CVehicle :: LTurn(int st)
{
    if (st=1){
        x = t / 2 *cos(theta);
        y = t / 2 * sin(theta);
    }
}
```

リスト１　車両クラスに関するコード

```
#include <iostrem.h>
#include "vehicle.h"
main()
{
.....
    vehicle integra;
    integra.LTurn();
.....
}
```

リスト２　シミュレーションに関するコード

スーパーユーザーがクラスの宣言，定義（リスト１）を作成し提供することにより、一般ユーザーはクラスの実装を知らなくても、メインプログラム（リスト２）さえプログラミングできれば、シミュレーションプログラムを作成できる。

５．結果

交通流クラスを定義することにより、交通流の多種多様な要素を分類・規定することができ、交通流の基本的なアーキテクチャを設定することができる。

従来の手法を用いたプログラミングでは、以前に作られたプログラムを再利用するのであれば、ユーザーはその内容をすべて把握しなければならず、変更・追加をするのは大変困難な作業であったが、属性と手続きを分離することにより、一般ユーザーは、クラスの実装部分のプログラムを理解しなくても、交通流シミュレーションモデルの構築ができ、逆にクラスの知識があるスーパーユーザーは、新しく交通流のクラスを追加・訂正することができ、容易に仕様の変更などが行える。

さらに、追従理論の定義だけが異なるモデルなどが容易に作成できるため、ロジックの比較，検討や、リソースの再利用が容易なため複数の開発者によるモデルの改良が可能となる。

[参考文献]

1）本位田 真一，山城 明宏：オブジェクト指向システム開発，日経ＢＰセンター

2）N.グラハム：ラーニング C++，海文堂

3）桜田 幸嗣，田口 景介：Visual C++4.0 プログラミング入門，アスキー出版

4）河西 朝雄：Microsoft VISUAL C++ 初級プログラミング入門 [上] [下]，技術評論社