Transportmetrica, Vol. 4, No. 3 (2008), 227-247

DEVELOPMENT OF MICROSCOPIC SIMULATION MODEL FOR HETEROGENEOUS TRAFFIC USING OBJECT ORIENTED APPROACH

K. VENKATESAN[1], A. GOWRI[2] AND R. SIVANANDAN[3]

Received 10 January 2008; received in revised form 22 March 2008; accepted 10 April 2008

Roads in India carry heterogeneous traffic with wide variations in static and dynamic characteristics of vehicles. The traffic flow is also generally devoid of lane discipline, with vehicles occupying any available road space ahead. Modeling of such heterogeneous traffic is generally complex in nature and simulation has been proven to be effective in studying such traffic. This paper presents the development of a model through object oriented programming (OOP) approach for simulating heterogeneous traffic. Several unique traffic characteristics have been considered in the modeling process. C++ language is used for this purpose. OOP leads to better structured codes for simulation and facilitates the development, maintainability and expandability of such codes. The first part of the paper explains the various logics of simulation such as vehicle generation, vehicle placement and vehicle movement. The second part of the paper describes the features of OOP, description of classes and class relationships. The last part of the paper presents the validation and advantages of the developed model.

KEYWORDS: Heterogeneity, traffic flow, simulation, object-oriented languages, computer programming

1. INTRODUCTION

The traffic in developed countries is homogeneous, which comprises mostly passenger cars and follows lane discipline. In India, traffic on roads is heterogeneous in nature with wide variations in physical dimensions and other vehicular and traffic characteristics. Heterogeneous traffic includes motorized vehicles such as motorized two wheelers (MTW), cars (including jeeps and small vans), buses, auto-rickshaws (three-wheeled motorized vehicles), light commercial vehicles (LCV), trucks, and non-motorized vehicles such as bicycles and tricycles, which share the common road space without any physical segregation. Speeds vary from about 5 kmph in the case of bicycles to about 60 kmph or more for passenger cars. Physical dimensions also vary significantly for different types of vehicles (lengths varying from about 1.5 to 10.0 m and widths 0.5 to 2.75 m). Due to wide variations of speed and physical dimensions it is difficult to impose lane discipline in heterogeneous traffic conditions, so vehicles occupy positions on any part of the road based on space availability. Small sized vehicles (MTW, bicycle) often use gaps (lateral and longitudinal) between larger vehicles in the stream to weave through traffic. These characteristics of heterogeneous traffic flow result in complex interactions among vehicles.

Modeling traffic flow is stochastic in nature due to randomness in variables such as vehicle arrivals and speeds. Due to this and due to complex vehicular interactions and their manoeuvres, it is extremely difficult to model the traffic flow through analytical methods. To study this type of complex traffic system and vehicle interactions, simulation is considered as an effective tool. Many traffic simulation models are available for homogeneous traffic, where most of the vehicles are passenger cars with

[1] Transportation Engineering Division, Dept. of Civil Engineering, Indian Institute of Technology Madras, Chennai 600036, India. Corresponding author (E-mail: venkatesankanagaraj@yahoo.co.in).

[2] Transportation Engineering Division, Dept. of Civil Engineering, Indian Institute of Technology Madras, Chennai 600036, India.

[3] Transportation Engineering Division, Dept. of Civil Engineering, Indian Institute of Technology Madras, Chennai 600036, India.

similar characteristics. Application of these models to heterogeneous traffic may not be able to capture the complex manoeuvres and interactions in such flows. Hence, the interest here is on developing a heterogeneous traffic simulation model using object oriented concepts. In recent years, OOP has gained popularity in the development of scientific codes (Fenves, 1990; Lee and Arora, 1991). OOP allows the programmer to produce more logical, maintainable and expandable codes with less effort and expense due to its powerful and flexible characteristics (Ross et al., 1992).

2. LITERATURE REVIEW

The object oriented programming (OOP) is a major shift from traditional method of software construction. In OOP, a system can be viewed as a collection of entities that interact together to accomplish certain objectives. The OOP technique offers a highly sophisticated versatile programming environment for the development of large scale and/or complex software system because of its inherent features of encapsulation, inheritance and polymorphism (Booch, 1994). The benefits of OOP are easy maintenance, enhanced modification and reusability of the programme. In addition, OOP facilitates bridging the gap between the real world system and the simulation model. Few attempts have been made related to the present study under homogenous traffic conditions using OOP concepts (Tsukamoto et al., 1994; Ben-Akiva et al., 1997; Wong et al., 1998; Ijaha et al., 2000; Li et al. 2004).

Commercial softwares such as VISSIM, PARAMICS, NETSIM, FRESIM, INTRAS, CORSIM and INTELSIM have been developed to simulate homogeneous traffic characteristics (Aycin and Benekohal, 2001; Chandrasekar et al., 2002; Fang and Elefteriadou, 2005). These softwares are not believed to be appropriate for studying heterogeneous traffic, due to wide variations in flow characteristics of heterogeneous traffic vis-à-vis homogeneous traffic. Vissim software can't do non lane-based simulation, non lane-based vehicle entries and simulation of undisciplined intersection very well. The developed model has the capability of incorporating the above features. The implementation details of Vissim are proprietary; it is hard to say exactly how it works. But, the developed model has the advantage of being open source and extensible. Though comprehensive commercial software is not yet available for studying heterogeneous traffic, attempts have been made by researchers to model various aspects of such traffic. Popat et al. (1989) developed a simulation model for uncontrolled T-Intersection using FORTRAN language. Murugesan et al. (1991) implemented a simulation model in GPSS to design the signal settings appropriate for saturated intersection. Raghavachari et al. (1993) proposed a simulation model, PEDVIM in FORTRAN language to study the interaction between pedestrians and vehicles at unsignalised urban intersection. Agarwal et al. (1994) made an attempt to simulate the mixed traffic flow of four legged, right angled uncontrolled intersection using FORTRAN language. Kumar and Rao (1996a) developed a simulation model to study the traffic flow and driver characteristics at a yield controlled intersection, using FORTRAN language. Kumar and Rao (1996b) designed a SIMJORT-KGP model in FORTRAN language for analysis of traffic operation on two lane highways. Rao and Rengaraju (1998) described a methodology of simulating the traffic flow to estimate the number of conflicts at uncontrolled intersections in C language. Chandra and Parida (2004) developed heterogeneous traffic simulation models for multilane urban roads using C language. Arasan and Koshy (2005) developed a simulation model for comprehensive study of heterogeneous traffic using C++ language. Lan and Chang

(2005) developed inhomogeneous cellular automata models to explain the interacting movements of cars and motorcycles in mixed traffic contexts. Bhavsar et al. (2007) simulated bus priority system for an urban corridor in Mumbai city using VISSIM software.

A related literature on lane change model was found, which explained a tactical lane change model for representing the driver behaviors of anticipation and sequential planning of lane change maneuvers in a traffic simulator (Webster et al., 2008). Car following models are microscopic models as they describe the behaviour of individual drivers when interacting with vehicles in their proximity. The classical model (General Motors theory) was represented by the following equation (May, 1990):

$$a_{n+1}(t+\Delta t) = \alpha_n \frac{[v_n(t) - v_{n+1}(t)]^k}{[x_n(t) - x_{n+1}(t)]^m}\,, \tag{1}$$

where $a_{n+1}(t+\Delta t)$ is the acceleration or deceleration rate of the following vehicle $n+1$ at time $t+\Delta t$, α_n is the sensitivity parameter, $v_{n+1}(t)$ is the speed of the following vehicle at time t, $x_{n+1}(t)$ is the position of the following vehicle at time t, and k and m are model parameters. For the last few decades, car following models were developed based on various theories. Gipps (1981) developed a new model for the response of the following vehicle based on the assumption that each driver sets limits to his desired braking and acceleration rates. Helbing and Tilch (1998) developed a generalized force model to manage situations like vehicles approaching standing cars while other models produce accidents. Jiang et al. (2001) developed a full velocity difference model for a car-following theory based on the previous optimal velocity model and generalized force model. This model predicts correct delay time of car motion and kinematic wave speed at jam density. Spyropoulou (2007) discussed the possible additions to the Gipps' car following model to simulate urban networks comprising signal-controlled junctions in an efficient way.

From the above review, it is seen that most of the heterogeneous traffic simulation models have been developed using procedural languages, such as FORTRAN. FORTRAN is an important tool for development of scientific and engineering computational codes. However, it has its own limitations: maintenance of programming is time consuming and difficult due to coupling among the procedures. Moreover, from the previous studies it is revealed that the models are developed for some specific traffic analysis and it is difficult to extend for newer applications and if they are extended, it would involve considerable time, effort and rewriting and debugging code. Only a little attempt has been made to implement heterogeneous traffic simulation model using C++. Unfortunately, very little information about the implementation of heterogeneous traffic simulation in object oriented environment is published in the literature and also, description of classes and class relationships are not explained in greater detail. The implementation and use of advanced features of OOP for heterogeneous traffic simulation are still limited to research applications. Therefore, the main objective of this work is to describe an approach to the design and implementation of a heterogeneous traffic simulation model using object oriented architecture.

This paper presents an OOP approach to develop a heterogeneous traffic simulation model using C++ language. A sample code is discussed in detail to demonstrate the implementation of OOP features, such as encapsulation, inheritance and polymorphism. Reusability, expandability and maintainability of code are also explained in the context of the simulation model.

3. DEVELOPMENT OF HETEROGENEOUS TRAFFIC SIMULATION MODEL

On the most Indian roads, vehicles move freely based on availability of space and ignore lane discipline. Smaller vehicles often weave through gaps between larger vehicles. Such features of traffic flow and wide variations in vehicular characteristics have been incorporated in the developed simulation model. In this model, the entire road space is considered as a single unit instead of individual lanes. The vehicles are represented as rectangular blocks, with dimensions, occupying a specified area of road space whose positions are defined by co-ordinates. This facilitates to mimic the real-life behaviour of vehicles in their placement and movement, without lane discipline. The simulation model is developed based on interval scanning technique with fixed increment time advance. Warmup time and warmup zones are included to eliminate the transient state of the system. Traffic statistics are collected once the system reaches steady state. Mid block section of a road stretch is considered in the simulation model. Figure 1 shows the overall framework of the simulation model.

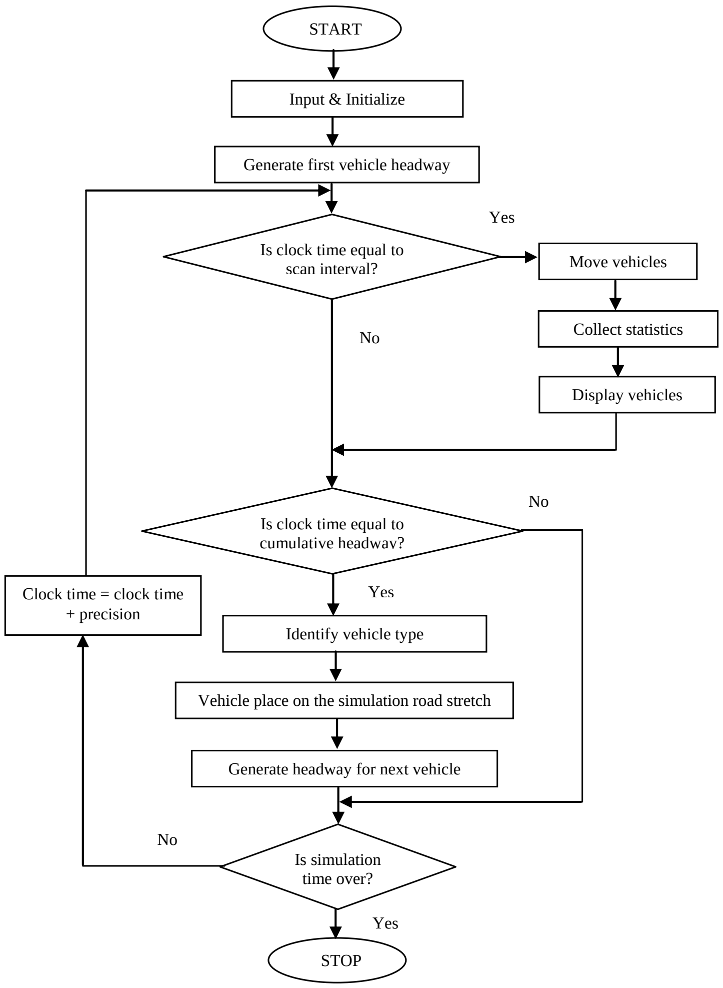
START
Input & Initialize
Generate first vehicle headway
Is clock time equal to scan interval?
Yes
Move vehicles
Collect statistics
Display vehicles
No
Is clock time equal to cumulative headwav?
No
Yes
Clock time = clock time + precision
Identify vehicle type
Vehicle place on the simulation road stretch
Generate headway for next vehicle
No
Is simulation time over?
Yes
STOP

FIGURE 1: General structure of simulation model

The inputs and initialization parameters are given to the model. When simulation clock time equals cumulative headway, vehicle is generated and placed on the road stretch. Then the simulation clock time is updated with precision of 0.05 s. At each time scan interval (1 s), the position of vehicle is updated, traffic statistics are collected and animation of traffic flow is displayed. The model shows headways of all the vehicles, average speeds of each category of vehicles, concentration (number of vehicles present on a given length of road section) and number of vehicles generated and their types as output. The proposed model contains the following three sequential steps:

3.1 Vehicle generation

Traffic process is stochastic in nature as vehicle arrivals are associated with randomness. Random numbers are used for generation of vehicle arrivals, identification of vehicle type and assignment of free speed of vehicles in the simulation program. Vehicles are generated at the starting point of the simulation road stretch according to some headway distribution. Headway distribution models developed for homogenous traffic conditions cannot be applied directly to heterogeneous traffic situations. The occurrences of zero headway are nearly impossible for homogenous traffic conditions because the observations are made on a specific lane, since the vehicles themselves are having a specific length. Whereas, under heterogeneous traffic conditions, smaller vehicles like two wheelers move in parallel even in a single lane, which create significant number of zero headways. Negative exponential distribution admits headways which are zero or close to zero. Headway observations made on entire width of road instead of lane wise which is appropriate for heterogeneous traffic conditions due to the absence of lane discipline.

Headways are generated based on the negative exponential distribution using the following expression

$$H = -\,(1/\lambda)\ln R\,, \tag{2}$$

where H is the headway or time interval between arrival of successive vehicles (*s*), R is the random number in the range (0 to 1), and λ is the mean arrival rate of the vehicles (vehicles per second). Whenever a vehicle is generated, the associated headway is added to all previous headways to obtain the cumulative headway. When simulation clock time equals to the cumulative headway, the vehicle is placed on the simulation road stretch and before the placement of vehicle, vehicle type and free speed are determined. Vehicle type is identified based on the composition of traffic. For this purpose, a random number is generated and checked with the range of numbers reflecting a particular category of vehicle based on the known composition. After that, generated vehicle is assigned with free speed. The free speeds of vehicles follow normal distribution. Hence, the standard normal deviates are generated using Box-Muller transformation method by using the equation

$$Z = (-\ln 2R_1)^{1/2}\,2\cos(\pi R_2)^2\,, \tag{3}$$

where Z is the standard normal deviate, and R_1, R_2 are the random numbers. The normal deviate (X) can be obtained as

$$X = \mu + \sigma Z\,, \tag{4}$$

where μ is the specified mean and σ is the standard deviation. X is assigned as the free speed of the vehicle. The speed of vehicles on simulation road stretch is based on two assumptions: (a) vehicle speeds will not be allowed to exceed their free speeds in the entire stretch, and (b) the vehicles are entering the simulation stretch at their free speeds.

3.2 Vehicle placement

Vehicle placement is based on the availability of transverse and longitudinal spacing on the road stretch. Motorized vehicles are placed from right edge to left edge of the road stretch, since the vehicles can move more freely and faster nearer to the median (in India, vehicles move on the left side of the road). Non-motorized vehicles are placed near the left edge of the road stretch (restricted to 2 m based on field observations); as the speeds of these vehicles are less compared to motorized vehicles, they are expected to use the left edge of the road (In homogeneous traffic conditions, the vehicles are placed only on appropriate lane. The above cited situations are unique for heterogeneous traffic conditions). Longitudinal and transverse spacings of vehicles are determined based on their current speeds. A motorized vehicle first looks for availability of longitudinal and transverse space in the right most section of the road stretch. If space to accommodate the vehicle is inadequate, it looks for similar spaces towards the left. Thus, longitudinal and transverse spacings are examined progressively from right edge to left edge of the road stretch. If such spaces are not available, the speed of the subject vehicle is reduced to that of its leader (car following rule). Again, similar checks for spaces are made, beginning from right most edge. If the available spaces are still insufficient, the vehicle is put in queue and it will be placed in the next scan interval. For non-motorized vehicles, similar checks are made from the left edge of the road (within 2 m).

3.3. Vehicle movement

In this simulation model, vehicle accelerates up to its free speed or posted speed limit of the road stretch, whichever is the minimum, if there is no slow vehicle in front of it. The position of vehicle is updated based on the following equations of motion:

$$S = ut + \frac{1}{2}at^2 , \qquad (5)$$

and

$$v = u + at , \qquad (6)$$

where S is the distance moved by the vehicle (m), u is the initial speed of the vehicle (m/s), a is the acceleration of the vehicle (m/s^2), t is the scan interval (s), and v is the speed of the vehicle (m/s) at the end of the scan interval. If the initial speed of the vehicle is equal to its free speed, then the distance S traveled by the vehicle in meters can be obtained using the following expression, since a is 0 for a vehicle traveling at its free speed.

$$S = u_f t , \qquad (7)$$

where u_f is the free speed (m/s), and t is the scan interval (s). When there is a slow moving vehicle in front of the subject vehicle, overtaking logic is invoked. During this stage, the subject vehicle (fast moving vehicle) checks for the free longitudinal and transverse spacings available on the right and left sides of the vehicle in front (slow moving vehicle). An overtaking manoeuvre is performed, if the spacing is found to be sufficient that is at least equal to the sum of movable distance of the vehicle intending to overtake and the corresponding minimum longitudinal spacing in the longitudinal direction and the minimum required lateral spacing in the transverse direction. Left or right overtaking is performed based on the position of centre line of overtaking vehicle. If the centre line of the overtaking vehicle is on the right side of the centre line of the overtaken vehicle, then the overtaking vehicle looks for availability of transverse and longitudinal spaces on the right side of the overtaken vehicle**.** If spaces are adequate on the right side, right overtaking is performed; if not, the overtaking vehicle looks for availability of such spaces on the left side, and if available, left overtaking is performed. A similar overtaking process is applied for the vehicle whose centre line is on the left side of the centre line of the overtaken vehicle. In this overtaking process, the overtaking vehicle attempt lateral motion to come parallel to the overtaken vehicle. This process takes cumulatively several scan intervals. The time taken for this lateral motion depends on the current vehicle speed, type of the vehicle and its maximum acceleration rate. The left and right overtaking manoeuvres are shown in Figure 2.

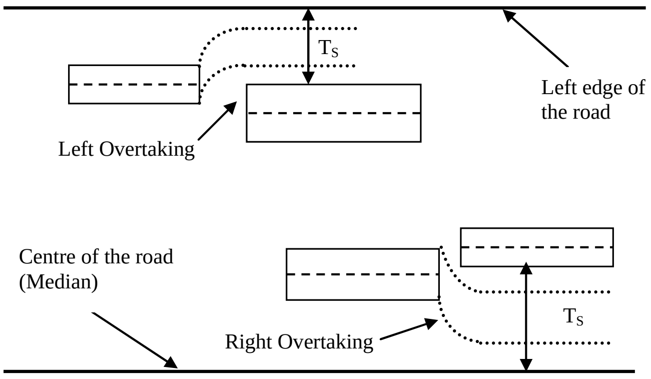

FIGURE 2: Left and right overtaking manoeuvres

The minimum required transverse spacing (T_s) for the overtaking vehicle is the sum of (a) width of the overtaking vehicle and (b) left side and right side clearances of the overtaking vehicle. Clearance is that any vehicle should maintain adequate transverse clearances in the traffic stream on the left and right sides with respect to other vehicles/curb/median to avoid collision. The vehicle performs the overtaking (passing) manoeuvre only if the overall transverse spacing is at least equal to $T_{s.}$ If transverse spacing is inadequate on both sides, overtaking is not performed and car following logic is invoked. Figure 3 illustrates the car following logic.

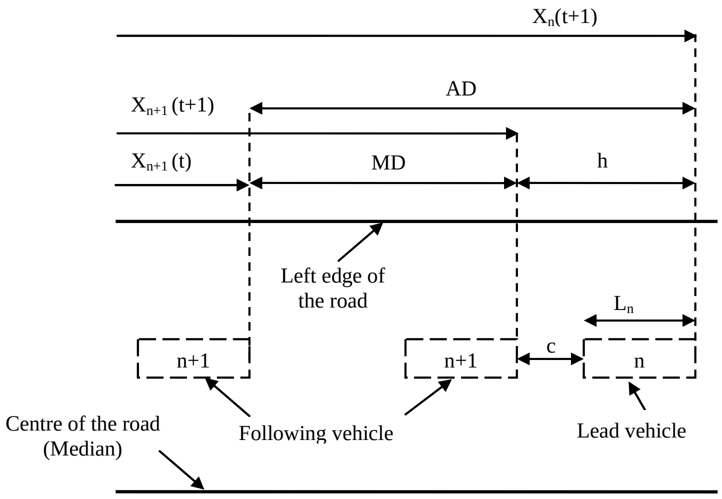

FIGURE 3: Car following logic

In this case, the speed of the following vehicle is reduced to that of lead vehicle, maintaining a safe spacing from it. The available distance (AD) in meters, is obtained as

$$\mathrm{AD} = x_n(t+1) - x_{n+1}(t)\,, \tag{8}$$

where $x_n(t+1)$ is updated position of the lead vehicle (m) and $x_{n+1}(t)$ is the position of the following vehicle before updation (m). The movable distance of the following vehicle (MD) in meters is calculated during each scan interval by the following equation

$$\mathrm{MD} = u_{\mathrm{F}}t + \frac{1}{2}at^2\,, \tag{9}$$

where u_{F} is the speed of the following vehicle at the beginning of the scan interval (m/s), a is the acceleration rate of the following vehicle (m/s^2), and t is the scan interval (s). The following vehicle will move freely if AD must be greater than or equal to distance (MD + safe space headway); otherwise, the following vehicle reduces its MD equal to the available distance minus safe space headway. In this case, the following vehicle follows the lead vehicle by decelerating to a speed equal to the speed of the lead vehicle ($V_{n+1} = V_n$) and it maintains the minimum safe space headway. The minimum safe space headway is calculated as

$$h = (c + L_n)\,, \tag{10}$$

where c is the clear longitudinal spacing between back of the lead vehicle and the front of the following vehicle (m), and L_n is the length of the lead vehicle (m). The c value is calculated by multiplying the reaction time of driver by the reduced speed of the following vehicle.

The logics of vehicle generation, vehicle placement and vehicle movement are amalgamated into one unit to simulate the movement of traffic on a road stretch. These logics are programmed coherently in OOP to develop a simulation model.

4. OOP FOR HETEROGENEOUS TRAFFIC SIMULATION

A program is a list of commands for performing some tasks on a given data. In conventional procedural approach, the emphasis is more on procedure and the data is taken as auxiliary factor. In procedural languages like C, PASCAL and FORTRAN, data and functions are not formally linked. So, it does not model the real world problems very well. In large programs, there are many functions and global data. Hence, there are large number of potential connections between data and functions, which lead to difficulty in conceptualizing and modifying the program (Lafore, 1995). But, on the other hand, in OOP approach, the data and the functions that operate on those data are combined into a single unit (object) and it allows decomposition of problems into number of objects. These objects closely resemble real world objects. Hence, OOP models are most suited for real world problems. The concepts of OOP and their similarities to the heterogeneous traffic simulation system are discussed below.

4.1 OOP concepts

4.1.1 Object

An object is the actual instance of a class that represents some real world entity. Objects are characterized by three properties - state, operations and identity. For example, object of class *MTW* (in the context of heterogeneous traffic simulation) has an internal state (position, speed, acceleration); it reacts to certain operations (movement, overtaking, car following); it has an identity that is different from other objects of *MTW* class. In the object oriented approach, for heterogeneous traffic simulation, objects of the classes: *MTW*, *Car*, *Auto-rickshaw*, *Bus*, *Truck*, *LCV*, *Bicycle*, *Link* and *Traffic* are the basic objects.

4.1.2 Class

A class represents a plane or a template for several objects and describes how these objects are structured internally. A class contains properties (data) and procedures (functions), which are applicable for all objects of that class. Since each object has own separate data items, separate data memory is allocated for the data items of each object (excluding static member data explained later). But, the member functions of the class are stored in memory only once, since member functions are identical for all objects of that class. In the context of traffic simulation, the vehicles are represented as objects of classes *MTW*, *Car*, *Bus*, etc. Figure 4 shows the description of *MTW* class. The properties of MTW, such as *length*, *width*, *mean free speed*, *standard deviation of free speed*, etc. are declared as private member data in *MTW* class. Only member functions or permissible actions of the class *MTW* (*freespeed ( )*, *fixacceleration ( )*) can access the private data variables of that object.

4.1.3 Encapsulation

Encapsulation is the property of OOP which enables bundling of related data and functionality within a single, autonomous entity (object). For example, each object of the *Bicycle* class encapsulates member data and functions of that class. The data items of the objects cannot be accessed directly; they can be accessed only by the member functions of the objects. Hence, the data is safe and hidden from accidental manipulation. In C++, this encapsulation protection mechanism is implemented using access specifiers, such as public, private and protected keywords. Public members can be accessed anywhere in the programme, whereas private members can be accessed only within the class. Protected members can be accessed within the class and subclasses. Generally, data within a class are private and functions are public and there is no rigid rule that data must be private and functions must be public. The access to attributes and methods associated with an object is permitted only through well defined public interface.

4.1.4 Inheritance

Inheritance is one of the most powerful features of OOP. Inheritance means that one class is a specialized version of another class (general class) with most operations identical or similar, but with few differences. Hence, specialized class (subclass) is inherited from the general class (superclass). A subclass inherits all the properties or data and procedures or methods from the superclass. For example, in heterogeneous traffic simulation model, classes such as *MTW*, *Car*, *Bus*, *LCV*, *Auto-rickshaw*, *Truck*, and *Bicycle* are derived from *Vehicle* class as shown in Figure 5. *MTW* class declaration (Figure 4) shown in the form of class *MTW*: *public Vehicle* indicates *MTW* class is derived from *Vehicle* class. The *MTW* class inherits member data (*headway, coordinates of vehicles* and *longitudinal spacing*) and member functions (*overtaking ( ), carfollowing ( )* and *updatespeed ( )*) from the base class *Vehicle* and also it has its own unique properties (*length*, *width*) and functions, as shown in Figure 4. In *Vehicle* class, the member data is declared as protected keyword; hence, these data can be accessed by member functions of a base class or any class derived from its own class. Inheritance allows a hierarchical organization of classes that eliminates duplication effect and allows sharing and reusing of data and methods.

```
enum Answer {no, yes};      //enum type
enum Done {notdone, yesdone};
enum Space {notenough, enough, notwithinrange};
enum Type {left, right};

//--------------------------------------------------------------------------------
class Vehicle             //Vehicle class
{
protected:
float headway;
float longitudinal;      //Vehicle member data
float lateral;
float free_speed;
float current_speed;
float clearance;
float acceleration;
float xll;                  //Vehicle coordinates
float xml;
float xul;
float yll;
float yul;
float centre_line;
float ptime;
public:
Vehicle ()            //Constructor
      { }
virtual ~Vehicle()   //Virtual destructor
    { }
virtual void freespeed (int) =0;      //Pure virtual functions
virtual void xclearance ()=0;
virtual void updateposition(float)=0;
virtual void show (float, float, int) =0;
virtual char type () =0;
virtual void fixacceleration () =0;
virtual void intialization (float, float)=0;
virtual Answer motorized ();        //Virtual functions
virtual Answer fix_y_coordinate (float, float,float);
virtual float get_position (Answer);
void fix_x_coordinate (float);
Space check_space (Vehicle **, float, int, int);
float updatespeed ();                   //Vehicle performance functions
Type overtaking(Vehicle**);
Answer leftovertaking (Vehicle**, float);
Answer rightovertaking (Vehicle**,float);
void carfollowing( Vehicle **);
};
//--------------------------------------------------------------------------------
class MTW: public Vehicle     //Motorized two-wheeler class
{
private:
static float length;       //Only one data item for all objects
static float width;
static float maxcl;       //Declaration only
static float mincl;
static float mean;
static float sd;
static float maxac;
static float midac;
static float minac
public:
MTW (): Vehicle ()
        { }
~MTW ()
        { }
void freespeed (int);
void xclearance();
void show (float, float, int);
void updateposition (float);
void fixacceleration ();
void intialization (float, float);
char type ();
};
//--------------------------------------------------------------------------------
float MTW:: length=1.8;            //Definition of static variables
float MTW:: width=0.6;
float MTW::mincl=0.2;
float MTW::maxcl=0.7;
float MTW::mean=0.00;
float MTW::sd=0.00;
float MTW::minac=1.35;
float MTW::midac=1.03;
float MTW::maxac=0.37;
//--------------------------------------------------------------------------------
class Bicycle:public Vehicle          //Bicycle class
{
private:
static float length;
static float width;
static float maxcl;
static float mincl;
static float mean;
static float sd;
static float minac;
public:
Bicycle():Vehicle()
    { }
~Bicycle ()
     { }
void freespeed (int);
void xclearance ();
void show (float, float, int);
Answer motorized ();
void updateposition (float);
Answer fix_y_coordinate (float, float, float);
void fixacceleration ();
void intialization (float, float);
char type ();
};
```

FIGURE 4: Description of *Vehicle*, *MTW* and *Bicycle* classes

4.1.5 Polymorphism

Polymorphism is the relationships of objects of many different classes by some common super class. Thus, any of the objects designated by this name is able to respond to some common set of operations in a different way (Booch, 1994). For example, classes such as *MTW*, *Car*, *Bus* etc. are inherited from *Vehicle* class. Each class has member function free speed (Figure 4), invoked by *v->freespeed ( )* command line. But

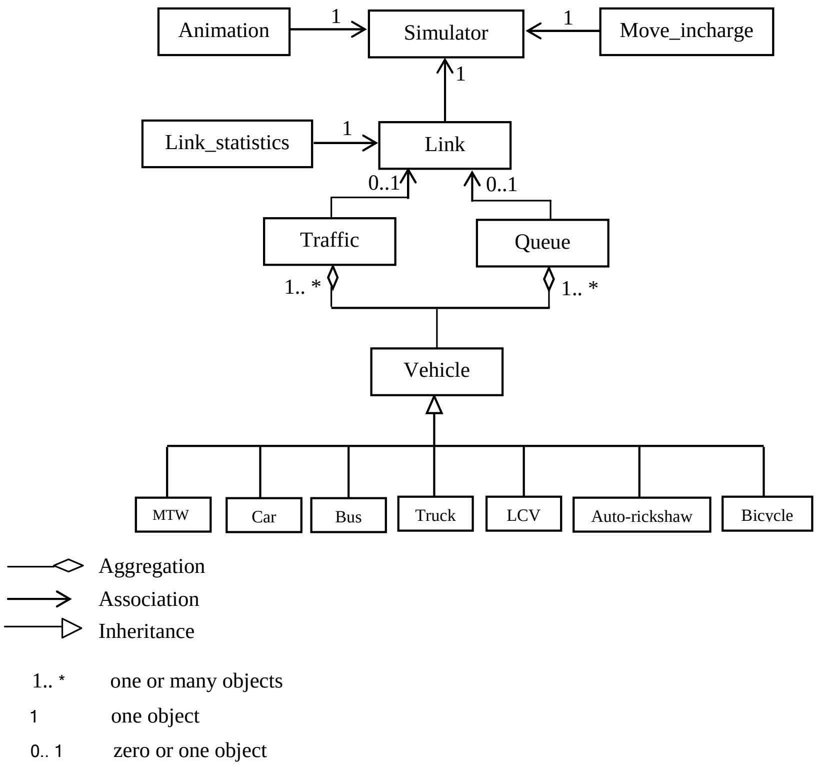

FIGURE 5: Class diagram for the heterogeneous traffic simulation

the compiler does not know which class has the vehicle pointer. At runtime, it is known which class is pointed to by *v*, and then appropriate function is called. This process is called as late binding or dynamic binding. Polymorphism works under two conditions: (a) Subclasses (*MTW*, *Car* etc.) are derived from superclass (*Vehicle*), (b) Function should be declared as virtual (*virtual Answer motorized ( )*) in base class *Vehicle*. Polymorphism is useful in writing generic, reusable code and ease of maintenance.

4.2 Description of the classes

The classes that implement the OOP approach in C++ for heterogeneous simulation model are discussed below.

4.2.1 Vehicle class

The abstract class *Vehicle* encapsulates the common attributes and behaviour of a vehicle. The common manoeuvres of vehicles such as movement, overtaking, and car following lie in this class. The hierarchy of *Vehicle* class is shown in Figure 5. Classes: *MTW*, *Car* etc. are derived from *Vehicle* class. The declaration for classes *Vehicle*, *MTW*, and *Bicycle* are given in Figure 4. The main task of vehicle is to perform movement, overtaking and car following operations which are achieved by member functions such as *updatespeed ( )*, *overtaking ( )* and *car following ( )*, respectively. In *Vehicle* class, *freespeed ( ), xclearance ( ), fixacceleration ( )*, etc. are defined as pure virtual functions. The calculation of these functions is done in derived class, i.e., vehicle is dynamically bound to its derived class during execution, which depends on the type of vehicle. Another benefit of virtual functions is that, if an object of a *Vehicle* class is created, the compiler will complain trying to instantiate an object of abstract class.

Vehicle class uses a keyword *Answer* to specify the data type for the return value of functions, *fix_y_coordinate ( ), motorized ( ),* etc. An *Answer* data type can be easily defined in OOP by using "enumerator data type" in C++ language, which can return two values, i.e., "no" or "yes", as shown in first statement code in Figure 4. To easily create a new data type (*Answer*, *Space*, *Type*) in C++, enumerated data type features of OOP can be used. Each derived class has specific data items (*length*, *width*) and virtual functions, which need to be modified accordingly. In derived class, data items are declared as static; hence only one such item is created for entire class, no matter how many objects are there. It is useful when all the objects of the same class share a common item of the information. For example, data items of an object of *MTW* class (*length*, *width*, *mean*, *standard deviation*) are similar for all objects of that class. The declaration and definition of static variables are different from ordinary variables. Ordinary variables are declared and defined in the same statement, but static variables are declared in class declaration and defined outside the class (Figure 4). *Bicycle* class overrides the base class member functions such as *motorized ( )* and *fix_y_coordinate ( )* based on the behaviour of bicycle. For example, bicycle is a non-motorized vehicle, so its class returns "no" value for motorized function, but other derived classes (*MTW, Bus*) returns "yes" value.

4.2.2 Traffic and Queue classes

These two classes are used to store vehicle in a doubly linked list and their definitions are listed in Figure 6. In *Traffic* class, vehicles are stored in linked list based on the positions of the vehicles from the exit end of the road stretch. The entire linked list represents an object of class *Traffic* and the individual links are represented by structure of vehicle list, and each structure contains a vehicle pointer. This class itself stores the address of first and last vehicles on the road stretch. It handles the list of operations, addition (*d_append ( )*), deletion (*remove_vehicle ( )*) and sorting (*sort ( )*) of vehicles. If the placement of a generated vehicle is not possible on the road stretch, then its placement is shifted to next scan interval and this vehicle is added in the vehicle list of *Queue* class.

4.2.3 Link and Link_Statistics classes

Link class represents data on length, width, traffic composition and traffic volume of the link. The description of the *Link* class is shown in Figure 7. The function *getinput ( )* gets input data from the user. In addition, this class has vehicle and traffic related functions. *Statistics* class is responsible for collection of statistics of vehicles. The data in statistics class contains *file pointer, speed* and *volume* variables. The member function, *collectstatistics ( )* is responsible for collection of data on speed, volume, headway and concentration of the vehicles.

```
struct vehiclelist
{
vehiclelist *prev;    //Pointer to previous vehiclelist
Vehicle  *v1;         //Vehicle pointer
vehiclelist *next;    //Pointer to next vehiclelist
};
//----------------------------------------------------------------------------------------
class Traffic
  {
  private:
    vehiclelist *first;     //Pointer to first vehiclelist
    vehiclelist *last;      //Pointer to last vehiclelist
    Vehicle *v;
  public:
    Traffic ()
        { }
     ~Traffic ()
         { }
    void d_append (Vehicle **);     //Add vehicle pointer to vehicle list
    vehiclelist* getvehiclelist (vehiclelist **);
    Vehicle* getvehicledata (vehiclelist **);
    vehiclelist* movetrafficlist (vehiclelist **);
    void sort (vehiclelist**);
    void remove_vehicle (float);  //Remove vehiclelist
 };
//----------------------------------------------------------------------------------------
class Queue
  {
  private:
    vehiclelist *start;   //Pointer to first vehiclelist
    Vehicle *v;
  public:
    Queue ()
         { }
     ~Queue ()
          { }
    void d_append (Vehicle ** ) ;
    Vehicle* getvehicledata ( );
    Answer isthere ();
   void list_interchange( vehiclelist**  ,int  );
 };
```

FIGURE 6: Definition of *Traffic* and *Queue* classes

```
class Link
  {
  private:
    float length;                          //Link details
    float width;
    float volume;
    float average_flow;
    float cumulative_headway;
    int dist;
    float twu,caru,busu,trucku,lcvu,autu,bicycleu;           //Composition of  traffic
    Traffic *t;
    Queue *q;
    Link_statistics *s;
 public:
    Link()
             { }
    ~Link()
             { }
    void getinput();                       //Functions related to Link, Traffic and Queue classes
    float returncheadway();
    Vehicle* vehicletype( );
    float get_position( Answer);
    Traffic* getvehiclelist(vehiclelist*);
    void  d_append( Vehicle*, );
    void sort( vehiclelist*, );
    Vehicle* getvehicledata( vehiclelist** );
    Vehicle* movetrafficlist( vehiclelist** );
    void list_interchange( Traffic** ,vehiclelist** );
    Traffic* gettrafficdetails( );
    Queue* returnqueue( );
     void remove();
  };
```

FIGURE 7: Definition of the *Link* class

4.2.4 Move_incharge and Animation classes

In the code architecture, a separate class *Move_incharge* was created, which is responsible for movement of entire traffic stream. *movetraffic ( )* method collects the vehicle data from *Traffic* class and then, the position of the vehicle is updated (*updatespeed ( )*). If updation of the vehicle is not possible, left or right overtaking (*leftovertaking ( )* or *rightovertaking ( )*) is performed using *decisionofovertaking ( )* method. In this updating process, all the vehicles in the link are updated at each scan interval. *Animation* class is responsible for graphical representation of traffic flow.

4.2.5 Simulator class

The main function is used to create objects of two classes such as *Simulator* and *Link. Simulator* class is responsible for entire simulation and drives the different stages of the process. The management of different stages of the process is performed by this major class, which represents the heart of the software architecture. The description of main function and *Simulator* class is shown in Figure 8. The different processes (placement, movement and animation of vehicles) in simulation are performed through the method *simulationprocess ( ).*

```
int main ()
   {
   clrscr();
   Link *l;              // Definition of object of Link class
   l->getinput()
   Simulator * s;        //Definition of object of  Simulator class
   s=new simulator (&l);
   return 0;
   }
//--------------------------------------------------------------------------------
class Simulator
{
private:
float precision;
float clock_time;
float scan_interval;
float TST;                           //Total simulation time
Link *l;
Move_incharge  *m;
Animation  *a;
public:
Simulator ()
      { }
~Simulator ()
      { }
void siumlationprocess ();           //Incharge for all simulation related process
Done placement (Vehicle*);
};
```

FIGURE 8: Definition of Main function and *Simulator* class

4.3 Class relationships

The class diagram which explains the classes in the system and their relationships in the context of the heterogeneous traffic simulation system is shown in Figure 5. The symbols used for representing the object relationship follow the Unified Modeling Language (UML) (Bahrami, 1999; Rambuch et al., 1999). In this system, the objects of the *Vehicle* class, *Traffic* class, *Queue* class and *Link* class are real world objects. On the other hand, the objects of *Simulator*, *Move_incharge*, *Link_statistics* and *Animation* classes are conceptual objects. The objects of *Link*, *Move_incharge and Animation* classes are associated with the object of *Simulator* class. An object of *Traffic* class and

Queue class is a collection of objects of *Vehicle* class. Objects of *Traffic, Queue and Link_statistics* classes are associated with the class *Link*. The classes *MTW*, *Car*, *Bus*, *Truck*, *LCV*, *Bicycle*, and *Auto-rickshaw* are specialized classes derived from *Vehicle* class. Table 1 describes the relationship between the objects in object oriented decomposition in the context of heterogeneous traffic simulation.

TABLE 1: Relationships between the objects in object oriented decomposition

Object 1	Relationship	Object 2	Object Oriented Property
MTW, Car, Bus, Truck, LCV, Auto-rickshaw and *Bicycle*	is a kind of	*Vehicle*	Specialization/inheritance
Traffic	is a collection of	*Vehicle*	aggregation
Queue	is a collection of	*Vehicle*	aggregation
Link	has	*Traffic*	association
Link	has	*Queue*	association
Link	has	*Link_statistics*	association
Simulator	has	*Move_incharge, Animation, Link*	association

5. NUMERICAL EXAMPLE - A CASE STUDY

The developed model should be validated to verify if it is able to imitate the field conditions. The data on volume, traffic composition, road length, road width, and free speed of each category of vehicle are given as inputs to the model. Other data such as headway distribution type, length and width of each type of vehicle, acceleration/deceleration characteristics of different categories of vehicles and transverse clearance are inbuilt model parameters. Traffic data was collected at a mid-block stretch of a divided roadway with 7.5 m wide road space for each direction of traffic flow, in Chennai City, India. Traffic movements were captured using videography during peak and off-peak periods. The video data was played in a video player on a computer and volume and composition of traffic were extracted. The vehicle composition for a traffic volume level of 3249 veh/h at a study stretch is shown in Figure 9. It can be noted that the major and minor proportion of traffic comprises of MTW (52%) and truck (1.42%) respectively.

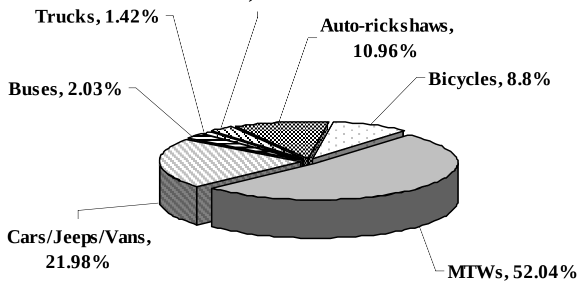

FIGURE 9: Traffic composition at a study stretch

The length, width and transverse clearance of each type of vehicle are given in Table 2. Transverse clearance of the vehicle is based on the speed of that vehicle, speed of the adjacent vehicle and also their respective types. This transverse clearance is an important parameter for overtaking (passing) manoeuvre. Transverse clearances were adopted from the studies of Nagaraj et al. (1990) and Isaac and Veeraragavan (1995) and these studies indicate the linear relationship of transverse clearances with speed and also give the maximum and minimum limits for transverse clearances. The transverse clearance was calculated by the following equation:

$$C = \frac{C_{\max} - C_{\min}}{V_{\max} - V_{\min}}(V - V_{\min}) + C_{\min}, \tag{11}$$

where C is the transverse clearance, $C_{\min}$ is the minimum clearance, $C_{\max}$ is the maximum clearance, V is the current speed, $V_{\min}$ is the minimum speed, and $V_{\max}$ is the maximum speed.

TABLE 2: Overall dimensions and transverse clearance limits for different categories of vehicles

Vehicle class	Length (m)	Width (m)	Minimum clearance at zero speed (m)	Maximum clearance at 60 kmph speed (m)
Bus	10.3	2.5	0.4	1.0
Truck	7.2	2.5	0.4	1.0
LCV	5.0	1.9	0.3	0.7
Car	4.2	1.7	0.3	0.7
Auto-rickshaw	2.6	1.4	0.2	0.7
Motorized two-wheeler	1.8	0.6	0.1	0.7
Bicylce	1.9	0.5	0.1	0.5[a]

[a] Maximum clearance at 20 kmph

To estimate the speeds of different categories of vehicles, time taken by vehicles to travel a trap length of 50 m was determined. Free speeds of different types of vehicles were collected during off-peak periods when vehicles moved freely on the stretch of a roadway. The mean and standard deviation of free speeds of various categories of vehicles are given in Table 3. The acceleration characteristics of vehicles vary widely under heterogeneous traffic conditions. Acceleration and deceleration rates at various speed ranges for different types of vehicles were adopted from previous study (Arasan and Koshy, 2005).

TABLE 3: Free speeds of different types of vehicles

Vehicle class	Free speed (kmph)	
	Mean	Standard deviation
Bus	53.03	7.50
Truck	52.52	6.80
LCV	49.80	6.50
Car	58.30	13.40
Auto-rickshaw	44.89	7.70
Motorized two-wheeler	45.40	12.10
Bicylce	15.98	2.99

The total length of road stretch for simulation was considered as 1200 m. The initial stretch (entry point) of 200 m was considered as warm up zone. The exit end of the road stretch of length 200 m was excluded for traffic analysis due to unstable flow of traffic conditions. The vehicle is deleted when it reaches the exit end of the road stretch; the immediately following vehicle accelerates to its free speed (unconstrained vehicle). This process creates unstable traffic flow at the exit end of the road stretch. The traffic statistics were collected after 100 vehicles reached the exit end of the road stretch in order to eliminate the initial transition state of traffic flow. For any level of traffic composition, the heterogeneous traffic simulation model can be simulated including the extreme case of “Cars only” traffic. The developed simulation model was validated to this extreme condition (homogeneous traffic). In this case, the model was simulated with 100% passenger cars on a single lane road with no passing. During the simulation runs, the traffic volume on the study stretch was varied from 100 to 1950 veh/h. The capacity of the simulated value (1850 veh/h) is close to single lane capacity of 1800 veh/h (as per Highway Capacity Manual, 1994). Speed-flow curve was developed and shown in Figure 10. The figure illustrates the trend and nature of the curve which resembles general shape of such curve.

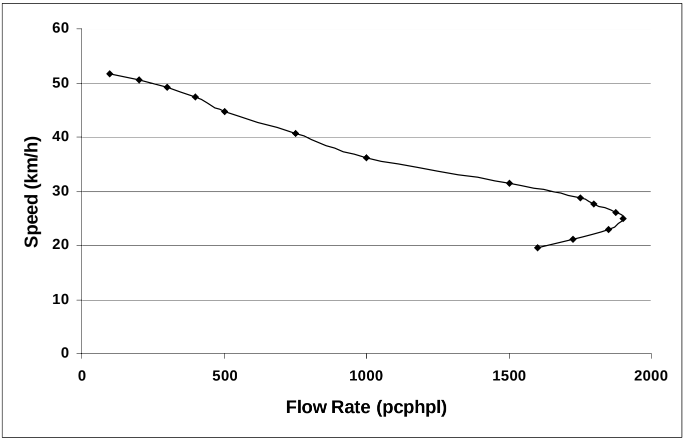

FIGURE 10: Speed-flow curve for homogeneous traffic

The model was again validated by comparing the observed and simulated headways at a volume level of 3249 veh/h (Figure 11). It is seen that there is no significant difference between the observed and simulated cumulative frequency distributions, confirming the validity of model. To further check the validity of the model, observed and simulated speeds were compared. Figure 12 shows the comparison of simulated and observed speeds of different types of vehicles. Buses have higher free speeds, but there is a significant amount of speed reduction for buses compared to other vehicles (as seen in Figure 12) because of their larger size. Speeds of motorized two wheelers and auto-rickshaws are not reducing much because of their smaller size and hence, they use lateral gaps between larger vehicles. Simulated speeds of bicycles closely resemble with observed speed values. A paired *t*-test of null hypothesis with no mean difference was performed at 0.05 level of significance. It was inferred that there is no significant difference between the simulated and observed speeds of different types of vehicles.

6. ADVANTAGES OF THE MODEL

From the literature it is understood that most of the models are developed for specific applications. If these models are extended to newer applications, it is difficult and time consuming for rewriting the code. The developed model can be maintained, reused and extended easily for newer applications. For example, the introduction of new type of

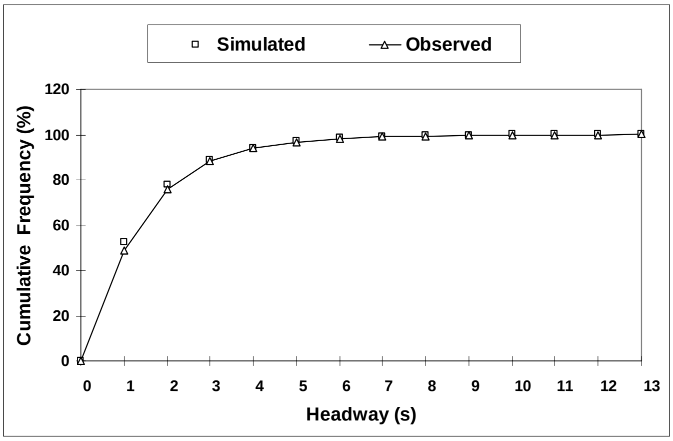

FIGURE 11: Comparison of simulated and observed headways

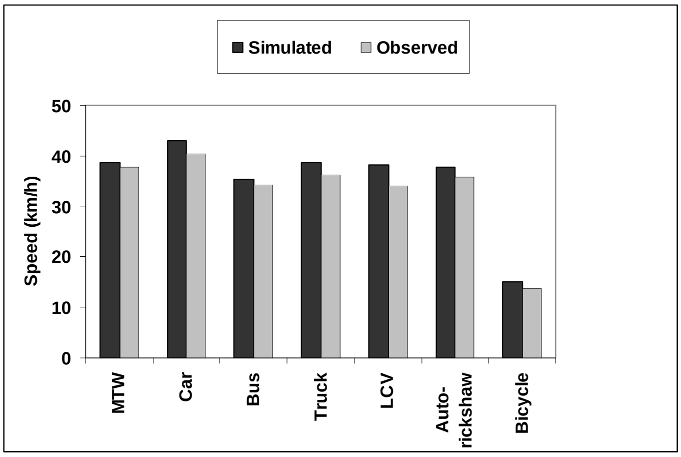

FIGURE 12: Comparison of simulated and observed speeds

vehicle or simulation of other elements of traffic engineering (intersections, exclusive bus lanes, and control measures) can be achieved easily through minor changes of existing classes or definition of new classes. The implementation of these applications in the heterogeneous traffic simulation model is explained below.

6.1 Introduction of new mode

If a new type of mode (tricycle) is to be introduced in the simulation model, new classes can be created and only minor modifications in the existing classes are needed. The new classes such as *Motorized Vehicle, Non-motorized Vehicle* and *Tricycle* classes are shown in Figure 13. Class *Vehicle* serves as a base class for *Motorized Vehicle* and *Non-motorized Vehicle* classes (intermediate base classes). *MTW, Car, Bus, Truck, LCV,* and *Auto-rickshaw* are inherited from *Motorized Vehicle* class. *Bicycle* class and *Tricycle* class are inherited from *Non-motorized Vehicle* class. Motorized and Non-motorized vehicle characteristics are contained in *Motorized Vehicle* class and *Non-motorized*

Vehicle class, respectively. The *Vehicle* class contains common characteristics of *Motorized Vehicle* and *Non-motorized Vehicle* classes.

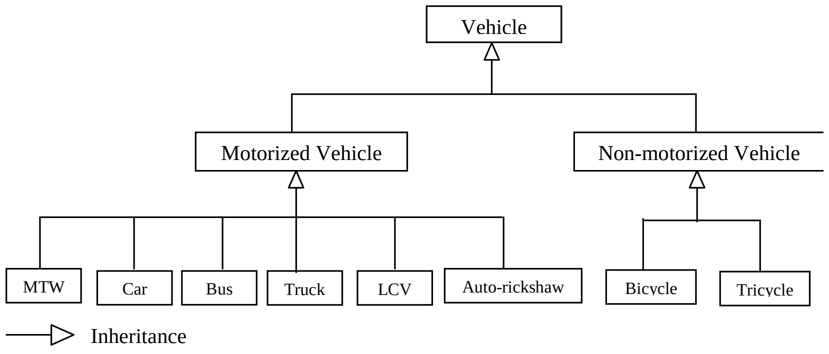

FIGURE 13: Class hierarchy in *Vehicle*

6.2 Exclusive bus lane

In this model, provision of exclusive bus lane can be achieved using the powerful approach of polymorphism. A virtual function, *fix_y_coordinate ( )* in the V*ehicle* class can be overridden in the derived class, *Bus* for implementation of exclusive bus lane. This function is invoked by a command line, v-> *fix_y_coordinate ( ).* The vehicle pointer, *v* contains the address of object of *Bus* class, its member function *fix_y_coordinate ( )* is called and operation of exclusive bus lane can be implemented.

6.3 Intersection simulation

This mid-block simulation model can be extended easily to develop intersection simulation model, by defining three new classes such as *Traffic_light* class, *Move_intersection* class and *Node* Class and, making minor modifications in existing classes.

From the above applications, it is understood that maintainability, reusability and extensibility of the code can be achieved using OOP. In any case, it does not require complete redefinition of the software architecture

7. CONCLUSIONS

In this paper, a heterogeneous traffic simulation model was developed for a mid block road section using OOP principles with C++. Encapsulation, inheritance, polymorphism and other features of OOP were incorporated in the model. It also incorporates the unique features of heterogeneous traffic conditions. Field values of speeds and headways were compared with simulated values for model validation. The validation of the model, based on headway distribution and speeds of different types of vehicles, indicates that the model is satisfactorily replicating field conditions. The benefits of improved maintainability, flexibility and ease of expansion of the program are achieved through systematic object oriented approach. Extensibility and reusability of the developed model may be exploited for intersection flow modeling and to simulate traffic management measures by modifying only a small part of the program or by defining new classes. In any case, it does not require the complete redefinition of software architecture. It has been proposed to develop the model further to incorporate the effects of intersection, intersection geometries, traffic, and traffic control conditions to build a program package which can be used to evaluate various traffic scenarios, which in turn can lead to improved traffic management measures, particularly for heterogeneous traffic conditions.

REFERENCES

Agarwal, R.K., Gupta, A.K., Jain, S.S. and Khanna, S.K. (1994) Simulation of intersection flows for mixed traffic. Highway Research Bulletin, Indian Road Congress, 51, 85-97.

Arasan, V.T. and Koshy, R.Z. (2005) Methodology for modeling highly heterogeneous traffic flow, Journal of Transportation Engineering, ASCE, 131 (7), 544-551.

Aycin, M.F. and Benekohal, R.F. (2001) Stability and performance of car-following models in congested traffic. Journal of Transportation Engineering, ASCE, 127 (1), 1-12.

Bahrami, A. (1999) Object Oriented Systems Development using Unified Modeling Language. Irwin/ McGraw-Hill, Singapore.

Ben-Akiva, M.E., Koutsopoulos, H.N., Mishalani, R.G. and Yang, Q. (1997) Simulation laboratory for evaluating dynamic traffic management systems, Journal of Transportation Engineering, ASCE, 123 (4), 283-289.

Bhavsar, J.N., Sharma, S. and Dhingra, S.L. (2007) Simualting bus priority for an urban corridor in Mumbai City. In Proceedings of the 11th World Conference on Transportation Research, University of California, Berkeley, USA.

Booch, G. (1994) Object Oriented Analysis and Design. Addison-Wesley, Reading, MA.

Chandra, S. and Parida, M. (2004) Analysis of urban road traffic through simulation. Indian Highways, 32, 87-102.

Chandrasekar, P., Cheu, R.L. and Chin, H.C. (2002) Simulation evaluation of route-based control of bus operations. Journal of Transportation Engineering, ASCE, 128 (6), 519-527.

Fang, C.F. and Elefteriadou, L. (2005) Some guidelines for selecting micro simulation models for interchange traffic operational analysis. Journal of Transportation Engineering, ASCE, 131 (7), 535-543.

Fenves, G.L. (1990) Object-oriented programming for engineering software development. Engineering with Computers, 6, 1-15.

Gipps, P.G. (1981) A behavioural car-following model for computer simulation. Transportation Research Part B, 15, 105-111.

Helbing, D. and Tilch, B. (1998) Generalized force model of traffic dynamics. Physical Review E, 58, 133-138.

Highway Capacity Manual (1994) National Research Council, Transportation Research Board, Washington, D.C., USA.

Ijaha, S.E., Winter, S.C. and Kalantery, N. (2000) HIPERTRANS: High performance transport network modeling and simulation. Lecture Notes in Computer Science, 1900, 869- 874.

Isaac, K.P. and Veeraragavan, A. (1995) Studies on mixed traffic flow characteristics under varying composition. PhD Thesis, Bangalore University.

Jiang, R., Wu, Q.S. and Zhu, Z.J. (2001) Full velocity difference model for a car-following theory. Physical Review E, 64, 017101.

Kumar, V.M. and Rao, S.K. (1996a) Simulation of traffic flow at a yield controlled T-intersection for establishing volume warrants. Highway Research Bulletin, Indian Road Congress, 54, 53-62.

Kumar, V.M. and Rao, S.K. (1996b) Simulation modeling of traffic operations on two-lane highways. Highway Research Bulletin, Indian Road Congress, 54, 211-237.

Lafore, L. (1995) Object Oriented Programming in C++. Waite Group Press.

Lan, L.W. and Chang, C.W. (2005) Inhomogeneous cellular automata modeling for mixed traffic with cars and motorcycles. Journal of Advanced Transportation, 39, 323-349.

Lee, H.H. and Arora, A.S. (1991) Object-oriented programming for engineering applications. Engineering with Computers, 7, 225-235.

Li, W.G., Yamashita, Y., Koendjbiharie, M.W., Jucá, R.C.M. and MacIver, A. (2004) The development and implementation of the operation system and data bank for the intelligent transportation system – Sitcuo. Journal of Advanced Transportation, 38 (2), 163-186.

May, A.D. (1990) Traffic Flow Fundamentals. Prentice Hall, New Jersey.

Murugesan, R., Venkataraman, T.S. and Moses, S.K.S. (1991) Performance evaluation and design of signal settings. Indian Highways, 19, 11-19.

Nagaraj, B.N., George, K.J. and John, P.K. (1990) A study of linear and lateral placement of vehicles in mixed traffic environment through video-recording. Highway Research Bulletin, Indian Road Congress, 42, 105-136.

Popat, T.L., Gupta, A.K. and Khanna, S.K. (1989) A simulation study of delays and queue lengths for uncontrolled T-intersections. Highway Research Bulletin, Indian Road Congress, 39, 71-78.

Raghavachari, S.K.M., Badrinath, K.M. and Bhanu, M.P.R. (1993) Simulation of an urban uncontrolled urban intersection with pedestrian crossings. Highway Research Bulletin, Indian Road Congress, 48, 29-49.

Rambauch, J., Jacobson, I. and Booch, G. (1999) The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, MA.

Rao, V.T. and Rengaraju, V.R. (1998) Modeling conflicts of heterogeneous traffic at urban uncontrolled intersection. Journal of Transportation Engineering, ASCE, 124 (1), 23-34.

Ross, T.J., Wagner, L.R. and Luger, G.F. (1992) Object-oriented programming for scientific codes, I: thoughts and concepts. Journal of Computing in Civil Engineering, ASCE, 6 (4), 480-496.

Spyropoulou, I. (2007) Simulation using Gipps' car-following model - an in-depth analysis. Transportmetrica, 3, 231-245.

Tsukamoto, K., Matsuoka, Y., Seki, H. and Itoh, H. (1994) A parallel program for an urban traffic analyzation and its evaluation. Computers and Industrial Engineering, 27, 233-236.

Webster, N.A., Suzuki, T. and Kuwahara, M. (2008) Tactical lane change model with sequential maneuver planning. Transportmetrica, 4, 63-78.

Wong, S.C., Yang, H., Au Yeung, W.S., Cheuk, S.L. and Lo, M.K. (1998) Delay at signal controlled intersection with bus stop upstream. Journal of Transportation Engineering, ASCE, 124 (3), 229-234.