

SE 450: Object-Oriented Software Development
Final Project
Traffic Simulator Design

By Scott B. Bradley
SE450, Section 905
June 9, 2005

Introduction

The Traffic Flow Simulation Project has been developed for SE450 as an exercise in the development of a significant object-oriented application utilizing modern design and development techniques. The functionality of the program is relatively simple, but the overall structure sufficiently complex to serve as a significant learning tool for object oriented development. This basic application could easily be expanded into a more complex application by continued development of program features.

Project Summary

The project goal was to develop a traffic flow simulation program that would provide a means for evaluating traffic flow in a simple road system. The program allows for the manipulation of a variety of simulation parameters related to the roads, vehicles and traffic signals. The program is developed with a text-based user interface, but a graphic view could easily be developed to enhance the current program.

Furthermore, the program's implementation of the road structure has been made to be expandable by decoupling the infrastructure from the data model, and so providing a simple means of further developing the existing program, including the development of more complex infrastructures.

The use of common design patterns has been emphasized throughout the design and development of the Traffic Simulator project. The following list provides an overview of known design patterns which have influenced the current program design.

- **Model-View-Controller**
An architectural pattern which has significantly impacted the overall structure of the program. MVC provides decoupling of the View from the program Model and facilitates further View development according to criteria that may be different from that of the current view. In particular, in conjunction with the command pattern, the MVC has been employed in a manner in anticipation of development of a GUI view for the program.
- **Observer**
The MVC pattern makes integrated use of the observer pattern. In addition, Vehicles must be configured as observers in order to behave properly, observing the states of other vehicles as well as Traffic Signals.
- **Strategy**
Strategy is used to select from a number of potential ways to perform the same action. Here, strategy is used to determine the appropriate equation for finding a vehicle's velocity. The strategy is accessible to the data model and the Vehicle class primarily through the static methods of the Velocity class.
- **Abstract Factory**
Road components are generally produced by means of the Abstract Factory pattern. Concrete factories for different types of road segments and intersections implement general factory interfaces and provide the needed instances of those infrastructure components.

- Command
The command pattern provides object encapsulated commands which allows for greater control over handling of the commands. Here, command objects which handle user input events as commands implement abstract commands based on the data type of the user input. This allows type-checking to be controlled by the command object rather than by the view. Command objects are passed to the view in a collection which allows the view to determine how they are handled in the user interface, thus anticipating implementation of a GUI interface.
- Decorator
The decorator pattern allows for object behaviors to be determined at run time. Here, the command pattern makes integrated use of decorator methodologies. Upon generation of command objects, the TrafficSimModel, which is responsible for the program parameters, decorates those objects with the functionality needed to allow the Controller to set the parameters according to the user input. This approach provides extensive flexibility for interfacing with the model and keeps the model in charge of data parameters.

Design Scenario

Because of the decision to use a complex command structure toward the development of a flexible and expandable program interface, it was determined that the user interface would be a high risk aspect of the design. The following use case and design scenario served in the early stages of development to mitigate that risk and to determine the basic program structure with regard to the user interface and its relation to the data model.

The following design scenario includes a use case, CRC cards, and the resulting class diagram and sequence diagram. It should be noted that although the design represented by this design scenario was made early in the design and development process, this high-risk scenario was instrumental in structuring the rest of the program and has been, in the end, implemented in a manner very similar to the design shown here. A particular note of difference is that the command collection has not been treated as an external class but has been wholly managed by the TrafficSimModel. A more cohesive design would probably involve refactoring the existing code to more closely match the design shown in this scenario.

Use Case

Name: Text-based Parameter Entry User Interface Use Case.
 Description: Upon startup of the Traffic Flow Simulation program, the user will be prompted to enter model parameter values.
 Actors:

- Traffic Flow Simulation User
- Traffic Flow Simulation program Model component
- Traffic Flow Simulation program View component
- Traffic Flow Simulation program Controller component

Trigger:	Traffic Flow Simulation program has been started from the command line.
Preconditions:	Traffic Flow Simulation program has been successfully started and the program banner has been displayed.
Incoming Information:	Program model parameter values.
Results:	Simulation model parameter values are set.
Post-conditions:	Valid model parameters have been entered and set for the simulation.

- Event Flow:
1. Use case begins when system has been started and the startup banner has been displayed.
 2. System Controller notifies System Model of setup status.
 - Required abstractions: TrafficSimModel, TrafficSimController
 - TrafficSimModel will require a public initialization method: setParams()
 3. System Model notifies System View of current prompt (*Frequency of Vehicle Arrival for Entry Point #1*).
 - Abstraction: TrafficSimView
 - Note integrated use of observer pattern in the MVC structure
 - Mechanism will be required to pass user input prompt messages, retry messages (in case of invalid data input)
 4. System View displays prompt for *Frequency of Vehicle Arrival for Entry Point #1*.
 5. User enters value for *Frequency of Vehicle Arrival for Entry Point #1*.
 - View itself will only handle String input. Intelligence for handling parsing of data type from input handled by mechanism external to view.
 6. System View notifies System Controller of entered value for *Frequency of Vehicle Arrival for Entry Point #1*.
 7. System Controller sets the value for *Frequency of Vehicle Arrival for Entry Point #1* in the System Model.
 - Parameter passing mechanism (which will be itself passed through the observer pattern notification process) will need to supply ability to Controller to update parameters in the Model
 8. Repeat steps 3-7 for each of the following parameter values (in lieu of *Frequency of Vehicle Arrival for Entry Point #1*): *Frequency of Vehicle Arrival for Entry Point #2, Frequency of Vehicle Arrival for Entry Point #3, Frequency of Vehicle Arrival for Entry Point #4, Vehicle Length, Maximum Vehicle Velocity, Vehicle Acceleration, Maximum Obstruction Influence Distance, Minimum Distance From an Obstruction, Signal Timing (Signal #1), Signal Timing (Signal #2), Signal Timing (Signal #3), Simulation Time Step, Number of Simulation Steps, Road Segment Length*.
 - Several data types will need to be handled
 - Design: use a Command pattern like implementation to encapsulate parameter setting behavior and logic of parsing input through the View. Abstract command definitions include the general UserInputCommand, and the data-type specific classes that subclass it: DoubleUserInputCommand, IntegerUserInputCommand, SignalTimingUserInputCommand. Note also, SignalTiming should be a type defined in its own abstraction.

CRC Cards

TrafficSimModel	
<i>Responsibility</i>	<i>Collaborators</i>
Update the view	TrafficSimView
Receive updates from the controller	TrafficSimController
Collect command objects	CommandController
Instantiate concrete UserInputCommand implementations	DoubleUserInputCommand
Instantiate concrete UserInputCommand implementations	IntegerUserInputCommand
Instantiate concrete UserInputCommand implementations	SignalTimingUserInputCommand

TrafficSimView	
<i>Responsibility</i>	<i>Collaborators</i>
Observe the model	TrafficSimModel
Notify the controller	TrafficSimController
Receive input from the user	Simulation user
Get commands for user display	CommandCollection

TrafficSimController	
<i>Responsibility</i>	<i>Collaborators</i>
Observe the view	TrafficSimView
Update the model	TrafficSimModel
Get parameter mutators for updating the model	CommandCollection

CommandCollection	
<i>Responsibility</i>	<i>Collaborators</i>
Aggregate the command instances	TrafficSimModel

Provide commands to user interface	TrafficSimView
Provide parameter mutators to controller	TrafficSimController

UserInputCommand	
<i>Responsibility</i>	<i>Collaborators</i>
Provide abstract definition of a command	DoubleUserInputCommand
Provide abstract definition of a command	IntegerUserInputCommand
Provide abstract definition of a command	SignalTimingUserInputCommand

DoubleUserInputCommand	
<i>Responsibility</i>	<i>Collaborators</i>
Provide partly abstract definition of a command for a Double input value	Concrete command implementation
Provide type for instantiation of concrete command	TrafficSimModel

IntegerUserInputCommand	
<i>Responsibility</i>	<i>Collaborators</i>
Provide partly abstract definition of a command for a Integer input value	Concrete command implementation
Provide type for instantiation of concrete command	TrafficSimModel

SignalTimingUserInputCommand	
<i>Responsibility</i>	<i>Collaborators</i>
Provide partly abstract definition of a command for a SignalTiming input value	Concrete command implementation; SignalTiming
Provide type for instantiation of concrete command	TrafficSimModel

SignalTiming	
<i>Responsibility</i>	<i>Collaborators</i>

Define SignalTiming Data type	SignalTimingUserInputCommand
----------------------------------	------------------------------

Class Diagram

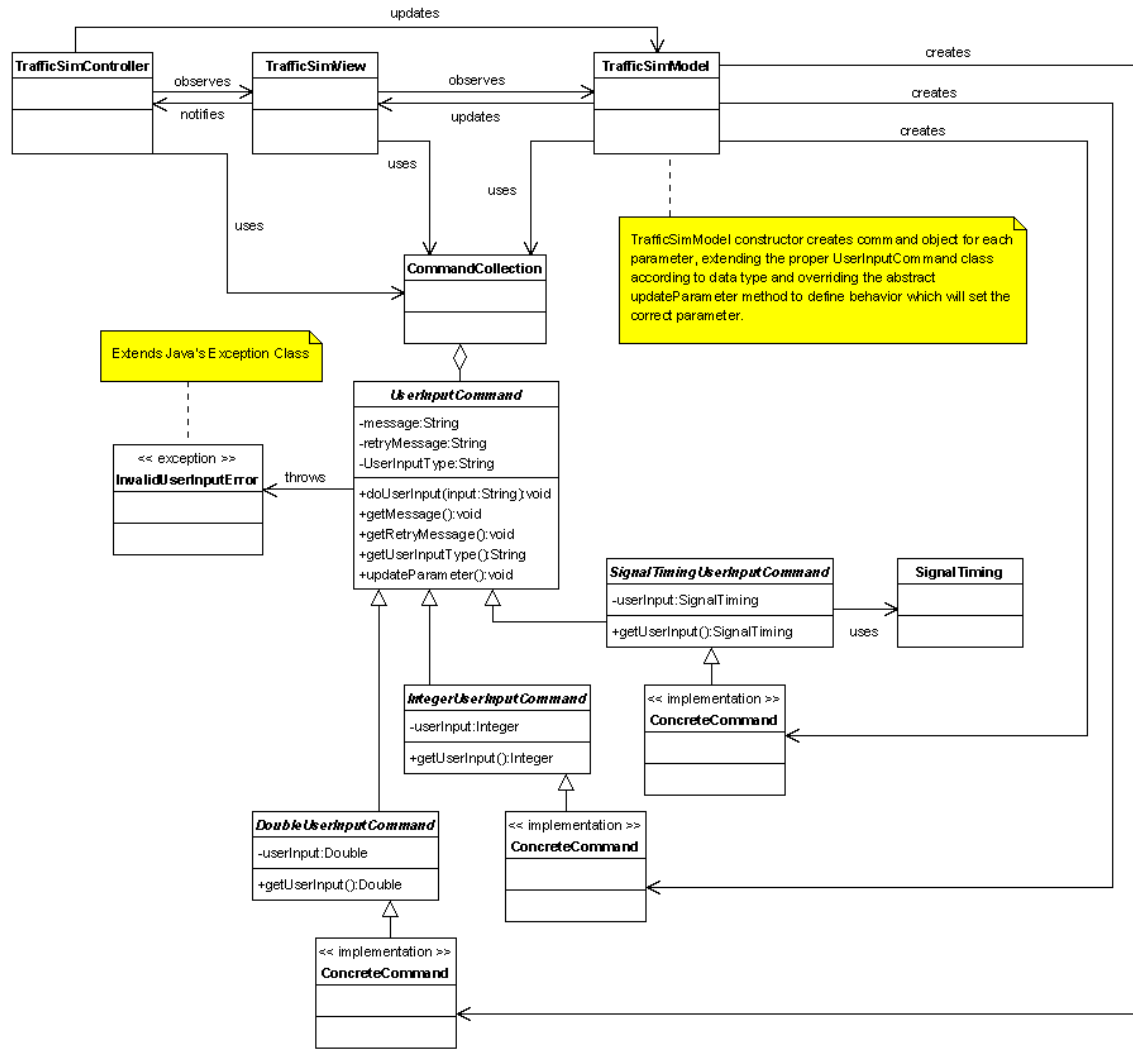


Figure 1. Class diagram demonstrating implementation of user input commands

Sequence Diagram

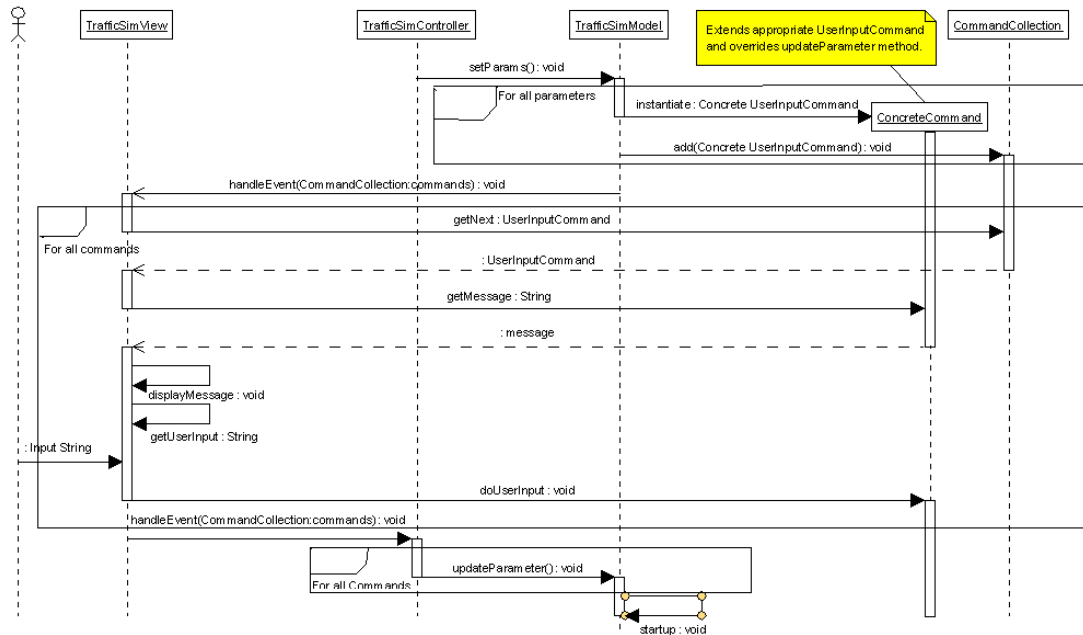


Figure 2. Sequence diagram for high risk user input use case

The Final Project Design

Project Packages

The project has been divided into a number of packages for improved object-oriented manageability. These packages include: model, views, controllers, velocityStrategy, infrastructure. These packages are briefly described below.

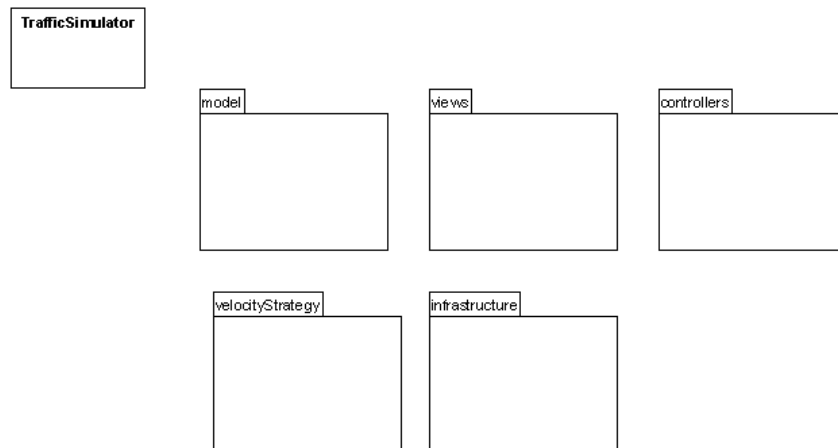


Figure 3. Traffic Simulator driver class and related program packages.

- **model**
The model package contains the primary structure of the overall program. It consists of the single class TrafficSimModel which contains the program state and is the basis for persistence of the program parameters. Interaction by the user with the program model is provided through a Model-View-Controller structured interface.
- **views**
A single view has been implemented. This view is a text-based console view which displays program states and gathers user input by a standard console. Provisions have been implemented for the facilitated development of a GUI based view if desired. Such provisions include:
 - Decoupling of the View from the Model via the MVC architecture.
 - Simplified parameter management through the View via the command pattern which passes a user-input command object to the view for minimal manipulation. This object provides the needed logic for error checking of the user input. Further, the command objects are passed to the view in a single collection in order to allow serial (such as a text view might require) or parallel (such as might be desired for a GUI interface) processing of the commands to occur.
- **controllers**
A single controller has been implemented. The controller is responsible for enacting the data changes in the user input commands via a generic method call. Decoration of this method by the model itself ensures proper modification of program parameters.
- **velocityStrategy**
The velocity strategy is used by a vehicle to determine the speed appropriate for the vehicle based on a number of factors. The velocity equation used is explained in detail in the glossary at the end of this document.
- **infrastructure**
The infrastructure package contains the factories and components necessary for creation of the road system infrastructure. Interaction between this package and the program model has been designed in a way to facilitate modification to the road system infrastructure for the testing of various road configurations. Such modification can be made by making changes to the InfrastructureFactory output without further modification of the model. Further development may lead to an abstract factory structure which would utilize polymorphism to create any number of infrastructures for testing via concrete factories with various structured outputs. The infrastructure package is the most interesting in terms of the visible structure of the

simulation program. This single package demonstrates the bulk of the relationships among program classes. These relationships are shown in the class diagram below which includes the infrastructure package and its relationships to the model and the velocityStrategy.

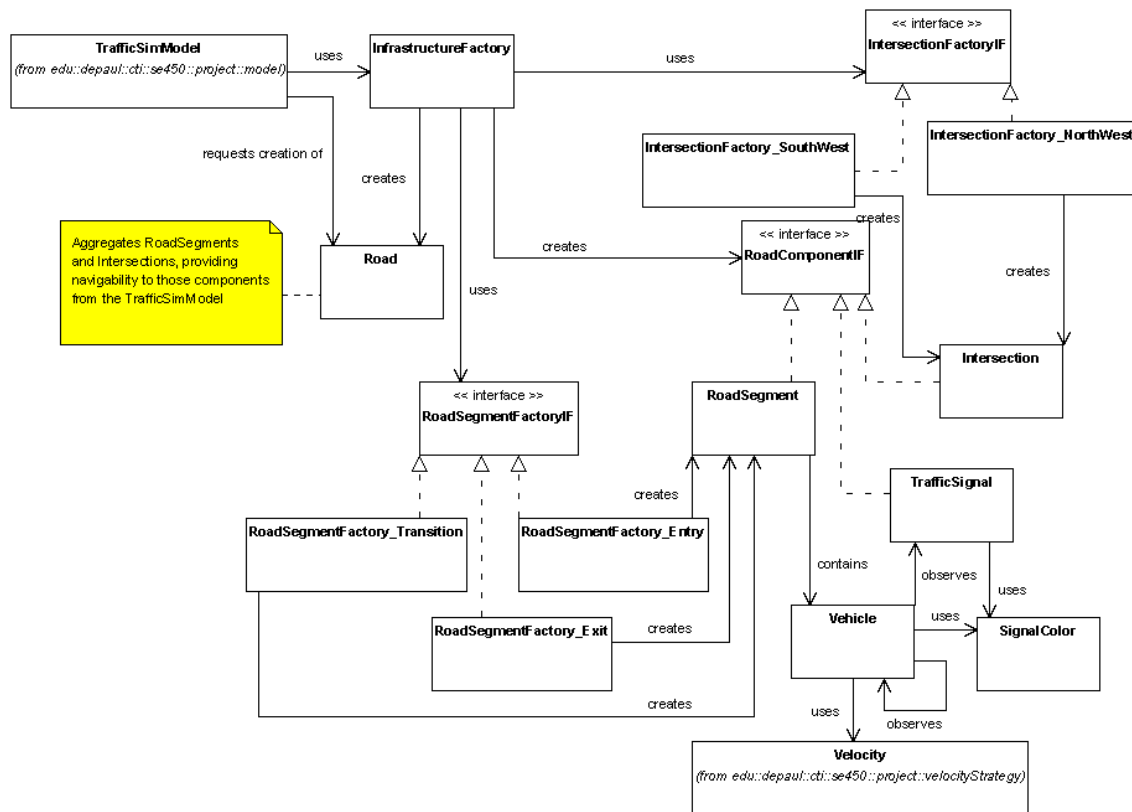


Figure 4. infrastructure package

Glossary

Controller:	System component which observes the system View for updates from the user and which updates the system Model accordingly.
Frequency of Vehicle Arrival:	Rate at which vehicles enter a road at the road entry point. Frequency is set individually for each road.
GUI	Graphical User Interface. Description applies to user view which is graphically rather than textually based.
Maximum Obstruction	
Influence Distance:	Parameter D in the Velocity Equation. Default value is 15 m.
Maximum Vehicle Velocity:	Parameter C_{limit} used by the Velocity Equation. Value is in meters per second. 13.9 m/s is the default value.

Minimum Distance

From an

Obstruction:

Model:

Parameter D_{min} in the Velocity Equation. Default Value is 1.5 m.

System component which holds the system state and related data.

Number of

Simulation Steps:

Road Segment

Length:

Vehicle

Acceleration:

Signal Timing:

Number of simulation time steps to be executed.

Length, in meters, of each road segment. Default value is 100 m.

Average acceleration of vehicle from a stop. Value is in meters per second per second. Default value is 2.4 m/s².

A set of timing profile numbers representing the color state times of a Traffic Signal in the format (red, yellow, green). Values are represented in seconds. Each Traffic Signal has an independent timing profile.

Simulation Time

Step:

Vehicle Length:

Velocity Equation:

Time increment in seconds for each iteration of the simulation.

Length of vehicle in meters. 5 meters is the default value.

Equation which describes vehicular velocity. The equation may be stated as:

$$\left\{ \begin{array}{ll} v_i = c_{limit} & \text{for } x \geq D \\ v_i = mx + b & \text{for } D_{min} < x < D \\ v_i = 0 & \text{for } x \leq D_{min} \end{array} \right.$$

where:

- c_{limit} is constant vehicle velocity (i.e. speed limit)
- x is distance from vehicle i to any obstruction (another vehicle or a traffic signal)
- D is maximum distance of obstruction influence on vehicle i
- D_{min} is distance at which vehicle i stops for the obstruction.

For purposes of implementation, the equation is understood to return constant values for the first and last parts (c_{limit} where $x \geq D$, and 0 where $x \leq D_{min}$) and to return a value according to the following equation where $D_{min} < x < D$:

$$v_i = (c_{limit} / (D - D_{min}))(x - D_{min})$$

View:

Part of the system accessible by the user which takes user input and provides that input to the system in the form of updates to the system Controller.