1 Implementation of Model

1.1 Why Java and XML?

We chose to implement our traffic model using Java for two main reasons. We wanted to use a language that allowed us to encapsulate our model in a class structure. Thus, the high-level language and object-orientated structure of Java was our foremost reason for implementing our model in Java. Additionally, we chose Java over other object-orientated, high-level languages because our familiarity with programming in Java out-weighed our desire to learn the ins-and-outs of an unfamiliar high-level language.
As for the persistence of our simulation summary data, we chose to use the data persistence functionality of Extensible Markup Language (XML) because of the ability within Java to easily convert data encapsulated in our class structure to XML data encapsulated in Java packages that handle Java-to-XML interactions. Furthermore, XML is a standard data persistence language that easily imports into many different types of data analysis software, such as Microsoft Excel.

1.2 Encapsulation of Model with Object Orientation

As mentioned above, we wanted to make use of object orientation to encapsulate our model into a class structure. Due to time constraints we minimized the class structure of our simulation program to include only the functionality and data encapsulation necessary to fully implement our model. Below is a breakdown of the functionality and data encapsulation for each class in our program.

1. `Car` This class encapsulates the initial speed of a car, determined by our normal distribution of car speeds, the current speed of a car, changed by our logic tree (see figure...), and the passing state of a car, either true of false.

2. `DataPersistence` This class encapsulates the functionality for persisting the simulation summary data into an XML file. This class handles the conversion of the encapsulation of the simulation summary data to data in an XML file using `java.io.*`, `java.xml.*`, `org.w3c.dom.*` packages.

3. `Main` This class is the entry point for the flow of execution of our simulation program. The interface for prompting a user for the simulation parameters and looping structure for running multiple simulations with the same parameters are encapsulated in this entry-point class.

4. `Map` This class encapsulates a collection of roads that a simulation tests. Each road in the `Map` represents a different road type that the simulation tests with a specific rule.

5. `Position` This class encapsulates a position on a road by the lane and slot (length-wise position) values.

6. `Road` This class encapsulates the properties of a road in our simulation, based on our assumptions. These properties include the number of lanes, the length of the road, the

traffic on the road (i.e., a collection of cars), and counts of the number of decisions, lane changes and slow downs that occurred during an iteration of our simulation program. Since our model assumes a hot air perspective, the array used to store the cars on a road has the dimensions of road, as seen from a hot air balloon, and the position of each car (in terms of lane and slot) are the indices of the car object in the array.

7. `Rules` This enumerator class encapsulates a designation of the rule used to navigate our logic tree (see fig...) during a simulation. The values stored in this enumerator class are `FREE PASSING, SINGLE PASSING, SINGLE DRIVING, NO PASSING`.

8. `Simulation` This class encapsulates a collection of different road types that different numbers of lanes and different traffic densities to test how these parameters, in conjunction with a rule, affect the flow of traffic on a road type.

9. `SummaryData` This class is an abstraction of vector that is used to store the summary data the we use in our statistical analysis. All of the variables defined as our summary data are encapsulated as private fields in this class.

1.3 Program Execution Flow

Now that we have discussed the overall class structure of our simulation program, we can explain the flow of execution of our simulation program. At the entry point of our simulation program, the user is prompted for the simulation parameters. Then, for each simulation specified by the number of simulations to run, our simulation program creates a new instance of the `Simulation` class and runs that simulation instance by calling the public instance method `Run()`. Next, within this public instance method (of the `Simulation` class), the collection of roads are generated and populated cars. The position of a car is randomly generated using a uniform distribution, and the speed of a is randomized using our normal distribution model for possible car speeds. After the roads have been populated with random cars, our simulation program begins looping through the number of iterations defined in the simulation parameters. Within each iteration, our simulation program implements our logic tree (see figure...) to calculate new positions for each car on each road. As soon as the execution of our logic tree (see fig...) finishes, our simulation program calculates the summary data for each road. Recall that each road in a simulation represents a different road type. After the simulation program finishes computing all the iterations specified by the simulation parameters, then the summary data for the whole simulation is calculated. At this point the simulation program is done computing iterations, so the simulation summary data is persisted in an XML file using an instance of the `DataPersistence` class. Finally, if there are multiple simulations defined by the simulation parameters, the simulation program iterates to the next simulation instance. For a graphic

1.4 Flaws and Improvements

Overall, the simulation program we developed implemented our model better than anticipated, especially within the time constraints. However, overall design flaws produced issues with the collection, persistence, and analysis of the simulation summary data. Initially, we

attempted to collect and persist data for every car on every road during every simulation. This method of collecting and persisting *all* data would have required terabytes of storage space, which we did not have access to, nor time to analyze. So, we changed the way in which we collected and persisted data so that only the summary data for each iteration was stored in output XML file. However, we encountered several problems, mostly from time limitations, trying to import and analyze the data in Microsoft Excel. Thus, due to time constraints, we decided to only output summary data for each simulation. Therefore, when we continue to investigate this problem, we plan to use a database, instead of XML files, to persist our traffic data. A database would allow our simulation program to persist more information with an organizational structure that makes the statistical analysis of the data easier. Additionally, using a database to persist more data with better organization, would allow us to develop an extension of our simulation program that plays back the movement of traffic our simulation program calculated. This extension would allow us to analyze the traffic flow visually, instead of purely analytically. This would present us with the opportunity to catch further design flaws in both our model and implementation of the model.