332:452: Software Engineering

Report 3: System Specification

Group 7

Vamshi Chilukamari

Akhilesh Maddali

Vladimir Samokhin

Sanket Wagle

Aditya Devarakonda

Project: Traffic Monitoring

URL: https://sites.google.com/site/452trafficmonitor/

5/3/2011

Breakdown of Contributions

All members contributed equally for this report.

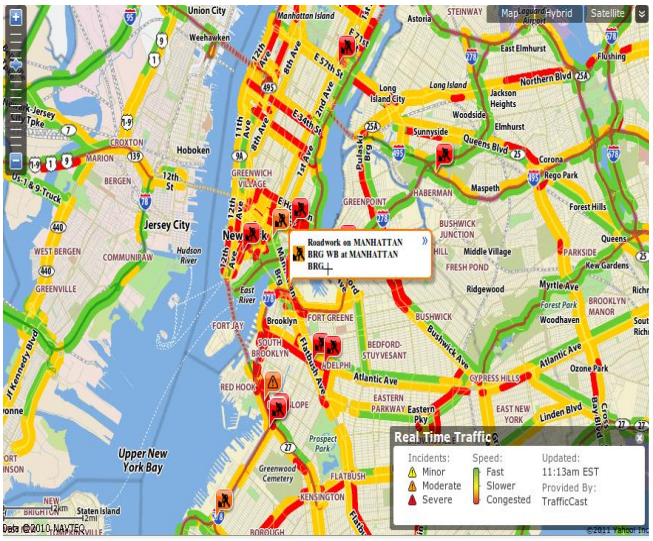
Table of Contents

Summary of Changes

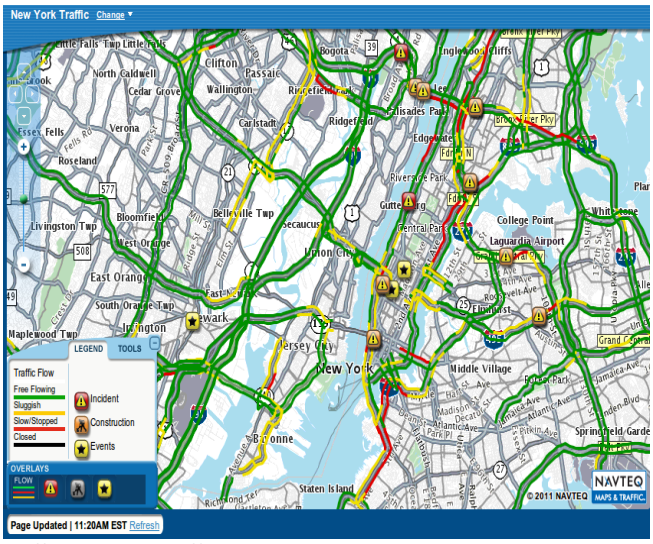
1. Changes to Customer Statement of Requirements to include the list of requirements and who the software is meant for.
2. Additions to the Glossary of Terms.
3. Changes to the Functional Requirement sections to reflect the final status of the project
4. Added clarification on why the Timer should be considered an actor.
5. Added a user case that was previously omitted. Traffic along a route use case
6. Addition of the "Extensions" and "Alternate Scenarios" sections to the Use Case descriptions
7. Updated system sequence diagrams with the system involved in all use cases.
8. Changes to the Nonfunctional Requirements to comply with FURPS+ criteria.
9. Added another diagram for Domain Model and changed the "Analyzer" component to the "DataAggregator" component
10. Added a descriptions about PHP symbols used in the Interaction diagrams.
11. Changes to the Interaction diagrams to reflect the final chain of events.
12. Additional information about how the calling process works in the interaction section.
13. Updated class diagrams, Data types and signatures and discussion about design patterns.
14. Changes to the System Architecture section, the figure is now colored to show classes provided by Google. The Database Schema has also been updated to show the final implementation
15. Algorithms section now contains descriptions on all of the iterations of the severity calculations and more information on how mapping from incidents to sectors is accomplished and how we differentiate between road intersections.
16. User interface section now shows a picture by picture log of our iterations on the GUI design and description about some issues with marker placement.
17. Added a History of work section with our views on the development and management process
18. Added a Conclusion and future work section which outlines our ideas and future potential for this software.

Customer Statement of Requirements

Many traffic monitoring services only generate information about current, live traffic incidents. In general, most of these services only collect data at specific intervals and use live traffic information to generate the incidence map. Current services such as, "Yahoo! Maps Live Traffic"[1] and "Traffic.com"[4] use this method to monitor traffic. Below are examples of two live traffic systems with incidents reporting on the Manhattan area of New York City.



Yahoo! Maps Live Traffic



Traffic.com Live Traffic

This project attempts to use historical traffic data to show the traffic trends along a particular route/area during various weather conditions and times of day. This new approach to traffic monitoring is necessary when considering the fact that, live traffic only reports incidents as they occur. If, for example, a route consistently has accidents and heavy traffic then it would be a bad idea to take that route. However, a live traffic monitoring system will only report incidents as they occur without notifying users that accidents occur along the route daily. Clearly this would be bad for the users of such a monitoring system. To rectify this problem, we intend to develop a system that will take historical traffic patterns into account. This represents a better monitoring method because it takes past trends into account in order to show the likelihood of encountering heavy traffic along the route/area regardless of whether there is currently an incident or not. In addition, to using the historical traffic data, one can use live traffic so that the users have all the necessary information in order to make a good route choice. In this project we seek to implement the historical traffic portion and, time allowing, extend the system to support live traffic. Google Maps[5] has a similar traffic monitoring system that combines historical traffic and live traffic.

Our system will support two methods of traffic reporting that users can choose from:

(1) “Incidents reports within an area”, where the user is given the choices to,
   A) “Select target area”. This target area will be one of
      I. “North”
      II. “Central”
      III. “South”
   B) “Time”. The time interval can be one of
      I. Intervals of 1 hour (example: “3AM”)
      II. “All”. Traffic reports for all time periods within the selected area are displayed.
   C) “Type of day”. This can be one of
      I. “Weekday”. The traffic reports spanning Monday-Friday are considered.
      II. “Weekend”. The traffic reports on Saturday and Sunday are considered.
   D) “Overlay Live Traffic”. The map will display live traffic in addition to the historical traffic if the user chooses this option.

Although the suggestion for this project was to focus on target area(s) in New Jersey, our system will be able to map all of New Jersey, except it will reduce the number of roads considered to major highways and thruways. As such, we intend to implement this service by partitioning New Jersey into 3 target sectors. This will allow us to sufficiently represent the traffic in a given area and give the users enough granularity so that they can monitor the incidents reports in their region. The idea here is that local roads are, in general, less congested than major highways and where accidents will have less impact due to lower speed limits and ample detours. This assumption will be verified throughout the course of the project but the principles behind the assumption seem accurate.

(2) “Incident reports along a route”. The route maybe selected from a drop down menu which lists several major highways and thruways.
   A) “Time”. The time interval can be one of
      I. Intervals of 1 hour (example: “3AM”)
      II. “All”. Traffic reports for all time periods within the selected area are displayed.
   B) “Type of day”. This can be one of
      I. “Weekday”. The traffic reports spanning Monday-Friday are considered.
      II. “Weekend”. The traffic reports on Saturday and Sunday are considered.
   C) “Overlay Live Traffic”. The map will display live traffic in addition to the historical traffic if the user chooses this option.

This service behaves in the same manner as the “Incident reports within an area”. The only difference is that it shows the traffic incidents along single routes. Given our idea of showing traffic along major highways, the “Incident reports along a route” service will give the option to look at the traffic patterns on one major highway. We intend to extend this portion such that the user will be able to zoom into their region (North, Central or South) while only looking at one route. However, this extension will depend on our progress throughout the semester.

The above services represent the user accessible front-end of our traffic monitoring system. The back-end of our system encompasses the "Weather collection" and "Traffic collection" services. These services will augment the front-end services by collecting the data, performing statistical analysis and returning the processed data to the front-end. Both of the back-end services are accessible and configurable by the administrator only, so users will not have access to the raw data nor the components that collect and process the data and requests.

The "Weather collection" service requires and allows update to the following parameters:

(1) "Name of highway". This parameter is used to retrieve weather along this route.
(2) "Time interval". This parameter will determine how long the collection service waits before requesting for more data.

The "Weather collection" service utilizes the administrator parameters to retrieve data from a weather forecasting service such as "weather.com" or "AccuWeather.com" along the specified highway. The service then parses the information that is retrieved and stores it in the database for future use in the statistical analysis by the system. The frequency of data retrieval is determined by the second administrator parameter (Time interval). The weather collection service will gather data at every time interval. Because our system is based on major highways, there is really no need to coordinate with the Traffic collection service because there are weather services that track weather conditions along major routes. This gives greater flexibility to our system because the weather and traffic collection services are independent and can perform their task simultaneously.

The "Traffic collection" service requires the same two parameters as the weather collection service.

(1) "Name of highway". This parameter is used to retrieve data along a specific highway.
(2) "Time interval". This parameter is used set the delay (sleep time) between two data retrieval cycles.

The "Traffic collection" service will access a live traffic monitoring service, such as "511nj.org" or "Yahoo! Live Traffic" in order to retrieve the incidents data. When the data retrieval is done, there are a few checks that the collection service needs to do before adding the data to the database. The first should be a check to see whether the incident already exists within the database. If the incident does exist then the service needs to check if the current incident report just retrieved was updated since the previous reference to the same incident. If this is also true then the database entry referring to the same incident is modified to reflect the update. If the condition was false and there was no update then the system should not add a new database entry and move on to the next task or next incident on the list. There is no dependency on weather so the traffic collection service can perform its task in parallel with the weather service. The only

requirement is that the administrator should configure both weather and traffic collection services to retrieve data on the same highway.

We envision that this traffic monitoring system will be useful to everyday commuters who use the major highways in New Jersey. Given the limited time-span that tourists would spend in the state, the system seemed irrelevant to their concerns given that they will not benefit from historical traffic when they tour for a very limited amount of time. Thus we have geared this project towards commuters, like ourselves, so that we can look at historical traffic along with live traffic in order to decide which route(s) to use. Here are some scenarios where we feel our system would be useful:

1. Travelling on major highways during rush hour. By viewing the historical traffic severity along the highway or in the region of interest the driver can make plans to take alternate routes in order to decrease travel time and get home faster.

2. Deciding which highways to use under specific weather conditions, time of day and day of week for road trips or long drives. Note: At this point our system distinguishes only between weekend and weekday but future extensions will expand upon this and will include individual days of the week as well as compound choices (weekday/weekend/holidays etc.).

3. This system would be useful for infrastructure improvements also, because we calculate severity taking all reported incidents along highways into account, so any construction process can take advantage of finding the most severe locations and making changes in those target areas.

Some interesting additions include the option to analyze routes for each individual. As of now the systems only shows traffic along defined, static routes. We intend to extend this system so that individuals can enter their starting point and destination and our system would calculate the travel time by taking into account historical incident data. This would estimate the actual travel time rather than the usual Google Maps travel time which only takes speed limits into account. Some other extensions could be to include small local roads into the severity calculations so that drivers will not have to guess which local roads they need to take to get to the major highways. Another interesting application would be to take social networking into account where users can report incidents via facebook or twitter, this would help with doing severity calculations for areas with few sensors or cameras.

List of Requirements:

- The system should collect traffic data for the roads being analyzed.
- The system should collect weather data for the regions being analyzed.
- The system should provide functionality for viewing statistical traffic data within a region (North, Central or South) of New Jersey.

- The system should provide functionality for viewing statistical traffic data along a route (highways/thruways) of New Jersey.
- The system should provide live traffic in order to facilitate the comparison between current traffic conditions and historical traffic conditions.

Glossary of Terms

Crontab:	A scheduling daemon which executes commands at administrator-set intervals. This is required by our monitoring system which collects weather and traffic data at defined intervals.
Graphical User Interface (GUI):	A type of interface that allows users to interact with the website using graphical components (buttons, checkboxes, etc.) and images instead of text-based commands.
Google Maps API:	An application programming interface (API) which provides a way for our system to access and use the maps from Google.
Weather Service:	A website or web service that contains weather data that can be collected, parsed and stored.
Traffic Service:	A website or web service that contains live traffic data that can be collected, parsed and stored.
MySQL Database:	A relation database management system that can be used to store the weather data and traffic data being collected. MySQL is one implementation of this type of system.
Administrator:	An entity which has privileges to customize, alter and update all facets of the traffic monitoring system.

User:	A person who intends to use the traffic monitoring system for the purpose of viewing historical traffic data in order to make a more accurate decision regarding route planning.
Severity:	A method of classifying the traffic data based on the type of incident and duration of the incident. Uses 5 levels of severity (Low, Low Moderate, Moderate, Moderate High and Severe).
Live Traffic:	Current traffic across New Jersey, as reported by Google Traffic. Includes roads that our system does not cover. Colors of the road outlines are green, orange or red based on current conditions.

Functional Requirements Specifications

By Users we mean any person who intends to use the traffic monitoring system for the purpose of viewing historical traffic data. This system is designed with commuters in mind but access is not limited to only commuters. Anyone can access it regardless of their purpose.

Stakeholders

- Users
- Administrator

Actors and Goals

- User
  - Initiating Actor
  - The user's goal is to access weather and traffic information pertaining to a chosen region or along a chosen route.
  - Includes but is not limited to: Commuters, Tourists, Drivers, Government officials for the purpose of collecting statistical data.
- Administrator
  - Initiating Actor
  - The administrator's goal is to set up the weather collection and traffic collection components of the traffic monitoring system to collect data for use in the statistical analysis process.
- Crontab
  - Initiating Actor
  - The goal of the Crontab is to automatically schedule the data collection system to collect and process data at administrator defined intervals. Crontab is not part of the system, because it is a fully-functional scheduling daemon provided by the UNIX operating system. Because this is not controlled by our system, we cannot count it as part of the system and ignore it as an actor.
- MySQL Database
  - Participating Actor
  - The goal of the database is to store the parsed weather and traffic information for future use in the statistical analysis process. The database is an intermediary type since the data collection side and the user interface must have access to it.
- Mapping Service (Google Maps)
  - Participating Actor
  - The goal of the mapping service is to present an interactive map used to display traffic information to the user.
- Weather Service (weather.com)
  - Participating Actor
  - The goal of the weather service is to provide weather forecast data, which the data

collection system can process and store for future use.

- Traffic Service (511nj.com)
  - Participating Actor
  - The goal of the weather service is to provide live traffic data, which the data collection system can process and store for future use.

Use Cases

Casual Description:

- UC1: ViewTrafficStatistics
  - Provides user with traffic conditions along a chosen highway or chosen region. See System Sequence Diagrams: UC1
- UC2: ScheduleWeatherCollection
  - Collect weather data based on administrator configurations at intervals specified, once again, by the administrator.
- UC3: ScheduleTrafficCollection
  - Collect traffic data based on administrator configurations at intervals specified, once again, by the administrator.
- UC4: AdministratorSettings
  - Let the administrator configure the database and Crontab for collection and storage.

Fully-Dressed Description:

- Use Case 1: View Traffic Statistics in a Region

Primary Actor: User.
Goal: To view traffic conditions within a region.
Stakeholders: Supporting actors are Google Maps, Database.
Precondition: Nothing important to mention.
Post condition: Nothing important to mention.
Main Success Scenario:

1. User enters his/hers location (North, Central, South) in New Jersey.
2. User enters the name of the highway he wants to travel on, day of the week, time of the day, weather condition, and incident severity.
3. System request Google maps for latitude and longitude using user's location.
4. System sends request to database for traffic and weather condition data along the highway.
5. System calculates statistics on the highway and retrieves

the map from Google maps.

6. System overlaps statistics and incident marks on the given map.

Extensions: The user selects an invalid sequence of inputs such as statistics

(Alternate Scenarios) along a route and within a region or invalid time option. The GUI will output an error and prompt the user to select a valid sequence of options.

- Use Case 2: View Traffic Statistics along a Route

Primary Actor: User.

Goal: To view traffic conditions along the highway.

Stakeholders: Supporting actors are Google Maps, Database.

Precondition: Nothing important to mention.

Post condition: Nothing important to mention.

Main Success Scenario:

7. User selects his/hers choice of highway in New Jersey.
8. User enters the name of the highway he wants to travel on, day of the week, time of the day, weather condition.
9. System requests Google maps for latitude and longitude using user's location.
10. System sends request to database for traffic and weather condition data along the highway.
11. System calculates statistics on the highway and retrieves the map from Google maps.
12. System overlaps statistics and incident marks on the given map.

Extensions: The user selects an invalid sequence of inputs such as statistics

(Alternate Scenarios) along a route and within a region or invalid time option.. The GUI will output an error and prompt the user to select a valid sequence of options.

- Use Case 3: Crontab for Weather Conditions

Primary Actor: Crontab

Goal: To collect weather data along the highway and store the data in our data base at certain interval of time.

Stakeholders: Supporting actors are weather channel, data base and administrator.

Precondition: Administrator.

Post condition: Current weather data is stored into the database.

Main Success Scenario:

1. Crontab initiates weather collection process.
2. System request weather channel for weather conditions along the highway at a particular interval of time.
3. System gathers data and puts the data into the database.

Extensions: **(Alternate Scenarios)**

1. The weather collection script fails to parse the weather information. The administrator can read the error logs in order to determine and fix the problem.

2. The weather collection script fails to add the data to the database. The administrator is notified by error logs and can fix the problem based on the information in the error logs.

Use Case 4: Crontab for Traffic Conditions

Primary Actor: Crontab.

Goal: To collect traffic data along the highway and store the data in our data base at certain interval of time.

Stakeholders: Supporting actors are weather channel, database and administrator.

Precondition: Administrator.

Post condition: Current weather data is stored into the database.

Main Success Scenario:

1. Crontab initiates traffic collection process.
2. System request Yahoo traffic for traffic conditions along the highway at a particular interval of time.
3. System verifies that each incident reported is new and then adds into database otherwise just ignores that incident so there is no duplication of data.
4. System gathers data and puts the data into the database.

Extension: **(Alternate Scenarios)**

1. The traffic collection script fails to parse the traffic information. The administrator can read the error logs in order to determine and fix the problem.

2. The traffic collection script fails to add the data to the database. The administrator is notified by error logs and can fix the problem based on the information in the error logs.

- Use Case 5: Administration

Primary Actor: Administrator.

Goal: To design traffic and weather collection criteria and create database to store the data.

Stakeholders: Supporting actors are database and Crontab.

Precondition: Nothing important to mention.

Post condition: Criteria are finalized and database is created.

Main Success Scenario:

1. Administrator supplies 511nj.org traffic and weather channel to supply data.
2. Configuring the database and then store data into it using Use Case 2 and Use Case 3.

Extension:

(Alternate Scenarios)

The collection system is not configured to parse traffic and weather. The administrator can physically change the code to reflect these changes.

Nonfunctional Requirements

- **F**unctionality Requirements for "Traffic Monitoring System":
  - "Traffic Monitoring System" shall provide the function of viewing traffic statistics along a highway.
  - "Traffic Monitoring System" shall provide the function of viewing traffic statistics within a region.
  - "Traffic Monitoring System" should allow the user to overlay live traffic in addition to the statistics.
  - The system should allow the user to select specific weather, day of interest (weekday or weekend) and time of interest (by entering the hour in 24-hour format).

- **U**sability Requirements for "Traffic Monitoring System":
  - "Traffic Monitoring System" should provide the user with an easy to use interface which requires minimal effort by the user. (*See User effort estimation for a detailed description)*
  - "Traffic Monitoring System" shall provide documentation of the user interface in order to help the user understand the functionality provided and how to use the interface provided. This document will be present in the form of an html page called "Help"

- **R**eliability Requirements for "Traffic Monitoring System":
  - "Traffic Monitoring System" shall have no failures caused by the software system itself. Any server downtime shall be rectified within 2 hours of first incident report. *(Note: Our system uses an ECE server, so we cannot guarantee server availability beyond what we can control with user-level access only. We do not have root privileges for the server).*
  - The downtime for our system shall be limited to hardware/server issues.

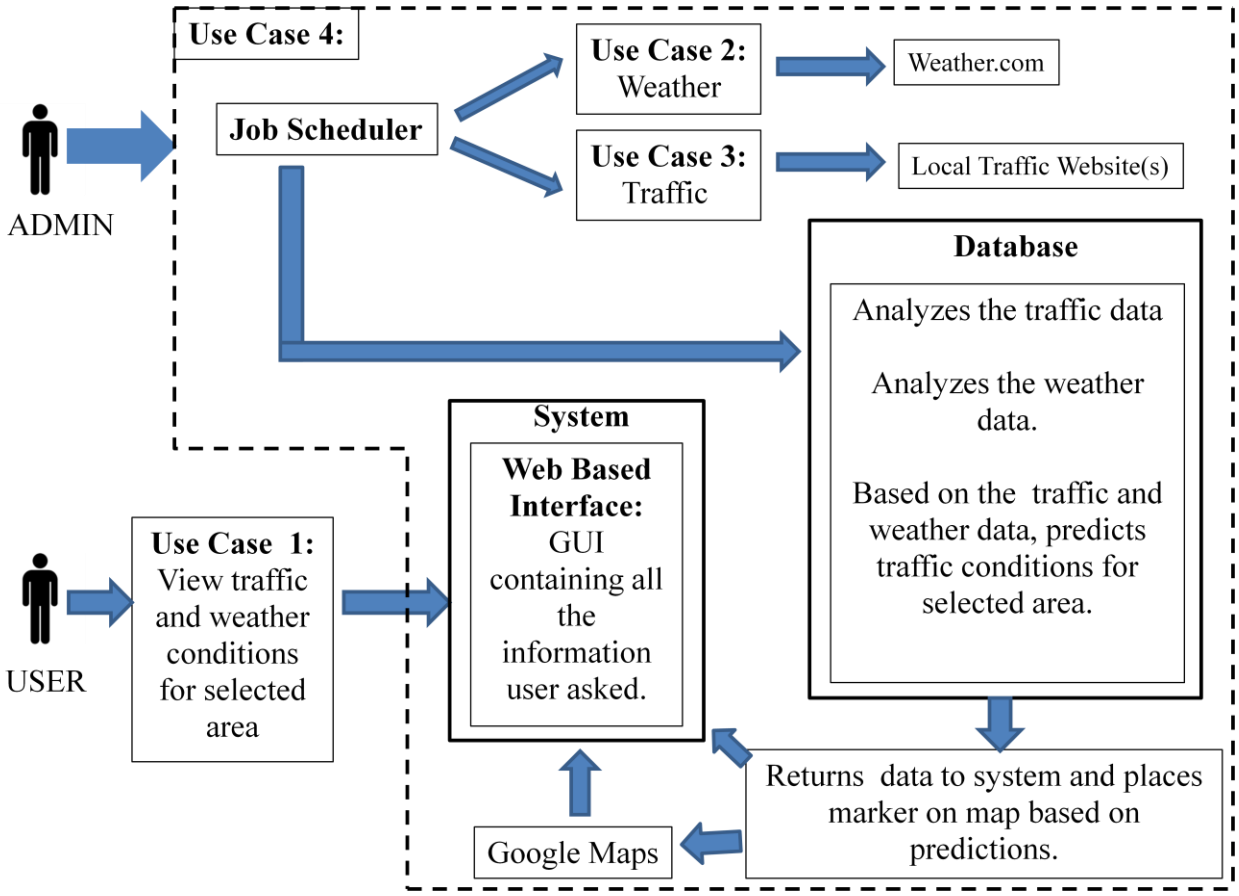
- **P**erformance Requirements for "Traffic Monitoring System":
  - "Traffic Monitoring System" should process the data and display the result within 15 seconds.
  - "Traffic Monitoring System" database should have sufficient amount of space for the data but not too large in order to decrease access time and increase throughput.
  - Multiple users should be able to access the website without experiencing slow response time.

- **S**upportability Requirements for "Traffic Monitoring System":
  - "Traffic Monitoring System" shall be designed in such a way that it requires little support for the data collection system. It shall be designed such that once the data collection system is configured; there would be no need to modify it other than to

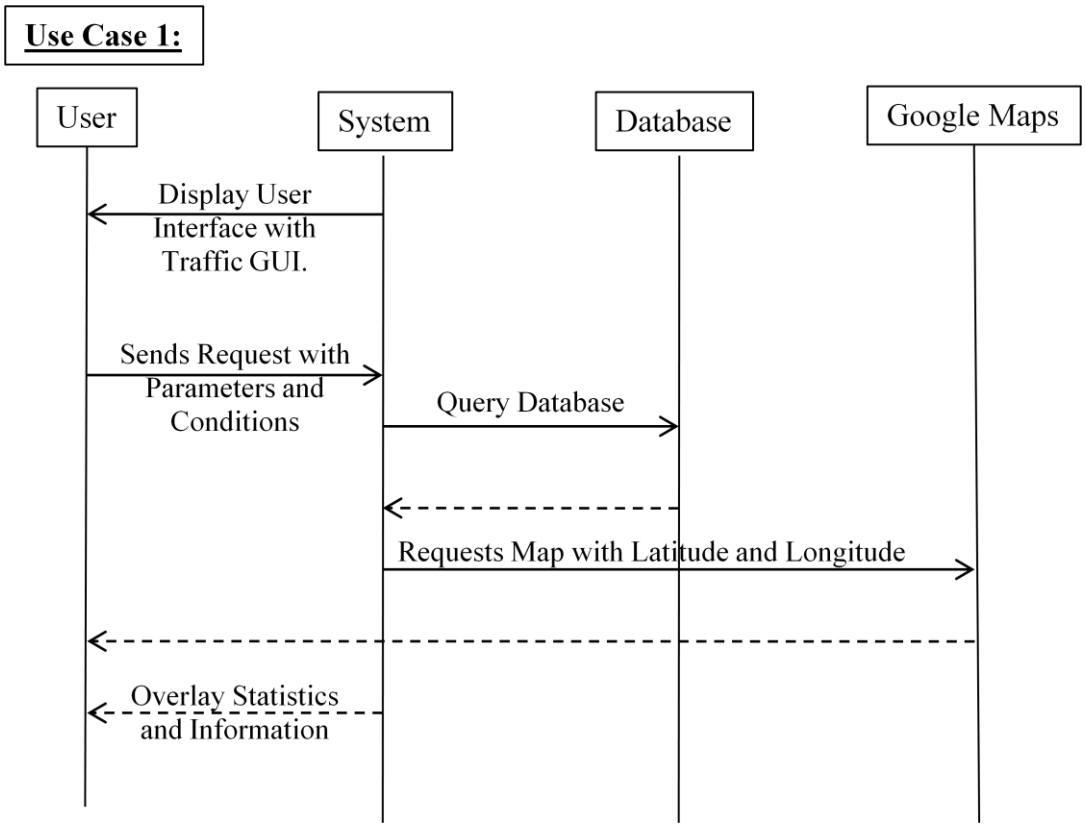
extend the scope of the project to include more highways.

- "Traffic Monitoring System" shall be portable in the sense that we can move the data collector system, GUI system and analysis system to another server without having to change the content of our code.

Use Case Diagram:

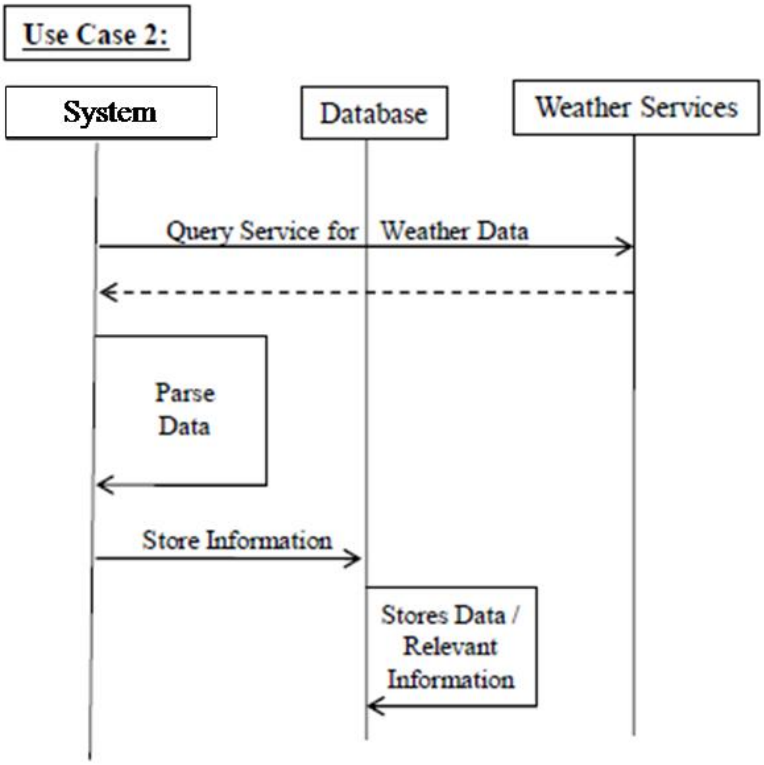


System Sequence Diagrams

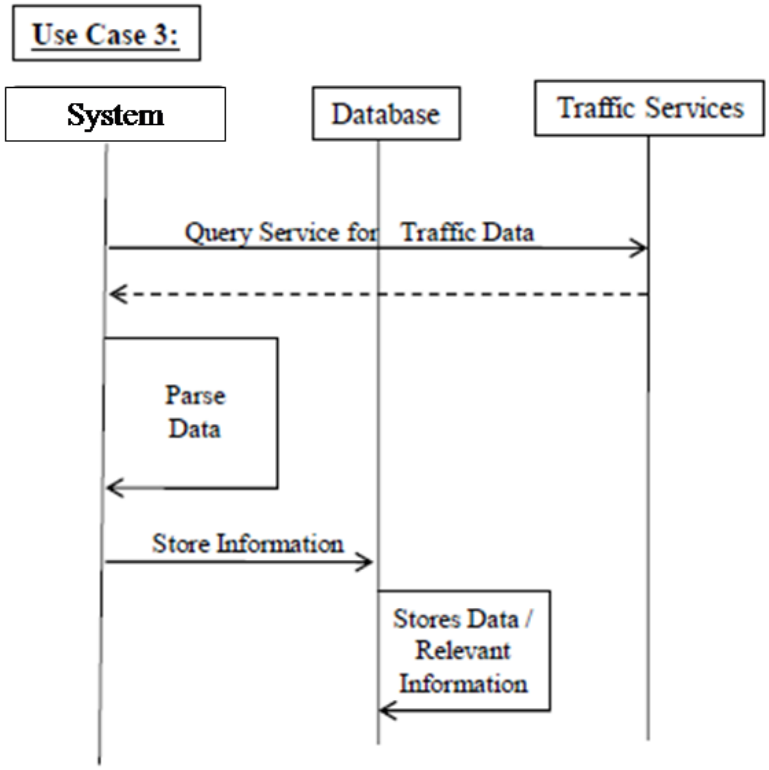


In UC 1 the system displays the user interface to the customers and requests the information. The user then enters the information and sends it back to the system. Then the system then does a query search in the database for the requested traffic data, and retrieves them information. The system then pings Google maps with the latitude and longitude to recover the map of the selected region. Next, Google sends this information back to the system, which is finally returned to the original user.

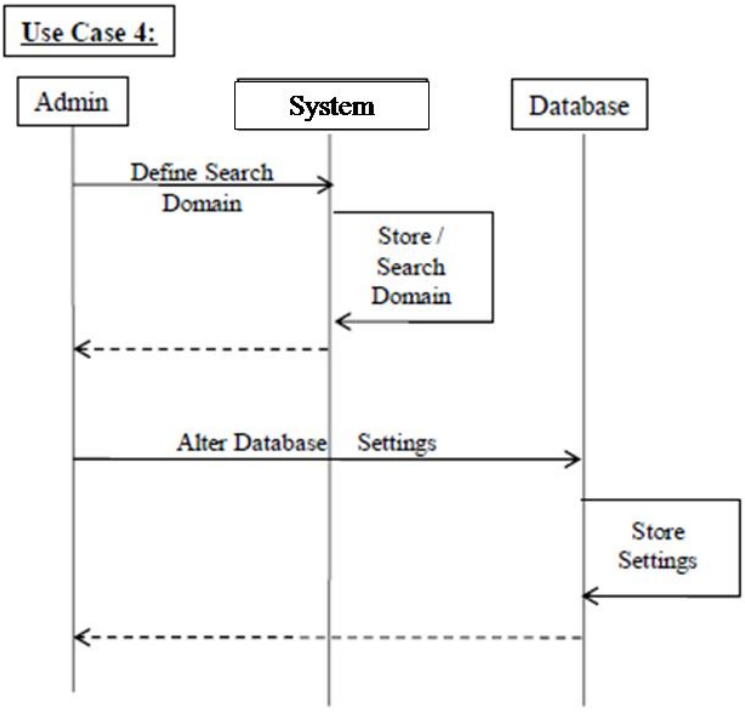
Note: This sequence diagram is valid for statistics along a route as well as statistics within a region. The only difference is that the final step: “Overlay Statistics and Information” will be different.



In UC 2 the System pings the weather services for the data for defined intervals. Then the weather service sends the data back to the System, which then stored it into the database. The System then parses through the data, takes the relevant data and sends it to the database. Finally the database stores the information into the database.



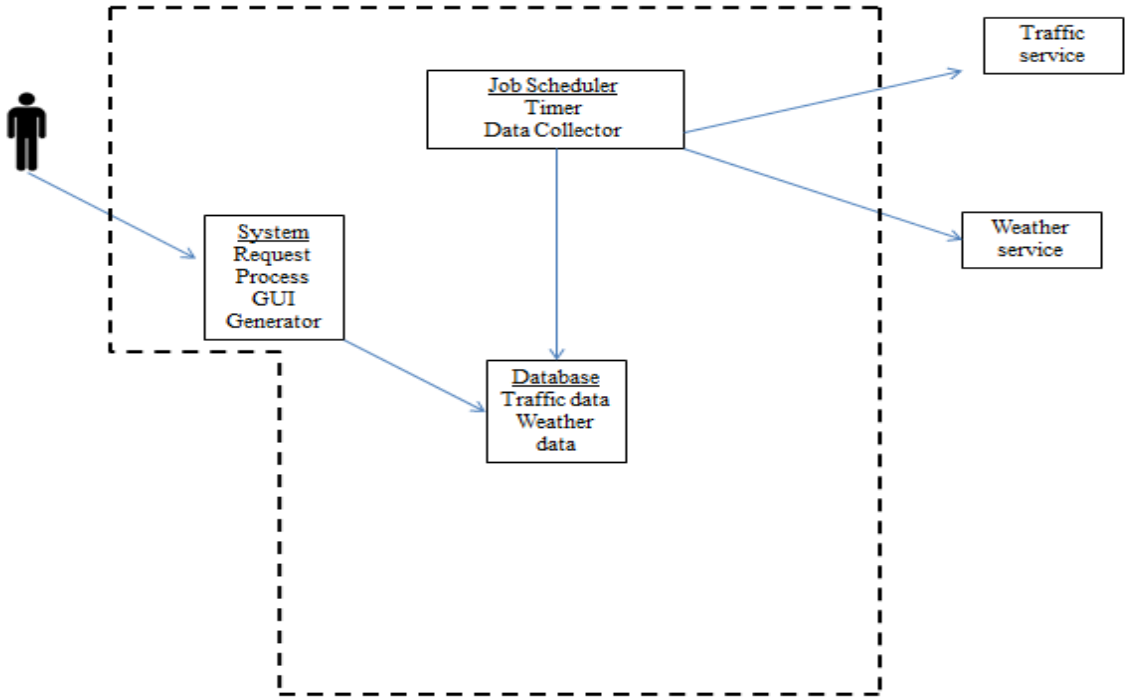
In UC 3 the System pings the traffic services for the data for defined intervals. Then the traffic service sends the data back to the System, which then stored it into the database. The System then parses through the data, takes the relevant data and sends it to the database. Finally the database stores the information into the database.



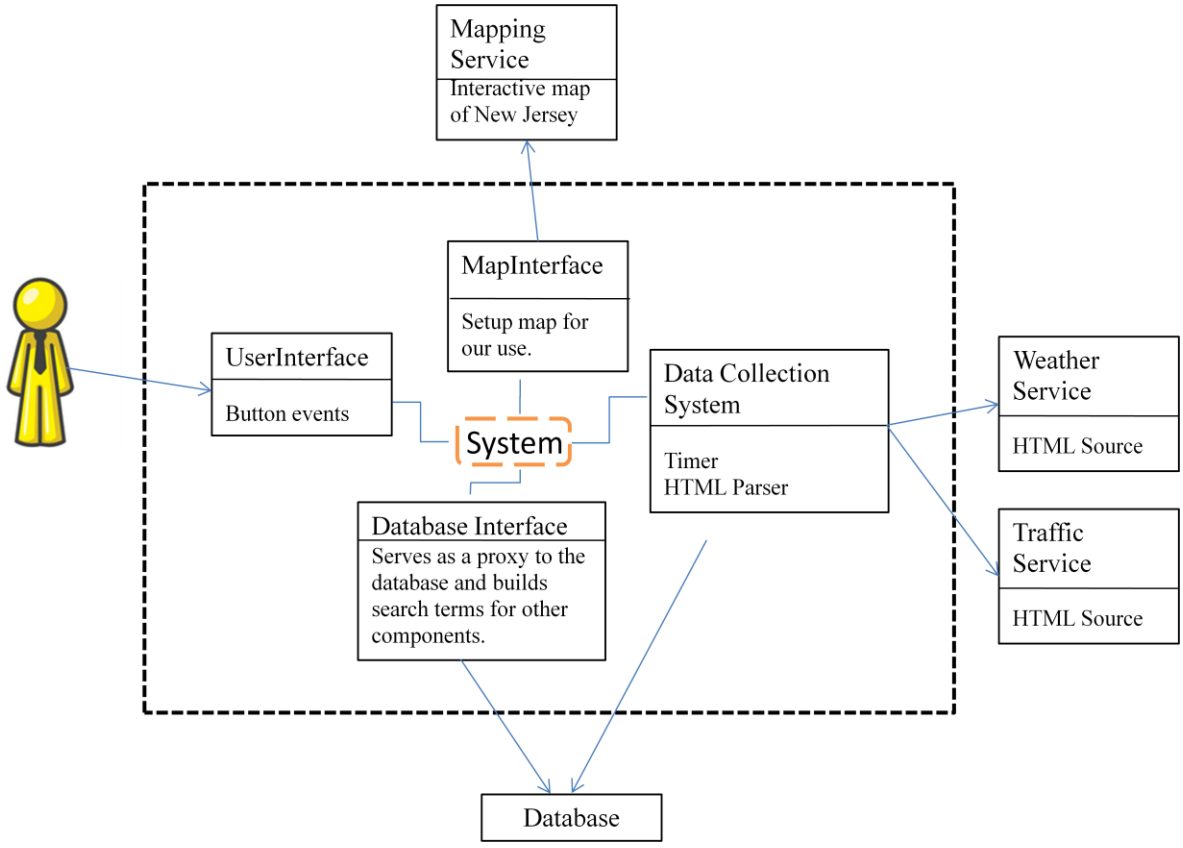
In UC4 the administrator provides search criteria to the System so it can gather traffic and weather condition data. System stores search domain setting and collects the data. Administrator has the access to database to fix the database if necessary.

Domain Analysis

Domain Model: Prior to 2nd Iteration



After 2nd Iteration



Concept Definition:

The Concept Definitions and all other descriptions have been updated to reflect the final design. However, we have provided the first iteration Domain Model equivalent of each concept.

System:

- Encompasses the following concepts
  - GeoCoder (MapInterface):
    - GeoCoder interacts with Google Map's API and makes function calls in order to place markers, set zoom levels and set map type (terrain or satellite).
  - Graphical User Interface (User Interface)
    - Displays the interface for the user. This includes radio buttons for selecting various key inputs such as region, road, interested weather scenario, interested day selection and an option to overlay live traffic.
  - Database Interface
    - Queries the MySQL database depending user selection made in the GUI. It will return all the relevant data corresponding to the user selection.
  - DataAggregator (Was not anticipated during Domain Model)
    - Takes the raw data from the Database Interface and maps it to markers based on Latitude and Longitude values and calculates the severity level of each marker.
  - Severity Calc (Was not anticipated during Domain Model)
    - Calculates the severity of a single marker by taking all the incidents mapped to the marker and calculates their severity. Once all incidents' severity are calculated they are averaged in order to get the severity for the marker.

Database (Same name):

- It is used to store traffic and weather data as database tables.

Job Scheduler (Data Collection System):

- Job Scheduler is used to schedule the data collection system to gather traffic and weather conditions at particular intervals of time. It encompasses the Crontab Timer and the Data Collection scripts.

Association Definition:

System:

System accesses data from the database and provides traffic condition at that point of the day to the user so that user can reach his destination using the highway with least traffic and reach safe on time. It also encompasses the Data Collection System which parses the HTML sources of the weather and traffic services.

Database:

It is used to store traffic and weather condition data along the highway and is accessed by the system to provide user with traffic conditions so it takes user least time to reach his destination safe.

Attribute Definition:

System:

1) User Interface: This is used to provide an interface so user can interact with the traffic monitoring system and supply the highway, day of the week , incident severity and location to the system for processing. GUI is used by the monitoring system to give users traffic predictions along the highway/region they want to view.

2) Database Interface: This is used to access the weather and traffic information stored in the database and passes it on to the Map Interface so that markers are placed accordingly on the map.

3) Map Interface: This is used to access the map provided by a mapping service (MapQuest, GoogleMaps). It will be used to place informative markers on the map based on the information extracted from the database interface.

4) Data Collection System:

1. Timer(Crontab): It is used by the collection system to initiate the parsing and data collecting process.

2. Parsing Scripts: Access the traffic and weather services for collection and parsing of data, once triggered by the timer it parses the html source codes, extracts key terms and saves the data to the database.

Database:

1) Traffic data: This is used to store traffic data.

2) Weather data: This is used to store weather data.

System Operation Contracts:

- Parse data:
  Precondition: Data is stored in the database

  Post condition: it makes sure data is parsed into weather conditions data and traffic condition data

- Service request:
  Precondition: There is enough data in the database to be provided to the user

  Post condition: This operation returns user with markers overlaying on the map. If there is no data in database for a particular highway or location, it returns map with no markers on it suggesting to the user that there is no data for the criteria provided by the user.

- Calculate Results:
  Precondition: Criteria provided by the user must be available in the database.

  Post condition: Statistical averaging of the target traffic area/route is computed and returned.

Notation/Symbols Description

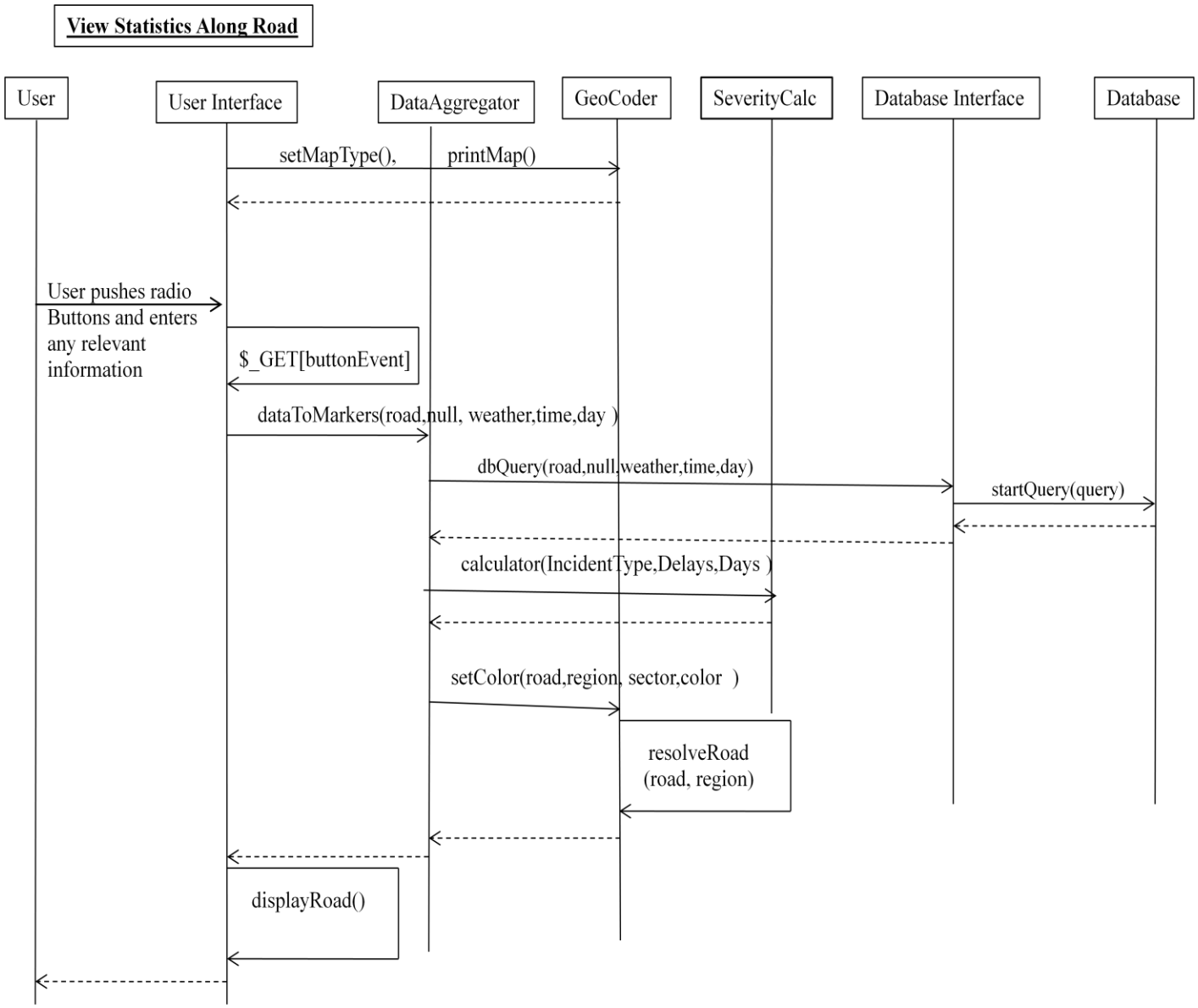
The interaction diagrams below uses some specific symbols related to PHP. We enumerate some of the functions here and their corresponding platform-specific relationship.

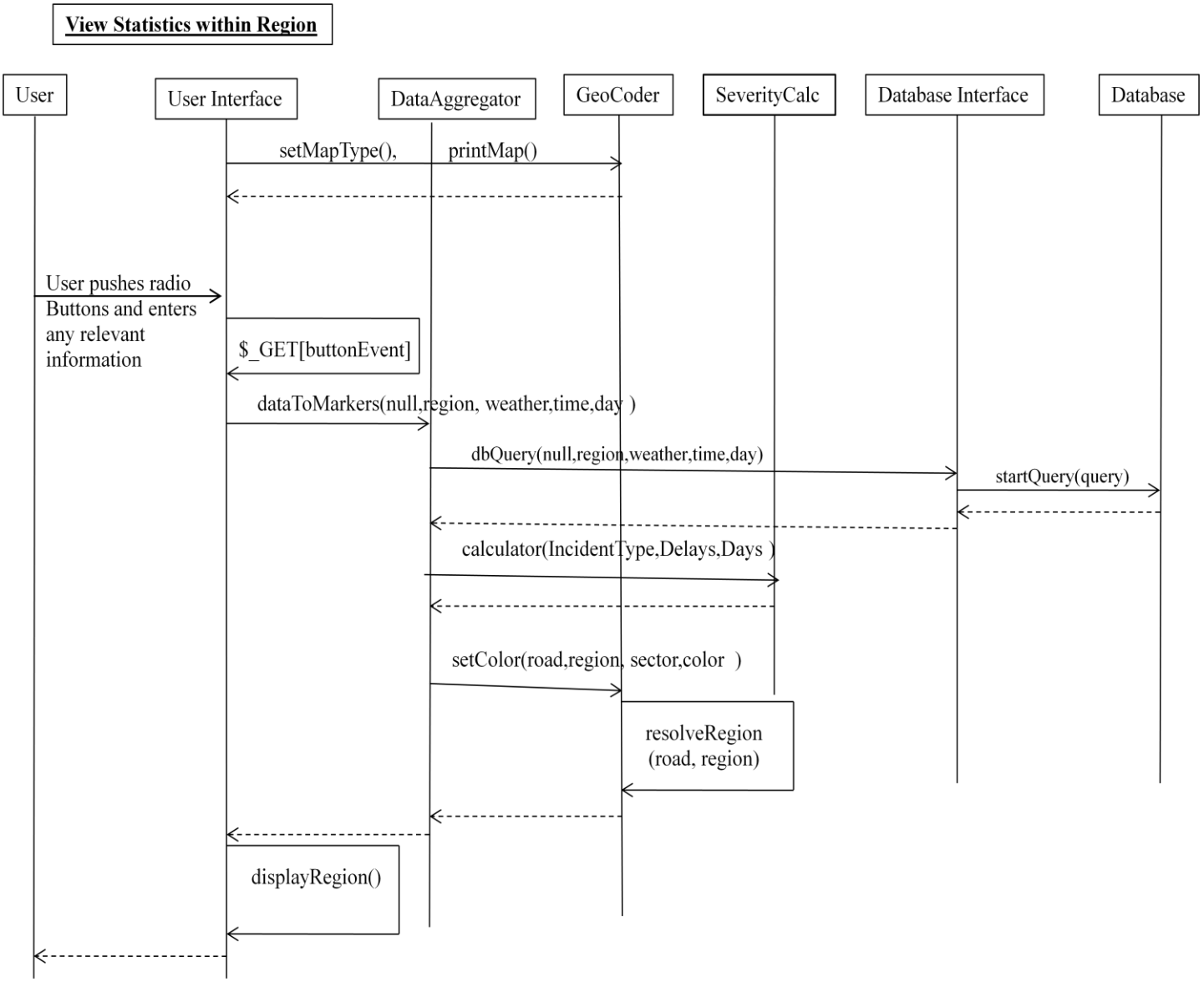
$_GET[…] – This is a PHP, built-in function which check if any GUI components (menus, radio buttons, checkboxes etc.) are triggered (i.e. clicked or selected by the user). Hence it behaves like an event handler for the GUI buttons. We can subsequently check which buttons are selected and make the appropriate function calls in order to process the request.

All of the other functions are similar to C/C++/Java functions and are platform-agnostic.

The '$' symbol is a PHP related syntax which specifies a variable. The '$_' specifies a GUI related function.

Interaction Diagrams





Note: For both of the interaction diagrams we have omitted the GoogleMap and JSMin classes. These classes setup the query to Google Maps repository to get and set the map. With the interaction diagrams we intend to show the interaction of our system with components that we have designed. The setMap and printMap methods are invoked from the GeoCoder which makes calls to GoogleMap and JSMin to finalize the map setup. These methods are implemented by Google and thus, we do not have an in depth understanding of how the methods work. We also, show the input arguments to dateToMarker, dbQuery, calculator and setColor functions in order to show the difference in input arguments for the Along a Highway and Within a Region options. The “null” values are key to differentiating between the two options hence we have emphasized them in the diagrams.

Interaction Diagram Descriptions:

Method	Description
$_GET[buttonEvent]	Called when the user changes the radio button options for the regions (North, Central, and South) or selects the "Road" radio button.
dataToMarkers(road,region,weather,time day)	This is the call to the DataAggregator module which sends the button and menu selections from the GUI as inputs. If any button or option is unselected then the corresponding input is set to null.
dbQuery(road,region,weather,time,day)	Form the database key terms based on the user inputs and differ control to the database interface class. Any input that is unselected defaults to null.
startQuery(query)	Interfaces with the database and returns the database tables containing the entries matching the user options. The "query" is a string written in MySQL query format. It attempts to get data from the database corresponding to the user's request.
setMapType(), printMap(), defaultView(),	It initializes Google maps and displays it on user interface. These are functions provided by JSMin and GoogleMaps classes. These are implemented and provided by Google.
Calculator(IncidentType,Delays,Days)	This method calculates the severity for events contained in a single marker. This function is called within DataAggregator(…) once the aggregator finishes mapping incident data to markers.
setColor(road, region, sector, color)	This function sets the severity level color for a single marker based on the value returned by the severity calculator.
resolveRegion()	This method sets up the GoogleMap marker settings to reflect the color described by setColor.
resolveRoad()	Similar to resolveRegion except it sets the Google Maps marker settings to reflect the color described by setColor.
displayRoad()	Sets the map returned by the geocoder to the road specified by the user and preloaded with the markers with corresponding colors.
displayRegion()	Sets the map returned by the geocoder to the region specified by the user and preloaded with the markers with corresponding colors.

Calling Process

When the web page is first loaded the setMapType(), printMap() and defaultView() functions are called to initialize the map display and create the GUI. The user fills in the form with the criteria for the traffic query and when a region is selected the $_GET[buttonEvent] function is called which sets local variables based on the user's GUI selections.

When a road is selected $_GET[buttonEvent] function is called, much like for a region selection. Once the $_GET function is complete the aggregator() function is invoked and the input parameters of road, region, weather, time and day_of_week are set. If any of the GUI buttons are unselected then all corresponding values default to null. This is an important aspect of the GUI because if any value is set to null then the type of query changes, hence the null values are very helpful within the DatabaseInterface class in order to determine which sets of queries need to be used.

The DataAggregator queries the database using the dbQuery(road, region, weather, time day) function provided by the DatabaseInterface. This function in turn uses startQuery() function to create a connection to the MySQL database and retrieve the necessary data corresponding to the input parameters. Once the data is retrieved the dbQuery function returns the data. The aggregator then maps all of the retrieved data into markers based on the Latitude and Longitude values as well as the Road_Name. The mapping of incidents to markers is discussed in greater detail following the Class Diagrams.

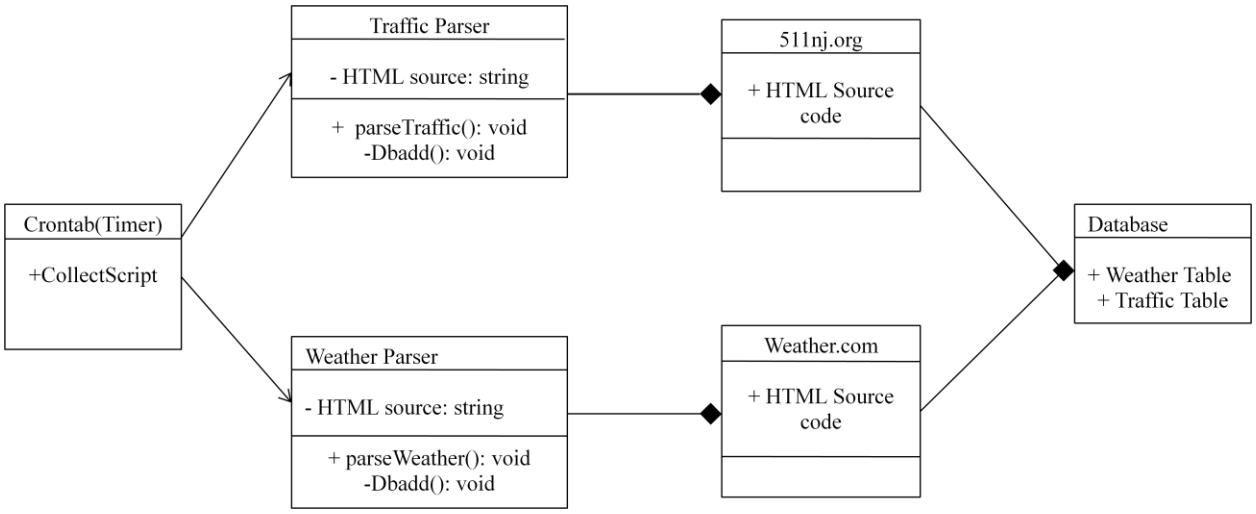
Once the mapping of data to markers is done, the severity values for each marker is calculated. This calculation is done by making a call to the calculator() function provided by the SeverityCalc class. The SeverityCalc class uses the Incident_Type, duration of delay and date since creation to determine which of the five levels of severity this marker belongs to. Note that the severity is calculated for each marker (i.e. it has more than 1 incident mapped to it) so the severityCalculator needs to calculate the severity of each incident and then average those values in order to get the final severity level of a single marker. This process is repeated for all markers.

Once the severity of each marker is calculated, the color for each marker is set by calling the setColor function and inputting the parameters: road name, region, sector and color. The setColor method sets the the image file corresponding to a marker to a specific color. After this process the aggregator is finished and the control shifts to the GUI which then makes a resolveRegion or resolveRoad call (based on initial user selection). These functions essentially set the GoogleMap markers to load the image files for a region or for a road. Once this is done, the final call to displayRoad or displayRegion are made. These functions print the map to our website so that it becomes viewable to the user.

Class Diagrams and Interface Specification

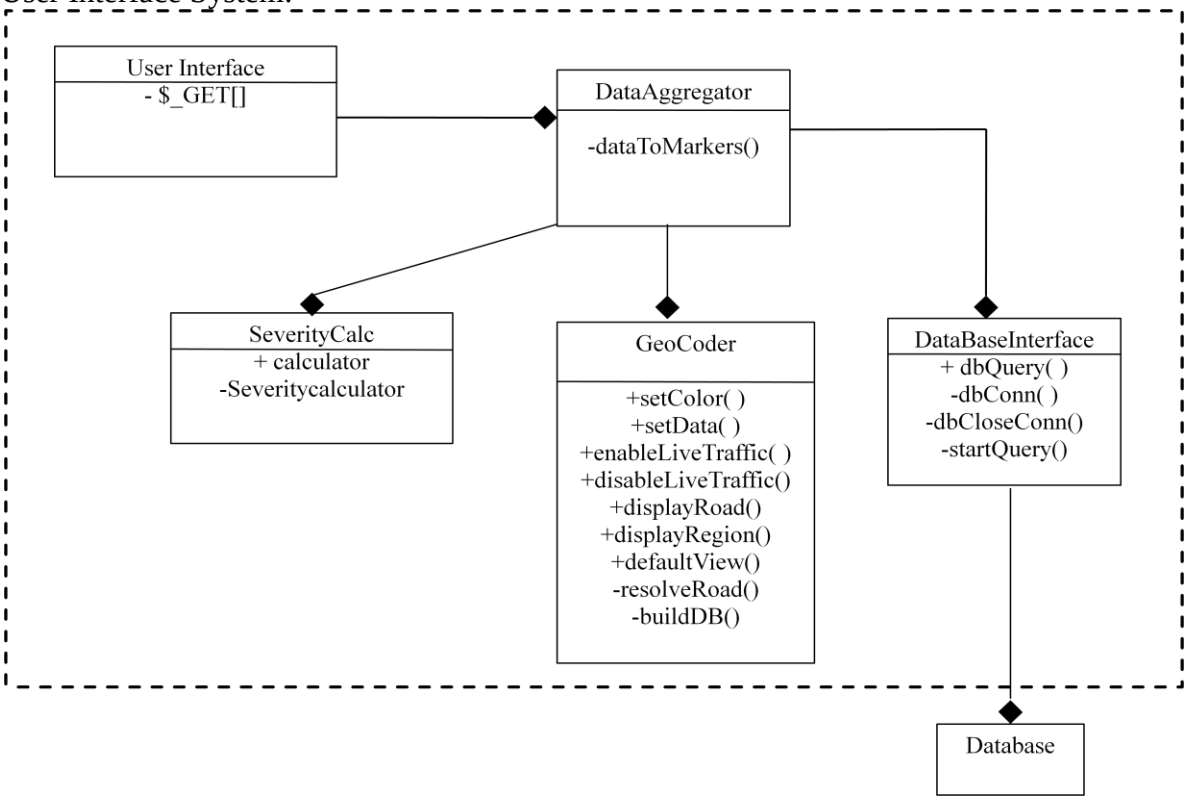
Class Diagrams:

Data Collection System:

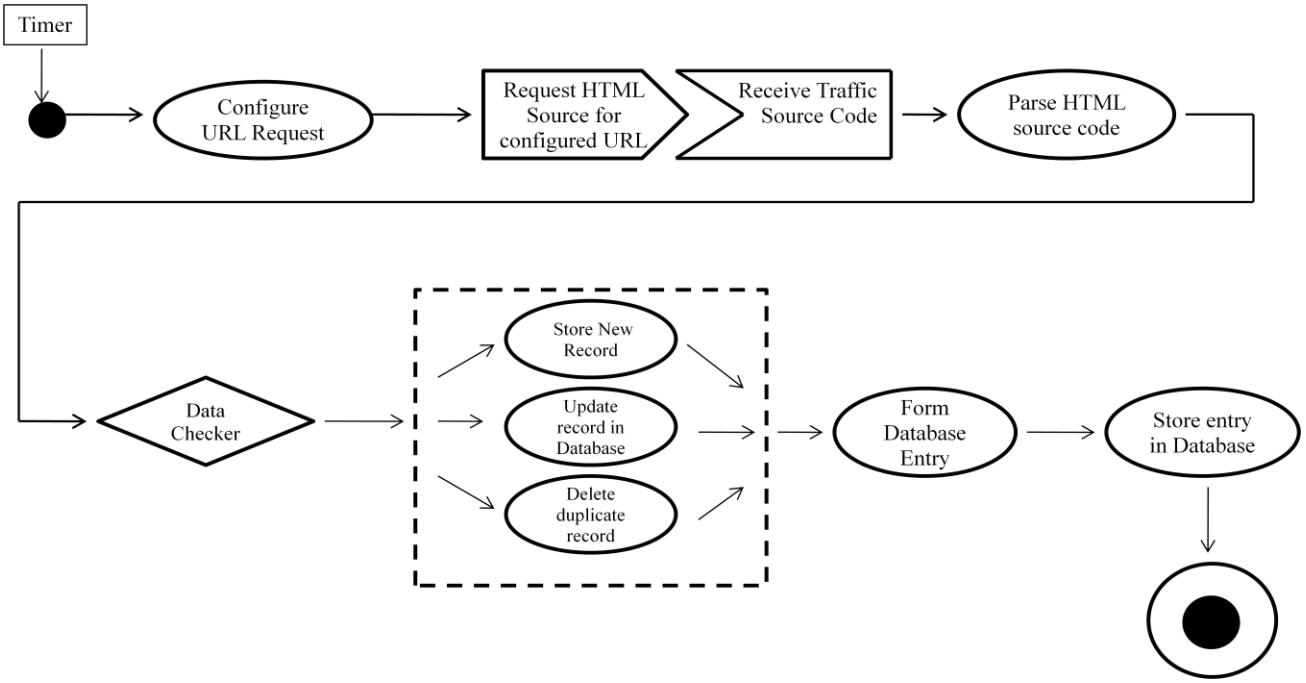


A more detailed description of the data structures and local variables is given in data types and operation signatures section. These diagrams have been condensed to convey the main goals and data structures which go into giving the system its functionality.

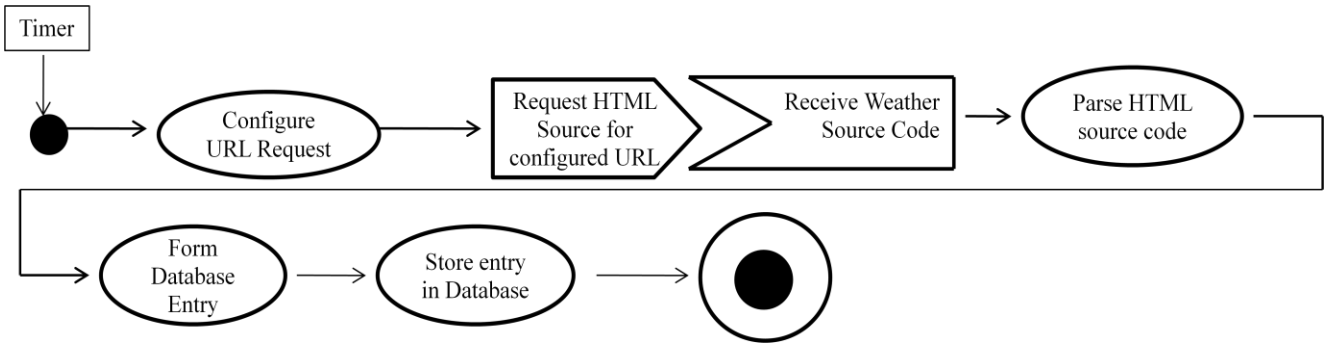
User Interface System:



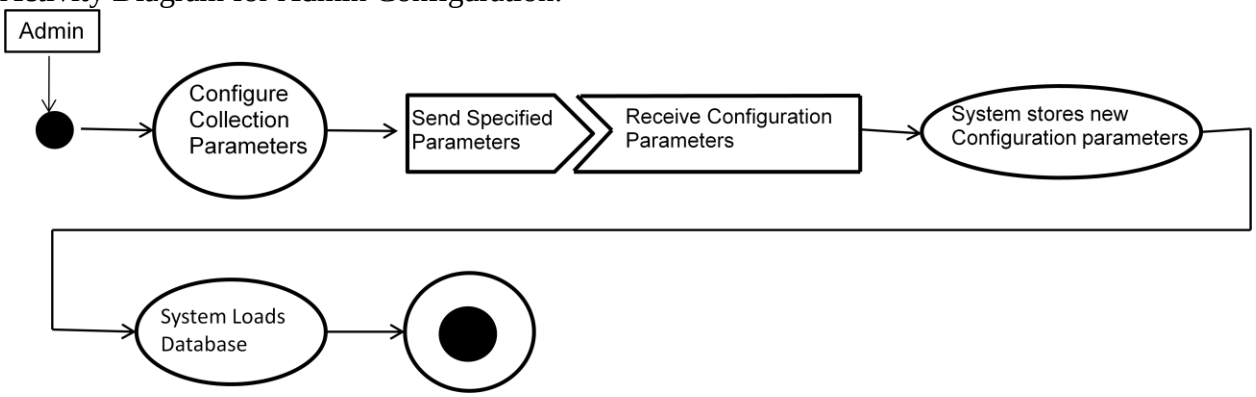
Activity Diagram for Traffic Collection:



Activity Diagram for Weather Collection:



Activity Diagram for Admin Configuration:



Data Types and Operation Signatures

User Interface System

UserInterface
- $MAP_OBJECT - $road - $region - $weather - $time - $day
-$_GET[]

DataAggregator
- IncidentArrays for all markers - IncidentCounters for all markers - IncidentArrays for all highways - IncidentCounters for all highways
- dataToMarkers()

GeoCoder
$MAP_OBJECT Multi-dimensional arrays for each markers containing the information: $lat,$long,$data,$color
+setColor() +setData() +enableLiveTraffic() +disableLiveTraffic() +displayRoad() +displayRegion() +defaultView() -resolveRoad() -buildDB()

DatabaseInterface
$sql $trafficData $weatherData
+ dbQuery() - dbConn() - dbCloseConn() -startQuery()

SeverityCalc
$incident5 :- array containing incidents which are given severity level 5. $incident4 :- array containing incidents which are given severity level 4. $incident3 :- array containing incidents which are given severity level 3. $incident2 :- array containing incidents which are given severity level 2. $incident1 :- array containing incidents which are given severity level 1.
+ calculator() - Severitycalculator()

Data Collection System

Timer (Crontab)
- tParse : Traffic Parser - wParse : Weather Parser

The timer method uses built in function and will not create new function utilized anywhere else. It will contain instances of the traffic parser and weather parser. This design decision was made with the consideration of portability. We can instantiate new Traffic Parsers or Weather Parsers anywhere else without worrying about having a pre-configured hourly data collector. Now, the callee program can instigate the collection based on other factors and events rather than just time.

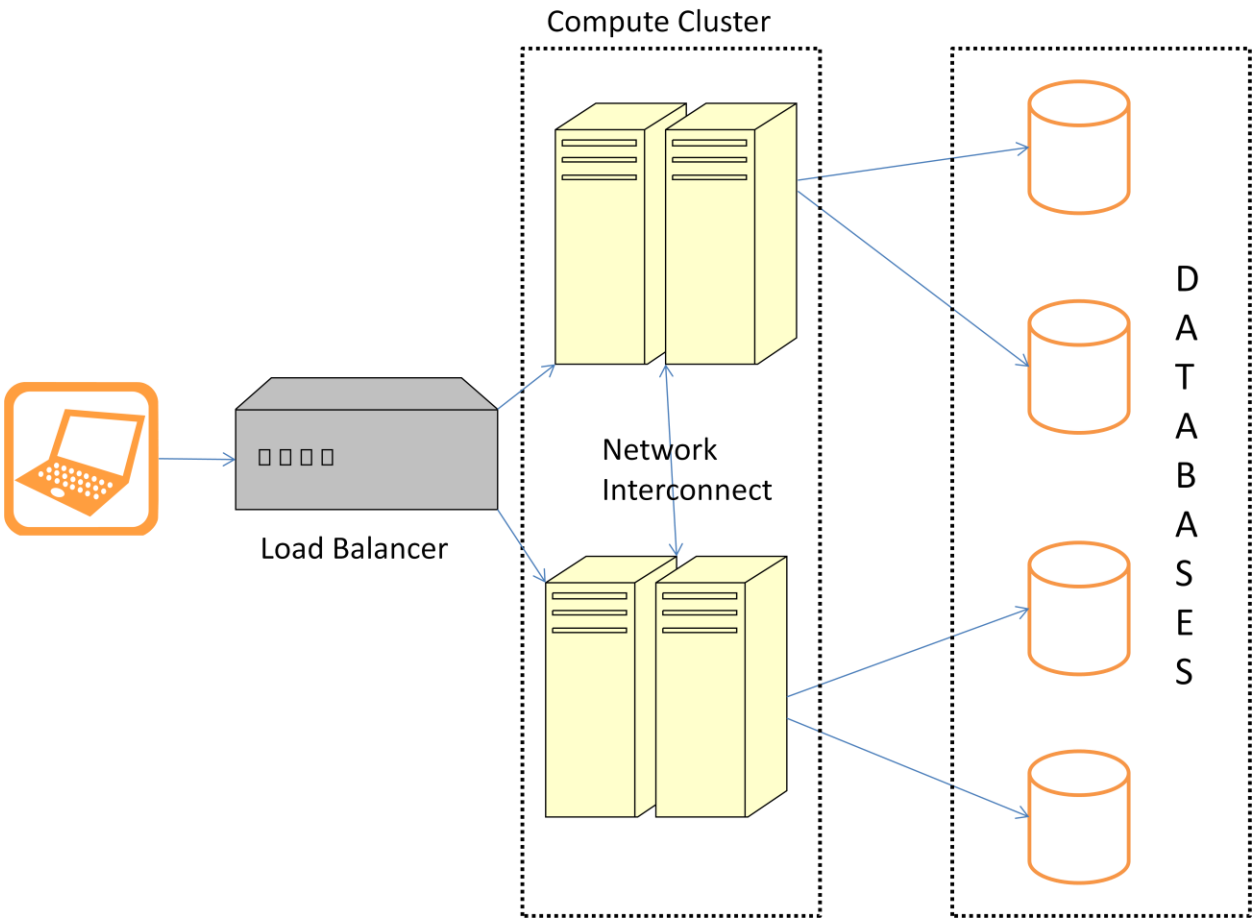
Traffic Parser
- myhtml : string - htmlTag : string - tagCount : int - incidentList : string - roadName : string - incidentDesc: string - Latitude : float - Longitude : float - sock : http socket
+ parseTraffic() - dbAdd()

Weather Parser
- myhtml : string - htmlTag : string - tagCount : int - parseList : string - cityName : string - condition: string

- Temperature : float - precipitationType: string - sock : http socket
+ parseWeather() - dbAdd()

Design Patterns:

In order, to improve this design for the future, we see the implementation of the multi-threaded and distributed computing design patterns. Due to the data parallelism present in our system we can take advantage of multiple threads to access the database and retrieve information. This will improve the performance of the severity calculations since multiple processors can be taken advantage of. We feel that a distributed computing paradigm would also be beneficial given the considerable amount of data we could potentially cache. Currently we have allowed the database to collect data for ~ 2 month and the current size of database is in the order of 100's of MB. So if we were to continue collection throughout the year, then we would easily have Gigabytes of data. If we were to employ a distributed system then we envision a system that is structured as follows:



System Architecture and System Design

Architecture style:

Traffic monitoring system follows client/server software architecture. There is a database server used to save data for traffic along the route and weather. Client communicates with the database server using the web page, which is saved in the database server. Server verifies the user criteria and processes the request to generate result and is displayed on the web page. The architecture has three subsystems: User Interface Subsystem, Collector Subsystem and Database. The User Interface Subsystem is used to display the output to user. The Collector Subsystem is to collect and parse weather and traffic data. The Database Subsystem stores the database tables for the collected traffic and weather data.

User Subsystem has following classes associated with it:

➔ class UserInterface , class DataAggregator, class SeverityCalc, class GeoCoder , and class DatabaseInterface.

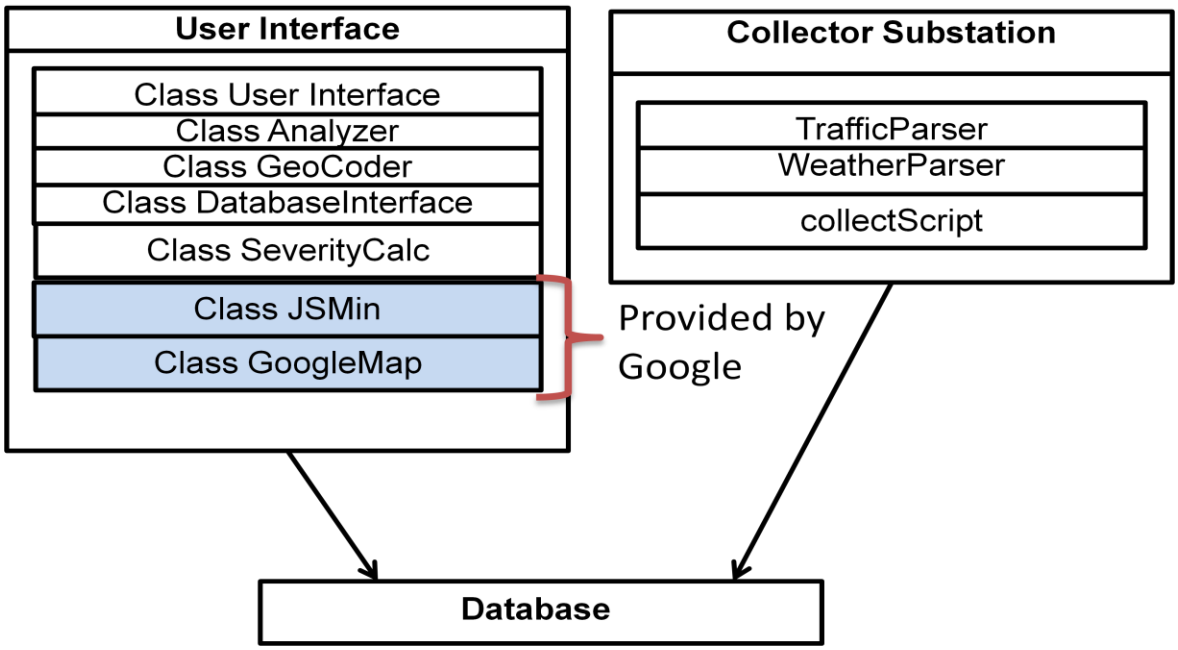
Collector Subsystem class has following classes associated with it:

➔ class TrafficParser , class WeatherParser.

Benefits of Client/Server architecture in Traffic Monitoring System:

- Since data is stored on a server, it offers greater security and there will not be that many errors in the data.
- Traffic monitoring system requires constant updates in weather and traffic data, with client/server architecture, it is easier to access the data on server and update it rather than constantly downloading and processing data on a local user machine.
- Client/server architecture provides ease of maintenance so that client is not affected by server maintenance and upgrade.

Identifying Subsystems:



Mapping Subsystems to Hardware:

The data collected in the "Traffic Monitoring System" will be stored in a database, which will be located in a server. In addition to the database, the graphical user interface will also be stored in the server. The server we are going to be using to store all our data as well as the GUI is a web-hosting service. The client will be able to access the GUI from the server and run the "Traffic Monitoring System" on the server. The client opens a TCP/UDP socket to access and communicate the server.

Persistent Data Storage:

The following database schemas correspond to the stored database tables for the collected weather data and traffic data.

Traffic Database Schema

Field Name	Type	NULL	Default
CREATE_DATE	date	YES	NULL
CREATE_TIME	time	YES	NULL
UPDATE_TIME	time	YES	NULL
UPDATE_DATE	date	YES	NULL
LONGITUDE	char(15)	YES	NULL
LATITUDE	char(15)	YES	NULL
INCIDENT_TYPE	varchar(255)	YES	NULL
ROAD_NAME	varchar(255)	YES	NULL

Weather Database Schema

Field Name	Type	NULL	Default
CREATE_DATE	date	YES	NULL
CREATE_TIME	time	YES	NULL
CONDITIONS	varchar(255)	YES	NULL
TEMPERATURE	char(30)	YES	NULL
CITY_NAME	char(45)	YES	NULL
ZIPCODE	char(10)	YES	NULL

The design decision to implement a database instead of a flat file arose due to performance concerns. We will potentially have megabytes of data and the database lookup will be computed much faster than parsing a file. We feel that once we collect significant amounts of data, the decision to use a database will be justified.

Network Protocol:

In Order for the "Traffic Monitoring System" to interact with the server and the client a HTTP protocol must be used. The same protocol is used to access "Google Maps" for the map images, "511nj.org" to collect the traffic information, and "Weather.com" to collected the weather information. An IPC socket connection is opened so that the server can interact with database. We are using an HTTP protocol because it is the most commonly used protocol and it is a standard protocol in any server as well as browser.

Global Control Flow:

- Execution Order: The execution of the Traffic Monitoring System begins with some sort of an event, in the form of a user selecting options from the standard GUI. The User Interface PHP class will react to the user options and begin calling respective function in DataAggregator and GeoCoder. So in our architecture User Interface is event-driven and everything else is procedure-driven. Both of the parsers (Traffic Parser and Weather Parser) is procedure-driven. It might seem like event-driven because of the Timer, but the Timer itself is written by us and uses system calls to sleep and call the parsers.

- Time Dependency: The Traffic Monitoring System is a real-time program in that it does the computations and processing when the user selects options to view statistics along a route or statistics in a region. Naturally, there will be a delay between the user request and the final processed display however we are very early in the implementation stage and will measure these statistics when appropriate. The two Parsers, which primarily collect and parse weather and traffic data are event-response. They respond to the Timer and collect and parse data when woken up. We have initially set the delay time to 1 hour but this can be re-configured easily buy modifying the Timer sleep time.

- Concurrency: We do not intend to use multiple threads in the User Interface or the Data Collection systems. We assume single threads for each system.

Hardware Requirements:

- Traffic monitoring system requires user to have a minimum bandwidth of 56 Kbps
- Screen resolution of 800 x 600
- We recommend user to have Microsoft Internet Explorer 7.0 and higher, Firefox 3.6 onwards, safari 3.1 and later, Google chrome because we are using Google maps and it supports only these web browser.
- 2 GB of free disc space per month for storing historical data
- Server provides following services:

PHP, MYSQL, Python with standard libraries and Apache HTTP server.

Algorithms and Data Structures

Algorithms

The most important algorithm we intend to implement involves the calculations of average traffic, severity and other metrics such as delay times, if supported. The current traffic website (511nj.org) uses very generic descriptions and simply states whether an incident has occurred or not. Based on these generic descriptions we need to develop a systematic way of deriving severity and average number of incidents statistics.

1) Average Incidents Statistics: The SQL database table for traffic will contain information (latitude, longitude and highway) which will differentiate the incidents based on the sectors of the highway they occur on. We count the number of incidents which occur within one sector of the highway by querying the database with the conditions, lat = sector_lat and long = sector_long and road_name = highway being queried. Once the matching results are gathered, we count the length of the array returned and then divide this by the total number of incidents on the highway in question. The resulting answer represents the average number of incidents which occurred within a particular sector of the highway. However this does not take into account cases where a highway has negligible traffic. If such a case is encountered then the averaging will be skewed. To remedy this we introduce time into the averaging. This way as more days go by without incidents the lower the calculated probability of encountering traffic. This model will now incorporate the number of days since the data collection started. Our final statistical averaging model will be formulated as follows:

   First Iteration:

$$\frac{Number\ of\ incidents\ in\ sector}{Number\ of\ incidents\ on\ highway * total\ days\ of\ data\ collection}$$

   This gives a true estimate of the traffic and takes into account the days when there is no traffic, in order to preserve accuracy.

   Unfortunately this gave us several problems during our experimental stage where we have values that were not predictable. However, the one takeaway was that this gave more weight to recent event than events which have occurred in the past.

   Second Iteration:
   Incident severity by categorization:

Level 5 (Most Severe)	Accidents, Accidents with Injuries, Vehicular Fire, Truck Fire, Overturned Vehicle etc.
Level 4	Disabled Truck, Disabled Car, Heavy Traffic etc.
Level 3	Accident Investigation, Construction

	Delays, Earlier Incident, Pockets of Volume
Level 2	Congestion, Police activity, Pothole repair,
Level 1 (Least Severe)	Roving repairs, Debris Spill, HOV Rules

This method gave us a uniform way of judging each incident by categorizing them and averaging the severity values of each incident gave us the severity value of a single marker. However, what we found was that most of the events belonged to Level 3 severity thus most markers were colored exactly the same. Clearly this would not be helpful for users so we further refined the model to our final implementation.

Final Implementation
Incident severity by categorization and weight:

The same five-level categories are used however we extending the process to include the CREATE_DATE of the incident and the duration of the incident. The table below shows how the duration and date of creating is taken into account:

0hrs <= Duration <= 3hrs; Create Date < 1 week	Incident gets full category value
Duration > 3hrs; Create Date < 1 week	Incident's category value is incremented by 1 (if category value is 5 then do nothing)
0hrs <= Duration <= 3hrs; 1 week < Create Date <= 2 weeks	Category value * 0.8
Duration > 3hrs; 1 week < Create Date <= 2 weeks	(Category value + 1)* 0.8
0hrs <= Duration <= 3hrs; 2 weeks < Create Date <= 3 weeks	Category value * 0.6
Duration > 3hrs; 2 weeks < Create Date <= 3 weeks	(Category value + 1)* 0.6
0hrs <= Duration <= 3hrs; 3 weeks < Create Date <= 1 month	Category value * 0.4
Duration > 3hrs; 3 weeks < Create Date <= 1 month	(Category value + 1)* 0.4
0hrs <= Duration <= 3hrs; Create Date > 1 month	Category value * 0.2
Duration > 3hrs; Create Date > 1 month	(Category value + 1)* 0.2

Once each incident is sifted we average the severities of all incidents to get the average severity value of a marker. Once the average severity value is calculated we further sift these values into colors. This final sift is shown below:

AverageSeverity <= 2.2	"Green"
2.2 > AverageSeverity >= 3.15	"Blue"
3.15 > AverageSeverity >= 3.45	"Yellow"
3.45 > AverageSeverity >= 3.75	"Orange"

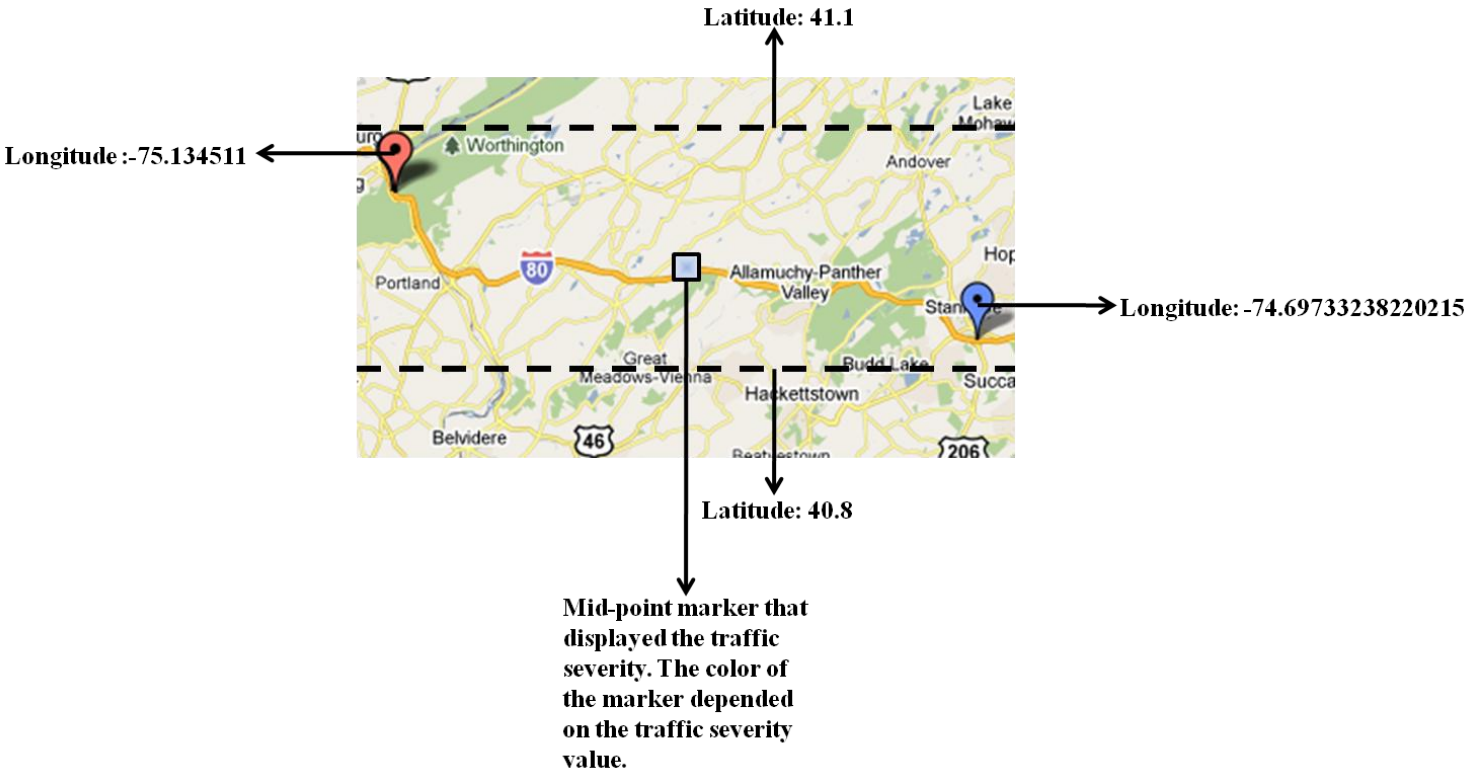
AverageSeverity > 3.75	“Red”

The reason the threshold values seem arbitrary is mainly because, we attempted to produce a Gaussian distribution of the severity levels. We ran the severity calculator several times and based on the values we saw most often we arranged our filter to simulate a Gaussian distribution.

If we had more time we would have actually taken several runs and used a spreadsheet to graph the distribution and come up with a mathematical formula which would filter the average severity. However, given the short time span we opted for a less elegant solution. If we were to extend this project then the severity calculator would be the first item to be updated and optimized.

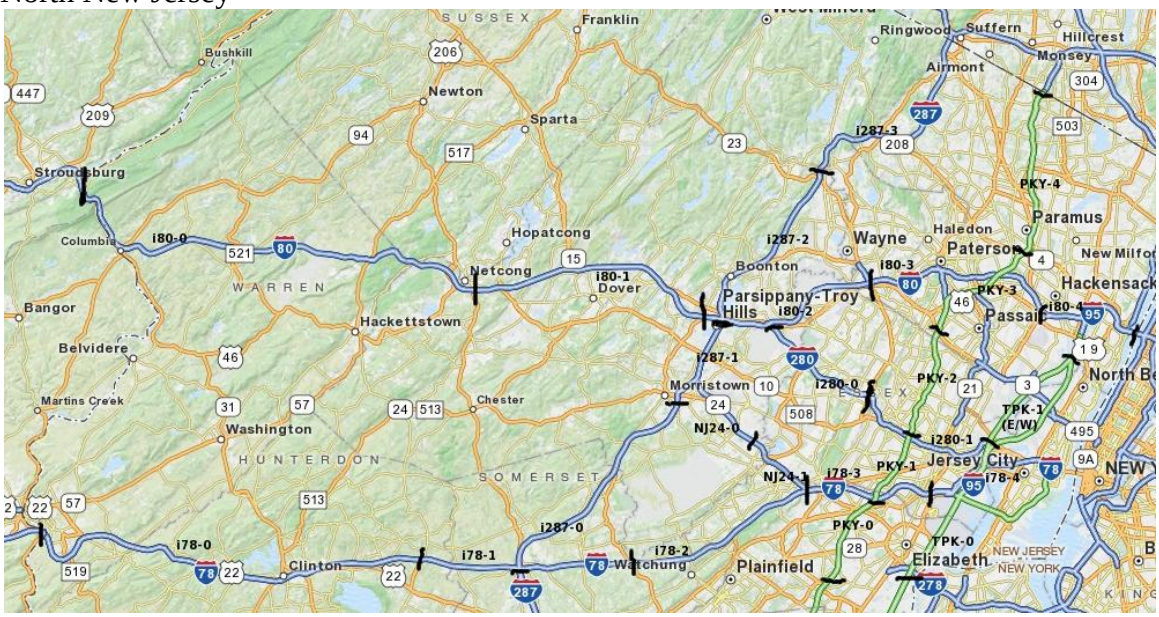
Mapping Incidents to Sectors:

To make accurate prediction for each highway across the state of New Jersey, each highway was broken down into various sectors. This was done because traffic incidents occur at various points of the highway and traffic prediction had to be made for each sector. The number of sectors depended on the length of the highway. For example, the New Jersey Turnpike has a length of over 120 miles and was divided into 10 sectors, whereas I-80 has a length of around 60 miles and was divided into 4 sectors. Using a map, each highway was divided to the appropriate number of sectors and their geographical coordinates were noted. For each sector, a mid-way point was selected and at this point a marker was placed to give the traffic prediction for that sector of the highway. The color of the marker tells the user the level of the traffic severity. A more detailed explanation on how the color is assigned is explained below.
(Note: No road in the state of New Jersey is a straight road. All the roads have various curves based on the geography of the region. For this reason, all highways that were east-west, predefined upper and lower latitudes were used that incorporated the entire highway. For highways that were north-south, predefined right and left longitudes were used to incorporate the entire highway In other words, each sector was made into a box. The picture on the next page depicts how this was done. I-80 is used as an example)



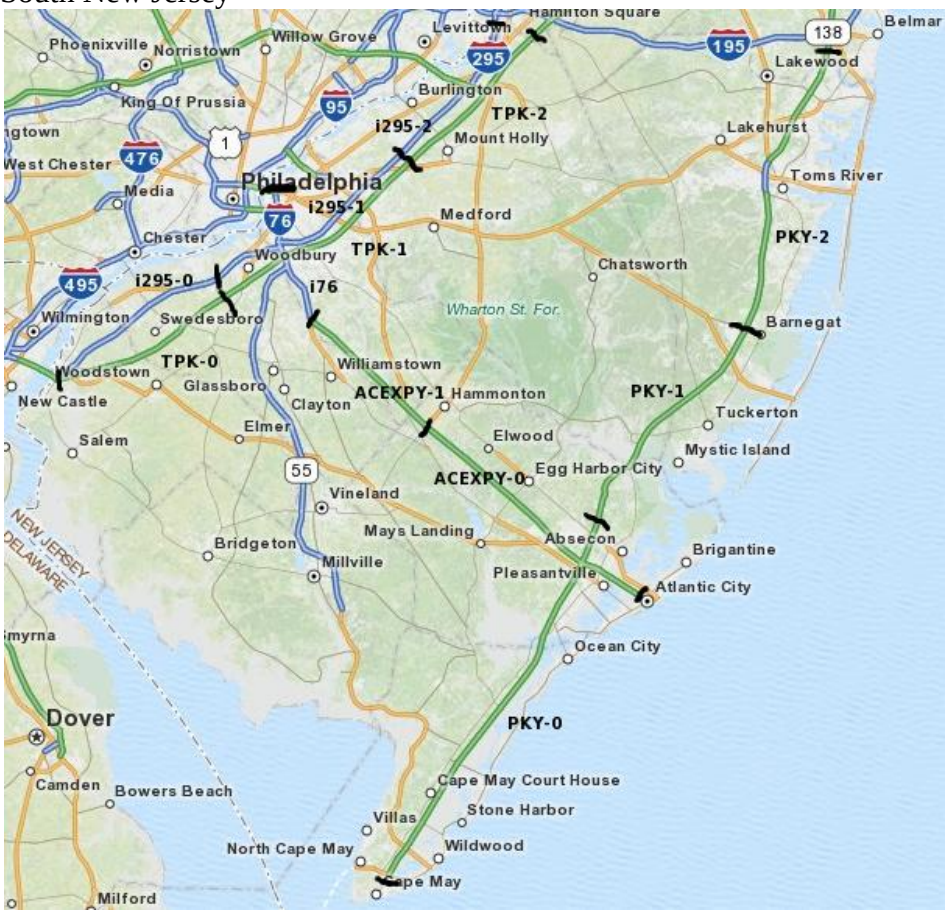
Once all the data was collected, this data had to be analyzed. When a user selects his or her desired preferences, a call is made to the DataAggregator in order to return necessary information to the user. The way the DataAggregator worked was that it makes a call to the database interface to get the required information. This database interface access the database and retrieves all the pertinent information that the DataAggregator asked for and returns it to the DataAggregator. Once the DataAggregator received all the data, using various set conditions based on the selected road/region, weather conditions, and the time of day, a traffic severity value is calculated and assigned to each sector marker of the selected highway. The traffic severity value has a range of 1 through 5. The following figures show the sector demarcations for each region of New Jersey.

North New Jersey



Central New Jersey

South New Jersey



Differentiating Road Intersections:

One minor concern that we were faced with, was how to differentiate between roads when they intersect. This would have been a problem if we simply used the Latitude and Longitude box demarcation for sectors. However, we took the ROAD_NAME parameter into account. Doing so ensured that even if several roads share the same sector demarcation their ROAD_NAME parameter would be different. This would let us easily differentiate between any convergence of roads.

Data Structures

The main data structure in our Traffic Monitoring system is the database. The database will hold all the collected weather data and all the collected traffic data. The contents of the database are outlined in the database schema. We have implemented lookup tables (Multi-dimensional arrays) in order to lookup the sectors for each highway. This way we can match incidents to their corresponding sectors. We do not anticipate any other major data structures (trees, linked list etc.) in future extensions.

User Interface Design and Implementation

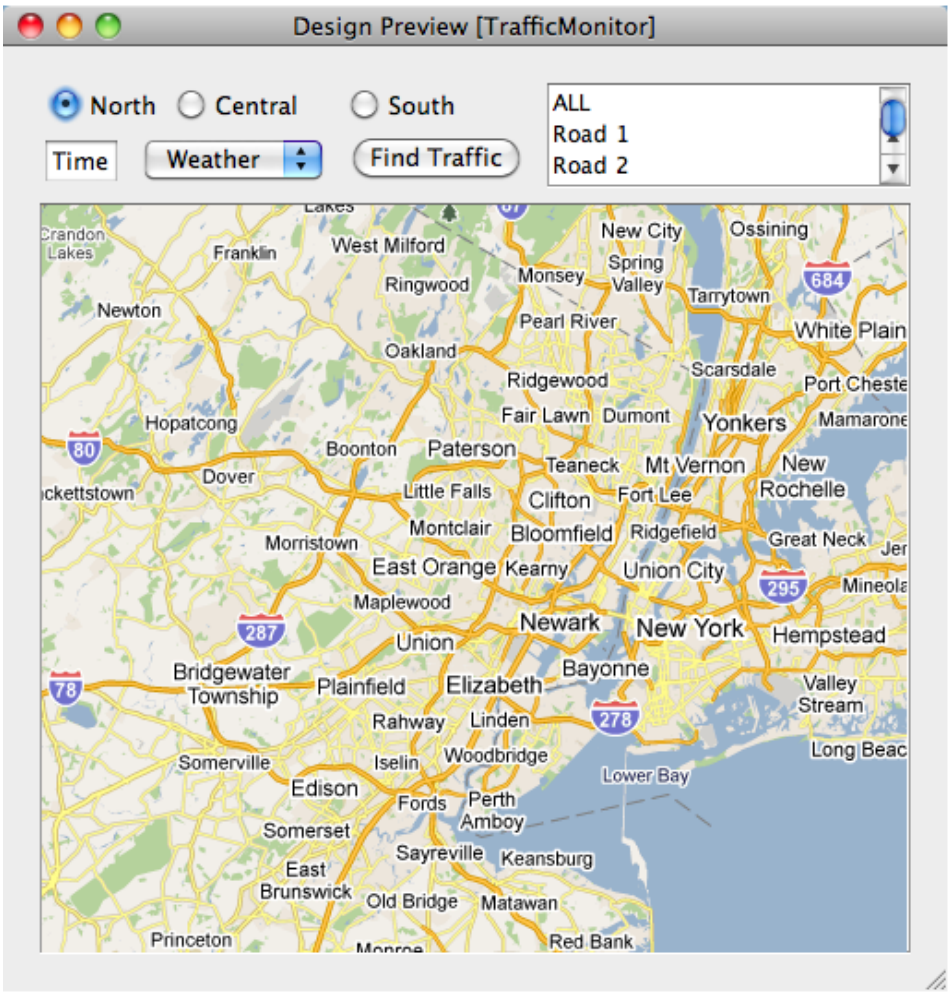
Preliminary design:

To access and use the Traffic Monitoring System, the user first opens their preferred internet browsing application / program and loads the project's website. The user enters all pertinent information on this page to find traffic and weather information. The user clicks one of the three radio buttons designated to the three different regions of the state of New Jersey - North, Central, and South to find information about the desired area the user chooses to travel. The user then enters the time of day. Using a drop down menu, the weather type is selected: rain, snow, sleet etc. or can select display all types of weather. The user then selects the highway or all highways to travel on using the scroll menu. The user then submits this information to be queried by the website to display information.
The user has an option to view three different sets of information. One choice is to see traffic predictions for the selected route. The second choice is to see traffic predictions for the entire selected region. The third option is to overlay the map with live traffic condition from a live website (such as Google traffic/NJ traffic etc.).

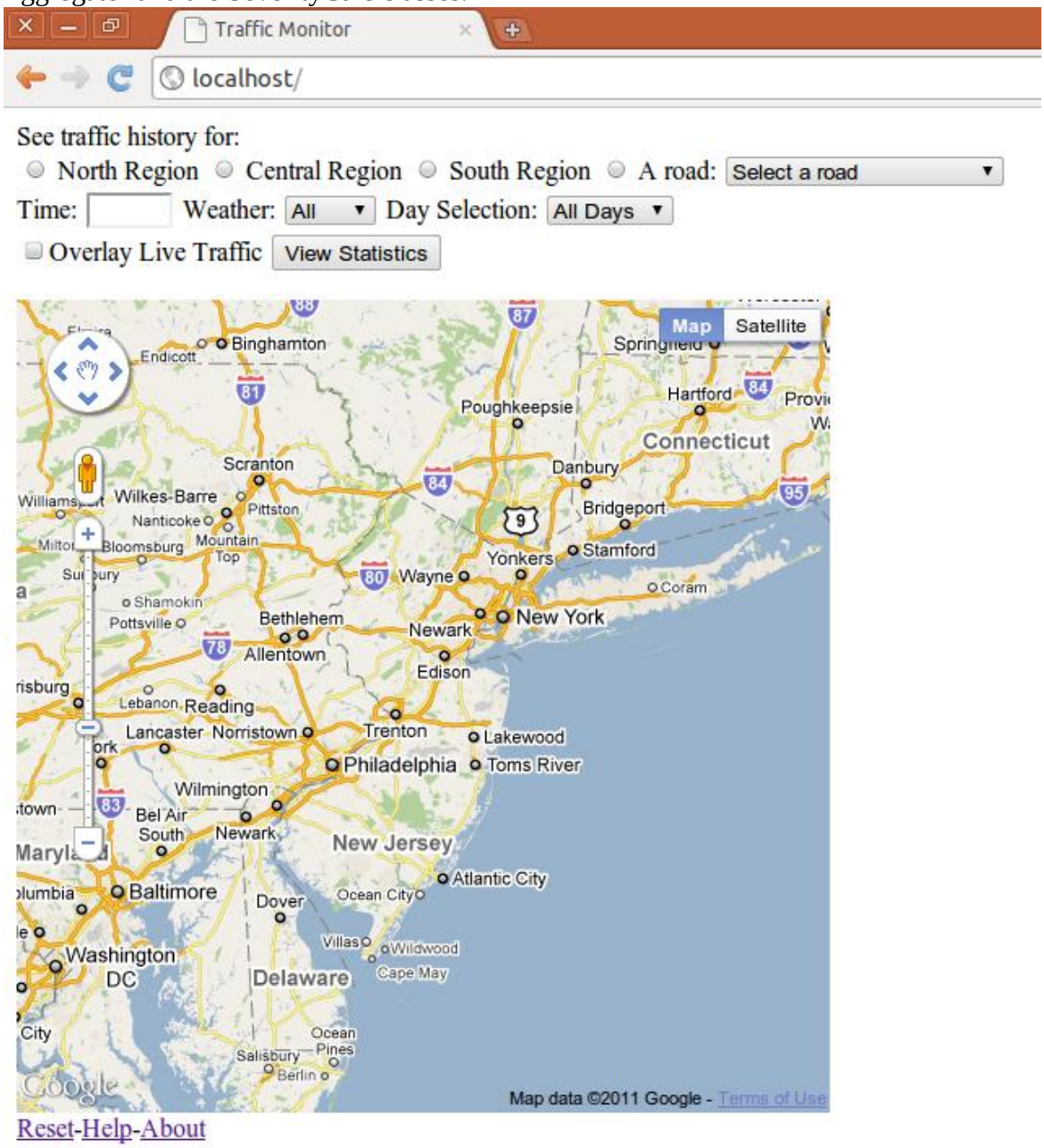
The user will select a region, time, weather, and optionally a road. Then the website will display a map plotting markers in sectors of the roads. These sectors represent a segment of the road that will have similar traffic conditions throughout. In rural areas the sectors are larger and they are small in urban areas. The map will display a marker of a certain color depending on the traffic prediction for that sector. The user interface has not changed visually since the last report. However, now it will be implemented in PHP and not Java or JavaScript because Google maps does not support Java.

Below are some initial mock-ups of the user interface design. The pictures are ordered such that it shows our progression in design until the final picture which represents the current, fully implemented design.

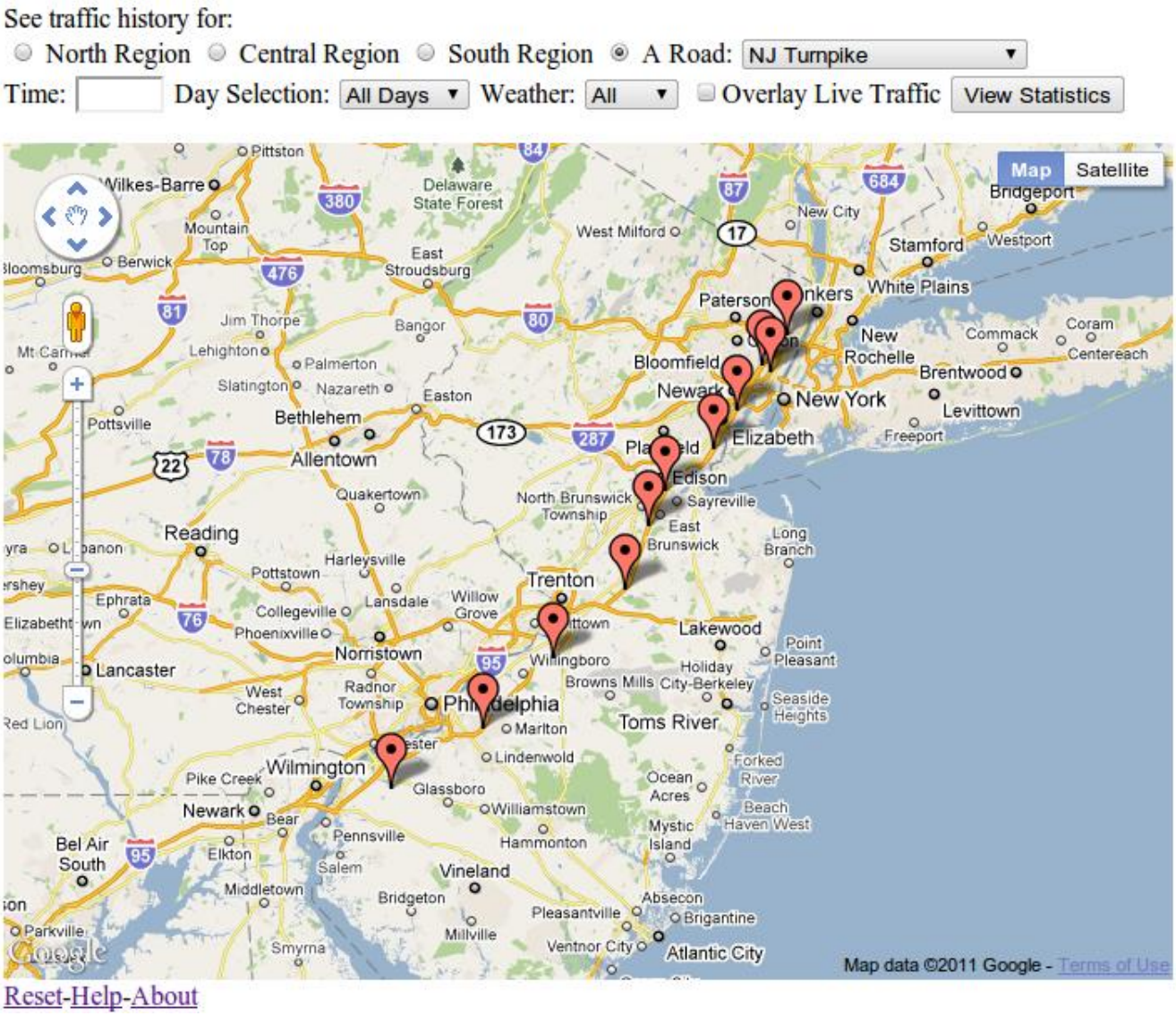
Initial Mock up using a Java GUI generator. This was done during the 1st iteration in order to see how our system may look in the future.



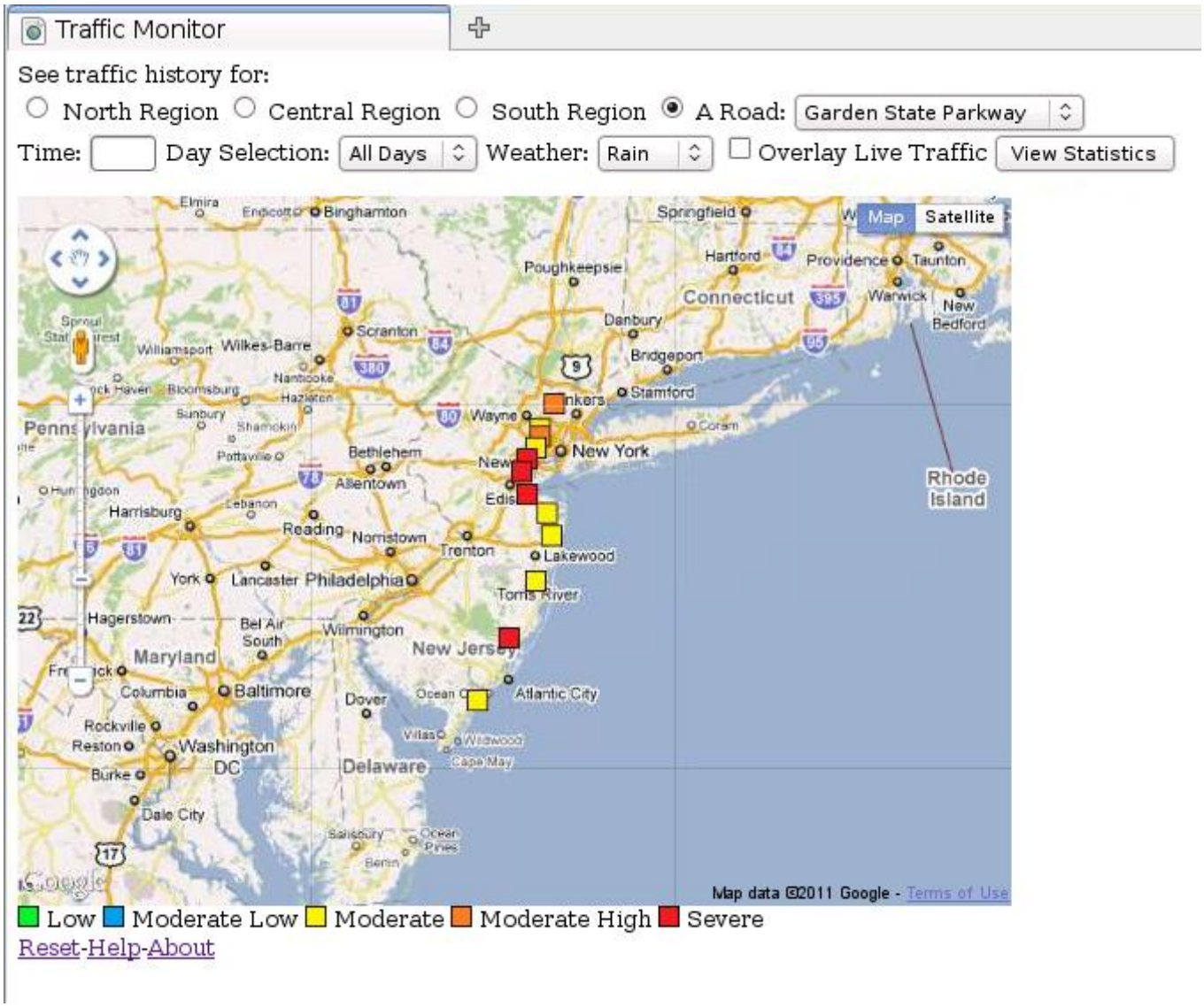
Before Demo 1. The group agreed upon this interface as our final design. At this stage we implemented the GeoCoder which could interface with Google Maps and use their API to zoom, place markers and overlay live traffic. At this stage we were working mainly on the Data Aggregator and the SeverityCalc classes.



During the 2nd iteration. We implemented the marker placement for each component. Selecting difference regions and highways showed these mock-up markers. Once again, we were in the middle of implementing our SeverityCalc and DataAggregator class.



This is a screenshot of our final implementation. We replaced the mock-up markers with the actual colored markers which differentiated severity levels. This was the point where we combined all classes into one system thus completing our project. However, we are not completely satisfied with the GUI and if we had more time we would have experimented with different markers in order to display a better picture.



Here, we enumerate some problems with the markers:

1. These marker images are hosted on a photobucket profile for the project because we could not host them on the ECE server (due to privacy concerns). We noticed that performance of the website dropped slightly, because GoogleMaps now had the overhead of retrieving the images from photobucket and then overlaying them at the Latitude and Longitude specified by our GeoCoder.
2. At 0% zoom, it seems that the markers are off center and not placed directly on the highway. The problem here is that Google Maps places the marker on the map at the largest zoom level and then reverts to 0% zoom. This means that as you incrementally

zoom out, the marker would seem to move off center. However, zooming in would show that the markers are not off-center and directly on top of the highway. This is a problem we could not fix since we relied on the Google Maps API.

3. Another interesting point to note, is that the last marker, for the NJ Turnpike (the marker at the southern tip of NJ Turnpike) might erroneously be attributed to bad placement since it looks like it's in the middle of no-where. However, this is not the case; zooming in would show that the Turnpike splits into I-295 and NJ Turnpike at that point. However, zooming out shows that the NJ Turnpike fork disappears from the map. So, the marker placement is correct but Google does not provide enough clarity at the one point to be able to physically see the NJ Turnpike part of the fork. We show this particular case with the following figures.

Marker seemingly off-center:

Marker after zoom:

User effort estimation:

The effort put in by the user to get all the information required, including mouse clicks and keystrokes. (***Using an example time period of 9:00A.M.***)

- **NAVIGATION**: Total of two mouse clicks, as follows
  - Open preferred web browser to load project website

  ---After completing data entries as shown below---
  - Click *"Show Traffic"* to display the traffic in the selected region.

- **DATA ENTRY:** Total of 5 mouse clicks and 6 keystrokes
  - Click the radio button to select region to travel in.
  - Press the *"Tab"* key to move to the text field (*"Time"*).
  - Press the keys *"9", "0", "0", "A", "M"* (Enter time in a 12-hour format without the colon for nine in the morning).
  - Click the drop down menu named *"Weather"* and select a weather condition by clicking on a choice.
  - Click on the preferred highway OR click on ALL to select all roads.
  - Click the radio button to display which map the user wants to see.

History of Work & Current Status of Implementation

Project Milestones	Predicted Date	Actual Date	Reason
Set up data collection system including weather and traffic collection.	March 6th 2011	March 18th 2011	A website for traffic information could not be found by 3/6
Set up a database server to store processed weather and traffic data	March 6th 2011	March 18th 2011	Same reason as above, a suitable traffic website couldn't be found.
Assemble GUI Interface	March 10th 2011	March 10th 2011	-
Develop the statistical averaging model for traffic prediction	April 17th 2011	April 29th 2011	There were several iterations of the averaging model which took a long time to develop.
Develop the Database Interface	April 10th 2011	April 11th 2011	Final code fixes took an extra day
Develop the DataAggregator for funneling incidents into sectors	April 28th 2011	May 1st 2011	We knew that the Aggregator would require a lot of code and as such it took us a while to complete it and test it.
Final Testing and Code Fixes	April 30th 2011	May 2nd 2011	We planned to test the code by the end of April but due to some delays with other components we had to delay.

Project Management

Some of the most problematic issues during the entire project were the synchronization needed to keep the project moving forward. There were several occasions where we would accidentally modify code that was already updated. Throughout the process we used Dropbox as a way of sharing the source code and making sure each person would maintain contact via email, phone calls etc. There were occasions where some team members did not know how to proceed with their tasks and that required some of the more experienced members to explain and help with the conceptual understanding of the component.

In the future we would look into a repository or some sort of version control in order to maintain the code and restrict any changes so that we wouldn't un-knowingly overwrite code. The development process was quite long and arduous especially when trying to combine components from various people with different coding styles. One such example was the Google Maps API where we had to modify our GeoCoder and User Interface components to fit the API provided. Otherwise we felt that our project was very strong and that we have provided a unique, complex piece of software in order to solve a key problem in the form of Traffic Monitoring.

Conclusions and Future Work

Our initial difficulties were during the Data Collection implementation stage where we could not find a suitable live traffic information website to use for parsing. We found that Yahoo and MapQuest have proprietary software which does not allow third party developers to use their traffic information. However, through extensive researching we found 511nj.org, a government maintained website for the major highways and thruways of New Jersey. They publish Incident reports for events ranging from constructions, delays, police work, lane closures to accidents. The website is updated approximately every 15 minutes with new content, if any, however we have decided to collect information hourly. Even though we found a website there was still the challenge of extracting this information.

We came up with a method of using Python scripts to parse the html tags from the website in order to find key terms such as “Latitude”, “Longitude”, “Incident Type” and “Road Name”. Once these terms were parsed we compared the road names to those we were interested in, if any did not match then they were discarded otherwise we added all of the relevant information including time stamp and date information “Create Date” and “Create Time” into the database.

Other sources of challenges came from the Severity calculations and the DataAggregator components. As described previously, the DataAggregator maps events from the database into predefined sectors for each region/highway. This aggregation process required a lot of code and proved to be the main bottleneck for our system. In the end, we are happy with the work that went into the aggregator however we felt that efficiency could be improved drastically through leveraging multiple threads and more efficient code structure. The Severity calculation algorithm was another challenge and we felt that we did not have enough time to completely address this component to our liking.

Our final version, although much improved from the previous two iterations, is still far from the standard we hoped it would be. Most of the other components were fairly straightforward and required less complexity in the code however the code needed to be extensive and sufficiently fault-tolerant. Further refinement may be needed for the GUI in order to make it more appealing to general users and use the API provided by Google more extensively in order to provide more user-friendly features such a pop-up bubbles with severity indices when users click on the markers.

In the end there seem to be a lot of improvements we could make given an appropriate amount of time to continue developing the project. However, we are quite happy with our level of implementation and the features we have made available.

Future Work

We have several ideas about future work and we would like to discuss some ideas in this section. We envision our software covering not just the state of New Jersey but perhaps all of the tri-state area where traffic has many socio-economical implications. Currently we only provide information on major highways however, any good traffic monitoring software should, in good faith, cover any road that the user may want to frequent. With this particular feature we have found several problems, especially the unavailability of information for small county roads or local streets. We propose that social networking could be a very important too in collecting information about incidents occurring in small local or county roads. This adds another dimension of complexity, namely the requirement of accessing, collecting and parsing hundreds or thousands of updates searching for key terms like “traffic”, “pile up” or “accident” to name a few. We feel that this would be a great contribution especially since we have not found references to any software using social networking to gather traffic information.

Some extensions would be less for the user and more for the system in order to improve efficiency and code maintainability. As discussed previously, we could potentially leverage multithreading in order to improve the throughput of requests and, if, the system is scaled up then the potential of using a distributed computing paradigm becomes more pragmatic from both a programming and a user perspective. Another possible extension would be to take future construction projects into account. We have noticed that 511nj.org posts information of future projects so we could, conceivable, access this information and warn the user about such events.

There are some very interesting applications for a historical traffic monitoring system such as this and clearly since there aren’t many instances of such systems (apart from Google Maps) makes this very appealing. We feel that our system is a robust, unique application which has a lot of potential for improvement. In the end, we enjoyed working on the project and it was great understanding the software development cycle from the inside as developers of a large-scale, maintainable application.

References

- Traffic.com Live Traffic System
  http://www.traffic.com/controller/routing?nvt_pointA=new%20brunswick%20new%20jersey&nvt_pointB=new%20york%20city%20new%20york#

- Ovi Maps Live Traffic Demonstration
  http://www.youtube.com/watch?v=o5wDz4E28x4

- Inrix Live Traffic iPhone App
  http://reviews.cnet.com/8301-13746_7-10305540-48.html

- Google Maps Historical Traffic Example
  http://maps.google.com/?ie=UTF8&ll=40.446947,-73.608398&spn=2.44537,4.938354&z=8&layer=t&tptime=144000

- 511nj.org Incident and Congestion Webpage
  http://www.511nj.org/IncidentList.aspx?listType=IncidentsCongestion

- 511nj.org Live Cameras (Click on any of the camera figures to see a live feed)
  http://www.511nj.org/cameras.aspx

- Weather.com RSS Feed Example
  http://rss.weather.com/weather/rss/local/08831

- Google Maps API
  www.code.google.com/apis/maps/index.html

- PHP Tutorial
  www.w3schools.com/php/default.asp

- Python Reference Manual
  http://docs.python.org/tutorial/

- SQL Database documentation
  http://dev.mysql.com/doc/refman/5.0/en/tutorial.html

- MySQL Tutorial (Downloadable PDF) very useful for future projects!
  http://downloads.mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf

- PHP 5 for Dummies by Janet Valade

- Gantt Chart Tutorial video on youtube:
  http://www.youtube.com/watch?v=HQwE0Xv1lAA&feature=relmfu

- Software Engineering by Ivan Marsic Statement of Requirement Page 64

- Software Engineering by Ivan Marsic Use Case Modeling Page 71
- Software Engineering by Ivan Marsic System Specification Page 78