ECE452: Concepts in Software Engineering

Report #3: System Specification & Design

Group #14: Jon Lipovac

Submission Date: 2007-04-27
Project Title: Traffic Monitoring
Project URL: http://ece452.2o1.net/

Breakdown of Individual Contributions

All team members contributed equally.

Table of Contents

Summary of Changes

"Customer's Statement of Requirements":
- Modified user choices description
- New information regarding Inrix's Bayesian technology

"Glossary of Terms":
- New more appropriate definitions

"Functional Requirements":
- New Administrator actor
- Modified Goals
- Modified Use Case descriptions
- New UC5
- Modified Use Case diagram

"Nonfunctional Requirements":
- Modified Quality requirements

"Domain Analysis":
- Modified sequence diagrams
- New text descriptions
- Modified domain model diagram
- Modified System Operation contracts

"Interaction Diagrams":
- New interaction diagrams
- New activity diagrams
- New design patterns
- Modified description

"Class Diagram and Interface Specifications":
- Modified class diagrams
- Modified Data Types and Operation Signatures

"System Architecture and System Design":
- Modified package diagram
- Modified descriptions

"Algorithms and Data Structures":
- Modified description

"User Interface Design and Implementation":
- Modified description
- New screenshot
- Modified user effort estimation

New "History of Work & Current Status of Implementation" section

New "Conclusions and Future Work" section

Customer's Statement of Requirements

This project aims to provide predictions of road congestion in a given geographical area for a specified day and time, based on historical traffic and weather conditions. This information would be useful for automobile drivers in choosing the best time to a travel a given route or the best route to travel at a given time.

"Most traffic information services (example: Yahoo! Maps and Traffic, Traffic.com) only provide current information about traffic conditions in a given area. While current information is essential, these reports are often incomplete since their sources frequently fail to report the current traffic conditions. Hence, the user cannot assume that there is not heavy traffic in a given location simply because that location was not reported by these services."[i] See Figures 1.1 and 1.2 for examples of current traffic conditions from "Yahoo! Maps and Traffic" and "Traffic.com", respectively.

Figure 1.1: Example of current traffic conditions from "Yahoo! Maps and Traffic."

Figure 1.2: Example of current traffic conditions from "Traffic.com."

Analyzing historical traffic data for patterns will yield locations of "hot spots." These are the locations or routes that appear more frequently in the traffic data. While these "hot spots" may not appear on the current traffic advisory, they are still likely to be subject to heavy conditions. Additionally, "hot spots" can be correlated with specific day/times and weather conditions. By using the historical weather data for the type of day and time range requested, the applicability of the "hot spots" can be refined further.

The complete system is comprised of 2 independent systems that share a common database of historical traffic and weather data: the "Traffic Map" system and the "Data Collection" system.

The "Traffic Map" system shall provide the user with the following services: "Statistics within the given area" and "Statistics along a given route." The "Data Collection" system shall populate the database with historical traffic and weather data. It shall provide the administrator with the following services: "Traffic collection" and "Weather collection."

For the "Statistics within a given area" service, the user shall make the following choices:

1. Address
2. Radius around Address
3. Time period (All *or* AM *or* PM)
4. Day type (All *or* Weekday *or* Weekend)
5. Weather (All *or* Clear *or* Cloudy *or* Precipitation)
6. Minimum incident severity (1 (Least) *thru* 5 (Most))

For example, the user may wish to know the traffic statistics of "Most severe (5)" incidents within 10 miles of Clifton, NJ for clear weekdays in the morning. With the user-selected criteria, the system shall extract the historical data and overlay the statistics of the given area on an interactive geographical map provided by a mapping service. "The numerical values of the mean and variance should be visually encoded by color and size of the graphical markers shown on the map. The marker color should range from green (level 1) for the smallest mean number of reports to red for the highest mean number of reports (level 3). The marker size should be used to encode the variance."[ii]

For the "Statistics along a given route" service, the user shall make the following choices:

1. Starting address
2. Ending address
3. Time period (All *or* AM *or* PM)
4. Day type (All *or* Weekday *or* Weekend)
5. Weather (All *or* Clear *or* Cloudy *or* Precipitation)
6. Minimum incident severity (1 (Least) *thru* 5 (Most))

This service shall function similar to the "Statistics across the entire area" service. A route shall be calculated using the user- supplied addresses and shall be displayed on an interactive geographical map provided by a mapping service. Traffic locations shall only be used in the statistical calculations and displayed on the map if they are within the confines of the route.

The "Data Collection" system should query the "Yahoo! Maps and Traffic" traffic information service and the "Wunderground.com" weather service to obtain current traffic and weather conditions, respectively. These conditions shall be stored in a database accessible to the "Traffic Map" system.

For the "Traffic collection" service, the administrator shall make the following choices:

1. Zip code(s)
2. Coverage radius for zip code(s)
3. Update frequency

For example, the administrator user may wish to collect traffic data for zip codes “07013” and “07702” at a radius of 100 miles at a frequency of 30 minutes. With the user-selected criteria, the system shall retrieve the appropriate data from the traffic server at the specified frequency. Each new traffic incident shall be compared to existing incidents. If an incident does not exist with the same Latitude, Longitude, Title and Start Date, it shall be added to the database. If the incident does exist in the database with that criteria but the Updated Date is earlier, then the existing record shall be updated with the new information. Otherwise, the new incident shall be discarded. Incidents in the “construction” category should not be discarded in case there is a need to utilize them in the “Traffic Map” system at a later date.

For the “Weather collection” service, the administrator shall make the following choices:

1. Zip code(s)
2. Update frequency

For example, the administrator user may wish to collect weather data for zip codes “07013”, “08901” and “07702” at a frequency of 60 minutes. With the user-selected criteria, the system shall retrieve the appropriate data from the weather server at the specified frequency. Weather conditions for each zip code shall be added to the database. Since this weather should correspond to the “Traffic collection” data, the administrator shall select zip codes that encompass the area radii selected in the “Traffic collection” service.

This system relies heavily on the quality and quantity of traffic reporting. Advancement in traffic prediction and forecasting technology ultimately depends on an improvement of traffic flow reporting. However real-time traffic reporting within the United States is currently in its infancy, primarily due to a lack of infrastructure. Methods of data collection that are currently widely employed include automated road sensors and driver’s reports via cell phones. Yet this information is often too coarse to comprehensively describe a traffic flow for a specific area. Ideally, all vehicles on the road should contribute to the model by continuously reporting traffic data. Only recently has this desire started to come into fruition as part of Inrix’s “Smart Dust Network”, in which 500,000 vehicles across the U.S. report their GPS location and speed.[iii] Inrix combines this technology with their proprietary prediction algorithms to provide real-time and forecasted traffic to broadcasters and navigation systems via Clearchannel’s “Total Traffic Network”.[iv] This proprietary prediction algorithm utilizes an advanced Bayesian statistical model originally developed by Microsoft[v]. Another company, Triangle, provides similar technology covering a number of metropolitan areas.[vi] See Figure 1.3 for an example of Triangle’s “Travel time forecast for Los Angeles to Long Beach, California.”

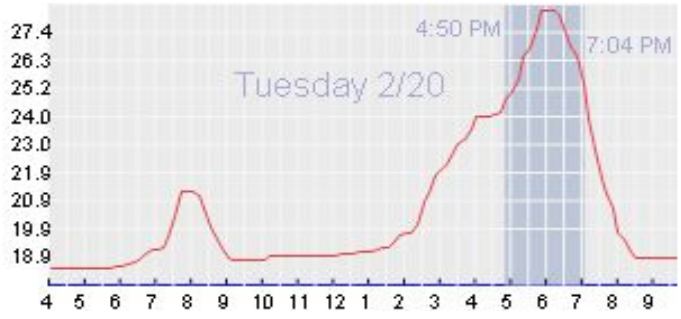

Figure 1.3: Example of Triangle’s traffic forecast

Glossary of Terms

AJAX:	A web technology, "Asynchronous Javascript and XML" -- used to refresh data on a web site without requiring the user to manually refresh the page
Bayesian statistical model:	A statistical model in which probability is defined as the degree to which a person believes a proposition
Cronjob:	A job (or script) ran on a time schedule
Geocoding service:	A web site that provides latitude/longitude to coordinate translation and/or routing services
JS:	A client-side web programming language, "Javascript"
Mapping service:	A web site that provides an customizable interactive geographical map
Perl:	A programming language, "Practical Extraction and Report Language"
PHP:	A web programming language, "PHP: Hypertext Processor" (or "Personal Home Page tools")
Polyline:	A sequence of latitude/longitude pairs
Routing Service:	A web site that provides polyline coordinates of a route between starting and ending addresses
SQL:	A database interaction/programming language
Traffic information service:	A web site that provides current traffic information for a given geographical area (ie: Google Maps)
Variance:	A measure of a random variable's statistical dispersion, indicating how its possible values are spread around the mean value
Weather service:	A web site that provides current weather conditions for a given geographical area

Functional Requirements

Stakeholders:

- User(s)
- Administrator

Actors and Goals:

- User
  - Initiating type
  - Goals: To lookup predicted traffic conditions within a given area or along a given route, within the "Traffic Map" system (involved in UC1, UC2).
- Google Maps
  - Participating type
  - Goals: To provide an interactive geographic map and geocoding service within the "Traffic Map" system (involved in UC1, UC2).
- Database
  - Participating type
  - Goals: To store historical traffic and weather conditions for use by both the "Traffic Map" and "Data Collection" systems (involved in UC1, UC2, UC3, UC4).
- Administrator
  - Initiating type
  - Goals: To configure the "Data Collection" system for the appropriate area (involved in UC3, UC4, UC5).
- Timer (Cronjob)
  - Initiating type
  - Goals: To run the "Data Collection" system periodically (involved in UC3, UC4).
- Yahoo! Maps and Traffic
  - Participating type
  - Goals: To provide current traffic conditions within a given area for the "Data Collection" system (involved in UC3).
- Wunderground.com
  - Participating type
  - Goals: To provide current weather conditions within a given area for the "Data Collection" system (involved in UC4).

Use Cases:

- Casual Descriptions
  - UC1: ViewStatisticsWithinArea
    - Provides user with predicted traffic conditions within a given area using user-supplied criteria (see Figure 1.4)

- UC2: ViewStatisticsAlongRoute
  - Provides user with predicted traffic conditions along a given route using user-supplied criteria (see Figure 1.5).
- UC3: CollectTrafficConditions
  - Collects traffic condition within a given area using criteria previously configured by the Administrator and stores the data in the common database (see Figure 1.6).
- UC4: CollectWeatherConditions
  - Collects weather conditions within a given area using criteria previously configured by the Administrator and stores the data in the common database (see Figure 1.7).
- UC5: ConfigureCollectionCriteria
  - Allows the Administrator to configure the "Data Collection" system for the appropriate area (see Figure 1.8).

- Fully-Dressed Descriptions (see Figures 1.4 through 1.8)

Use Case UC1:	**ViewStatisticsWithinArea**
Primary Actor:	User
Actor's Goal:	To view predicted traffic conditions within a given area using supplied criteria
Stakeholders:	*Supporting Actors*: Database, Google Maps
Preconditions:	None worth mentioning.
Postconditions:	None worth mentioning.
Main Success Scenario:	
	1. **User** supplies address, radius of interest, time period, day type,
-->	weather and minimum incident severity
	2. **System** queries **Google Maps** for latitude/longitude coordinates
<--	from supplied address (geocoding)
<--	3. **System** queries **Database** for applicable traffic and weather data
<--	4. **System** calculates statistics for applicable area
<--	5. **System** retrieves geographical map from **Google Maps**
	6. **System** overlays statistics and incident markers over
<--	geographical map
Extensions (Alternate Scenarios):	
	4a. No data for **User**-supplied area exists in the **Database**
	1. **System** notifies **User** that supplied area is outside data
<--	scope
	2. Same as Step 1 above

Figure 1.4: Use Case Description for UC1

Note: Due to the unavailability of a free online route calculation service, this use case may not be implemented by the final demo

Use Case UC2:	**ViewStatisticsAlongRoute**
Primary Actor:	User
Actor's Goal:	To view predicted traffic conditions along a given route using supplied criteria
Stakeholders:	*Supporting Actors*: Database, Google Maps
Preconditions:	None worth mentioning.
Postconditions:	None worth mentioning.
Main Success Scenario:	
-->	1. **User** supplies starting address, ending address, radius of interest, time period, day type, weather and minimum incident severity
<--	2. **System** queries **Google Maps** for latitude/longitude coordinates of route "polyline" from supplied starting and ending address (geocoding)
<--	3. **System** queries **Database** for applicable traffic and weather data
<--	4. **System** calculates statistics for coordinates on route
<--	5. **System** retrieves geographical route map from **Google Maps**
<--	6. **System** overlays statistics and incident markers over geographical route map
Extensions (Alternate Scenarios):	
	4a. No data for **User**-supplied route exists in the **Database**
<--	1. **System** notifies **User** that supplied route is outside data scope
	2. Same as Step 1 above

Figure 1.5: Use Case Description for UC2

Use Case UC3:	**CollectTrafficConditions**
Primary Actor:	Timer (Cronjob)
Actor's Goal:	To collect traffic conditions within a given area using criteria previously configured by the Administrator and store the data in the common Database.
Stakeholders:	*Supporting Actors*: Yahoo! Maps and Traffic, Database, Administrator
Preconditions:	Timer counter reaches update frequency previously configured by Administrator
Postconditions:	Current traffic data is stored in the Database
Main Success Scenario:	
-->	1. Script handling "Traffic Collection" service is initiated
<--	2. **System** queries **Yahoo** server for traffic conditions for area(s) previously configured by **Administrator** (see UC5)
<--	3. **System** (a) verifies that each reported traffic incident is new and (b) inserts the incident into the **Database**
Extensions (Alternate Scenarios):	
	3a. Reported traffic incident already exists in **Database** with older "update time"
<--	1. Update **Database** record with new incident data
	3b. Reported traffic incident already exists in **Database** with same "update time"
<--	1. Discard new incident data

Figure 1.6: Use Case Description for UC3

Use Case UC4:	**CollectWeatherConditions**
Primary Actor:	Timer (Cronjob)
Actor's Goal:	To collect weather conditions within a given area using criteria previously configured by the Administrator and store the data in the common Database.
Stakeholders:	*Supporting Actors*: Wunderground.com, Database, Administrator
Preconditions:	Timer counter reaches update frequency previously configured by Administrator
Postconditions:	Current weather data is stored in the Database
Main Success Scenario:	
-->	1. Script handling "Weather Collection" service is initiated
<--	2. **System** queries **Wunderground** server for weather conditions for area(s) previously configured by **Administrator** (see UC5)
<--	3. **System** inserts the weather data into the **Database**

Figure 1.7: Use Case Description for UC4

Use Case UC5:	**ConfigureCollectionCriteria**
Primary Actor:	Administrator
Actor's Goal:	To configure traffic and weather collection criteria and initialize the common Database for collection.
Stakeholders:	*Supporting Actors*: Database
Preconditions:	None worth mentioning.
Postconditions:	Database is initialized and collection criteria is configured.
Main Success Scenario:	
<--	1. **Administrator** supplies traffic and weather criteria for area of interest
-->	2. **System** configures **Database** and collection scripts (UC3, UC4).

Figure 1.8: Use Case Description for UC5

- Use Case Diagram (see Figure 1.9):

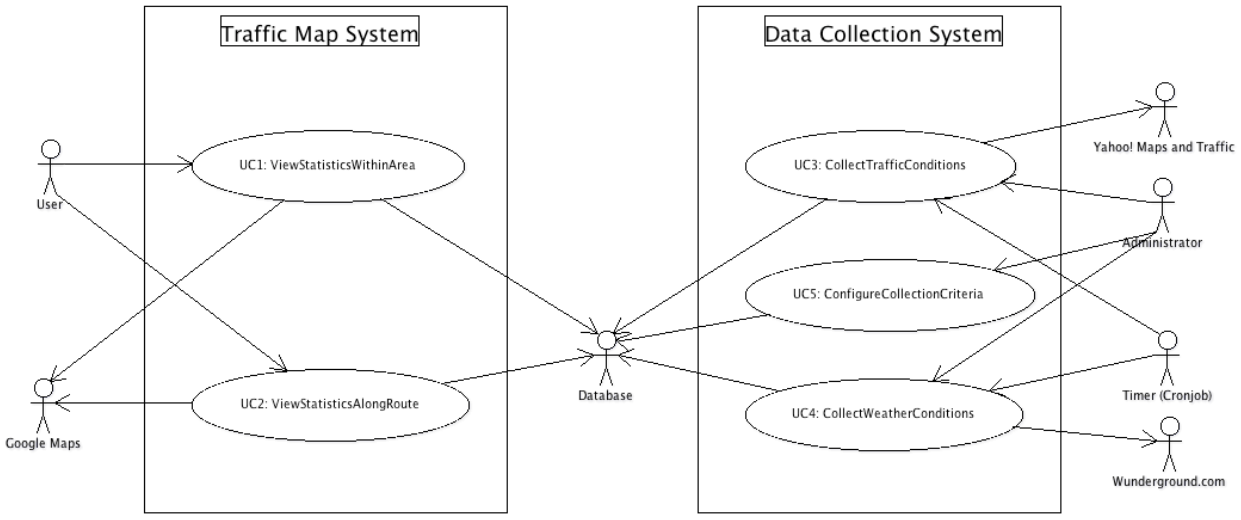

Figure 1.9: Use Case Diagram

Nonfunctional Requirements

Quality requirements for "Traffic Monitoring":

- Any user familiar with the "Google Maps" service should be able to use "Traffic Maps" without the user manual. [Usability requirement]
- "Data Collection" should store all Administrator-configurable options within a single configuration file [Usability requirement]
- "Traffic Maps" should handle invalid and out-of-scope user input gracefully [Reliability requirement]
- "Traffic Maps" shall guarantee accurate output after 1 month of data collection in the applicable area. [Reliability requirement]
- "Traffic Maps" should complete database operations and statistical analysis within 30 seconds for each user. [Performance requirement]
- "Traffic Maps" should handle 5 consecutive users at one time. [Performance requirement]
- "Traffic Maps" and "Data Collection" should each be able to be upgraded without breaking the other system. [Supportability requirement]

Constraints for "Traffic Monitoring":

- All software associated with "Traffic Maps" shall be written using object-oriented PHP, to ensure portability. [Implementation requirement]
- All software associated with "Traffic Maps" shall implement JS and/or AJAX technology within the user interface, to ensure compatibility with the Google Maps API. [Implementation requirement]
- All software associated with "Data Collection" shall be written using Perl and its associated CPAN modules, to ensure portability. [Implementation requirement]

System Sequence Diagrams:

In UC1, the user supplies the criteria for their search and submits it to the system. The System converts the address into latitude/longitude coordinates by means of Google's geocaching service. The system assembles a SQL query using these coordinates and the other user-supplied criteria and searches the (joined) traffic and weather databases for applicable records. Statistics are calculated simply by adding up the severities at each specific coordinate. The system builds the JS for the geographical map from Google and concatenates the JS for the incident markers and their corresponding color and sizes (based on the statistics). The JS for the overlay map is then returned to the user (see Figure 1.10).

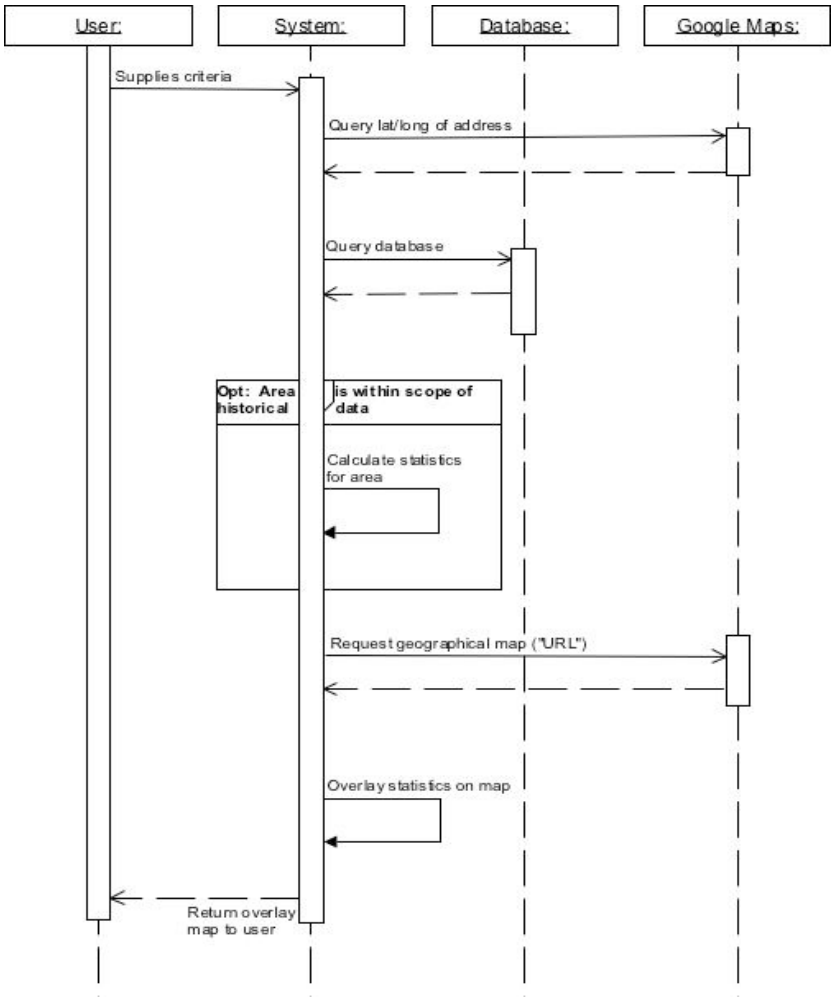

Figure 1.10: System Sequence Diagram for UC1 (ViewStatisticsWithinArea)

In UC2, the user supplies the criteria for their search and submits it to the system. The System converts the starting and ending addresses into a sequence of latitude/longitude coordinates by means of the routing functionality of Google's geocaching service. The system assembles a SQL query using this sequence of coordinates and the other user-supplied criteria and searches the (joined) traffic and weather databases for applicable records. Statistics are calculated simply by adding up the severities at each specific coordinate. The system builds the JS for the geographical map from Google and concatenates the JS for the incident markers and their corresponding color and sizes (based on the statistics). The JS for the overlay map is then returned to the user (see Figure 1.11).

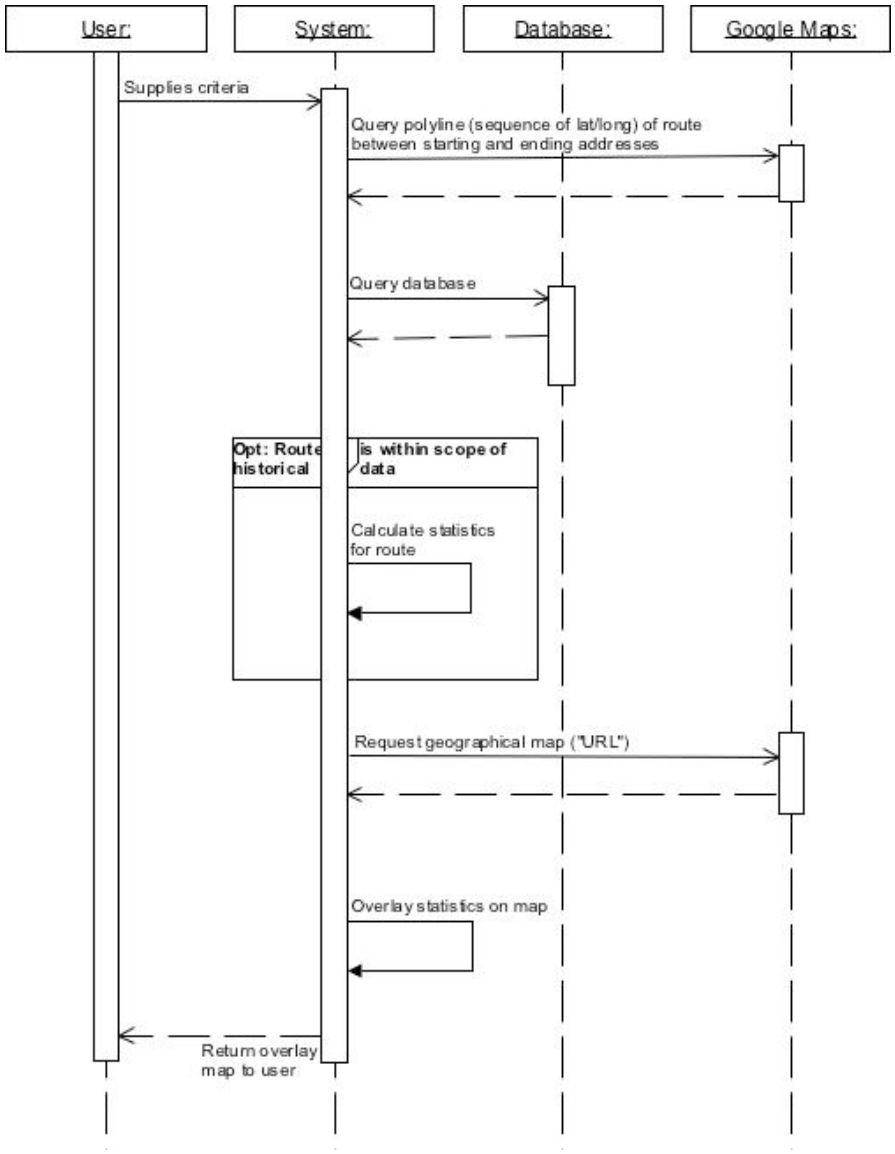

Figure 1.11: System Sequence Diagram for UC2 (ViewStatisticsAlongRoute)

In UC3, the timer triggers the traffic collection script. The System forms a request URL for a configured zip code and radius. The URL is sent to Yahoo! Maps and Traffic and a XML response is returned. The System parses the XML and for each record it checks for a duplicate record already in the database. If the record is not a duplicate it is stored in the database. If it is an updated version of an existing record, the existing record is updated in the database. This entire process is repeated for each configured zip code with a 2 second delay between each iteration. Control is then returned to the timer (see Figure 1.12).

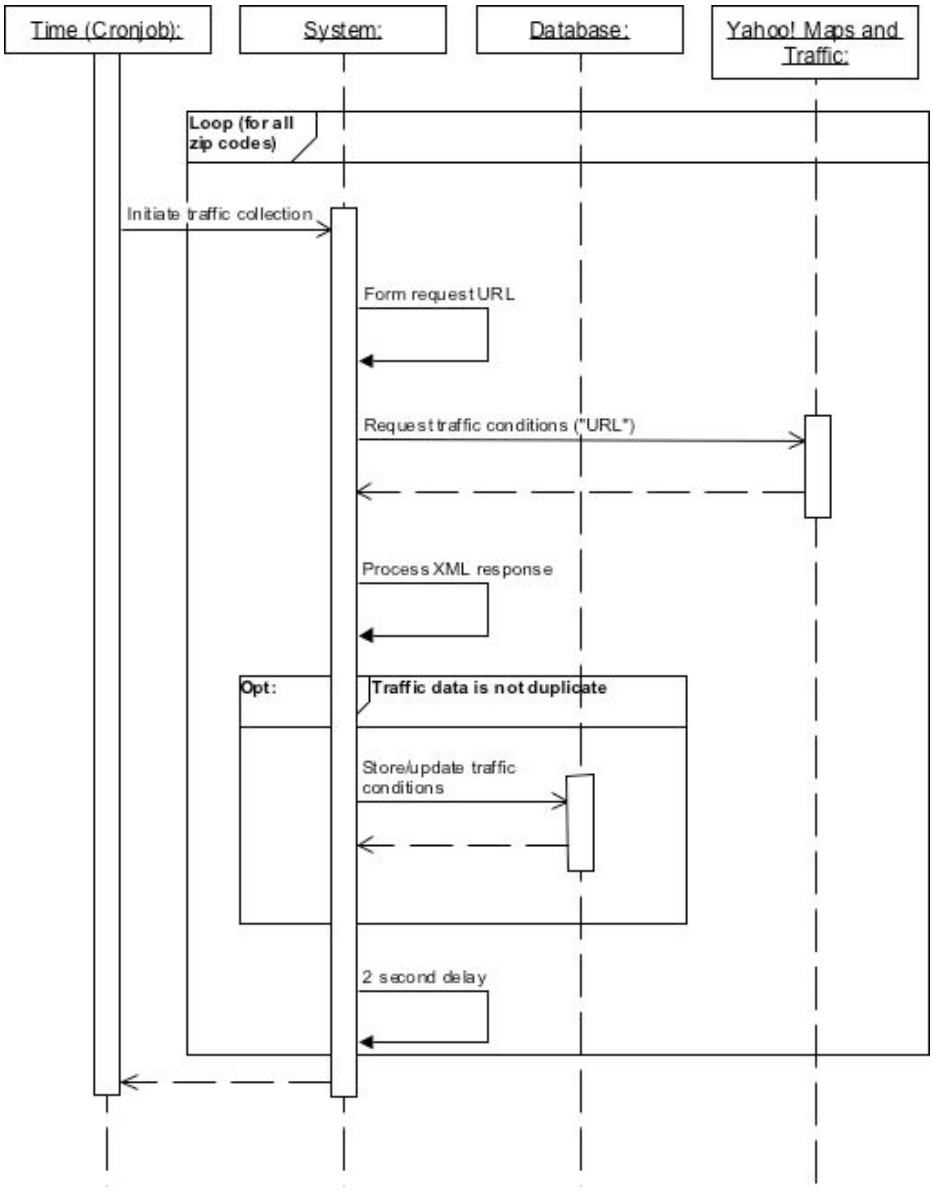

Figure 1.12: System Sequence Diagram for UC3 (CollectTrafficConditions)

In UC4, the timer triggers the weather collection script. The System forms a request URL for a configured zip code. The URL is sent to Wunderground.com and a RSS (XML) response is returned. The System parses the XML and stores each record in the database. This process is repeated for each configured zip code with a 2 second delay between each iteration. Control is then returned to the timer (see Figure 1.13).

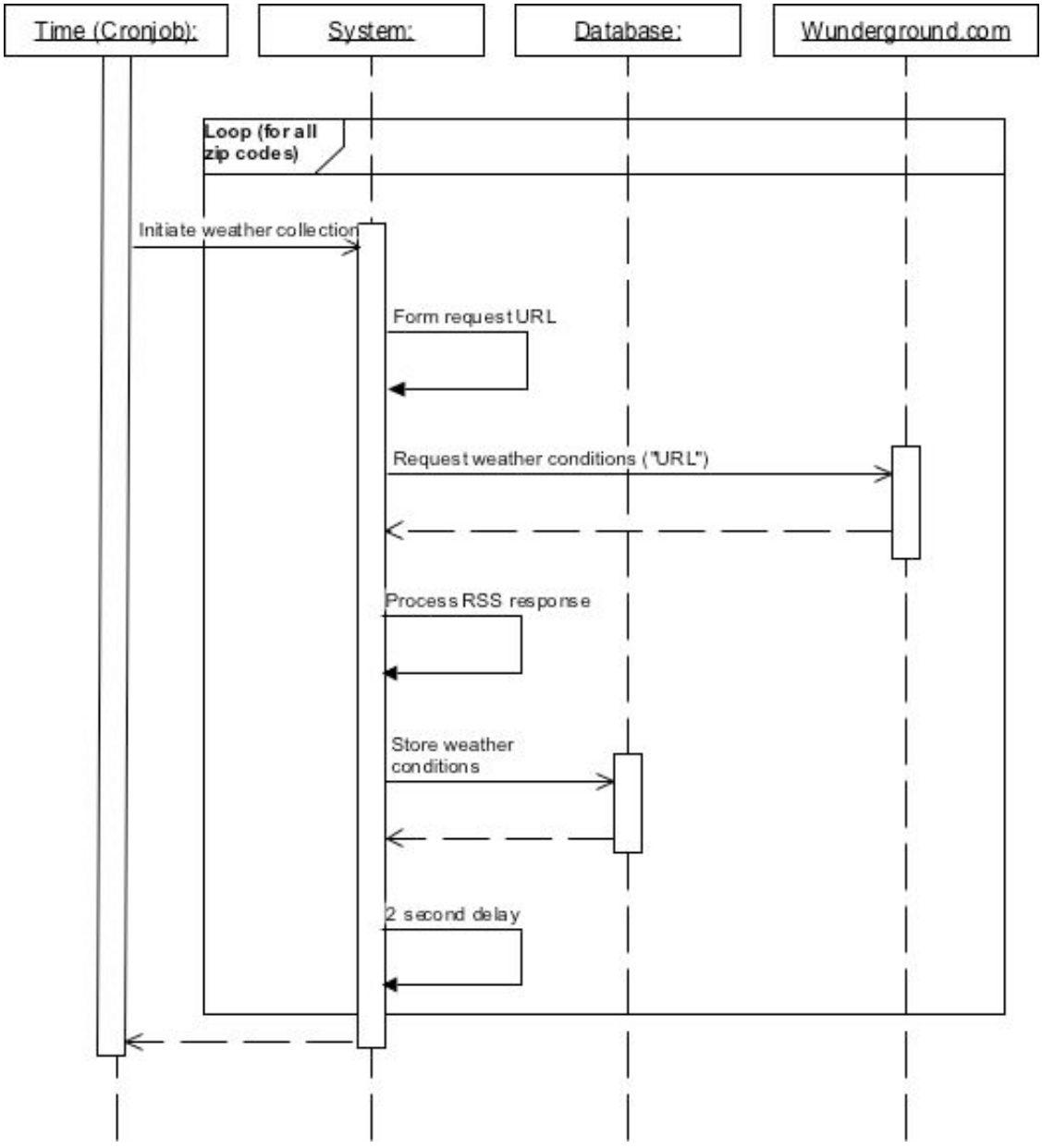

Figure 1.13: System Sequence Diagram for UC4 (CollectWeatherConditions)

In UC5, the Administrator supplies configuration data to the System. This data includes traffic zip codes and radius, weather zip codes and the frequency at which the timer should operate. The system stores this configuration data in the collection scripts. The system also sets up the database with the hard-coded weather and traffic data storage schemas. Control is then returned to the Administrator.

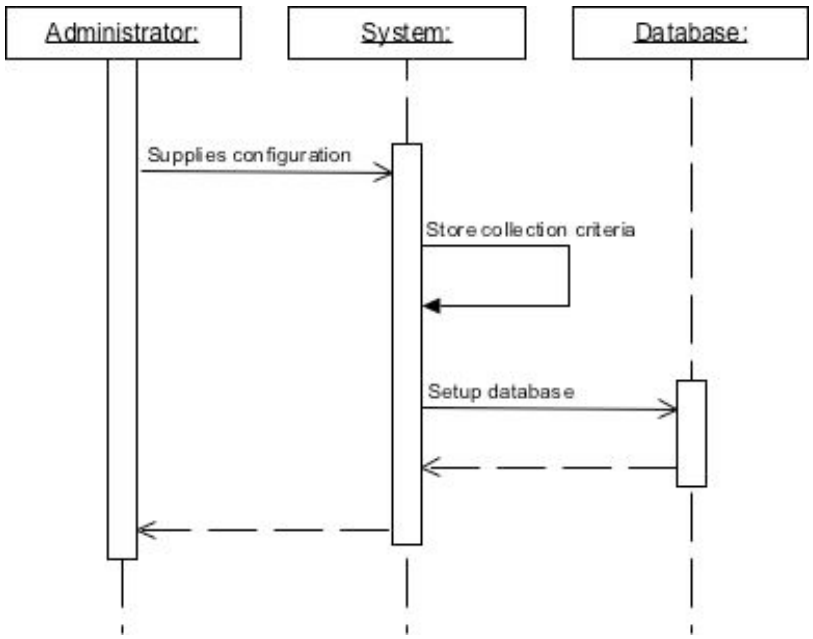

Figure 1.14: System Sequence Diagram for UC5 (ConfigureCollectionCriteria)

Domain Model:

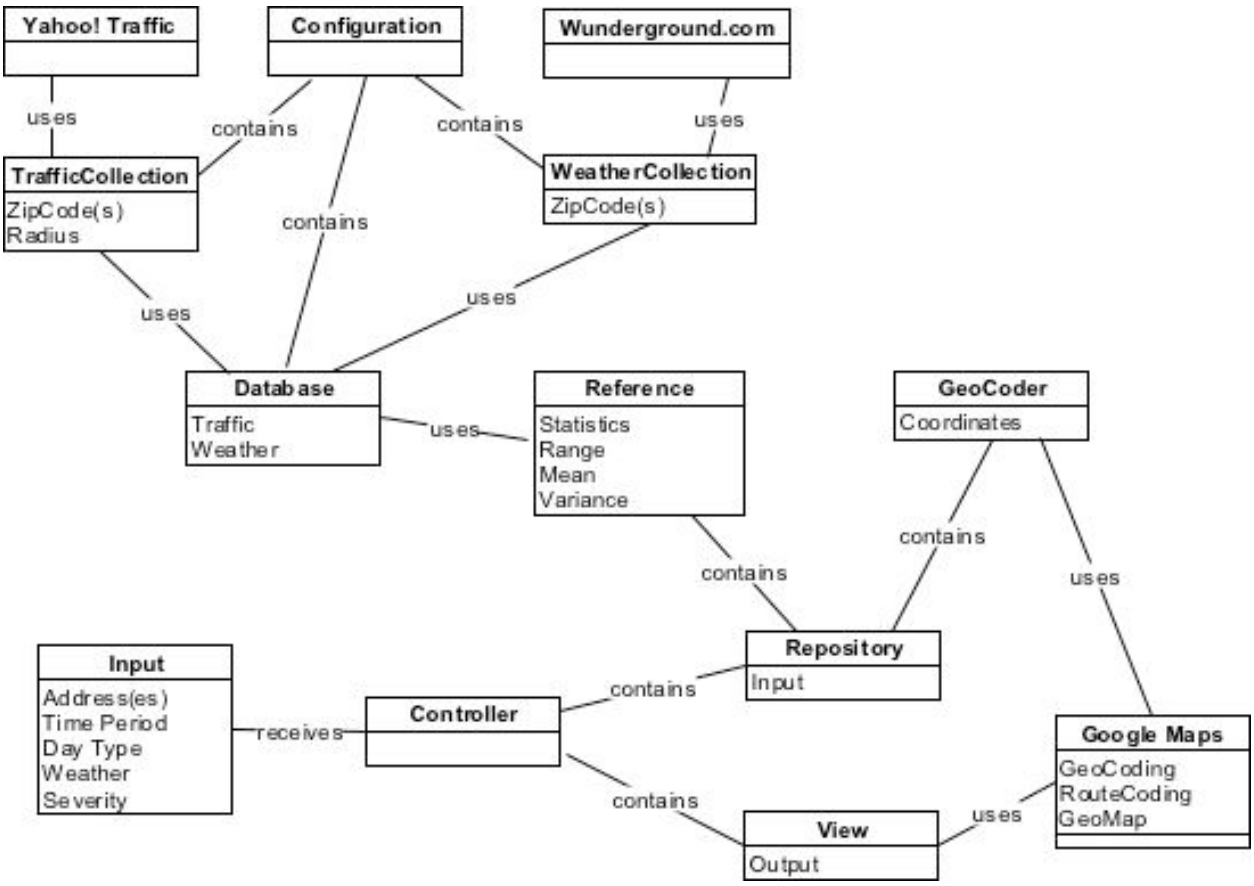

Figure 1.15: Domain Model for all use cases ("Traffic Maps" and "Data Collection")

System Operation Contracts:

Operation	ProcessRequest
Preconditions	Database is sufficiently populated with data
	Avg. case: Overlay map is returned to the user with markers
Postconditions	Worse case: Overlay map is returned to user with no markers

Operation	SantizeLocalData
Preconditions	Local data is stored in object
Postconditions	Data is within bounds and safe to use in SQL queries

Operation	CalculateStatistics
Preconditions	Input is within scope of historical data
	Statistics are stored in array indexed by latitude/longitude.
	Variance of entire sample is stored in variable. (See
Postconditions	"Algorithms" section

Interaction Diagrams

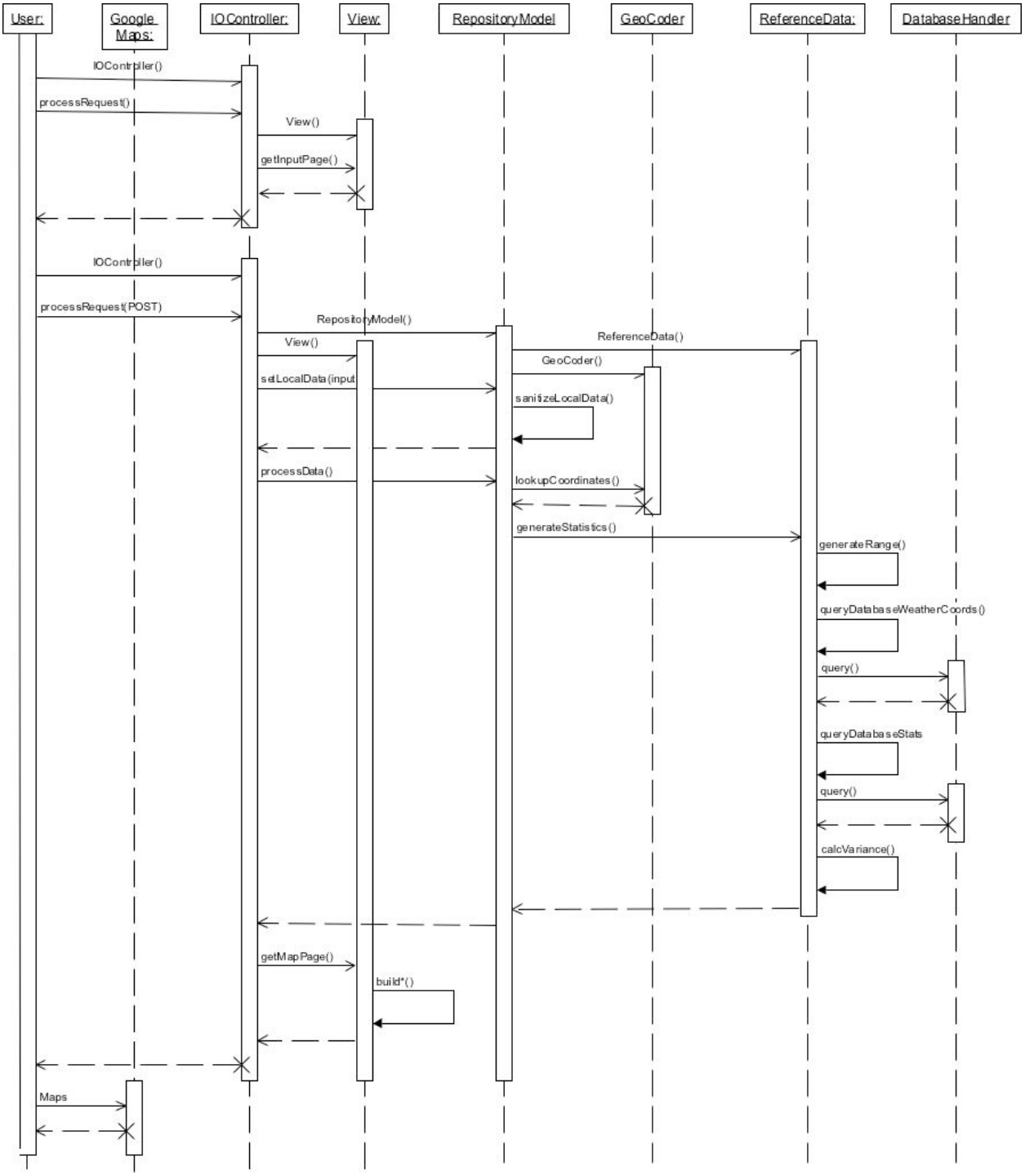

Figure 1.16: Interaction diagram for UC1 ("ViewStatisticsWithinArea")

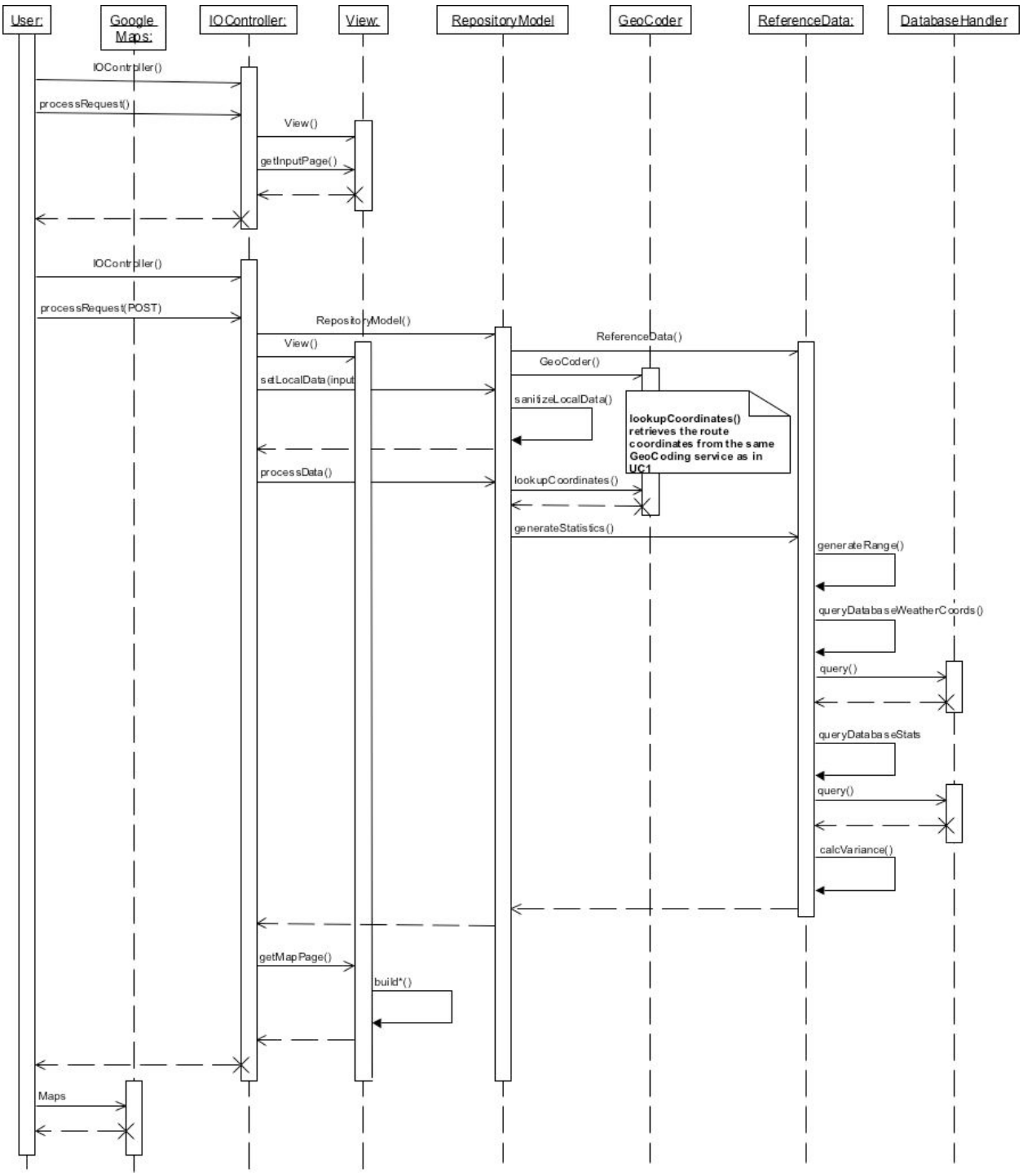

Figure 1.17: Interaction diagram for UC2 ("ViewStatisticsAlongRoute")

Multiple design principles were involved in the design of UC1 and UC2 (see Figure 1.16 and 1.17). When the client initially contacts the server, "IOController" requests the input page from View. The follow-up request from the client involves processing user data so "IOController" employs "RepositoryModel" in addition to "View". This is characteristic of both the "Expert Doer Principle" (knowing who should so the task) and "High Cohesion Principle" (balancing the workload and keeping objects focused). The "High Cohesion Principle" governs the rest of the system past "IOController". The "RepositoryModel" object is responsible for "GeoCoder", "ReferenceData" and "DatabaseHandler" (external). The "View" object is isolated and is fed data by the "IOController" in order to build the output. The aforementioned objects are functionally cohesive and attempt to represent a functional[vii] design pattern.

Additionally, a proxy[viii] design pattern is used in regards to the DatabaseHandler object. The administrator can change SQL server vendors (ie: PostgreSQL, MySQL) as well as the server's network location without requiring any change to the code.

In the Interaction Diagrams for UC1 and UC2 (see Figure 1.16 and 1.17), physical constraints limited the listing of some of the private methods involved in each class. For example, the View class's method "getMapPage()" calls 14 individual "build" functions (see Figure 1.25). In the "ReferenceModel" class, there are individual private methods to build each component of the SQL query.

Also of note is the that the "GeoCoder" method "lookupCoordinates()" is overloaded to process both single location and route translations. Similarly, functions that handle an array of coordinates differentiate between the 2 Use Cases by type of array. A single array indicates a location ("ViewStatisticsWithinArea") whereas a multidimensional array indicates a polyline route ("ViewStatisticsAlongRoute").

Since UC3, UC4 and UC5 are coded in a procedural fashion (and written in the PERL scripting language), their flows are best represented by activity diagrams (see Figures 1.18 thru 1.20). UC3 ("CollectTrafficConditions") utilizes simple SQL logic to determine whether to store, update or discard each record. UC4 ("CollectWeatherConditions") stores all records unconditionally. UC5 ("ConfigureCollectionCriteria") provides a simplified way to represent the configuration of the collection scripts and the database. The work entailed in UC5 is performed from a UNIX shell by the Administrator.

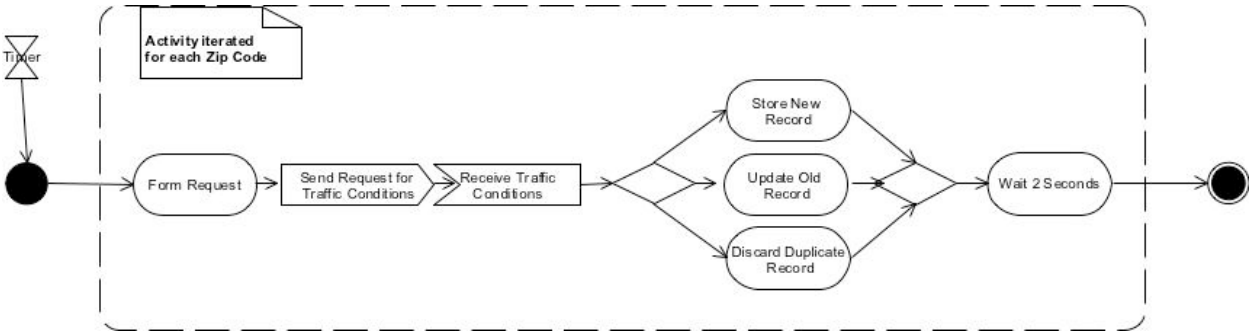

Figure 1.18: Activity diagram for UC3 ("CollectTrafficConditions")

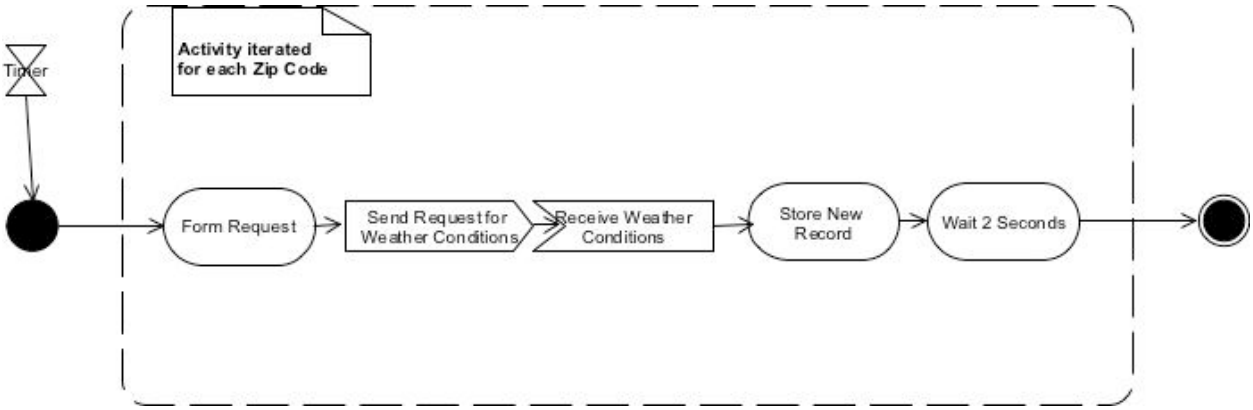

Figure 1.19: Actvity diagram for UC4 ("CollectWeatherConditions")

Figure 1.20: Activity diagram for UC5 ("ConfigureCollectionCriteria")

Class Diagram:

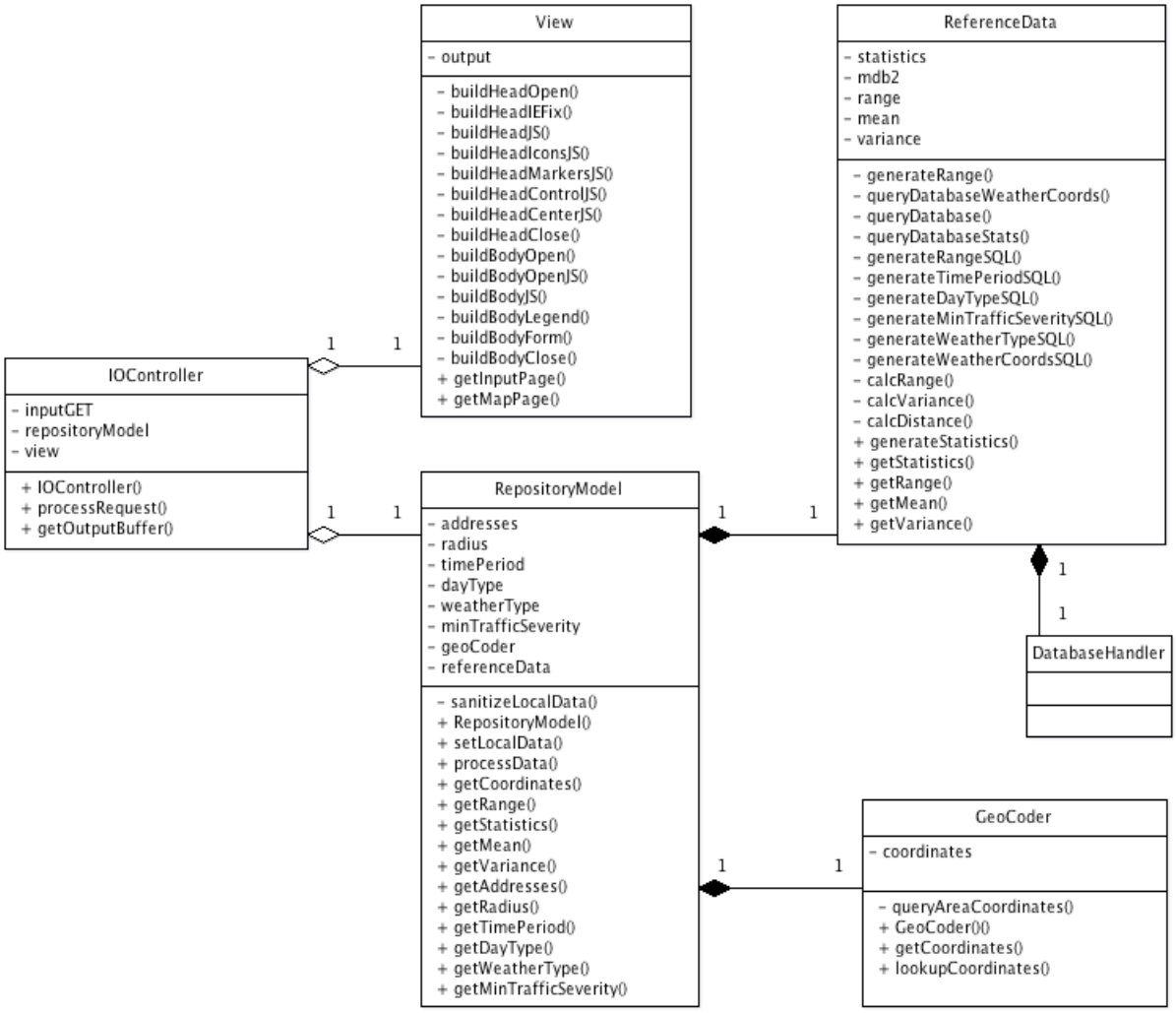

Figure 1.21: Class diagram for the "Traffic Map" system

In the "Traffic Map" system, variables have been specifically typecast due to PHP's inability to naturally enforce data types. Floating-point variables have been employed for latitude/longitude coordinates and statistical calculations such as variance and mean. A multi-dimensional array is used to store the statistics in the following structure: each element of the array has a unique key comprised of the latitude longitude and the value is the location's "weight" (see the "Algorithms" section). Strings are used for the Address(es) section of the user input. The remainder of the user-supplied input uses integers which map to specific values within the ReferenceData class. All data is stored privately and can be requested (in a cascading fashion) from the main RepositoryModel class by public "get" functions (see Figure 1.21). The DatabaseHandler implemets a "proxy" design pattern. This allows the database server to move to a different network location transparently. This design pattern also allows the Administrator to change SQL server vendors seamlessly (see "Interaction Diagrams" section).

The "Data Collection" system is procedural and contains scripts which employ "foreach" loops. Objects are used only to simplify database access and XML parsing (see Figure 1.22).

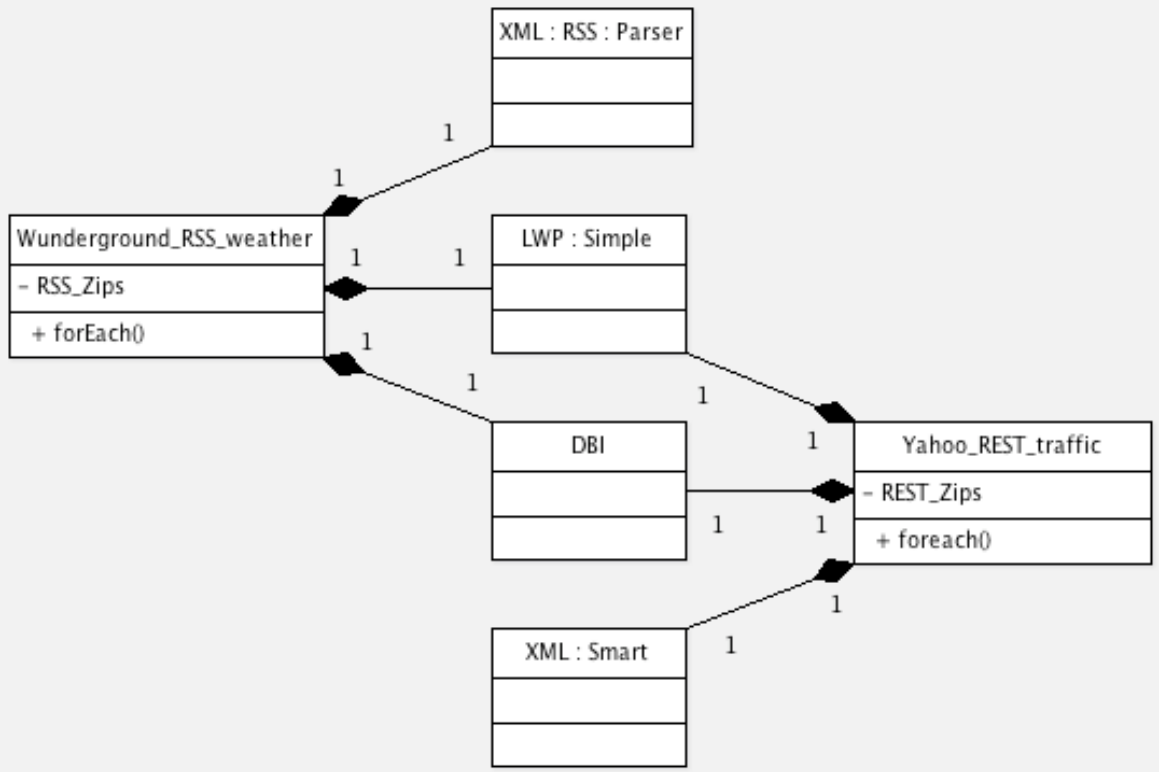

Figure 1.22: Class diagram for the "Data Collection" system (procedural design)

Data Types and Operation Signatures:

IOController
- input : string[] - repositoryModel : RepositoryModel - view : View
+ IOController() + processRequest(string[]) + getOutputBuffer()

Figure 1.23: Data types and operations for "IOController" object

RepositoryModel
- addresses : string[] - radius : int - timePeriod : int - dayType : int - weatherType : int - minTrafficSeverity : int - geoCoder : GeoCoder - referenceData : ReferenceData
- sanitizeLocalData(string, string, int) + RepositoryModel() + setLocalData(string[]) + processData() + getCoordinates() + getRange() + getStatistics() + getMean() + getVariance() + getAddresses() + getRadius() + getTimePeriod() + getDayType() + getWeatherType() + getMinTrafficSeverity()

Figure 1.24: Data types and operations for "RepositoryModel" object

View
- output : string
- buildHeadOpen() - buildHeadIEFix() - buildHeadJS(float[], float, float, float[][int], float, float) - buildHeadIconsJS() - buildHeadMarkersJS(float[][int], float, float) - buildHeadControlJS() - buildHeadCenterJS(float, float, float[]) - buildHeadClose() - buildBodyOpen() - buildBodyOpenJS() - buildBodyJS(int, int) - buildBodyLegend(float, float, string[], int, int, int, int, int) - buildBodyForm() - buildBodyClose() + getInputPage(boolean) + getMapPage(float, float[], float[][int], float, float, string[], int, int, int, int, int)

Figure 1.25: Data types and operations for "View" object

ReferenceData
- statistics : float[][int] - mdb2 : PEAR::MDB2:MDB2_Driver_mysql - range : float[] - mean : float - variance : float
- generateRange(float[], int) - queryDatabaseWeatherCoords(float[]) - queryDatabaseStats(int, int, int, int, float[]) - generateRangeSQL() - generateTimePeriodSQL(int) - generateDayTypeSQL(int) - generateMinTrafficSeveritySQL(int) - generateWeatherTypeSQL(int) - generateWeatherCoordsSQL(float[]) - calcRange(float[], int) - calcVariance() - calcDistance(float, float, float, float) + generateStatistics(float[], int, int, int, int, int) + getStatistics() + getRange() + getMean() + getVariance()

Figure 1.26: Data types and operations for "ReferenceData" object

GeoCoder
- coordinates : float[]
- queryAreaCoordinates(string[]) + GeoCoder() + getCoordinates() + lookupCoordinates(string[])

Figure 1.27: Data types and operations for "GeoCoder" object

Object Constraint Language (OCL) Contracts:

context RepositoryModel::sanitizeLocalData **post:**

addresses->sanitize_string()
range->santize_int()
timePeriod->santize_int()
dayType->santize_int()
weatherType->santize_int()
minTrafficSeverity->santize_int()

All other constraints are obvious from the interaction and class diagrams.[ix]

System Architecture and System Design

Architectural Styles:

The “Traffic Maps” system is decomposed into an architectural style best described as “Model/View/Controller”, a subset of the general “Repository” architectural style. The “model” subsystem maintains both domain knowledge and a central data repository. It is implemented via the “RepositoryModel” class and it’s associated classes (“ReferenceData”, “GeoCoder” and “XML_Parser”). The “view” subsystem displays output to the user. It is implemented via the “View” class and depends on the “RepositoryModel”. The “controller” subsystem manages the sequence of interactions with the user and is implemented with the main “IOController” class.

Identifying Subsystems:

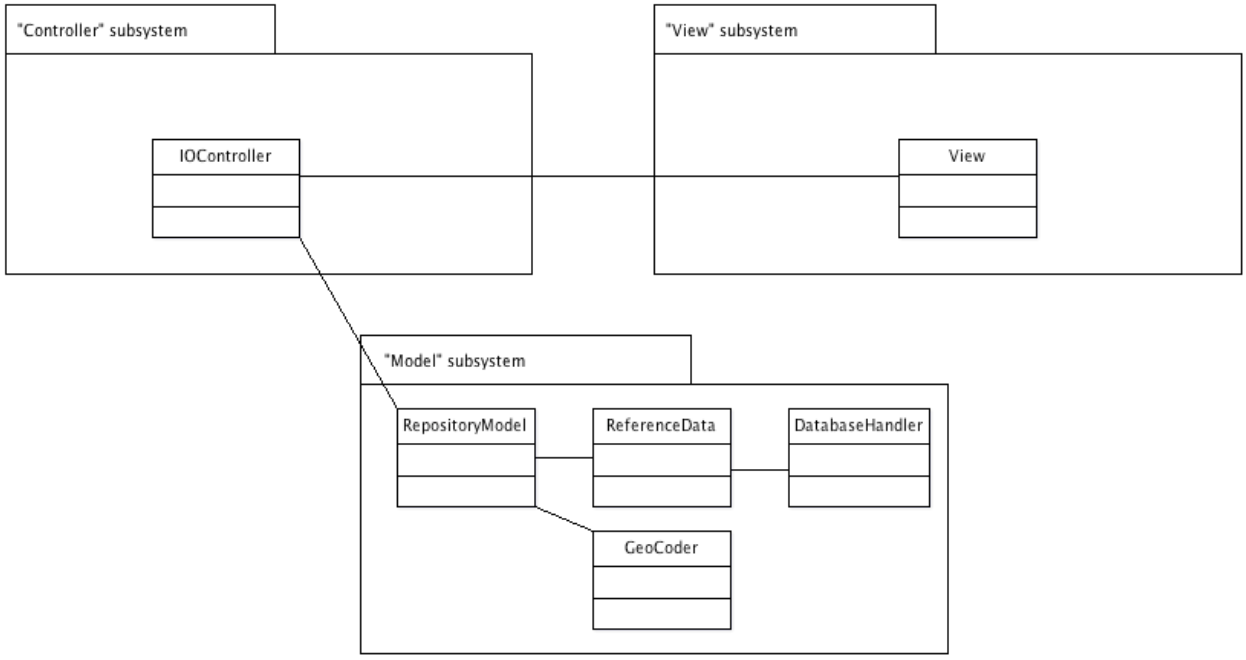

Figure 1.28: Package Diagram of “Traffic Maps” system

Mapping Subsystems to Hardware

The “Traffic Maps” and “Data Collection” systems both run on the server. The “View” subsystem of “Traffic Maps” generates dynamic Javascript code that runs on the client’s desktop. The client’s desktop communicates with “Google Maps” directly using data provided by the server.

Persistent Data Storage:

Historical traffic and weather data are stored in a relational database with the following schemata:

Field	Type	Null	Default
id	int(11)	No	
_Modified	timestamp	Yes	CURRENT_TIMESTAMP
_Created	timestamp	Yes	0000-00-00 00:00:00
_Title_HASH	varchar(32)	No	
_Description_HASH	varchar(32)	No	
Title	varchar(255)	No	
Description	text	No	
Latitude	decimal(9,6)	No	0.000000
Longitude	decimal(9,6)	No	0.000000
Direction	varchar(255)	No	
Severity	int(11)	No	0
Category	varchar(255)	No	
ReportDate	timestamp	Yes	0000-00-00 00:00:00
UpdateDate	timestamp	Yes	0000-00-00 00:00:00
EndDate	timestamp	Yes	0000-00-00 00:00:00

Figure 1.29: Database schema for the "Yahoo_REST_traffic" table

Field	Type	Null	Default
id	int(10)	No	
_Modified	timestamp	Yes	CURRENT_TIMESTAMP
_Created	timestamp	Yes	0000-00-00 00:00:00
ZipCode	varchar(5)	No	
Latitude	decimal(9,6)	No	0.000000
Longitude	decimal(9,6)	No	0.000000
Temperature_F	decimal(4,1)	No	0.0
Humidity_PERCENT	int(11)	No	0
Pressure_INCH_HG	decimal(5,2)	No	0.00
Conditions	varchar(255)	No	
WindDirection	varchar(255)	No	
WindSpeed_MPH	decimal(4,1)	No	0.0
pubDate	timestamp	Yes	0000-00-00 00:00:00

Figure 1.30: Database schema for the "Wunderground_RSS_weather" table

Network Protocol:

Within the "Traffic Maps" system, the HTTP protocol is used for all communication between both the client and the server and the client and "Google Maps". Within the "Data Collection" system, the HTTP protocol is used for all communication between both the server and "Yahoo! Maps and Traffic" and the server and "Wunderground.com". HTTP (via the POST method) was chosen because it is the standard web server and browser protocol.

The server communicates with the database via a local UNIX socket. The database service may be moved onto a different physical server by modifying the implementation of the "PEAR::MDB2" class ("Traffic Maps" system) and the "DBI" class ("Data Collection" system) to use the MySQL network protocol.

Global Control Flow:

Execution order:

The "controller" subsystem within the "Traffic Maps" system is event-driven. The implementation of both the "view" and "model" subsystems within the "Traffic Map" system are procedure-driven. The "Data Collection" system is entirely procedure-driven.

Time Dependency:

The “Traffic Maps” system is a real-time system with time limitations of 30 seconds in processing requests. The “Data Collection” system is of the event-response type and by default is triggered every 30 minutes by a timer (the frequency is configurable).

Concurrency:

Threaded processes are not used by either the “Traffic Maps” or “Data Collection” systems.

Hardware and Software Requirements:

Client:

- Internet connection with minimum bandwidth of 56Kbps.
- Screen resolution of at least “1024 x 768” is required.
- Javascript functionality must be enabled in the web browser.
- The following web browsers are supported (Google Maps requirement[x]):
  IE 6.0+, Firefox 0.8+, Safari 1.2.4+, Netscape 7.1+, Mozilla 1.4+, Opera 8.02+

Server:

- Internet connection with minimum bandwidth of 56Kbps per concurrent client.
- 100MB of free disk space per year for the storage of historical data.
- The server must provide the following services:
  Apache HTTP Server 1.3, PHP 5 with PEAR extensions, MySQL Server 5, Perl 5.8 with DBI, XML and LWP modules

Algorithms and Data Structures

Algorithms:

In the "Traffic Maps" system, the "ReferenceData" class is responsible for calculating basic statistics. The SQL query returns an array that is indexed by unique latitude/longitude coordinates. Each key in the array holds an integer value that is the total sum of incidents at that location multiplied but the respective severity – this is labeled the "weight". The variance of the entire sample is calculated and stored as a floating-point variable. The mean and variance attributes are used by the "View" subsystem to generate coordinate markers of varying size and color.

Data Structures:

In the "Traffic Maps" system, the "RepositoryModel" subsystem utilizes several simple data structures. Multidimensional arrays were chosen to compose the structures due to their flexibility and ease of manipulation in the PHP language.

User Interface Design and Implementation

Preliminary Design:

The user starts by navigating their web browser to the project's URL. For both Use Cases, criteria is entered on the same input page. Address(es) and radius are entered via a text box; Time Period, Day Type, Weather and Severity are chosen via radio buttons. User selects submit to begin processing. For the "View Statistics Within Area" Use Case, the map is displayed with statistical markers overlaid on a geographical map centered at the address provided. For the "View Statistics Along Route" Use Case, the map is displayed with statistical markers overlaid on a geographical map with the provided route highlighted. User may show/hide severity markers at their convenience by clicking checkbox. They may also revise their search from the output page following a similar procedure as above. See Figure 1.31 for a screen mock-up of the "View Statistics Within Area" service.[xi] See Figure 1.32 for a screen mock-up of the "View Statistics Along Route" service.[xii]

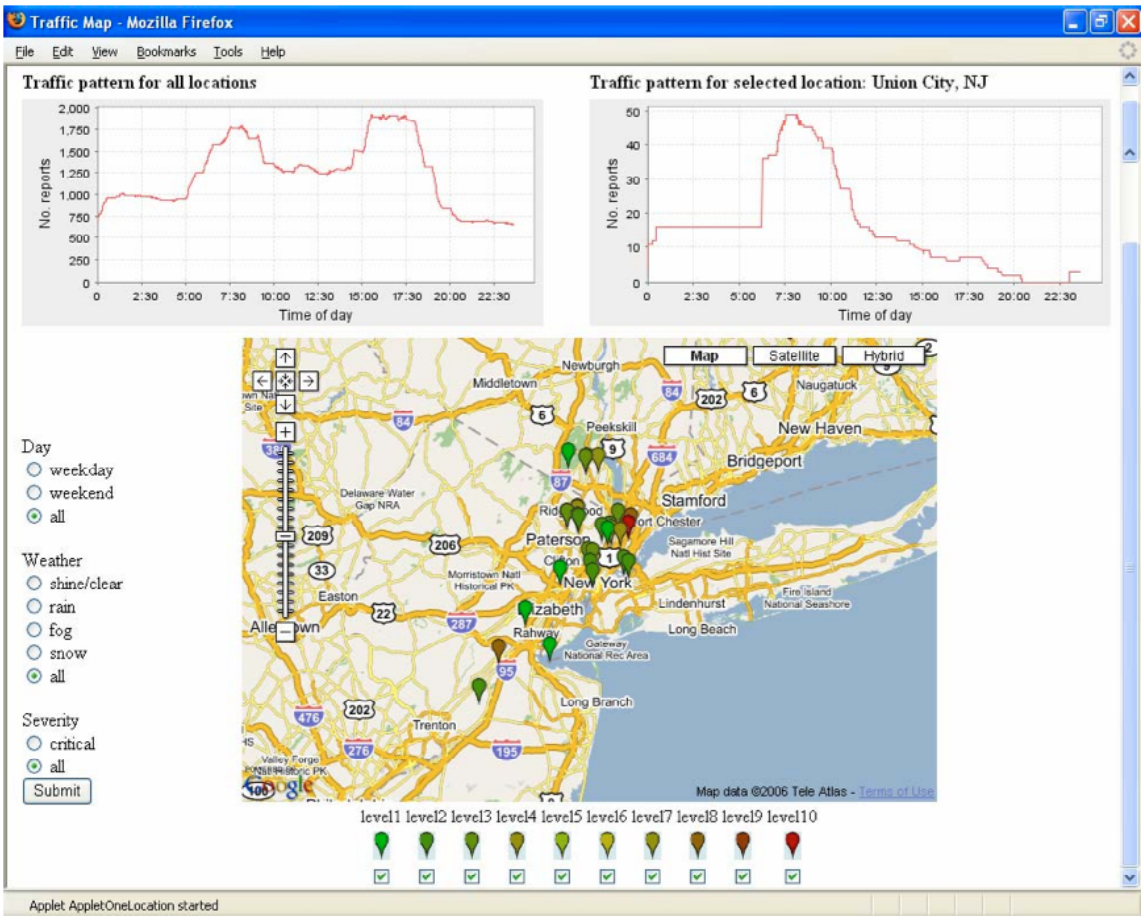

Figure 1.31: "View Statistics Within Area"

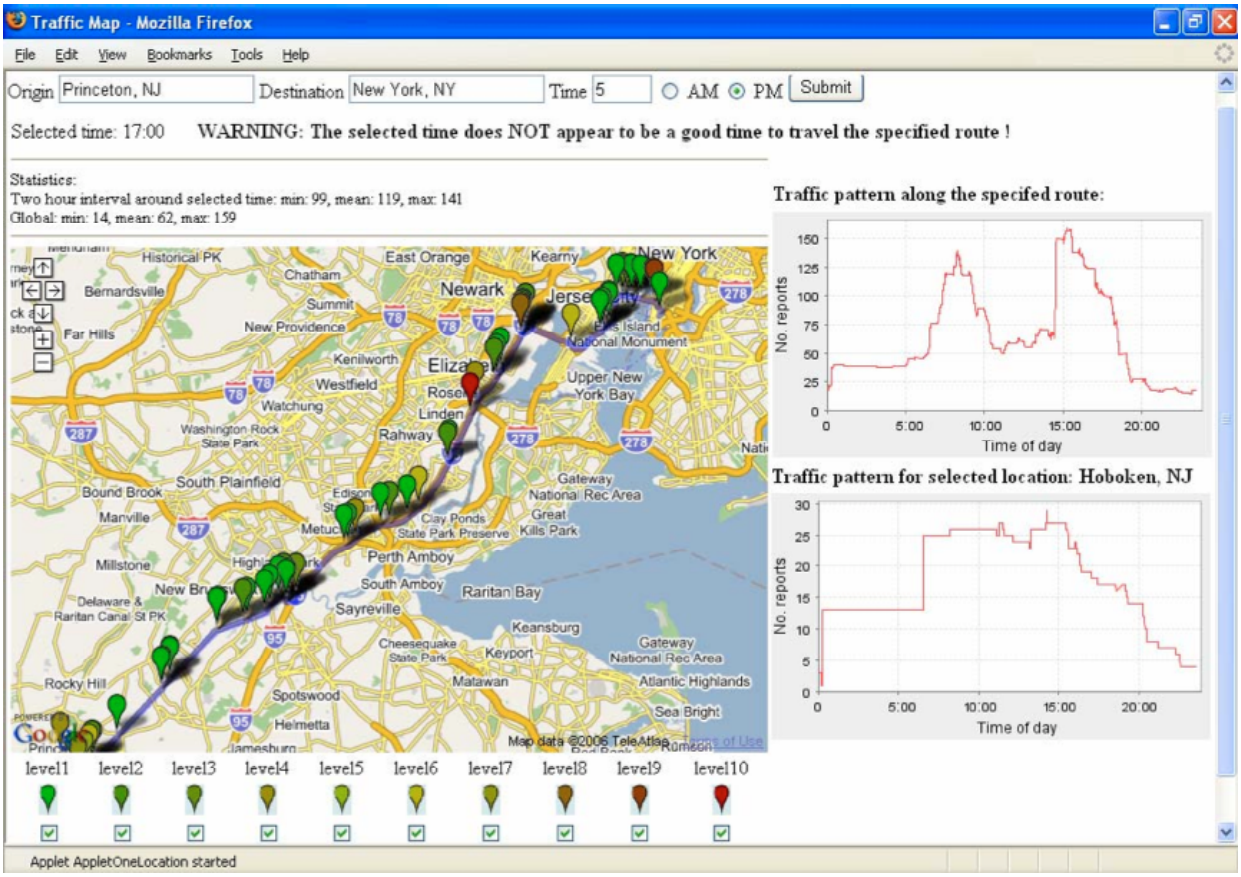

Figure 1.32: "View Statistics Along Route"

Design and Implementation:

The input form and map output have been designed with "ease-of-use" as the primary goal. Switching between the 2 Use Cases ("ViewStatisticsWithinArea" and "ViewStatisticsAlongRoute") on the input form will be handled via client-side scripting. The map output page will contain the input form for easier re-submission. The map markers will have detailed statistical information contained in clickable pop-ups. Links, such as current traffic conditions and weather for the applicable location, will be located on the map output page to aid users in their travel planning. See Figure 1.33 for a screenshot of the "ViewStatisticsWithinArea" Use Case.

Testing has revealed that the map loading time is unacceptable with more than 100 active markers displayed on the map. Therefore trivial markers will be consolidated or dropped, especially in the "ViewStatisticsWithinArea" Use Case.

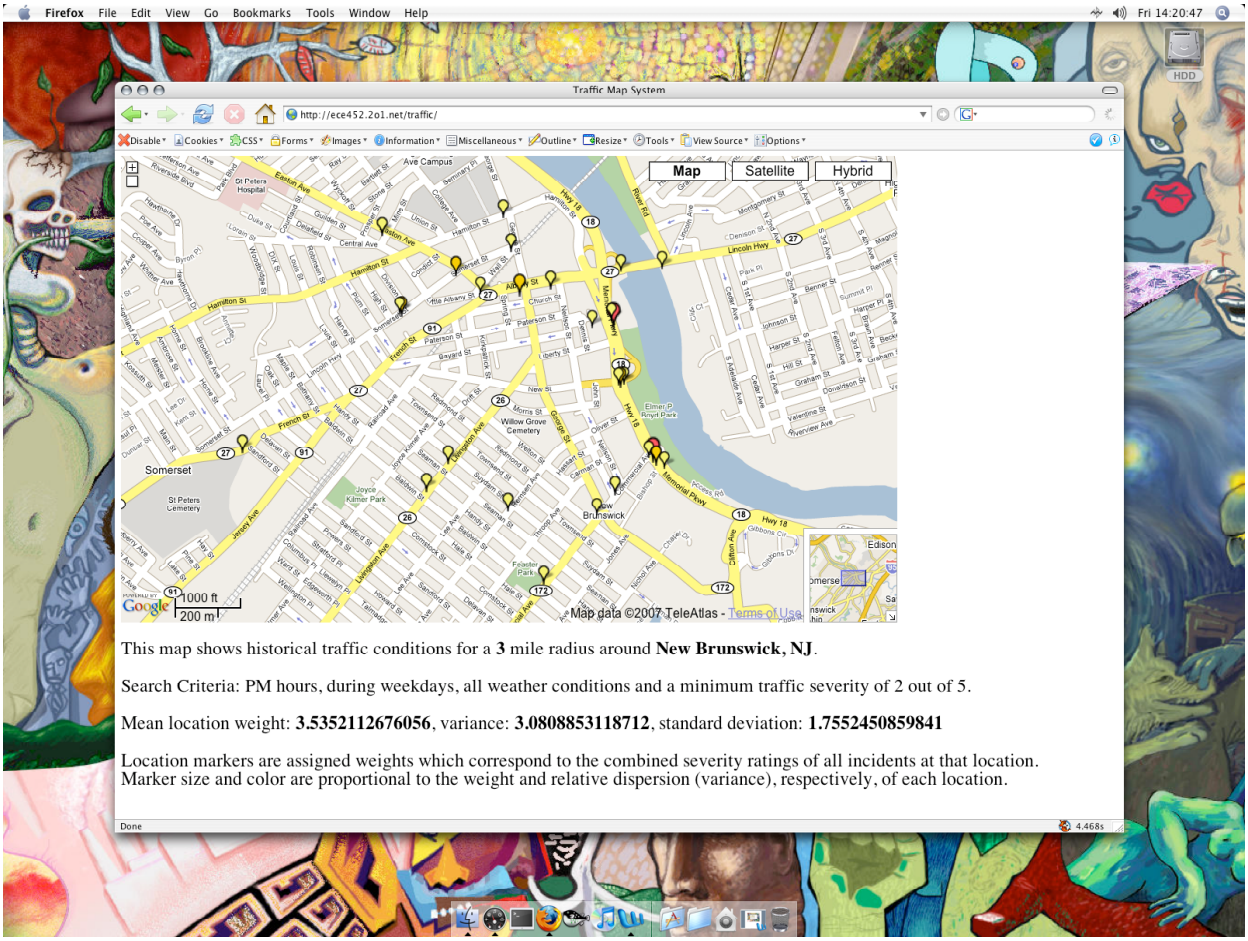

Figure 1.33: "View Statistics Within Area"

User Effort Estimation:

UC1 (View Statistics Within Area) used as a typical example.

1) NAVIGATION: total 2 mouse clicks, as follows
   a. Open browser
      -- after completing data entry as shown below --
   b. Click "Submit"

2) DATA ENTRY: total 6 mouse clicks and 7 keystrokes
   a. Click cursor to "Address"
   b. Press the keys "0", "7", "0", "1" and "3"
   c. Click cursor to "Radius"
   d. Press the keys "1" and "0"
   e. Click "Time Period" button
   f. Click "Day Type" button

g. Click “Weather” button
h. Click “Severity” button

Approximately 87% percent of the input effort goes towards data entry. The remaining 13% goes toward user-interface navigation.

History of Work and Current Status of Implementation

History of Work:

UC1 (ViewStatisticsWithinArea): Implemented and function on 2007/03/12
UC2 (ViewStatisticsAlongRoute): Partially implemented
UC3 (CollectTrafficConditions): Implemented and functional on 2007/02/19
UC4 (CollectWeatherConditions): Implemented and functional on 2007/02/19

UC2 is under active development; the entire subsystem has been written. However, no suitable routing service has been found to provide address to polyline coordinate mapping. Once a solution has been found, it will be a simple integration into the GeoCoder class. Previously working routing solutions involving the undocumented features of "Google Maps" have been mitigated by the vendor..[xiii]

Historical traffic and weather data collection has now been operating for a couple months; see Figures 1.34 and 1.35 for more details.

Row Statistics	
Statements	**Value**
Format	dynamic
Collation	latin1_swedish_ci
Rows	55,381
Row length ø	256
Row size ø	358 Bytes

Figure 1.34: Maturity of historical traffic table

Row Statistics	
Statements	**Value**
Format	dynamic
Collation	latin1_swedish_ci
Rows	24,752
Row length ø	81
Row size ø	129 Bytes

Figure 1.35: Maturity of historical weather table

Milestones set out in Report #1 and Report #2 have all been met successfully, with the exception of UC2 ("ViewStatisticsAlongRoute").

Breakdown of Responsibilities:

Jon Lipovac is the sole group member.

Conclusions and Future Work

Numerous technical challenges were encountered in the development of this software product.

Time constraints of the SQL query:

The realization of an efficient query to pull the applicable traffic and weather data from the database was quite difficult. Joining the 2 tables with the constraints of the user-supplied criteria would only provide timely results (less than 30 seconds) if the range/radius was kept to a minimum (about 20 miles in suburban areas, 10 miles in metropolitan NYC). The table indexing was optimized and the queries have been refined multiple times but the only solution found for large radiuses so far is to limit the amount of records involved in the table joins. This provides for a less than optimal traffic prediction distribution and it is recommended that the user keeps the range selection as low as practical.

Load constraints of the client browser:

The placement of over 100 markers within the client-side JS code for the interactive Google map is not reliable. Often when the amount of markers exceed this threshold, the browser either slows to a crawl and the JS terminates or the entire browser dumps its core (older versions of Mozilla). This has been mitigated by the automatic deactivation of excessive markers; only the 100 highest "weighted" locations are displayed in this scenario.

Downtime of the weather or traffic condition web sites:

This software utilizes Yahoo! Maps and Traffics' REST API for pulling current traffic conditions. For a couple weeks in February and March 2007 the Yahoo site was failing to provide updated traffic incidents and this resulted in a less effective sample set for making future traffic predictions. The web site has since begun functioning properly and the sample set has grown enough to mitigate any inconsistencies as a result of the downtime. Similar scenarios, although less likely, could also occur with the weather conditions web site.

Flexibility of PHP:

Programming this software in PHP provided an unanticipated challenge; while PHP supports Object-Oriented programming it is often too relaxed. It is far too easy to sway from the principles of OO and implement purely procedural PHP code as a temporary hack. Extreme caution has been taken to follow the "rules" of OO and maintain that consistency throughout the code.

The software engineering techniques that were taught in this course were invaluable when it came time to code the project. Most of the time it was merely a matter of coding the interfaces by reading off the class diagram. The UML notation required in this course turned out to be the most important skill. Managing even a small project like this would be overwhelmingly impossible without UML-aided design. The design patterns taught were also a useful tool although it seems that in order to utilize them properly and efficiently, one must be very experienced with identifying when and how they should be used. Some design patterns were implemented in the code without even realizing they were a pattern; for example, the database-handler object as a "proxy" pattern (part of the standardized PHP PEAR library).

There is certainly "domain" knowledge that is necessary to implement a traffic monitoring and prediction system. Some of the knowledge was learned in the first "Specification" stage but additional information would have been helpful -- for instance, whether incidents involving construction should be completely discarded. Perhaps it would be helpful to include them but give them less weight. Perhaps the statistical model implemented was too simple and provides useless results. Traffic prediction may only be properly modeled by an advanced statistical technique such as the state-of-the-art "Bayesian" learning model.

There are many possible directions for the future work on this project. One direction would be improving the statistical analysis model. Another direction may be integrating this software on a mobile device. However, the code in its present state already overloads the average database server – it would only be practical for a mobile device to display the statistics using a server that makes extensive use of caching, especially if the server is available to many devices.

References

[i] "Software Engineering", page 23 (Ivan Marsic)
[ii] "Software Engineering", page 25 (Ivan Marsic)
[iii] http://www.inrix.com/techdustnetwork.asp
[iv] http://www.clearchannel.com/Radio/PressRelease.aspx?PressReleaseID=1443
[v] http://microsoftstartupzone.com/blogs/ebt_success_stories/archive/2006/09/20/830.aspx
[vi] http://www.trianglesoftware.com/about.aspx
[vii] http://en.wikipedia.org/wiki/Functional_design
[viii] http://en.wikipedia.org/wiki/Proxy_pattern
[ix] "Object-Oriented Software Engineering", page 371 (Bruegge & Dutoit)
[x] http://maps.google.com/support/bin/answer.py?answer=16532
[xi] Image taken from "Software Engineering", page 29 (Ivan Marsic)
[xii] Image taken from "Software Engineering", page 30 (Ivan Marsic)
[xiii] http://groups.google.com/group/Google-Maps-API/browse_thread/thread/9301342f991808d0