Masters Project Report

The Prediction and Notification Of Bus Locations Using Live Data

Karl Connon

A thesis submitted in part fulfilment of the degree of

MSc Advanced Software Engineering in Computer Science

Supervisor: Dr. M. Ó Cinnéide

UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

April 29, 2011

Abstract

As GPS chips become more ubiquitous in society, new applications and location based services that take advantage of this data source are becoming as ever-present as the devices themselves. One of the major and most obvious areas that has benefited from this growth is travel. Cars have satellite navigation systems that can guide their users to their destinations and entire fleets of taxis can be remotely tracked from a remote destination to provide a fast and efficient service to their clients. With this GPS data available it seems a natural progression that users of a bus service to be able to tell how far away their bus is based on live data rather than relying on a static timetable. This thesis documents the attempt at using the GPS data of a fleet of buses, combined with the provided timetable, to provide such an alerting system.

Table of Contents

List of Figures

List of Tables

Chapter 1: **Introduction**

This report documents the development of an automatic bus location prediction and notification system. Each chapter presents a certain stage or aspect of the systems development and discusses its various issues or needs. Chapter 2 is mainly concerned with the high level abstractions that the final system will hang off. Structures such as the representation of a map and database are decided upon here and some of the lower level implications and necessary supporting objects are discussed. In chapter 3, development begins on a basic structure that can effectively and efficiently generate notifications. At this stage, it is the basic process involved in creating the predictions that is under scrutiny, and not the accuracy of the predictions created. An emphasis is put on allowing the system to be extended, so that accuracy can be addressed through easy modification and extension. With the first results obtained from this initial version, a series of metrics are defined that can be used to assess the systems performance between different iterations. It is then in chapter 4 that domain and application level issues are addressed in order to improve accuracy. With each new or altered component, a simulation is run, allowing the collection of data that would have been generated over a specific week while using the new configuration. The fact that each configuration is run on the exact same data ensures that the results obtained are directly comparable to each other. As the system progresses each set of results are anaylsed. Chapter 5 looks at the other, more generally applicable considerations, that were of importance during the construction of the project. Strategies employed to ensure the system remained easy to maintain, debug, extend and scale are examined. In chapter 6, the construction and implementation of an appropriate front end to allow users interact with the system is detailed. The benefits and drawbacks of the technologies used are listed and the supporting webservice is described. Finally in chapter 7, conclusions are drawn about where the system succeeded and failed, and possible future work is sugested.

Chapter 2: **Representing The Map And Data**

The representation of a map, the locations that it defines and their appropriate representation in a database were all issues that were necessary to approach before any estimation could be attempted. An inadequate abstraction and design will filter down into the implementation and add greatly to the effort it takes to fix any resulting issues. In this chapter, these concerns are broken down into higher level design issues and the scenarios of lower level implementation.

2.1 High Level Design

In this section, the the more abstract higher level details of the system are discussed. These were the initial elements that were researched and designed and are not tied down to any specific implementation.

2.1.1 Designing The Map

In order for the system to be able to analyse and understand bus locations, it must be able to comprehend them in relation to something native to the system. Kidwell proposes building a series of zones along a bus route, each a four sided polygon enclosing a specific area of the route [1]. These zones would then be associated with the most recently observed transit time through it. Summing up the times of each zone in between a bus and its destination would yield a prediction of its arrival.

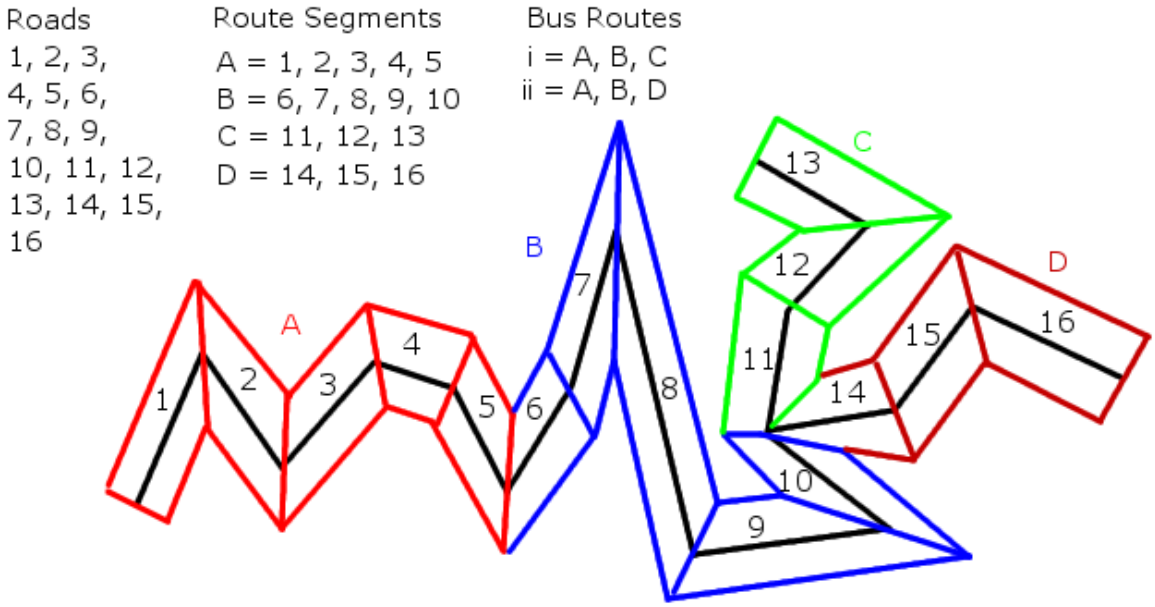
Although an interesting way to approach the issue, the widely encompassing area of a zone was not of a fine enough granularity to segregate unrelated sections that closely wound about each other, nor had the ability capture the more complicated diverging bus routes involved in this solution. This gave rise to a series of constructs, that although followed Kidwells basic design, allowed for a more accurate description of the breakdown of a bus route. Fig 2.1 shows an example of this dissection, illustrating how diverging bus routes are allowed share sections.

Map Components

Point

Points are the lowest level of description in the map. They represent a single point on the map and have no area. At a higher level of abstraction they can be used to represent bus stops and the vertices of bounding areas.

Figure 2.1: Example Breakdown Of Bus Routes into Route Segments and Roads



Bus Stop

Represent a circular area surrounding a physical bus stop. If a bus enters this area it is deemed to be at the bus stop.

Road

This describes a hexagonal area surrounding a line segment. Usually represents a section of road that a bus may travel through. It has no direction associated with it and its main use is for positioning the bus on the map.

Route Segment

This is an ordered list of roads. As it is comprised of roads it can be almost any shape, it can twist turn and even cross over itself. Unlike roads, route segments have a direction associated with them, thus the same group of roads travelled in opposite directions must be represented by two separate route segments. The purpose of a route segment is close to that of a zone in Kidwells solution, however the fact that it is defined by many smaller areas rather that just one all encompassing quadrilateral, allows a much more defined shape that can follow specific roads. Journeys through route segments will be recorded so that their duration can be used in estimations.

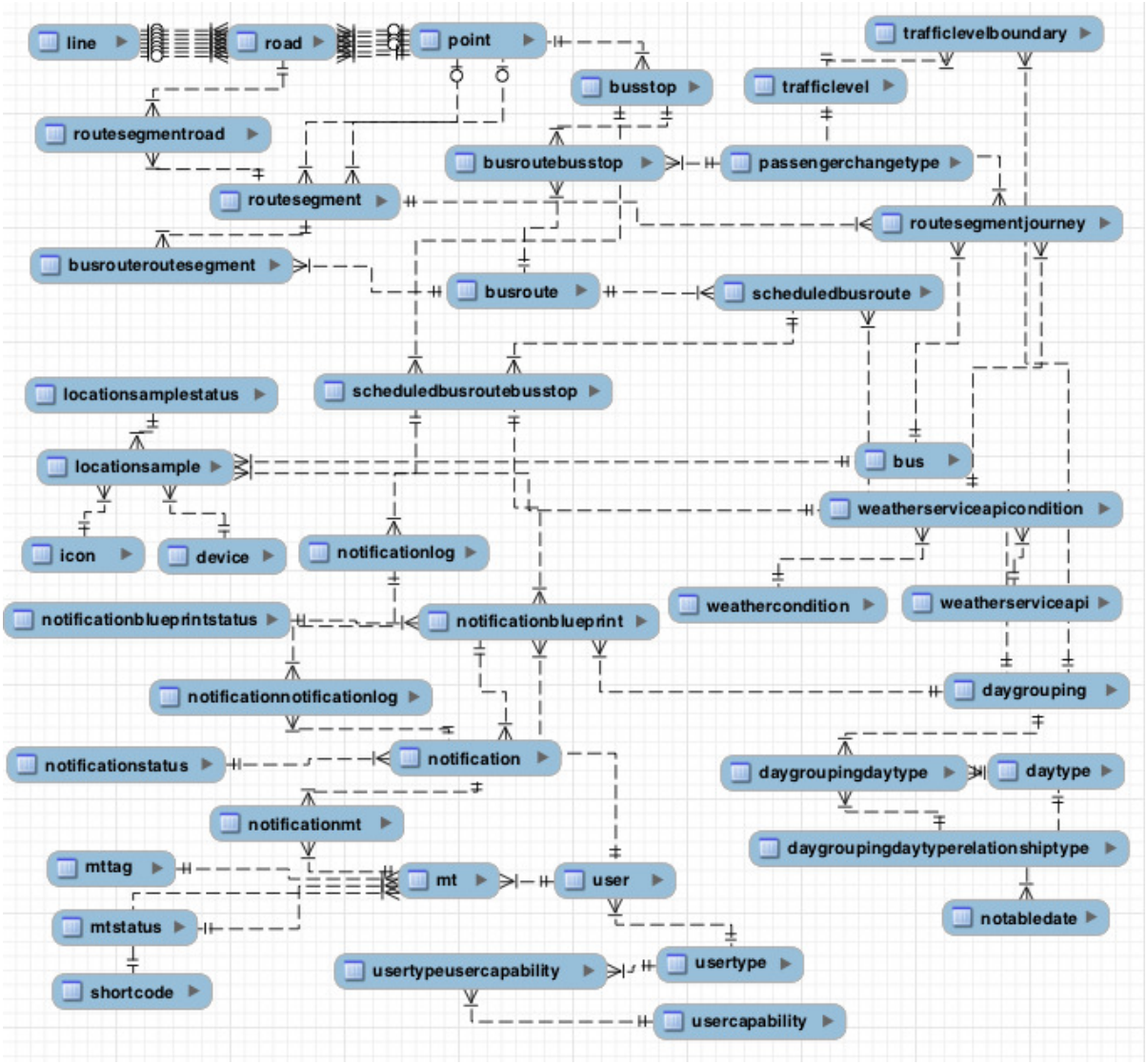
Bus Route

Represents an ordered list of route segments, bus stops and the associated passenger exchange level of the stop(i.e. If a bus stop is only set down and hence no new passengers will be picked up). This is the highest level of abstraction in the system and is able to represent all variations of actual bus routes.

2.1.2 Design Of The Database

In very early tests, most of the data was contained within a few XML files. However this quickly became unmanageable and the entire dataset was moved to a MySQL database to ensure quick access and scalability.

Figure 2.2: Database Structure



Normalising To Ensure Good Structure

The structure of the database was designed to be normalised to the 3rd normal form. Spending more time on this allowed for the natural organisation of entities to emerge. Some areas that would have caused issues later on were able to be identified here so that they would not cause any problems in the business logic itself. For example some unforeseen structures emerged with the representation of the timetable in the database; having the information of order encapsulated to *BusRoute* and *BusRoutesBusStop*, and having timing considerations separated out into *ScheduledBusRoute* and *ScheduledBusRouteBusStop* but linking *ScheduledBusRouteBusStop* to *BusStop* instead of *BusRoutesBusStop* as was expected, guaranteed that no phantom conflicts would crop up in the main process due to the replication of data

that could possibly differ from each other. This can be seen in Fig 2.2

Large Scale Tables

Good database design was also a priority due to the fact that some of the tables are guaranteed to become quite massive. At the time of writing, the LocationSamples table has grown to over two million records, yet certain subsections of this must be able to retrieved in an efficient manner to stop it slowing the entire system down and becoming a bottleneck. The proper use of foreign keys and indexing on columns where necessary has ensured that this has not happened.

2.2 Low Level Details

Here the lower level implementation issues are discussed, from aiding the development of the map to actually retrieving the location samples of the fleet. Although they do concern the implementation, they do not directly form part of the main estimation process.

2.2.1 Tools to Aid In Map Implementation

Automatic Generation Of Fine Details

Because the representation of the map is more complicated than just dividing the map into four sided polygons, the manual input of a lot of the finer details to describe the many roads needed is extremely time consuming and very error prone. Each road needs six points and seven lines to describe it, and every road that connects to another must match up exactly to the the roads that are linked to it. For this reason, a helper program was written that would allow an administrator to enter the abstract details such as the structure of a route segment as expressed in roads and then just the the bare minimum of details to describe each road (the two end points of a road and the maximum perpendicular distance from the connecting line that must be inside the road). The program will then automatically generate the remaining points and lines. A benefit of this approach is that changes to map do not necessitate a big time investment or risk. A custom LineUtils class will provide all the algebraic functions needed for this task.

Visualisation Of The Data

Another set of tools were written to make defining the map easier to understand and debugging of issues a more visual experience. They were written using the Google Maps API , and create html pages to show points and boundaries on an actual map rather than looking at numbers in a table.

- **All Location Sample Points**
  This module takes a sample of 300000 locations that were recoded by the fleet and places them on a series of maps each dealing with a vertical section of the entire range. This allows the inspection of the physical routes the buses take, the internal map can then be configured to ensure all areas are covered. Example shown in Fig 2.3

Figure 2.3: Showing All Location Samples Travelling Through A Section

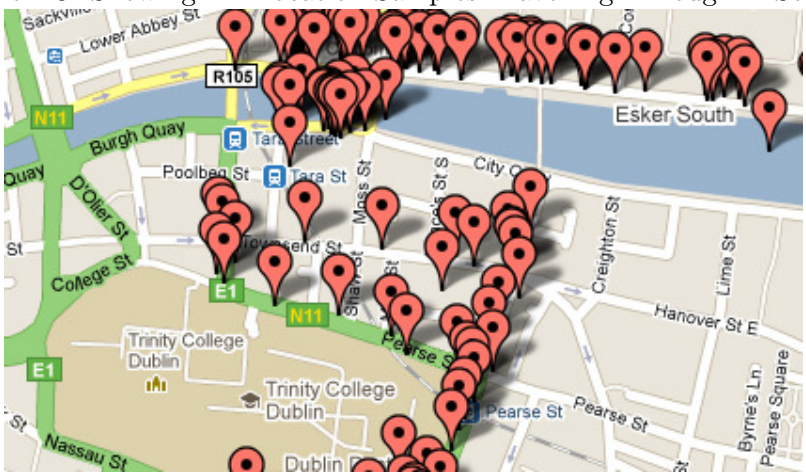


Figure 2.4: Showing Points(Red), Roads(Blue), Bus Stops(Yellow)

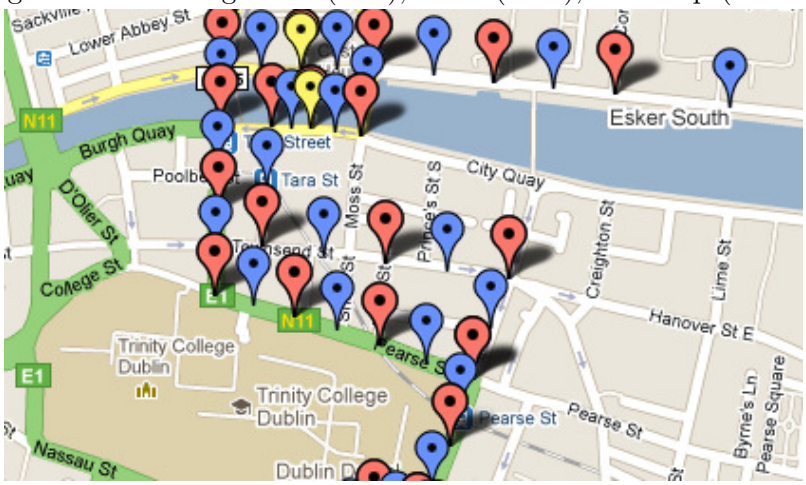
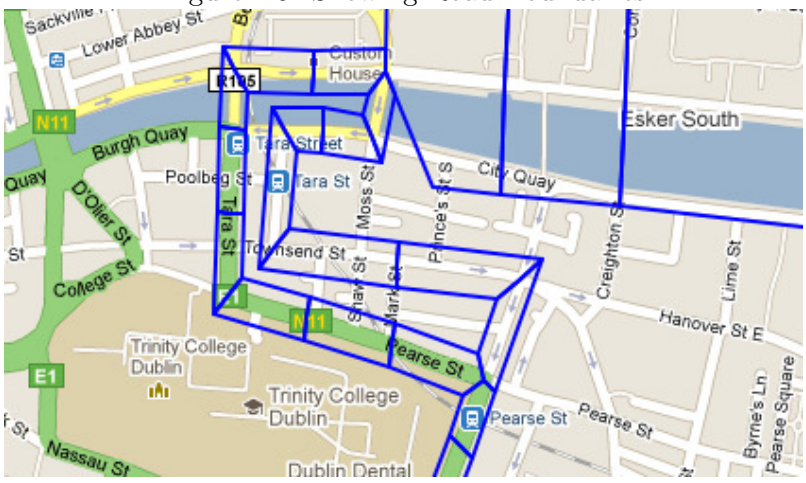


Figure 2.5: Showing Road Boundaries



- **Points And Roads On Map**
  This module allows a clear view of the bus stops, roads and adjoining points. Hovering over each marker shows the associated point and road identifiers. This allows a visual

inspection to confirm that each road is connected to the points they are expected to be and are generally in the right position. Example shown in Fig 2.4

- **Road Boundaries**
  This module shows the user the breakdown of area within the system. It is probably the most important because it aids in ensuring that no two roads overlap each other. If one road is too wide and overlapping with another, the maximum width of the roads can be altered and the details of the map regenerated to fix the issue. Example shown in Fig 2.5

2.2.2 Lower Level Database Choices

The Use Of Stored Procedures

MySQL 5.0 added the ability to use stored procedures instead of having to code all the SQL code within the Java. These stored procedures have been used extremely heavily within the TextMeMyBus System, they allow for quick development of SQL calls from within a native environment. MySQL Workbench provides instant feedback to errors within stored procedures as they are being written without having to start off any process to test it. Stored Procedures also provide the benefit of an extremely fast way of changing the behaviour of an application without having to alter Java or rebuild any code. For instance, if a single device in the network begins uploading bad data that's corrupting averages, the stored procedure that logs the data can be quickly altered to ignore that devices information or the stored procedure that gets the averages can be altered to omit anything received from that device. It also provides a means for encapsulating many sequential operations that perform a single task into a single component. Some data intensive operations can be very expensive to perform in Java due to the large amount of data that may have to be transferred, allowing these behaviours to be performed on the server itself can greatly improve performance.

2.2.3 Gathering Location Samples

Data Source

As with most Automatic Vehicle Location systems, each bus in the fleet is equipped with an internet connection through which it can upload its current location. Many ways of determining this location are available such as a signposting system, in which RFID chips are activated as they pass set locations along the route. The location in this scenario is determined by a GPS chip, when the device is turned on, it transmits its current location to the main server of the client every 75 seconds so that near constant tracking is available. One downside to this is that the signal is cut off in tunnels and long blackouts can appear in the data.

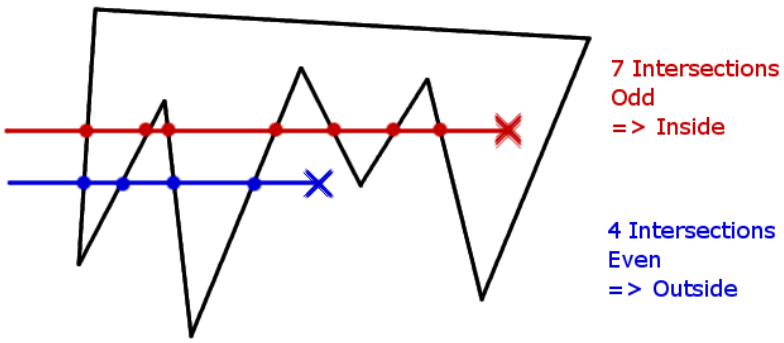
Data Retrieval

Once the data is uploaded to the server, the TextMeMyBus system uses the Apache HTTP-Client library to log in to the clients server, open and maintain a session and access the information. This raw data is provided in an XML format and provides information such as the device id, bus registration number, location and the time that the location was recorded.

Ray Casting To Place Points On The Map

Once these location samples enter the system, there must exist a way of understanding where in the map they fall. On the lowest level, this will be achieved by checking if the point falls within a polygon that describes a road. A well tested method of achieving this is ray casting. It is used extensively in graphics and satellite navigation systems to see if a point falls within a regular polygon and fits this situation perfectly. The process involves drawing an imaginary line through the point in question that completely cuts across the entire polygon(figure 2.6). The count of times the line and polygon intersect before the line reaches the point is calculated, an odd number implies the point is inside the polygon and an even number implies it lies outside the boundaries. The 'LineUtils' class mentioned in section 2.2.1 is used for all the algebraic needs of ray casting.

Figure 2.6: Ray Casting



2.3 Conclusion

A lot of time was put into developing a good abstraction to represent both the constructs of a map and how the necessary information should be modeled in the database. As this serves as the basis for the entire project, badly modeling this level due to a poor understanding, would have filtered down to the implementation and led to wasting time writing code that didn't behave as expected when processing read world information.

Chapter 3: **A Basic Estimation And Notification System**

At first a basic model was developed. Its aim was not to focus on accuracy but to discover the structure of a good framework for generating estimations and notifications. From there, issues with the domain could be identified and then new components written and inserted to attempt to solve these issues.

Dailey states that predictions based solely on the current motion parameters detected by an automatic vehicle location system(speed, direction), can produce only very approximate results[2], for this reason historical results are collected by the system, both on a volatile level such as the recent activity history of the bus and on a more permanent system wide basis such as the details of each observed trip though a route segment. Six basic needs were identified:

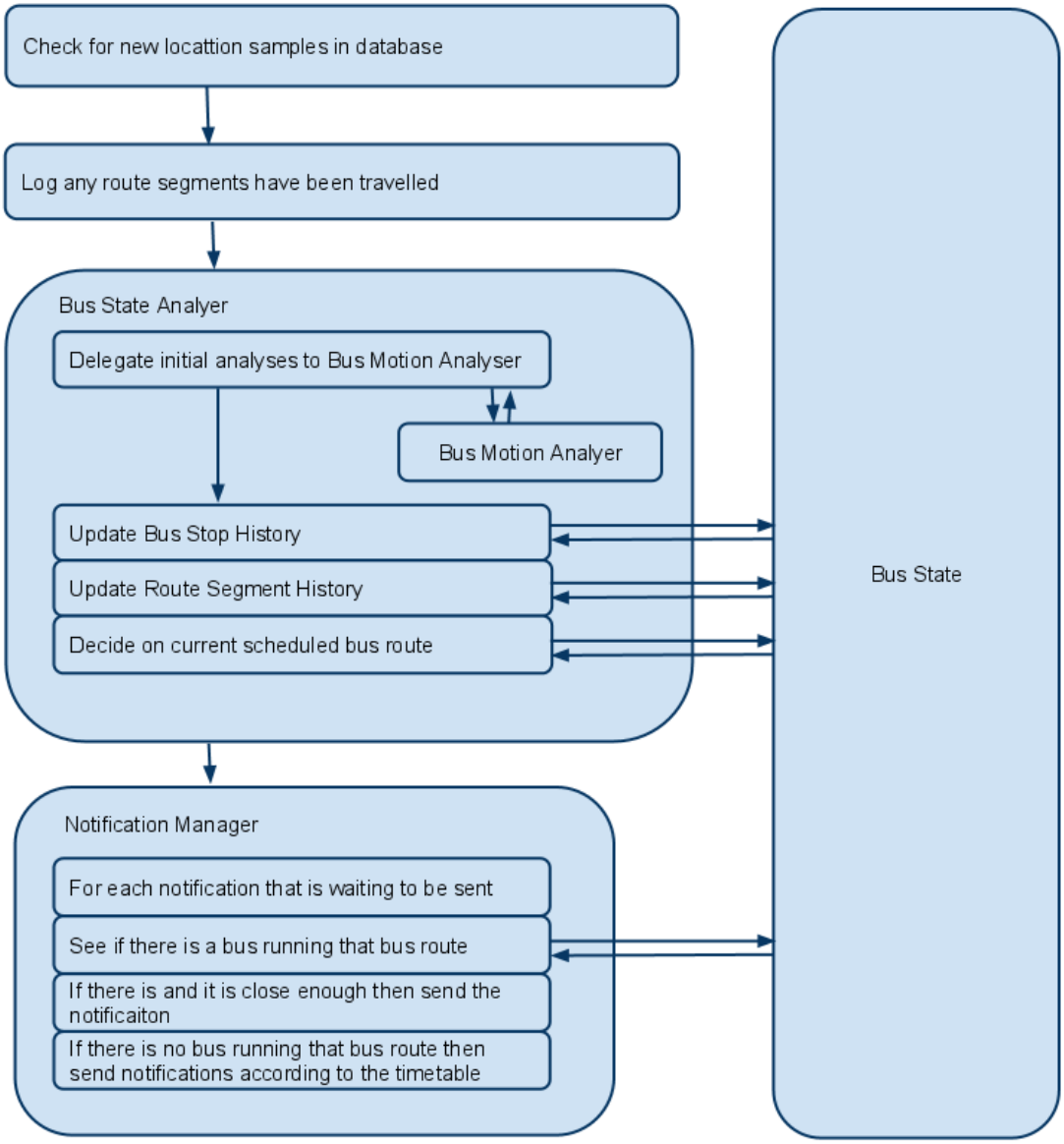
- A simple and easy way to represent and access the knowledge the system has of its environment, as it makes assumptions and builds a view on what the buses are doing(section 3.1).
- A way of abstracting the meaning of location samples before they are passed to components for analysing(section 3.2).
- A way of determining and permanently recording how long it takes buses to make individual trips through each route segment(section 3.3).
- An extendable way of analysing a bus over time, so that a current picture can be built up of its behaviour(section 3.4).
- A way of determining when notifications need to be sent to users(section 3.5).
- A way of determining the level of quality and success the estimation process had(section 3.6).

At the end of this chapter, the results achieved are documented and discussed. Figure 3.1 shows the general process taken by the program by the end of this stage.

3.1 Context

At the heart of the TextMeMyBus system is its singleton context object. Any object that contains information, either known or guessed, about the systems environment should be accessible through the context. All estimations of bus arrivals are based off the information stored within the contexts sub components. The two most important components contained within the context are the model and the bus states.

Figure 3.1: General Process Of The Basic Version



3.1.1 Model

Nearly all aspects of the program need access to the configuration and map data contained in the database, in order to provide a single non complicated point of access to this information, a domain model is created and data such as the descriptions of map structures and expected timetables are loaded into the appropriate objects within the model when the context is initiated. Any one to n relationship within the database is represented with the single object containing the references to all its owned objects, this allows for quick access to any constituent parts from any area of the program. Another benefit of this single point of access is that the TextMeMyBus model ensures that all equivalent objects point to the same object in memory, hence equivalency can be tested with a simple ==.

3.1.2 Bus State

This object represents everything that is currently known about an individual bus that is uploading data. Information such as the current location of the bus and the time that location was recorded are all stored in here. Later in the process, when more complicated approaches are used, different sets of information are also stored in here. If a component derives or estimates something about a particular bus, that information is stored in here, where it can be accessed by other components to make further estimations that must also be stored here.

3.2 Bus Motion Analyser

The raw location samples that travel through the system can have some issues associated with them. If each were to be blindly processed in order, some valuable insights would be missed that would throw off the validity of any results. For instance, if a bus enters a route segment and proceeds to stop transmitting before it leaves the route segment, the analysing component will continue to retrieve the same location for that bus until it starts transmitting again, unless it is known exactly what had happened, the system may log that that bus has taken an extortionately long time to traverse that route segment instead of realising what had actually happened.

Therefore before any detailed interpretation of the incoming location samples can be undertaken, it is important that the act of filtering the location samples be carried out. Components carrying out more complicated analysis of location samples do not have to worry about understanding the basic meaning of location samples. They deal with the higher abstractions of 'valid location samples' and location samples that have triggered some form of notification, such as a bus stopping its upload of details or a bus not moving for an extended period of time. For this purpose, bus motion analysers exist to determine the basic states of movement of a bus and the continuity of the data associated with them. They act as filters through which 'meaningless' location samples are filtered out, and instead notify components of events concerning the states of the data and motion of the bus.

3.2.1 Types of Data States

Four data states were identified. These states refer to how continuous the location samples are and point out when there have been gaps that must be dealt with by other components.

- **OK** - There has been no gaps, and this location sample follows on normally from the last
- **GAP IN RECORDING RECORDS** - If the bus enters a tunnel or goes through an area of bad coverage this will stop newly recorded locations from being inserted even though the bus is still moving. These type situations need to be processed by other components.
- **GAP IN INSERTING RECORDS** - If for some reason the process which downloads and logs location samples goes down, there will be a gap in the insertion of location samples. If the notification process comes across this situation, it needs to be processed by other components as it will affect assumptions.

- **UNKNOWN** - The starting point of the data status.

3.2.2 Types of Movement States

Three states of movement were defined.

- **MOVING** - The bus is actively changing location.
- **STOPPED** - The bus has stopped moving and has been in a single location for longer than the allowed time.
- **UNKNOWN** - When there is an issue with data state, the movement state is unknown until a subsequent location.

3.2.3 Letting This Flow To Other Components

Although bus motion analysers do this low level of interpretation, they themselves to not do anything with the information; they are merely delegate objects that analyse the low level meaning of location samples and inform the delegating components of this meaning via method calls. To this end, on their creation, they are passed an object implementing the bus motion analyser delegator interface. Any component wishing to perform deeper analysis of location samples, simply creates an instance of the bus motion analyser, registers itself as the delegator, passes all location samples it receives to the bus motion analyser and only processes the location samples the bus motion analyser passes back with the relevant method calls.

The amount of time a problem needs to be occurring for, before its notification is passed back to the delegator, is configurable per bus motion analyser. Therefor it is possible for different sets of components with different tolerances of error to use different bus motion analysers and receive different notifications for the same set of location samples.

3.3 Route Segment Journey Analyser

With each bus route divided into route segments, the system must be able to decide how long it takes a bus to get through each of these route segments. The first step in this process is to build up a large sample of journeys it has observed buses making through each one. A route segment journey analyser is employed here for just this purpose. It accepts location samples, matches the location to a road on its map and finds all the route segments this road could possibly be on. If one of the route segments that it could have possibly been on before, is not in this new set, it checks the likelihood that the bus had actually been on that route segment and has just finished traveling though it. This likelihood is calculated on the number of the route segments roads the bus had travelled through and the order in which they were travelled. If the likelihood is over a predetermined value then this route segment journey is passed back to the database logger that owns it and an entry is made to the database. In this basic version no interpolation is carried out with the road the bus is currently on nor is their any interpolation of time the bus entered and exited the route segment.

3.4 Bus State Analyser

It is in this component that all assumptions about the history, current state and future of each bus is decided and entered into the bus state. As the bus state objects are the most valuable resource during the process of estimating bus stop arrival times, it is of utmost importance that every assumption and guess the bus state analyser makes are as accurate as possible.

3.4.1 Expansion Through Delegation And Composition

The only pieces of bus state information that the bus state analyser sets itself, are those that can be directly read from the current location sample. All other analysis is done through other components contained within. All of these constituent components that add to or alter a bus state based on a new location sample, must remain in sync with one another. For this reason the bus state analyser has one bus motion analyser and never passes any unprocessed location samples to its member components, instead all of these components must also implement the bus motion analyser delegate interface, and all notifications from the single bus motion analyser are passed down to the appropriate delegate methods in the relevant order. In this way the bus state is built by the bus state analyser in a step by step manner, some steps relying on the ones that come before.

3.4.2 Components Of The Bus State Analyser

In this basic version, these are the components that extend the bus state analyers.

Route Segment Journey History Analyser

This component uses the exact same route segment journey analyser talked about in section 3.3. Only instead of the result being logged to the database, the route segment journey history analyser simply adds the completed route segment journey to the bus state. This history is built up in order to aid the more complicated bus state analyser components.

Bus Stop History Analyser

This component aims to document the passage of a bus through bus stops. It continually checks if a bus is within the range of a bus stop and if it already is then it checks to see if it is now outside the range of this bus stop. These bus stop entrances and bus stop exits are both recorded in the bus state. In addition to this, if the bus stop exit is the first bus stop on a route or the bus stop entrance is the last bus stop on a route these details are also logged into the state at the same time. In this initial version, a bus may only be inside a single bus stop at a time. The main point of this object building the history is also to aid the more complicated bus state analyser components.

Scheduled Bus Route Analyser

Some automatic vehicle location systems concerning a public transport system, transmit the active route number of the bus transmitting the location. In this AVL system no such

information is received, so dynamic assignment of scheduled bus routes must occur. One such approach, suggested by Predic et al.[3], is to base the assignment on the bus stops the vehicle passes through. The initial version of this scheduled bus route analyser would, on passing a bus stop, check to see if it was within a given time frame of a scheduled stop at that bus stop. If it was, then it required that at least one other bus stop on the same scheduled bus route, was passed by the same bus within the same give time frame of its schedule. If a bus had a scheduled bus route already assigned to it, then in order for the bus to stay on that scheduled bus route it must be within a road boundary of the bus route, further down the line from the last passed scheduled bus stop.

3.5 Notification Manager

The notification manager is the component that actually deals with generating estimates and sending notifications. At this stage the bus state has been updated to represent the state of the fleet of vehicles as the system sees it.

3.5.1 Creating Notifications From Blueprints

A notification blueprint is the result of a user requesting a timed alert for a bus stop on a scheduled bus route. This notification blueprint contains information such as the user id and advanced notification needed in minutes. A regularly timed event occurs in the program that asks the database to check all notification blueprints for entries that will occur within a given look ahead period. This creates notifications that get loaded into the model. These notifications are instances of the blueprints that are currently waiting for a bus to be the right time away from its stop before they will be sent.

3.5.2 Assuring Efficiency

In order to stop unnecessary calculations, only those scheduled bus route bus stops that are indicated in the queued notifications are assessed in the notification manager; all notifications are also processed in order of scheduled bus route and then scheduled bus route bus stop, this stops duplication of processing that is common to each, and stops the system becoming strained under a large number of notifications.

3.5.3 Generating Estimations

If a notification exists for scheduled bus route that is currently being served by at least one bus, the system will attempt to generate an intelligent estimate for that notification. For each bus serving the route, the system asks a 'bus route route segment interpolator' to return all the route segments and their percentages between bus and the bus stop. The average time through each route segment is then multiplied by the percentage of that route segment that is included between the two distinct points, and all the results are summed up. The bus that will reach the bus stop first is always the one that take precedence in the final estimate.

If there is no known bus serving a the route then the system must fall back to relying on

the timetable to generate the notifications. The notification sent informs the user of the fact that the estimate is not based off live data.

3.5.4 Route Segment Journey Averager

It is the route segment journey averager that returns the average time through a route segment. In the initial attempt this object simply makes a call to the database and returns the average of all journeys through the route segment with no special sub selection at all.

3.5.5 Sending Notifications

An agreement has been set up with a company named Zamano, to allow the TextMeMyBus Service to have free access to their services during the development of the project. As a service provider, Zamano has links with all the major Irish mobile networks and can send free and premium rate text messages through the APIs of the various networks. Using Zamano as an intermediary, the TextMeMyBus system can send all notifications to users via free text messages.

3.6 Defining Metrics

In order for this initial configuration to be measured against the future improvements, a method must be developed of defining how well the system has done its job. Cathey et al. recommend an approach of measuring how long before the estimation a person must leave in order to have a 90% chance of reaching the bus[4]. Although this is an interesting metric to consider, it falls short for bus services that run less than once an hour, and it doesn't fully encompass the entire range of behaviours seen within the notification system. Other issues such as the number of notifications that were able to be estimated rather than having to fall back on the schedule, the consistency of estimations as a bus approaches a specific bus stop and the amount of time a person will be left waiting for the bus to arrive if they follow the notification, are all concerns that must also be evaluated.

3.6.1 Percentages Of Notifications Estimated

This shows how good the system is at understanding which bus is on which bus route at any given time.Without the knowledge of a bus traversing a route, the system must rely on the timetable and base all its estimations off this static data. Some bus stops, such as the starting off points are never estimated as the bus has not started the route yet, and some of the scheduled bus routes have days where either there aren't any buses running them, or the buses that are aren't broadcasting their position. For this reason, the percentage of notifications estimated will never reach 100, but the higher the figure the better equipped the system is.

3.6.2 Average Inconsistency Of Estimations

As a bus approaches the bus stops on its route, many notifications may be generated about when the system expects the bus to arrive. This metric checks to see how consistent these estimations are; if they move erratically about, then this is an indication that the buses are not traveling in a manner system is expected them to. This metric is only calculated against sets of notifications for a scheduled bus route bus stop that have all been estimated and none of which fallen back on the time table.

3.6.3 Latency Of Notifications

If a notification is to be sent five minutes before a bus arrives, it is important that the time that the notification gets sent, is close to the five minutes before the system thinks the bus will arrive. This result measures on average how far apart the desired notification times and times they were actually sent were. A different average is produced for each desired notification amount. Large results here can mean that the system has a poor understanding of where the bus is in between the times it receives new location samples.

3.6.4 Accuracy Of Estimations

This set of metrics deal with how accurate the estimations themselves are. Issues dealt with are

- If a user were to follow either follow the notifications or purely the timetable
  - How many of the buses would they catch if they allowed for the bus being thirty seconds early?
  - How many buses would the same user miss?
  - What would that users average waiting time be?
  - How much extra time would a user need to leave to achieve a 90% chance of catching the bus?[4]

As there is no actual data about when a bus reaches its stops, a special notification of just one minute is inserted for each scheduled bus route bus stop and this estimate is taken as the time the bus arrived. Because this estimate waits until the bus is almost on top of the bus stop, its relies heavily on the actual recorded locations and only lightly on the estimation aspect, this level of accuracy has proved to be useful as a benchmark for of the project.

3.7 Results

In order for these tests to be run, every scheduled bus route bus stop in the system had 5 notifications set against it, with 20, 15, 10, 5 and 1 minute for their advanced notification. A simulation of one week was then run and the systems performance evaluated against its resulting output.

Table 3.1: Percentages Of Notifications Estimated(Config.1)

Total Notifications	7740
Percent Estimated	34.70%

Table 3.2: Inconsistency Of Estimations(Config.1)

Average Inconsistency Every 5 Minutes(measured in minutes)	1.27
Based On # Of Notification Sets	334

Table 3.3: Latency Of Notifications(Config.1)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	2.03	2.76	3.14	2.84	2.50

Table 3.4: Accuracy Of Notifications(Config.1)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	136	164	203	266	315
Buses Missed	247	234	247	261	300
Average Wait Time(mintues)	3.42	2.38	0.90	0.23	9.01
Missed 90% Catch Rate By	-6.75	-4.12	-2.53	-1.73	-17.77

Tables 3.1, 3.2, 3.3 and 3.4 show the results from the system running with the basic configuration. With these results, issues like the fact that only 34.7% of the notifications are estimated and that notifications get sent generally 2 minutes late, can now be targeted for improvement. It is also worth noting that because so few estimations were made, this sample set may not be large enough to create averages accurate enough to represent the entire system, until a larger sample set is produced there is not much point in comparing to the performance of the time table, although all results are provided.

As a side note, the reason that the sum of 'buses caught' and 'buses missed' for the timetable is greater than the sum of buses caught and missed for other notifications is because for the timetable, a bus is only registered as being caught or missed if there exists a 1 minute estimation for it, but for the other levels of estimation, both the 1 minute and X minute estimation must be present to decide whether the bus was caught or missed.

Chapter 4: **Improving Initial Attempt**

This chapter documents the domain and application level issues that degrade the systems performance. Attempts to solve these problems, by inserting and replacing components in the framework developed in chapter 4, are then discussed and the resulting performance analysed.

4.1 Issue: Bus Veering Too Far Off TimeTable & Weakness Of Scheduled Bus Route Analyser

From the results seen with config.1, it is clear that one of its greatest failings is that is does not make enough estimations and has to fall back on the time table too often. The reason for this is due to both a weakness in the component that dynamically assigns scheduled bus routes to buses as they are travelled, and the fact that at times, the time a bus arrives at a bus stop can be quite far from the scheduled time.

4.1.1 Remedy: Scheduled Bus Route Analyser Based On Start Time

From this information a new scheduled bus route analyser was developed to replace the original. This improved version places a much higher emphasis on the time a bus left a terminus and whether that bus is currently in the area of a road within an appropriate bus route, that leaves that terminus at that time. As this analyser does not require that the bus has passed at least two bus stops, it is also faster at assigning routes, becuase it can be done as soon as a bus leaves a terminus.

4.1.2 Results And Conclusion

As can be seen from tables 4.1, 4.2, 4.3 and 4.4 , the number of notifications that are now estimated has doubled, there have also been improvements in almost every other metric, with the exception of a higher percentage of buses being missed, this is most likely due to the fact that sample set has doubled and is now providing a more accurate description of the system. Although the time table is allowing for catching much more of the buses, its interesting to see that in order to catch 90% of them, a user would have to leave on average 7 minutes before the scheduled time, as opposed to almost half that for even the 20 minute estimation.

Table 4.1: Percentages Of Notifications Estimated(Config.2)

Total Notifications	7740
Percent Estimated	70.52%

Table 4.2: Inconsistency Of Estimations(Config.2)

Average Inconsistency Every 5 Minutes(measured in minutes)	1.04
Based On # Of Notification Sets	923

Table 4.3: Latency Of Notifications(Config.2)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.66	0.98	0.89	0.82	0.93

Table 4.4: Accuracy Of Notifications(Config.2)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	328	297	300	410	865
Buses Missed	619	714	734	721	298
Average Wait Time(mintues)	2.04	1.93	0.88	0.3	9.3
Missed 90% Catch Rate By	-4.1	-3.65	-2.8	-1.82	-7.07

4.2 Issue: Positions Too Old at time of Estimation & Gaps In Recorded Location Samples

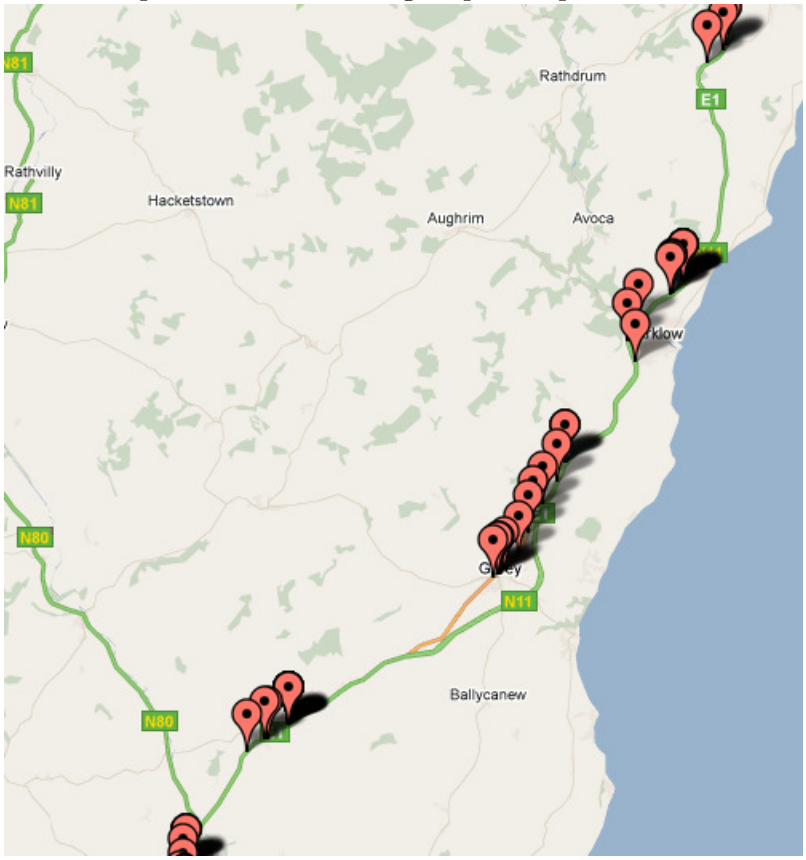
An important issue with the system as it stands in config.2 is that the system has no way of telling where the bus is in between new location samples being uploaded. As the gap in buses uploading new locations is about one a minute, that means that the system sees the bus jumping along the route instead of having many smaller discrete movements. Also it is not uncommon for a bus to stop uploading locations for up to five minutes at a time because of either a lack of internet coverage or perhaps going through a tunnel(the port tunnel is included in four of the six bus routes in this situation) , this has the side effect that as estimations are being generated, they can be based on locations that are quite old and hence no longer accurate. Figure 4.1 shows an example of a bus leaving gaps in uploading location samples

4.2.1 Remedy: Dead Reckoner

To address this, a dead reckoner was inserted into the solution that is run just before any estimation is made. It is the responsibility of the dead reckoner to interpolate a new position for a bus at any given time, based on

- The last known location samples off the bus.
- The time that location was recorded.
- The current time.
- The averages that have been recorded for route segments on the route.

Figure 4.1: Example Of A Bus Leaving Gaps In Uploaded Location Samples



4.2.2 Results And Conclusion

As can be seen from tables 4.5, 4.6, 4.7 and 4.8 the biggest effect of this configuration is that the latency of notifications has been significantly reduced. The dead reckoner allowed for estimations to continue to be based off new information in between new location samples from the buses. All metrics in relation to the timetable have stayed roughly the same (All notifications for timetabled values have no latency as there as there is no major scheduling issue).

Table 4.5: Percentages Of Notifications Estimated(Config.3)

Total Notifications	7740
Percent Estimated	71.83%

Table 4.6: Inconsistency Of Estimations(Config.3)

Average Inconsistency Every 5 Minutes(measured in minutes)	1.06
Based On # Of Notification Sets	943

Table 4.7: Latency Of Notifications(Config.3)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.32	0.32	0.35	0.36	0.42

Table 4.8: Accuracy Of Notifications(Config.3)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	313	279	306	425	870
Buses Missed	656	749	748	724	311
Average Wait Time(mintues)	2.23	1.97	0.89	0.34	9.03
Missed 90% Catch Rate By	-4.32	-3.67	-2.83	-1.9	-7.22

4.3 Issue: Inaccuracies Of Individual Route Segment Journeys

It is the responsibility of the route segment journey analyser to identify, and determine the details of, individual traversals of the route segments. In the original configuration this route segment journey analyser did not perform any interpolation. For calculating the time spent within a route segment, it only dealt with the definite times of:

- When a bus was first recorded inside a route segment.
- The next time the bus was recorded outside that route segment.

And for the percentage of each route segment travelled it only considered the roads that it had been seen to occupy while inside that route segment.

4.3.1 Remedy: Full Interpolating Route Segment Journey Analyser

To try and improve this approach three more route segment journey analysers were developed.

Time Interpolating Route Segment Journey Analyser

This class determines the probable time that a bus crossed the boundary into a route segment and also calculates the probable time that it crossed the boundary leaving the route segment. Therefor the duration spent within a route segment can be more accurate.

Road Interpolating Route Segment Journey Analyser

This class works by analysing the road the bus is in both before and after it is in the route segment in question. From these two extra data points, it interpolates the roads in the route segment that the bus is likely to have travelled as it entered and exited the route segment. Because this is likely to yield higher percentages of how much of the route segment was travelled, it allows the minimum likelihood a bus must have of traveling through a route segment to be significantly hired, thus filtering out segments that may not have been fully traversed.

Full Interpolating Route Segment Journey Analyser

This class is simply an amalgamation of the previous two, it provides the most accurate view the system has of individual traversals of route segments. It is this component that replaces its basic predecessor in this configuration run.

4.3.2 Results And Conclusion

The results from this configuration (Tables 4.9, 4.10, 4.11 and 4.12) show a significant improvement in the number of buses that would be caught by people following the various estimates made along the way. There is also a noticeable reduction in the amount of extra time needed to have a 90% chance of reaching the bus. Adding to the accuracy of the individual route segments has provided an important step in catching up to the ratio of buses caught to buses missed of the time table.

Table 4.9: Percentages Of Notifications Estimated(Config.4)

Total Notifications	7740
Percent Estimated	72.45%

Table 4.10: Inconsistency Of Estimations(Config.4)

Average Inconsistency Every 5 Minutes(measured in minutes)	0.94
Based On # Of Notification Sets	952

Table 4.11: Latency Of Notifications(Config.4)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.29	0.30	0.28	0.33	0.40

Table 4.12: Accuracy Of Notifications(Config.4)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	479	409	397	553	874
Buses Missed	502	625	662	601	311
Average Wait Time(mintues)	2.29	2.05	1.14	0.41	8.96
Missed 90% Catch Rate By	-3.13	-2.67	-2.2	-1.63	-7.35

4.4 Issue: Route Segment Duration Averagers Too Generic

The strategy taken by the route segment duration averager in the basic configuration, was to simply return the average duration of all journeys through a given route segment. This does not take other factor into consideration and ignores the patterns that occur due to several factors.

4.4.1 Remedy: Full Route Segment Duration Averager

This basic approach was extended by the development of three new classes to replace the basic version.

Weather Route Segment Duration Averager

One factor that can severely affect traffic patterns is the weather. As dry roads turn wet or slippery and as visibility levels drop, traffic can become slower and more congested. In order to take advantage of this fact, the weather conditions for the immediate area are recorded as location samples are logged to the database. Any new weather service conditions from an API are categorised by an administrator into one of four driving weather conditions:

- Good Driving Conditions: Dry Roads
- Poor Driving Conditions: Moderately Wet Roads
- Bad Driving Conditions: Wet Roads Or Reduced Vulnerability
- Treacherous Driving Conditions: Slippery Roads

There are many location based weather service APIs publicly available, with these you can send either a location name or lat/long position and have the weather from the nearest available weather station returned. In the TextMeMyBus implementation, a 'WeatherMan' object is created that will first attempt to retrieve the weather from google and if that fails, move on to the world wide weather online API.

Traffic Route Segment Duration Averager

Another factor that can have have a large impact on traffic patterns is peak traffic hours. Morning and evening rushes severely restrict traffic flow in areas that are susceptible to congestion, this necessity of factoring traffic into a solution is put forward by Jeong[5]. This component factors this into its averages by only returning the average of route segment journey durations that fall into the same on/off peak traffic time boundaries. For example, if an estimate is needed for a time between 08:15 and 09:45 or 16:30 and 18:00 on a non bank holiday weekday, then only journeys through the route segment in a similar peak level time are returned in the average. This gives the system a more acurate way of predicting the motion of a bus during a given time of day. These boundaries are configurable via the databases TrafficLevelBoundary table and bank holidays or any other similar days can be added in the NotableDate table.

Full Route Segment Duration Averager

This component amalgamates the functionality of both the other two route segment journey duration averagers, and provides the most accurate way of returning an average that is specific the current environment of a bus. It is this component that is used in config.5.

4.4.2 Results And Conclusion

Increasing the accuracy of choosing averages has not provided a similar increase as in improving the accuracy of the individual route segment journeys, a small gain can be seen in

the average wait times, but that is about the only gain. It is possible the effect may be more pronounced during times of unusually bad weather but that has not been ascertained in this set of results. Results are shown in tables 4.13, 4.14, 4.15 and 4.16.

Table 4.13: Percentages Of Notifications Estimated(Config.5)

Total Notifications	7740
Percent Estimated	72.42%

Table 4.14: Inconsistency Of Estimations(Config.5)

Average Inconsistency Every 5 Minutes(measured in minutes)	0.93
Based On # Of Notification Sets	951

Table 4.15: Latency Of Notifications(Config.5)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.29	0.31	0.31	0.35	0.44

Table 4.16: Accuracy Of Notifications(Config.5)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	484	390	400	550	869
Buses Missed	499	639	655	601	313
Average Wait Time(mintues)	2.25	1.85	0.97	0.38	8.99
Missed 90% Catch Rate By	-3.02	-2.57	-2.03	-1.55	-7.42

4.5 Issue: Fuzzy Location Samples & Detours

Although using GPS to track the location of a fleet of buses is very flexible, it also has some issues associated with it that must be addressed. One such problem is that the accuracy of the location is variable and depends on numerous factors such as the urban canyon phenomenon[6] or interference from the close transmission of digital or analog television signals[7]. If an inaccurate location sample is uploaded, especially during part of the route where the route segments are narrow and intricate, this has a chance of knocking the bus off the bus route. If a bus is running late, and it is knocked off its route in error, even for one cycle, this will cause notifications to be sent based on the timetable instead of the live data.

Another issue that is related to this by its remedy, is the fact that buses can take detours. Any deviation a bus takes from a bus route will cause it to be knocked off the bus route and the system will revert to the timetable. This is not a desired situation if the intent of the bus is to return to the original bus route.

4.5.1 Remedy: Percentage Based Course Supervisor

As both of these issues concern the error of a bus being knocked off its bus route, they can both be solved by a component which will ensure that the scheduled bus route a bus is

traveling, is maintained by allowing a margin of error in the incoming location samples. In order for these 'course supervisors' to not overstep their mark and keep buses on routes that they have correctly left, they will only make corrections if a bus has been traveling a route for at least 10 minutes and it will only provide corrections for 15 minutes before releasing the bus from the route. Two implementations were developed, their only real difference being where on the route the course supervisor will place the bus during a correction.

Dead Reckoning Course Supervisor

This approach simply uses the same extrapolator as used by the dead reckoner discussed in 4.2.1. If a bus moves off the route, the dead reckoning course supervisor will use its last location within the bus route and the amount of time it has been out of the route to extrapolate a new position and replace the one that fell outside. Although this approach works well for the fuzzy location samples, it is not as well suited to detours as it does not take into consideration how far away from the next stop a bus currently is, and hence if the detour is long enough, it can place the bus at the next bus stop despite the fact that the current location reported by the bus is nowhere near it.

Percentage Based Course Supervisor

This approach has proved to work better for both of the issues named above. Instead of dead reckoning the bus, it works out the percentage distance the bus is from the last passed bus stop on the route, in relation to the distance to the next bus stop on the route. It then places the bus that percentage distance between the two stops on the actual route. In this way, both small deviances from fuzzy location samples and large deviances caused by detours can be returned to the route in an appropriate position for estimating time remaining to subsequent bus stops. It is this component that is supervising the bus routes in config.6

Multiple Bus Stop History Analyser This approach necessitated the improvement of the component that compiles the bus stop history of a bus. In its original form, only one bus stop was allowed to be occupied at a time, this proved unsuitable as it did not account for bus stops that were close together such as the north and south bound UCD stops. This new version allows for this and was employed alongside the Percentage Based Course Supervisor.

4.5.2 Results And Conclusion

As expected the results from this configuration show an increase in the number of estimated notification by about 2.5%, this represents an increase of about 200 notifications that would have reverted to the timetable but are now being fully estimated (results shown in tables 4.17, 4.18, 4.19 and 4.20).

Table 4.17: Percentages Of Notifications Estimated(Config.6)

Total Notifications	7740
Percent Estimated	74.86%

Table 4.18: Inconsistency Of Estimations(Config.6)

Average Inconsistency Every 5 Minutes(measured in minutes)	0.96
Based On # Of Notification Sets	1035

Table 4.19: Latency Of Notifications(Config.6)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.30	0.27	0.31	0.34	0.44

Table 4.20: Accuracy Of Notifications(Config.6)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	507	408	429	588	919
Buses Missed	536	687	698	630	316
Average Wait Time(mintues)	2.28	1.91	1	0.4	9.05
Missed 90% Catch Rate By	3.18	-2.7	-2.13	-1.58	-7.13

4.6 Issue: Live Details Ignored

As a bus travels through a bus route, the route segments it has travelled are logged in its history. If the bus is moving through these segments at a faster or slower pace than the route segment averager says it should should, and this pace is kept, then estimations will be not be accurate because they are not being altered by this live knowledge. Chien et.al propose that in order to improve accuracy, the prediction error should be monitored in real time[8].

4.6.1 Remedy: Adaptive Averager

For this sake an adaptive averager was added to the estimation process. The purpose of this adaptive averager is to alter estimations based on the most recent live knowledge that is being collected, so that estimations will reflect the most up to date resources available to the system.

If a bus has completed at least two route segments in the past 20 minutes, then the ratio of how fast these were travelled in relation to how long they were expected to take, is applied to the average times of subsequent route segments involved in any new estimations. Because the speed at which a bus travels through route segments can be erratic at times, its effect on the estimates must be constrained. For this purpose, the adaptive average ratio generated is bound by +0.5 and -0.5, and for each subsequent route segment down the line from the bus, the effect of the adaptive average ratio is halved so that the further away from the environment that caused the speed up/slow down, the less of an effect it has.

4.6.2 Results And Conclusion

This configuration shows an increase in the number of buses that would have been caught, the effect is more pronounced the closer the notification is to the bus arriving, this is due to the reduction in the applicability of the adaptive ratio the further away the stop is. As expected, the average inconsistency of the estimations has jumped slightly, but constrain-

ing the adaptive ratio has meant that this is not outside acceptable levels. Results of this configuration be seen in tables 4.21, 4.22, 4.23 and 4.24.

Table 4.21: Percentages Of Notifications Estimated(Config.7)

Total Notifications	7740
Percent Estimated	74.88%

Table 4.22: Inconsistency Of Estimations(Config.7)

Average Inconsistency Every 5 Minutes(measured in minutes)	1.07
Based On # Of Notification Sets	1036

Table 4.23: Latency Of Notifications(Config.7)

AdvancedNotificationNeeded(minutes)	1	5	10	15	20
Minutes Late Sending Notification	0.29	0.28	0.32	0.38	0.47

Table 4.24: Accuracy Of Notifications(Config.7)

Estimation Used	20 Minute	15 Minute	10 Minute	5 Minute	TimeTable
Buses Caught	518	461	503	634	923
Buses Missed	526	635	625	585	313
Average Wait Time(mintues)	2.39	1.86	1.04	0.46	9
Missed 90% Catch Rate By	-3.15	-2.72	-2.07	-1.53	-7.27

4.7 Conclusion

In this chapter the performance of the system has been shown to improve with each issue addressed. Although the ratio of buses missed to buses caught did not catch up with that of the timetable, it far surpassed it in other areas such as the average waiting time and the amount of extra time needed to catch 90% of the buses. If users were advised of the extra time to allow, the system could be quite useful in an actual implementation.

Chapter 5: **Other Considerations During Development**

The TextMeMyBus system was developed with considerations other than just obtaining a result. Aspects such maintainability, scalability, ease of debugging and ease of extension were high on agenda and dictated a lot of the design choices. In this chapter, how those issues were approached are discussed.

5.1 Maintainability

If software is to stay in production for any length of time, individual segments of code within the program must be able to be modified, extended or refactored without causing unnecessary side effects or issues. One way to achieve this is to employ tried and tested approaches to problems that can be adapted and molded to the current set of needs. One of the things some these design patterns achieve is that they increase flexibility by encouraging the loose coupling objects through the use of interfaces and abstract classes[9]. This has allowed behaviour of the program to be changed by swapping objects with specific concerns rather than delving in and changing code that is completely interwoven with other aims. These patterns have not only ensured that the resulting components work in an expected manner, but will also clearly inform any subsequent developer of the original intent, due to the fact that these software patterns have almost becomes a common language between software engineers.

5.1.1 Software Design Patterns Used

Delegate

The delegate pattern describes a situation where an object uses a separate entity to carry out one of its tasks. The benefit here is that it provides a means of separation of concerns. The delegating object does not need to have knowledge of how the operation gets carried out, just that it has been carried out successfully, this allows each component to focus on its respective areas of expertise. Also as the delegating object is not tightly bound to the its delegate, it can easily swap the delegate out to fit whatever situation arises. In the TextMeMyBus system, a good example of the delegate pattern is in the use of bus motion analysers. Any component wishing to perform a deeper analysis of location samples, first delegates the responsibility of filtering through the location samples to a bus motion analyser, this frees the more abstract analyser to focus on higher level details.

Strategy

There are times when classes only vary in aspects of their behaviour. In such cases, it can be a good idea to define families of algorithms that provide these bahaviours, encapsulate each one and make them interchangeable. The strategy pattern encourages that behaviour should

be defined by composition rather than inheritance. This allows certain aspects of a class to defined at runtime. The TextMeMyBus code adapts the strategy pattern somewhat to alter the behaviour of the main scheduler. When the program is started, the administrators intent could be to have it run as a live service or be run as a test by quickly analysing all the location samples in a given time period(this is how all the configurations in chapter 4 were run against the exact same data). In order for this different intent to be realised, only a few aspects, such as getting the current time and deciding when to stop needed to be altered. The individual methods needing altering were so small that instead of each being encapsulated into an interface of its own, they were encapsulated as a group instead. When the scheduler starts up, it reads a value from a configuration file and loads the correct strategy for running as live or a test.

Factory

The factory pattern is probably the most used pattern within the project. Its intent is to abstract the creation of new objects and hide the class of the new object behind an interface. It states that classes to not need to know what implementation of an interface they have, just that they have an implementation. This decoupling ensures that when a new implementation is developed it will be able to be swapped into the live environment without any obstructions. Every component that analyses location samples in the project is created using a factory pattern, this, coupled with an external configuration file allowed the deciding of which classes to load occur at run time instead of design time, this level of flexibility was crucial to ensuring that the system was extendable.

Singleton

The singleton pattern is a simple but useful pattern. Using it ensures that only one instance of an object will ever be instantiated. In the project, the context class has a private constructor, a private instance of itself and a static method that will instantiate this object if needed and return it. As the context acts as a single reference point for getting objects that define the systems environment, it was important that there was only one instance of it so that components wouldn't be working with multiple instances while assuming they were all working from and altering the same one.

5.2 Scalability

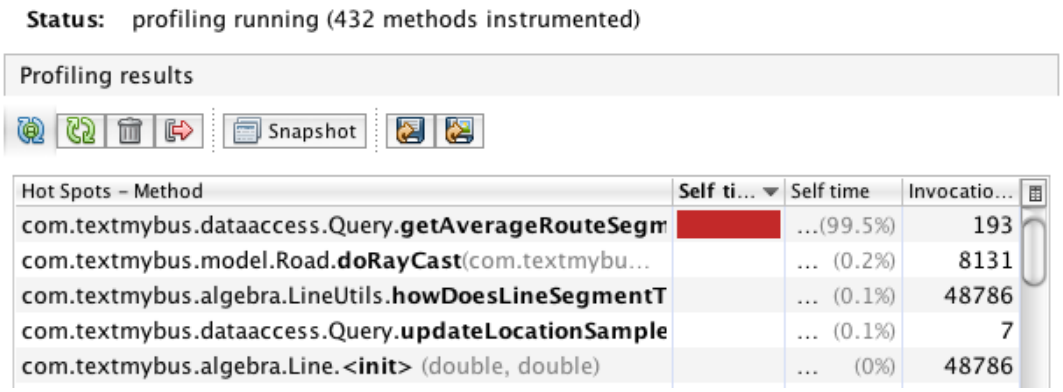
If the TestMeMyBus service was ever to open up to the public it would be a priority that that the system scaled well under the pressure of heavy use. In section 3.5.2, the organisation of code to ensure that duplication of work is kept to a minimum was discussed, but in order to more fully explore this avenue of thought, some profiling was carried out to highlight hot spots and bottlenecks in the code.

5.2.1 Profiling Java

The java code was profiled using a simple open source profiler named VisualVM. Although this profiler lacks some features of other proprietary profilers it is easy to use and good at quickly spotting hotspots. As the size and complexity of the code base grew, the speed at

which location samples were processed during tests slowed down considerably. At the time, this was assumed to be a side effect of new components and a growing structure, however as can be seen in figure 5.1, 99% of time was spent waiting on the database to return route segment journey averages from the database. As the route segment journey table grew larger, the constant requests for the same averages was slowing down the database server,which was in turn, bottlenecking the java process. To remedy this issue, all possibly needed database averages(not individual journeys) are requested and loaded into memory once every twenty four hours and a new set of route segment journey analysers were written to request averages from this cache rather than the database. Because they were built by factories and all employed the same interface, there was no issue inserting these new classes into the system.

Figure 5.1: Java Profiling Results



5.2.2 Profiling Database Stored Procedures

Another stored procedure identified by VisualVM as causing a bottleneck, was the stored procedure for getting new location samples from the database. MySQL has a basic profiling feature built in which can be switched on with "*set profiling=1;*". The initial form of the stored procedure, followed a process of selecting rows for returning, inserting their ids into a temporary table and then only returning the rows that haven't already had their status changed to 'picked up' by another process. This method is useful for stopping database locks occurring through concurrent access of rows. However the breakdown of its execution in figure 5.2 can be seen as spending over 20 times the duration just inserting into the temporary table, than just selecting and returning the rows(figure 5.3). Because of this poor performance the stored procedure was altered to just return a plain select(figure 5.3).

Figure 5.2: MySQL Profiling Results Showing BottleNeck

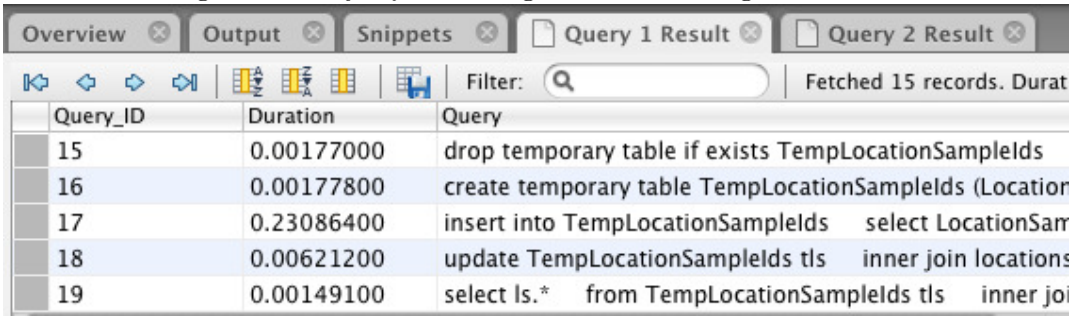
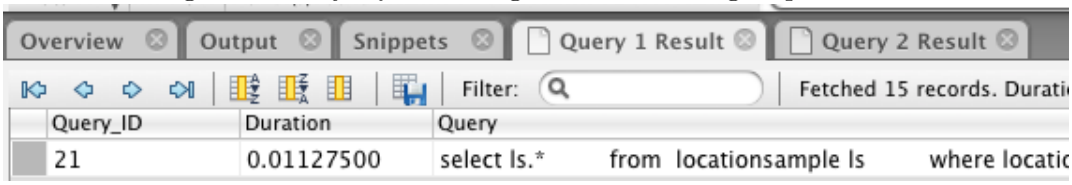


Figure 5.3: MySQL Profiling Results Showing Improvement



5.3 Ease Of Debugging

In order to be able to see how each component is acting as the system is running, each outputs live information, this causes the issue of an individual components output getting lost in a stream of unfiltered information, which makes spotting problems in a specific component extremely difficult.

5.3.1 Log4J

Log4J offers a simple solution to this problem. A set of loggers are defined in a configuration file that gets loaded and monitored by log4j at runtime. In this configuration file, a logger is defined for each family of components, which can then be loaded by the objects themselves. Each logger can output to the standard out and a specific file in the folder structure, but where it actually outputs can be altered in the configuration file without having to restart any processes. This allows for all logging to go to the relevant files, while allowing one or two components to also log to the standard out so live inspections can be carried out with ease.

5.4 Ease Of Extension

As the initial development of the TextMeMyBus system focused on building a basic structure that could then be easily extended and modified to increase accuracy and performance, the ability to extend the programs usefulness was the biggest factor that influenced the basic architecture of the system. The use of patterns to decouple objects has also greatly aided in this.

5.4.1 Replacing And Adding Components

If a component within the existing system is to be replaced with a newer more accurate version, all it must do is implement the interface of that type of analyser, the factory for that component type can be quickly altered to allow returning of the new component. As long as all necessary interfaces are implemented the rest of the system will not even realise that a new class is being used. New classes that do some new form of analysing need only be placed in a position appropriate to its desired scheduling i.e on the arrival of a new location sample, just before estimations are made, or during the estimation process.

5.5 Conclusion

From the outset, the future development of the system was in constant consideration during its growth. Because of this, it has retained a level of flexibility and efficiency that could be hard to regain without breaking some aspects of the program during the refactoring process.

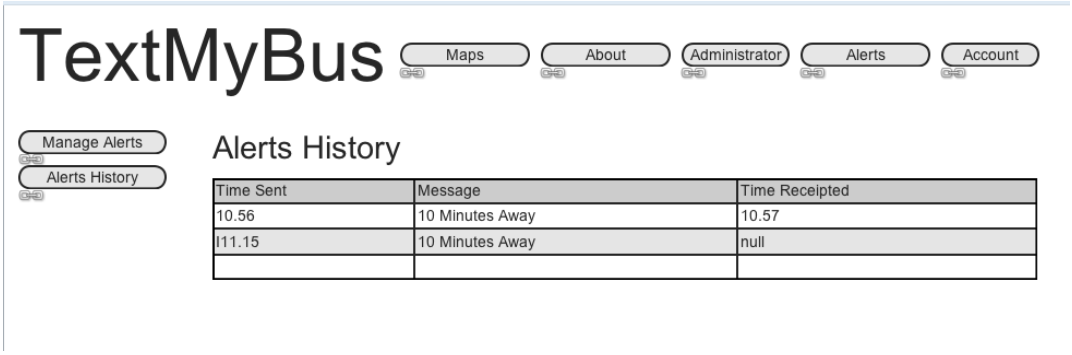
Chapter 6: **Frontend**

As this system is designed to be used by people who could have very little technical knowledge, the implementation of an easy to use, user facing interface was a necessary step in its completion. This frontend consists of flex based web application and a webservice running on a tomcat server that supports all of the interactions with the database. This chapter provides a description of its development process along with justifications for decisions made.

6.1 Mock Up

The first step was in defining the basic structure and navigation of the front end. This was achieved using a website mockup tool. From here, the components for effective interaction with the system were identified and design issues such as the means of navigation were decided upon. Figure 6.1 shows an example page from this mockup.

Figure 6.1: Showing The Mock Up Of The Alerts History Page



6.2 Flex

This design was then implemented using Flex. Flex allows for the creation of flash based 'Rich Internet Applications'. These applications are designed and programmed using a variable mix of actionscript3.0 and MXML. MXML is an XML extension developed by adobe, to allow for the quick construction of user facing components, as well as defining and creating some of the more development oriented constructs such as variable types and web services. The considerations that went into picking flex for the front end are documented below.

6.2.1 Benefits of Flex

Application Rather Than Web Page

The biggest draw Flex has, is that it truly allows development from the perspective of a developer instead of a designer, the official development environment is even a custom version of the eclipse IDE. Because of this developer oriented perspective, the TextMeMyBus website is able to use an MVC architecture to completely separate the the individual parts of the application.

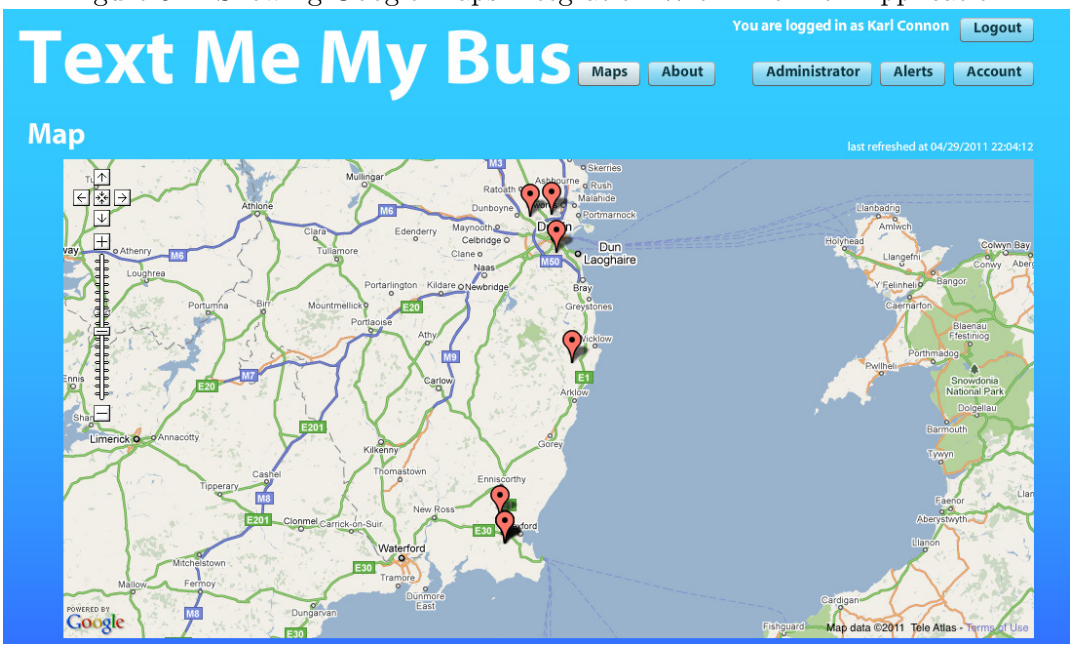
Bubbling

Flex employs an event bubbling technique that allows custom events to be fired from views. These events 'bubble' up through the parent objects of the firing component, until they are caught and processed by the intended recipient. It is this aspect of flex that allows for the TextMeMyBus views, to send the interaction and input they receive from a user to the controller that the entire view structure hangs from. This ensured that the view was completely separate from the main business logic and could be swapped out without having to reimplement any of the basic functionality.

Official Google Maps Libraries

As the TextMeMyBus application is essentially a location based service, the inclusion of a map showing the locations of each bus was a necessity. Google provides an official library for the integration of google maps with a flex application, this was a prerequisite for using flex. Figure 6.2 shows the final integration of Google Maps with the flex application.

Figure 6.2: Showing Google Maps Integration With The Flex Application



Good User Experience

Flex allows for the inclusion of subtle transitions in between different states of the application. Visual effects such as fading, sliding or scaling can be applied to visual components with relative ease in comparison to some other web technologies. These transitions give a smoother experience to users and help to not break a users flow.

Possible Local Distribution

A web application built using flex has the added advantage that it can be compiled to run as a regular desktop application instead of through a web browser. This even allows for the inclusion of automatic updates that will keep the application up to date with the latest version.

6.2.2 Drawbacks of Flex

No Database Access, Needs A Webservice

One of the biggest drawbacks of flex is that for security reasons there is no native support for database access. If a flex application needs access to a relational database, a layer must exist in between the two to provide the information required. This is somewhat balanced out by the excellent web service support provided by flex, but it still requires an extra layer to be built if it does not already exist.

Compatibility

Adobes recent dispute with Apple has meant that there is no support for flash based applications on the iPhone or iPad.

6.3 Webservice

A webservice was built with the sole purpose of providing database access to the flex application. A class was written, encapsulating each call to a stored procedure that the the application would need, into its own method. These methods simply pass the stored procedure call to a single method, which in turn uses a number of other classes to obtain and wrap the results in XML, this result is then passed back up to the exposed method. This approach allowed for the easy addition of new stored procedure calls, as all the code that obtains and wraps the information is generic enough to be reused for every call.

6.3.1 Using Tomcats DBCP Resources

As the web service was to be deployed using tomcat, this allowed the use of tomcats built in database connection pooling resource to free the Java code from having to manage the database connections manually. Once the connection is defined and named in tomcats context files, a request for this resource can be made inside the java by accessing the initial

context. When the process no longer needs the connection it can be released back to the pool that is managed by tomcat itself. Another benefit of using tomcats built in database connection pooling, is that parameters such as maximum active connections can be defined in configuration files to allow for the easy balancing of performance and resources as the project scales.

6.4 General Design

In Donald Normans 'The Design Of Everyday Things'[10], he documents the properties of good design, these are not all specific to user interfaces but some are very applicable and were adhered to during the development of the front end. Three relevant examples are

- Objects should give hints to their use in the way they look, so anything that is clickable should look like a button.
- There should be constant feedback to the user about the state of the system. If a user clicks on a button and the system is waiting on a reply from a webservice, this should be communicated to the user and they should not have to wonder whether the system registered their button press. In the TextMeMyBus system, a spinning cursor indicates that the application is currently processing a request.
- The current position within the navigation structure of the system should be immediately obvious to a user. In the TextMeMyBus frontend, the currently selected menu item is always highlighted so that it is obvious which menu the user is browsing.

6.5 Conclusion

The TextMeMyBus web application provides all the administrative tools a user needs for managing their interaction with the system, it adheres to good usability principles and the code behind it has been designed from the stand point of an application rather than a collection of webpages.

Chapter 7: **Conclusions**

During the development process of this project, many separate components have been created and described, some proving more successful than others, and each with their own concerns and issues.

- The high level abstractions such as the representation of the bus routes in terms of increasingly smaller composite components, worked well at capturing a framework for analysing and recording the movements of buses.
- The strategy of developing a system that was easily extendable, was successful at allowing the behaviour of the system to be altered with ease through the addition of various new components, that refine estimations.
- Most instances of resolving domain level issues, such as gaps in a bus uploading new location samples, did lead to gains of the systems performance as measured by the metrics.

Even though the results of the simulations showed better performance than the timetable in relation to the sum of the average waiting time and amount of extra time needed to catch 90% of the buses, it never caught up to the timetables ratio of buses caught to buses missed, and it is this that is most important when deploying a live notification service that people are going to rely on.

For this reason future work is needed to refine the process further and continue to increase the accuracy of the estimations. Some ideas would be

- To restrict the averaging of route segment durations, by only referencing more recent durations recorded, or possibly to assign weights so that the more recent a route segment journey is, the more of an effect on the 'average' it has.
- The Google Maps API could be exploited to request how long it thinks a bus has until it reaches a bus stop instead of using the collected averages.
- The idea of informing users of prediction intervals could be explored[5]. If a user is informed that the prediction is accurate to +-X minutes, they will be able to make informed decisions about any extra time needed.

Bibliography

[1] Brendan Kidwell, *Predicting Transit Vehicle Arrival Times*, GeoGrahpics Laboratory, 2001

[2] D.J. Dailey, Z.R. Wall, S.D. Maclean, F.W. Cathey, *An algorithm and implementation to predict the arrival of transit vehicles*, Intelligent Transportation Systems, 161-166 , 2002.

[3] B Predic, D Stojanovic, S Djordjevic-Kajan, A Milosavljevic,D Rancic, *Prediction of Bus Motion and Continuous Query Processing for Traveler Information Services*, Lecture Notes in Computer Science, Volume 4690/20007, 234-249, 2007.

[4] F. W. Cathey and D. J. Dailey, *A prescription for transit arrival/departure prediction using automatic vehicle location data*, Transportation Research, Volume 11, Issue 3-4, 241-264 , 2007.

[5] R.H. Jeong, *The prediction of bus arrival time using automatic vehicle location systems data*, PhD Thesis, 2004.

[6] Youjing Cui, Shuzhi Sam Ge, *Autonomous vehicle positioning with GPS in urban canyon environments*, IEEE Transactions on Robotics and Automation, Volume 19, Issue 1, 15-25 , 2003.

[7] Beatrice Motella, Marco Pini, Fabio Dovis, *Investigation on the effect of strong out-of-band signals on global navigation satellite systems receivers*, GPS Solutions, Volume 12, Issue 2, 77-86 , 2008.

[8] Steven I-Jy Chien, Yuqing Ding, Chienhung Wei, *Dynamic bus arrival time prediction with artificial neural networks*, Journal of Transportation Engineering, Volume 128 429-438 , 2002.

[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994.

[10] Donald Norman *The Design of Everyday Things*, 1990.