Software Requirements Specification of WhenBus

Arjun, Shashikanth, Vinay, Harsha and Hemanth Robbi

Dept. of Computer Science and Engineering
IIT Madras
email : WhenBus.software@gmail.com

Abstract

The idea of the product is to provide accurate real-time info regarding MTC bus timing through crowdsourced data from the users. As more and more users rely on MTC for their daily commute, it is important that we can provide accurate timing of bus arrivals, rather than customary timings from the MTC time table. This paper describes the Software Requirements Specification of WhenBus, an application that uses crowd-sourcing to provide a good estimate of MTC bus times.

Introduction

WhenBus is a crowd-sourced application that provides accurate bus timings. The key motivation for this application is that, even though there are a lot of applications that provide the standard bus times available on government sites, it is highly unlikely that the buses do follow the same due to various factors like traffic, weather etc., . As the number of people who are taking the public transport is increasing and with mobile technology available, it is possible via crowd-sourcing to get accurate bus schedules based on user feedback.

The application relies on the commuters, who allow anonymous location data or provide some qualitative feedback which will be sent from their smartphone whenever they are using public transport. By collating this data together, WhenBus offers more realistic information about bus routes. While a bus may be due in ten minutes, a rider currently on that bus might be located more than ten minutes away, indicating that the bus is not on time. The app can then suggest a quicker route for users, based on source and destination stops quoted by the user.

Crowdsourcing allows businesses to use the input of multiple sources, both within the corporation and externally, to develop solutions for strategic issues or to find better ways to complete tasks. This new culture of innovation, supported by crowdfunding for worthwhile projects, allows for idea collaboration and technological innovation for the greater good. Further, our increasingly mobile world population allows for people from anywhere, and with any background, to give their input on a project.

Thus, WhenBus being propelled by Crowdsourcing, if provided sufficient input from the commuters should provide accurate bus schedules.

General Description

WhenBus is intended to provide an accurate bus information to the user. From an organizational point of view there are three main components : the Commuter, Feedback Providing User (FPU) and the Administrator.

Administrator

The administrator will enter the details of bus routes, bus information, bus stops, its coordinates etc into the database. There is a provision for changing and resetting the timings of the buses. Also, he can add a new bus stop, bus service etc., to the database.

Commuter

The commuter can enter the bus number or he can simply choose the source and the destination. The app will suggest the nearest bus stop for the user based on the GPS location. Once the user enters the necessary information the system will recommend the best bus (first arrival) based on the live tracking information available with the system. The application will store the information about bus stops [a map between names and coordinates], this helps in reducing many data calls, and the user can always be directed to the nearest bus-stop, without any data-connection.

Feedback Providing User (FPU)

The commuter who provides Feedback, is called a Feedback Providing User (FPU). The user can provide two types of feedback, Qualitative (Discrete feedback (yes/no type)) and Quantitative (Continuous GPS location, Velocity etc.,). The WhenBus System will fire a prompt when it thinks the user is about to board a bus. Once the user answers to the prompt, the feedback along with the relevant bus number is sent to the system for further processing.

The system acquires feedback from multiple users and performs calculations considering traffic, distance etc., that helps in updating the information in the database.

Thanks to the Feedback Providing User, we can achieve real-time bus tracking, thereby helping other commuters as well.

The following image represents the high level architecture and the interface between WhenBus server and the application of the commuter.

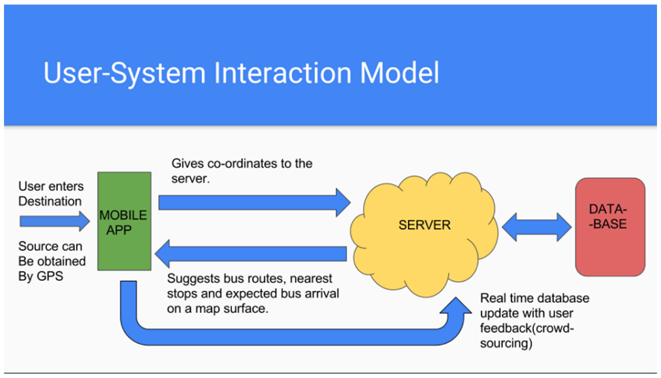
User-System Interaction Model
User enters Destination
Source can Be obtained By GPS
MOBILE APP
Gives co-ordinates to the server.
Suggests bus routes, nearest stops and expected bus arrival on a map surface.
SERVER
DATA--BASE
Real time database update with user feedback(crowd-sourcing)

Figure 1: User-Server interface

Requirements Specification

The requirements of the WhenBus application are presented in detail below.

Functional Requirements

The main functional requirements of the WhenBus are the following:

1. The WhenBus system should provide a means for storing the information about
   - Bus Timings
   - Bus Routes
   - Bus Stop information
   - Avg travel time
   - Bus information
2. The system should be able to detect the nearest bus stop based on the location of the user.
3. WhenBus should be capable of enlisting the possible bus numbers based on the source and destination input provided by the user.
4. The system should recommend the best possible bus and average time it will take to reach the destination based on the live tracking information.
5. WhenBus should possess the feature of allowing the user to give the bus number as the input and get the live tracking information about the bus.
6. The user interface displays a map to show the present location of the bus queried by the user.
7. The application should be synced with the server, to get the knowledge of various bus stops available, to ensure off-line features.
8. The system should be able to infer when the user boards the bus and ask for feedback.
9. The system collects the feedback after the user has boarded the bus.
10. WhenBus should notify the user when the bus is about to reach the destination.
11. The system should take the GPS location of the user and send it to the server to update the location of the bus time to time.
12. WhenBus server should store the feedback given by the multiple users and process it to get the accurate information about the bus location.
13. The system should provide the means to reset the timings of the bus, addition of bus stops, addition of bus routes etc. It should be scalable.

Non-functional Requirements

The key Non-Functional requirements for the WhenBus application would be as follows :

1. The client side application would be for Android 4.4 and above, The client's mobile handset should have GPS and a network connection.
2. Client side application will be written in the Java programming Language.
3. The interface will use Google's Map API.
4. The server would be written in NodeJS, using the express framework.
5. MongoDB would be used for storing all the information as stated in Functional Requirements.
6. The wrapper mongoose over MongoDB will be used, for the server to access the Database and define Schemas for collections.
7. Essentially, We would be developing a MEAN Stack.

Use Case Modeling

As part of the modeling process, the functionality of WhenBus has been defined using use cases and scenarios. At a high level of abstraction the entire functionality of the crowd-sourcing application WhenBus is captured in the use case diagram shown in Figure 2, a couple of major parts of WhenBus functionality are presented in the detailed use case section. We have relied extensively on use cases to define the expected functionality of WhenBus and better describe its intended behaviour.

Use Case Diagram

A use case diagram is a visual representation of the relationships between customers and use cases together that documents the systems intended behavior.

The use case diagram is important for visualizing a system. However, a textual description of the sequence of transactions of a use case is also needed for us to understand what really happens in a use case (see detailed use case).

The key purposes of use case diagrams can be as follows:

- Gather requirements of a system.
- Used to get an overview of a system.
- Identify factors influencing the system.
- Show interactions among actors and the system.

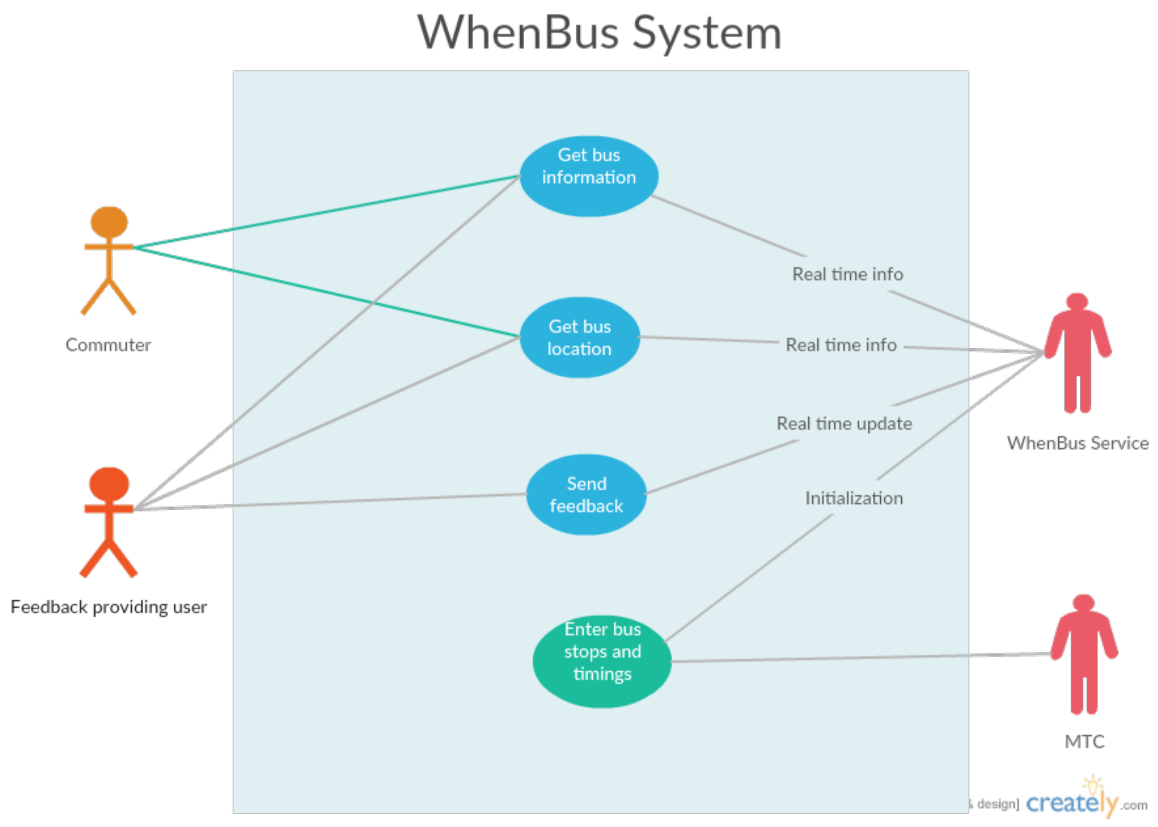

Figure 2: User Case Diagram of WhenBus

Detailed Use Case

This section shows the detailed flow of events in a particular use case.

Use Case: Enter Bus Information
ID: WB1
Actor: MTC
Precondition: 1. Bus Scheduling information need to be entered or modified
Flow of Events: 1. The use case starts on regular intervals 2. The system authenticates the user, as it has the ability to modify the database. 3. The system prompts the MTC administrator (or Automated scraping) to enter the bus timings, routes, stop names etc., 4. The system validates the entry. 5. The system saves the information in the database.
Secondary scenarios: 1. The newly entered information clashes with the existing information. 2. Required information is missing. 3. Aborts of the transaction.
Postcondition: 1. Entered bus information is saved in the database, with the required schema.

Use Case: Get Bus Information
ID: WB2
Actor: Commuter, FPU
Precondition: 1. User enters the bus number to get the details.
Flow of Events: 1. The use case starts when the user enters the bus number. 2. The system validates the query. 3. The system fetches appropriate data of the bus number. 4. System displays the results obtained on the user interface.
Secondary scenarios: 1. Required information is missing. 2. Requested data not available in the system. 3. Transaction fails to complete.
Postcondition: 1. The requested bus information is displayed to the user.

Use Case: Send Feedback
ID: WB3
Actor: FPU
Precondition: 1. The user has GPS and also possess a data connection on the mobile device
Flow of Events: 1. The user is prompted to allow a Quantitative Feedback, if he does not we resort to displaying Qualitative Feedback Questions. [more about in Feedback Protocol] 2. The feedback got from the user will be used to update the information available
Secondary scenarios: 1. Conflicts in sent Feedback. 2. Feedback based information update fails. 3. Data connection is lost.
Postcondition: 1. This update information could now be accessed by any other commuter to get real-time information.

Feedback Protocol

This application will have two types of feedback that the user can give back to the server, for real-time updates.

1. Qualitative Feedback
2. Quantitative Feedback

Qualitative Feedback

Questions like *did the bus 5E arrive?* with categorical feedback, can be used to update the database, based on user feedback. Although, Quantitative Feedback can provide significant information, we would also at times want to use Qualitative Feedbacks.

Quantitative Feedback

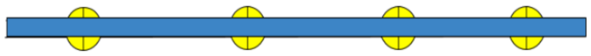

Figure 3: Continuous Feedback obtained from application

Let the blue lane represent the road. At each yellow marker, we poll the moving passengers GPS co-ordinate, calculated velocity etc., and tag the acquired position along with the position of the bus the user is in. This helps our service to update the bus timing considering any possible delays that could have occurred. However, there are a few intricacies involved such as when to stop following the user, say he got down from the bus before the destination etc., In such cases we keep a threshold distance from route, beyond which we drop the user.

The solution we would like to implement would be to set a threshold distance from the route beyond which the user feedback will be cutoff. In the Figure 4, the GPS positions in blue are considered OK and the ones in yellow are dubious and the application stops sending the feedback to the server.

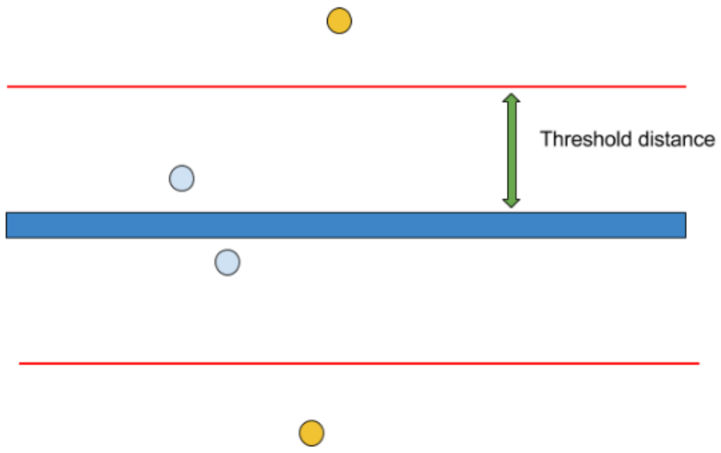

Figure 4: An image indicating a Threshold beyond which the user is unfollowed