Abstract

The purpose of this project was to create an flexible ticket system based on using PDA.

The train/bus is able to recognize whether the customer is on the board. After the customer left the train/bus, the system is able to recognize that and charge customer from his balance that is stored on account.

The customer can check current price of journey. Using PDA, customer can also search for journey and show the fastest route to get there.

Our system is using the ring zone system based on which we can calculate the final price of the journey.

The customer has to create the account through the website where he can see the history of journeys as well as he can search for new journey. The website is also the interface for adding the money to his balance

Contents

# Introduction

This is one of the most important components of the report. It should begin with a clear statement of what the project is about so that the nature and scope of the project can be understood by a lay reader. It should summarise everything you set out to achieve, provide a clear summary of the project's background, relevance and main contributions. The introduction should set the context for the project and should provide the reader with a summary of the key things to look out for in the remainder of the report. When detailing the contributions it is helpful to provide pointers to the section(s) of the report that provide the relevant technical details. The introduction itself should be largely non-technical. It is useful to state the main objectives of the project as part of the introduction. However, avoid the temptation to list low-level objectives one after another in the introduction and then later, in the evaluation section (see below), say reference to like "All the objectives of the project have been met...".

# Identifying Complications with the Implementation

In this chapter we try to identify some of the overall problems to be solved in order for the Requirements Specification to be upheld. The problems are explained at a conceptual level and any further detailing of the individual problems will follow in later chapters.

The problems we’ll encounter in development are as follows:

|  |  |  |
| --- | --- | --- |
| Problem | Description | Reference |
| Adding Clients and removing clients to the train/bus computer | A way for the clients to logon to the transport system of their choice using their PDA, and later log out using their PDA. |  |
| Handling multiple clients logging on/out at once | Concurrently connecting clients to the train/bus server must be handled efficiently with little communication footprint from and to the main server. |  |
| Charging multiple clients when logged off | Multiple concurrent transactions with the main server to charge and update user entries from different train/bus servers and the website must be handled without risk of data loss/corruption by concurrency. |  |
| Finding the fastest route from point A to point B | A solution to finding the fastest path between two stations/stops using the available public transportation lines or walking. |  |
| Creating a website for user management and route planning |  |  |

# Problems in Detail

The following chapter seeks to explore the identified complications in detail by raising more technical issues with each. Then suggest and criticize the possible solutions, and finally conclude on the chosen implementation.

## Adding and Removing Clients to Bus/Train Server

The problem in detail

### Getting Server address

Premise:

The user has to be able to connect with a specific transport vehicle.

Issues:

The address for each train/bus has to be unique on the network.

The network has to be readily available and not require extra cost on the users’ part.

Solutions:

The first issue of uniqueness on the network leaves three options:

* Either using IPv6 on the Internet which includes the MAC address of the individual bus or train computer to ensure global uniqueness.
* Or use a local area network on which a range of IPv4 or IPv6 addresses can be used to connect with a range of vehicles all within range of the user.
* Or to use a local area network on which a single specific IPv4 or IPv6 can be used to connect with any vehicle. Each vehicle has a “code” that the client uses to single out a specific vehicle.

The second issue of availability and cost both speak in favor of implementing a local area network for communications.

In 2014 Wi-Fi will continue to become the primary network for smartphones[[1]](#footnote-1) and even though the implementation of free Wi-Fi Internet is on the rise, there is still no guarantee for Internet connection or data provided.

On the contrary a LAN Wi-Fi connection to the train or busses would not require data cost for the user or a wireless data connection from the users service provider.

Taking the path of the local network solution, there is still a choice between a range of IP addresses each unique to a specific bus or train, or a single address with a code unique to a specific bus or train.

Even though a code offers a larger range or possibilities, the number of possible IP addresses is beyond the number of possible vehicles. Therefore both solutions offer equal merit.

However, for testing purposes during development, the use of a code instead of a range of IP addresses is more feasible. Since we can test many random codes on a single computer, yet not use many random IP addresses and expect a connection.

Conclusion:

The bus and train server will all use the same specific IPv4 address and port number. But each implements a specific code to connect to only one unique bus/train.

### Connecting to the server

Premise:

The user has to know when to connect, and perform the connection.

Issues:

The bus or train computer has to choose when to accept new connections to not get false connections between stations.

The user subsequently has to know when the bus or train is listening.

Solutions:

The first issue of choosing when to accept connections can be solved in three different ways:

* Closing the server socket while between stations would stop any incoming connections.
* Changing the internal server code between stops, making possible to connect but impossible to register a connection.
* Have an internal server state that, when set to “left station” denies user registration.

The first solution of closing the server socket has the issue of throwing exceptions both in the server when the blocking await function is forced to terminate, and in each client when they attempt connection.

The second solution of chancing the connection code between stops would effectively deny user connections. As long as the between stops code is random, it won’t be a security issue either.

Using states to solve the issue is equally effective in denying users and also without security issues. Furthermore, it has a less difficult implementation than setting a random code (since the code mustn’t be unchanged, nor can it be the same as any other train or bus nearby) and offers more flexibility as using states allows for alternate connection handling between stations rather than no connections.

The second issue of knowing when to connect can be solved by either constantly attempt connection for a brief period or for the train or bus to message the user than it is listening.

The approach of bombarding the server with TCP requests is not favorable and should be avoided. The idea of having the train/bus message to all clients that they can connect is preferable. The way to do this is to multicast a UDP datagram to all listening clients and have them connect back with a TCP request.

The combination of using UDP to connect the clients along with using the flexible states implementation allows for fully automatic registration and deregistration with the train/bus server.

This is possible because the server knows whether a user is near the train/bus by sending out a UDP signal. This way it also knows whether a user has traveled with the vehicle by requesting the user to connect at two subsequent stops, if a user connects at both, then the user is actively on board.

If a user that has been active is suddenly not responding, the server can assume the client has left the train/bus and has to be charged.

The server code can furthermore be sent out with the UDP datagram, which removes the need for the client to know anything about the server TCP IP or code. Only the UDP port and IP is needed.

Conclusion:

The server implements a UDP transmitter to contact all potential clients near stops which sends them the needed code, and TCP IP address. The client can then connect automatically to the requesting server and go back to listening.

A user is active if it connects to the same server at two subsequent stops.

A user is charged if it used to be active and stopped replying to UDP requests.

Registrations cannot happen unwanted due to a randomized code and internal server state (whether left or arrived at station).

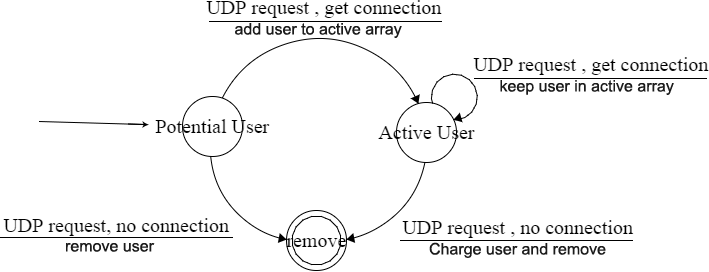


Figure 1

Figure 1 shows a finite state diagram of the suggested solution for automated user registration and deregistration.

## Charging multiple clients when logged off

*Premise*

The users have to be charged after they leave the train an log off.

*Issue*

The bus or train has to know when the whether the clients are still at the train/bus or not.

The system has to charge the right client.

Charging and updating has to be handled without data loss or corruption by concurrency.

Make communication between Train/Bus server and Main Server

*Solution*

The first solution was to use RMI to send a message from Train Server and execute the charging of user at Main Server, which is connected to Database.

The second solution is to use Map, using User key to identify the user and Value key to withdraw the charge from balance.

Using map is easier to implement

*Conclusion*

We decide to use Map

1. <http://www.wired.com/2014/01/collision-course-wi-fi-first-role-changing-mobile-communications/> and <http://www.pcmag.com/article2/0,2817,2425853,00.asp> [↑](#footnote-ref-1)