The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco, CA, U.S.A., March 1-2, 2013
Doi: 10.7321/jscse.v3.n3.133 e-ISSN: 2251-7545

Software Engineering for Smartphone: a Car Sharing System Case Study

Yoram Haddad
Dept. of Computer Science
Jerusalem College of Technology
Jerusalem, Israel
haddad@jct.ac.il

Yuval Cohen
Dept. of Computer Science
Jerusalem College of Technology
Jerusalem, Israel
yuvalc@jct.ac.il

Ronen Goldsmith
Dept. of Computer Science
Jerusalem College of Technology
Jerusalem, Israel
goldsmit@jct.ac.il

Abstract **- Widespread use of smartphones has, in parallel, opened a wide range of opportunities for new applications. To be sure, the ever-increasing use of digital information processing and communications devices has created a commensurate increase in electricity consumption. However, one can also develop applications which encourage and spread ecologically friendly behavior. This article presents the design and implementation of a car ride-sharing application for a mobile environment. The application enables users to share automobile transportation in an efficient and simple way. Use of this system can significantly reduce the number of private automobiles on the roads, thus yielding substantial ecological, economical, and social benefits. Since the application is designed for smartphones, the *sharing* facility may be implemented in realtime, from anywhere, anytime. The application is based on an algorithm for finding subroutes in a user-defined path, according to the number of matched points along the path. This application differs from existing car sharing applications in several, crucial ways. The article will describe both the system and the specific differences from other, existing software.**

Keywords: Mobile communication systems, Real-time and embedded systems, Software engineering for Internet projects, Location dependent and sensitive

I. INTRODUCTION

Population growth and increasing population density, particularly in metropolitan areas, have brought about an increase in the number of vehicles on the roads, by a few percentage points per year [1, 2] (3.6% increase in 2010 alone [3]). The cumulative effect of this phenomenon is staggering. The main derivatives of this situation include (in addition to direct economic expenditures on car maintenance, insurance and fuel):

- Traffic congestion – in USA drivers spend an aggregated total of up to one month each year in traffic jams [4]. This is the equivalent of the loss of a meaningful number of working days per year per person, with obvious consequent significant economic damage [5]. Congestion also brings about a huge waste of fuel, increased emission of carbon dioxide and pollutants and severe environmental damage. Traffic jams have other impacts on social or driving behavior, health (stress, anxiety, blood pressure and psychological effects [6].

- Parking – is another obvious problem in large, crowded cities. Various solutions have been implemented or proposed - e.g. "fast lane" , which offers free passage to vehicles with 4 or more passengers. In London, a heavy daily fee is exacted from commuter cars entering the city center.

- Public transportation – in some places the needs of the population exceeds the availability of public transportation, especially in some developing countries. The lack or inefficiency of public transportation is itself a stimulus to automobile purchases.

- Environmental concerns – Congestion has heightened the awareness of the importance of environmental protection and there is a worldwide search for new, energy efficient ways to manage our daily mobility. Unfortunately, none of these efforts has made a significant contribution to the situation.

The remaining structure of this article is as follows. In the next section we present some of the existing car sharing applications and stress what has not already been done in comparison to our work. Then in section 3 we present our system and in section 4 we present the design and algorithm for our application. Specific details on the implementation are provided in section 5. Finally a whole section on the future works and improvements is presented in section 6 and section 7 concludes the article.

II. Related Work

Today, there are some different programs for car sharing, and these can be divided into two main categories – static and dynamic applications. Amey [7] described dynamic ride sharing as:"a single or recurring rideshare trip with no fixed schedule, organized on a one-time basis, with matching of participants occurring as little as a few minutes before departure or as far in advance as the evening before a trip is scheduled to take place". The static method is one that requires users to wait a while for an answer, sometimes even for a few days or weeks. This is a crucial difference when it comes to the usage in real life transportation. Both methods are in use today although dynamic ride sharing is often referred to as a new, improved ride sharing technique in comparison to the outdated static method.

A static car sharing method is mostly common in dedicated websites while mobile applications usually use the dynamic method.

A. Static Ride Sharing Applications

A static ride sharing application consists of an interface where a user is invited to offer or request a ride sharing, while other users could see his request. A successful match for a ride is one where a user finds another user who matches his request, based on what they published and described.

The key to success of these applications is that they gather a large amount of data from users, which gives each user the ability to narrow his search for passengers or for a ride. Web applications can display all their data at once and their search options are many and diverse (by determining more accurate preferences, depending on the specific website), while mobile applications have to display less data at a given time without interrupting the intuitivity and functionality of the application. One mainly used feature of ride-sharing is the forum, where users can tell about their experience and exchange information, tips, and recommendations.

In this context, several static applications for ride sharing have been developed around the world, which share similar characteristics. Here are some examples for operative, widely used, static, ride sharing programs, that make use of some or all the features mentioned above.

"Backseatsurfing.com" is a carpool / hitchhiking website that divides its page layout into new rides, free rides, and the last minutes rides. Besides giving the option to find a ride, the site is meant to be a social network for hitchhikers, and it includes profiles, galleries, and videos. "Digihitch.com" is another website application which has a platform for hitchhiking, mainly between countries in Europe. In this site you cannot see the exact departure place or even the city, without contacting the driver first.

1. *Drawbacks of Static Ride Sharing Applications*

There are a few disadvantages for these systems. First of all, they use a static method that does not work in real time. For example, if a driver is delayed or has decided that he wants to postpone his trip, he will not always come back to his published offer to update it, which will cause the application or website to be unreliable and outdated. If a driver enters his trip in the website as a regular trip that he does every week, but occasionally it does happens that he skips a week, or changes his path- again, his offer / request becomes outdated. In addition, in websites, there is no direct interaction between users via the website itself, but only with emails, while it may take some time for the users to view the messages they receive, especially if there are any last minute changes.

B. Dynamic Ride Sharing Applications

The advantages of the smart mobile phone are many. First of all, the mobile phone is carried by a person at all times, and so the user can access and update his information at any given time. With smartphones users can easily pass data to others without the need to access the nearest computer. In addition, the embedded GPS chip allows detecting one's current exact location and speed, which can be very useful when it comes to location- based algorithms and applications. Due to these advantages, some dynamic ride sharing applications have been developed which rely on the above characteristics.

Another example of a mobile ride sharing application is the well known, Avego Driver [8]. This is a real-time ride sharing application combined with automated payment transaction management that uses web accounts as the method of payment to drivers by the passengers, while the fee is determined by the distance they shared during the trip. Users can see each other's last activities, and all the rides around them, even before specifying a destination. On the other hand, for a match between a driver and passengers to be done, they need to insert the exact same route, otherwise the application might miss the connection, and the passenger needs to choose from a list of pre defined pick up and drop off spots. i.e., there is a difficulty in giving a ride just for a part of the way, as we would like to allow in our application. Zebigo [9] is another known but different application that serves the same purpose. It is simple to work with and understandable, but users cannot provide as a destination or origin, a general name of a place such as a city or interchange, but instead they can only enter a specific detailed address, which might be less convenient when fast and easy usage is required. This requirement could

Doi: 10.7321/jscse.v3.n3.133 e-ISSN: 2251-7545

prevent people from using the application, while users tend to prefer a more flexible and convenient one. The less limitations it has, the more users using it.

I. Our system

A. Description

The proposed system is a dynamic car-sharing application. This application differs from others by that it finds overlapping of routes and analyzes the longest common route between a few given routes. It does not rely on a similar origin or destination to match routes, but actually finds a correlation between the paths, even when they start and end elsewhere. This allows us to find matching trips for users, even for a part of the way.
The system works as follows. After downloading the software and registering as a user (see section 3.6 for details on how to obtain a registration code), the user enters his personal details that include different identifiers for the program to relate to later on when combining routes. In addition, the user can enter his car details which will be saved and will appear as the default details of the user's car when creating a route. After creating a user profile, the user chooses between two views - "driver" and "passenger". The driver view is in general to create a route and the passenger view is to join one.

B. Driver view

The driver view (screen) has two main options, creating a new route or updating a route. When pressing to create a new route, the driver needs to enter all the information that is requested for the route. If the driver had saved a route in the past or had some details saved before, they will appear as the default. The driver will need to enter the origin (the default will be his location) and the destination, and the time of departure. The other compulsory details are car model and color, license plate number (so that the passenger will be able to clearly identify the driver at the meeting point), smoking / non smoking and special requests (e.g. only male/female passengers, no food in the car, no pets...). The rest of the details will be the default values unless the driver has defined something else (e.g. number of available seats will be 3). After entering the details, a map (of the chosen route) will appear on the screen. The driver can then choose to alter the route, and different routes will be displayed.
After choosing the desired route, the data of this specific ride will be saved in the database. From this point on, this ride will be visible to all users. When a passenger wishes to join that ride, the driver will be noticed and could accept or reject the request. At a specific time before the start of the ride, the driver will receive a message showing the places he needs to stop at. Drivers can also save their common routes and departure times for future use. This is the driver's point of view.

C. Passenger view

When looking for a ride, a passenger needs to enter his source location (The default value being his current place, as determined by GPS), destination location and desired departure time. Then the screen will display all optional rides sorted by the algorithm used by the system to find the best match between the routes (defined further in section 5.1). The passenger will only see rides that are relevant for him. After choosing a route and a meeting point where the passenger will join the car, the number of available seats for this ride in the database will decrease by one. A ride reaching zero available seats will not appear anymore.

D. Interaction between users scenario

The way the application works is described in more detail in the sequence diagram in Fig. 1. In this figure, we show the process of interaction between the users and the system, and what the system performs in order to match and find a ride the users could share. After the driver enters his origin and destination, the system stores the selected route, and waits for a passenger to perform a search for a ride according to his own desired locations. After finding the suitable routes for the given passenger's path, we have to compare these routes to the drivers' routes, in order to find the most suitable match, according to destination, departure time, and estimated arrival time. The system then activates an algorithm for comparing routes and sub-routes (see further in section 5.1), and if it finds a match, the system will connect those users and monitor the trip (until the passenger indicates that he exited the vehicle and that the ride was successful). We can see that a driver has to approve his final route, because there could be many available routes between the same origin and destination, and he might want to go through a certain route rather than the other, unlike the passenger who doesn't usually have a preferred route.

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco, CA, U.S.A., March 1-2, 2013
Doi: 10.7321/jscse.v3.n3.133 e-ISSN: 2251-7545

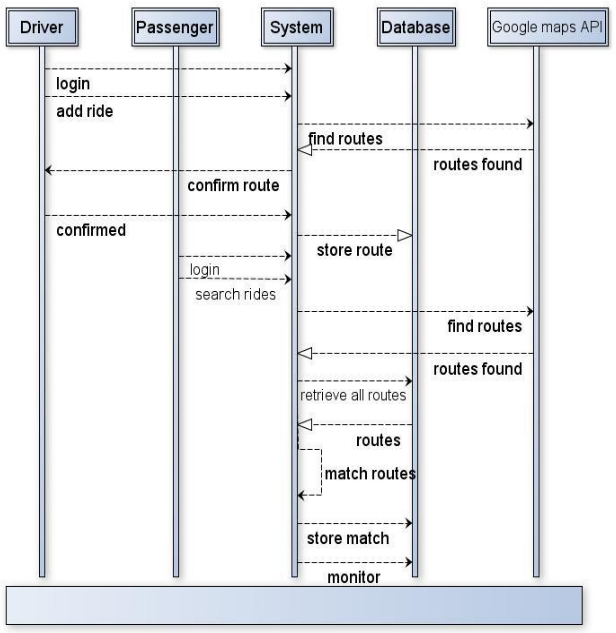

Figure 1: Sequence Diagram-matching a shared ride for a driver and a passenger

This diagram show only one driver and one passenger, but the same process will take place when it comes to multiple users.

E. Multiple users

The idea of the relationships between multiple drivers and passengers is described in the next two diagrams. In Fig. 2 it is shown that all users' data, drivers and passengers, is stored in the database. Each driver in the diagram has his own arrow color to demonstrate a different entry in the routes table. The passengers can connect to one of these drivers and the system will give them the same unique symbol their driver has (in our case the same color of arrow). In Fig.3 we can see that a passenger who was previously matched with a driver, interacts with his driver and could join his ride at an accepted location along the way.

F. Advantage of Our System

The advantages of our system are manifold:

1. The system can find a combined route between driver and several passengers using an algorithm that was specially developed for that purpose. Unlike other car sharing applications, where the passenger needs to choose from a list of drivers who can sometimes be irrelevant for him, our application automatically computes the best overlapped route that can be found. The order of the routes displayed to the passenger derives from the longest common path, the distance from the origin and to the destination and a few more parameters that are taken into consideration.

2. The GUI is designed to be as user-friendly as possible, without interference with the user's regular mobile phone activities. When planning the algorithm we took into consideration how to simplify the use and to decrease the number of actions a user needs to perform in order to achieve his goals.

3. Both driver's and passenger's locations are shown in real time on the map. This feature provides users a more liable and accurate use. When users can see each other on the screen, it adds to users' safety and to easier recognition.

4. Personal security is an essential need, especially considering the openness of the application and the unidentified users behind the usernames. Our system incorporates user- friendliness features along with protecting privacy and ensuring security. One of these features is the need for a registration code. After downloading the application, a registration code must be entered in order to activate the application. This registration code can be obtained in one of several ways. The most common one is by receiving the code from a friend, where the introducing friend is already a registered member and has accumulated a certain amount of credits, which allow him to generate registration codes to distribute to others. Other ways are by belonging to all sorts of communities that have been approved by the system's operators for the purposes of distributing registration codes (student groups, companies, etc.)[7].

5. Recommended drop off locations can be displayed, or could be added by the users themselves to make the system

more updated and dynamic.

```
Algorithm 1 Matching Route
1: driverUser ← getting user details from database
2: driverSource ← receive departure point of the driver
3: driverDestination ← receive destination point of the driver
4: driverRoute[] ← CalculateStoppingPoints(driverSource, driverDestination) {analyzing coordinates along the path}
5: deviation ← get driver's predefined maximum deviation allowed from database
6: passengerUser ←getting user details from database
7: passengerSource ← receive departure point from the passenger
8: passengerDestination ← receive destination from the passenger
9: passengerRoute[] ← CalculateStoppingPoints(passengerSource, passengerDestination)
10:
 for each 2 routes do {combining the passenger's route to the driver's and the opposite}
   point destination = route1.finalPoint, point meetingPoint = null, point dropoffPoint=null, point $i$ = route1.startPoint, point
   $j$ = route2.startPoint
   for each point in route2 do
     if distance(i,j) + distance(j,destination) $\leq$ deviation then
       add to route1 point j after i, $i = j$, meetingPoint = $j$, $j$ = nextPoint()
     else
       dropoffPoint = $i$, $j$ = nextPoint()
     end if
     store route1 in DB
   end for
 end for
 for each route in DB do
   route $r$ = find longest route()
   if r.distance $\leq$ driverRoute.distance+deviation then
     finalRoute=r return
   else
     remove r
   end if
 end for
```

G. Drivers motivation

The motivation for drivers to offer car rides can be by achieving rating points (which will grant discounts and offers). Payment from passengers is another option (although not necessarily allowed everywhere, because of restrictions of insurance policies). Businesses may encourage their employees (financially) to use this system, which will reduce the number of cars some companies have to provide to their workers, and reduce their expenses on gas and transportation.

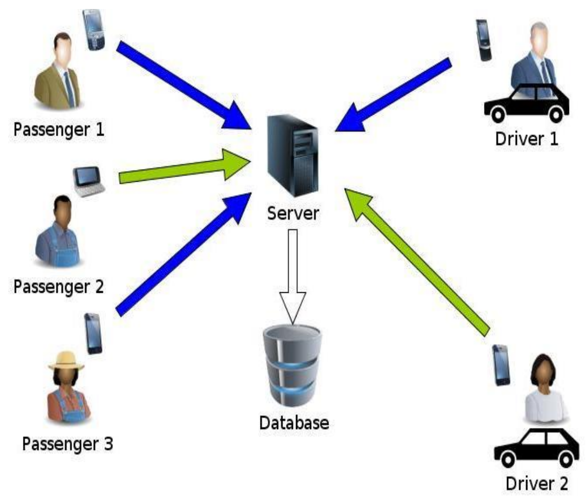

Figure 2: Matching Users' Routes at the Database(shown in same color of arrows)

III. SYSTEM DESIGN

The system is built in a client-server model, which allows us to save all important and private information remotely on the server, instead of locally on the mobile device. The device itself holds only user preferences. In order to make the software simple, we designed it using a modular approach, in which the components of the application are divided according to their purpose. The main component contains the matching algorithm, and the ability to access the web service, and connect with Google maps API. In addition, there are the parts that are in charge of the input / output stream from / to the database, and the GUI section that handles the display, according to the assumption that the software's display has to be separated from the logic or implementation of the application. The relations in between these parts are described in the following object diagram (Fig. 4).This figure demonstrates the main components of our system, and how the control communicates with all other parts, such as the database (for storing or retrieving data), the display (GUI), and other mobile functions (such as the GPS). In Fig. 6 we present the main part presented in Fig. 4 which is the control itself, but divided into its main functionalities, which are: registration, configuration, adding rides, searching for routes for defined origin and destination, matching and binding routes, and monitoring trips that are already on the way. All these functions are activated by the system management, while users can address some functions, such as adding or searching rides. Other functionalities will be processed by the management itself, such as the matching routes function, and monitoring. Passengers and drivers address the system management, either when requesting a ride or when offering one, and the system will then call and direct them, according to their needs, to the specific functions that handle each request. The management does not handle those requests itself, in order to prevent overload on the system which might interfere with other users' actions.

IV. IMPLEMENTATION

The application was built in Eclipse IDE, using JAVA language for Android based devices. It has been tested using AVD manager emulator. We chose to use Android for multiple reasons. First, it is supported by various types of devices. Second, Android has reached 52.5% of the global smartphone market share as of November 2011 [10]. Finally because of its open source approach, which grants us access to the phone's internal functionalities.

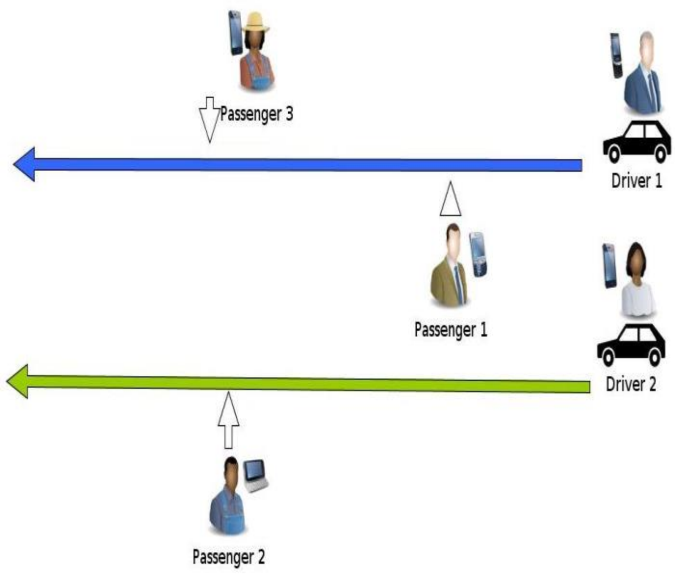

Figure 3: Passengers Join Their Corresponding Drivers

A. Algorithm

In this section we describe our algorithm (See Algorithm **Error! Reference source not found.**) which receives two routes as inputs, and searches for the longest similar path between both routes. The routes mentioned in this algorithm were received before from Google API after sending requests to calculate each route (of passengers' as well as drivers') on its own, and then stored in our database. The algorithm does not state the previous process of calling, parsing and storing these routes in the database, and only shows how it retrieves and makes all calculations on the existing routes, after querying them from our database. This is done by iterating and examining coordinates of those two routes and storing the overlapping coordinates. While receiving Google maps's routes, there is no way of knowing all coordinates along that route, and even if it would be so, it would take an enormous amount of time and resources to calculate and compare every single one of these points. Instead, we noticed that Google only stops or starts new steps of the given route when there has been any change- a turn has to be made, the road number has changed, and so on. This concept narrows down the number of coordinates that are a possible meeting point between the users, to the amount of changes in either the driver's route or the passenger's. The algorithm works bi-directionally: first, it matches the passenger's given coordinates to the driver's coordinates, and then does the opposite. We start by establishing a new path for the driver that starts as before, but when he has arrived to the nearest coordinate to the passenger's first coordinate (according to the distance between them), the path then changes and is appended with the passenger's first coordinate. If that deviation exceeds the driver's predefined maximum deviation, the path changes back, and we proceed to the passenger's next coordinate and try to add it instead. Now, the driver's new path is composed of his original route with an additional coordinate located as nearest as possible to the original path. The same is done with all coordinates of the passenger's route that do not exceed the driver's predefined maximum distance deviation. Because we cannot know whether the driver's original route passes by some coordinates of the passenger's route or if maybe it is the opposite, we have to do the above calculation twice: once starting with the driver and afterwards starting with the passenger. At the end, we will have a new route containing elements from both original routes, which will represent the easiest way to go from A to B with an acceptable range of a detour meant for picking up or dropping off a passenger. Adding the driver's route coordinates that exist in the passenger's route (and thus extending the driver's route) while they are distant up to a defined distance away from the driver's original route is optional. The driver can decide not to deviate at all by predefining that distance to zero. The minimum requirement for a successful match is the matching of at least two coordinates. The first matched point will be defined as the advised meeting spot, and the last corresponding point would be the preferred drop off point. If they were no two matching points at least the algorithm will return no results.

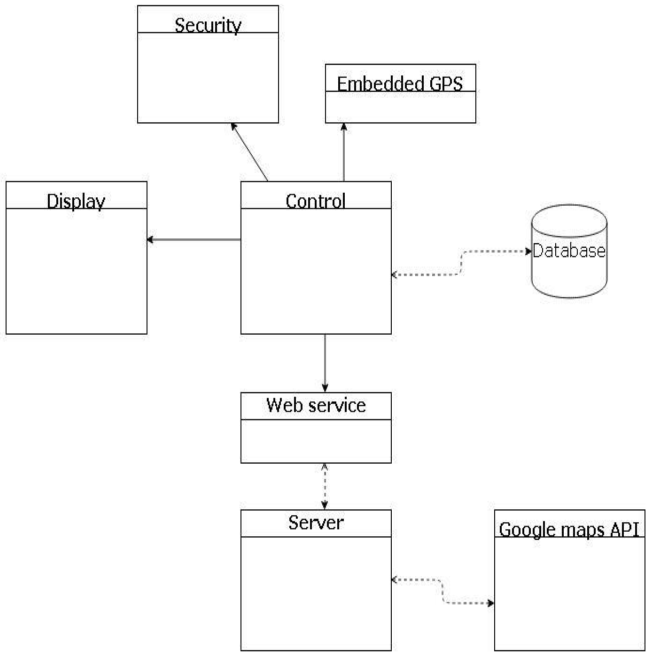

Figure 4: Object Diagram

This method is based on a well known computer science problem, which is finding the longest string (or strings) that is a substring (or are substrings) of two or more strings. This is called the "Longest common substring problem". For example, the longest common substring of the strings "ABABC", "BABCA" and "ABCBA" is string "ABC" of length 3. Other common substrings are "AB", "BC" and "BA" but they are not the longest, as their length is only 2 characters.

One way of solving this problem is by building a suffix tree, while we have to find the deepest internal nodes which have leaf nodes in the tree. The suffix tree scan takes a linear time, so it is a fast algorithm, but suffix trees are generally very large, and take a lot of space. That is why we preferred using a dynamic programming algorithm.

Dynamic programming is a way of solving complex problems by dividing them into simpler subproblems where possible. In the longest common substring problem, this method uses recursive bottom-up calculations to solve the general problem. The longest common suffix is found through function "*LSf*" as following:

$$LSf(S_{1..p}, T_{1..q}) = \begin{cases} LSf(S_{1..p}, T_{1..q}) + 1 & \text{if} \\ 0 & \text{oth} \end{cases}$$

The idea is to scan the strings from the end, add 1 to the total amount of matching characters if there was equality between the strings' ending characters, and call the function again on the strings without their endings.

When computing, we are building a table of *mxn* (*m*, *n* are the strings lengths), and filling the table with the correct values, according to the above formula. In addition to the cell value, for each cell we add an arrow that points to the upper-left if its *S*[*i*], *T*[*j*] (*i* for rows, *j* for columns) are equal. Else, when they are not equal, if the cell above it has a greater value than the cell to its left, the arrow points up. Else, it points to the left.

After constructing the table, we will find the greatest value in the table, start scanning from it according to the arrows along the path, and this will lead us to the longest common substring.

For example see fig. 5 that shows how to find optimal LCS length which is in this case *c*[*m*,*n*]=3.

	J	→	1	2	3	4	5
I		y_j	B	D	C	A	B
↓	x_i	0	0	0	0	0	0
1	A	0	0↑	0↑	0↑	1↖	1←
2	B	0	1↖	1←	1←	1↑	2↖
3	C	0	1↑	1↑	2↖	2←	2↑
4	B	0	1↖	1↑	2↑	2↑	3↖
5	A	0	1↑	1↑	2↑	3↖	3↑

Figure 5: The optimal LCS length

B. Database

Our database is meant to store multiple users, and multiple routes. The main table is the routes table. The routes received from the Google maps API are stored after parsing in the form of coordinates, while some may also have a specific name, such as intersections, universities, etc. Each route is saved by a unique id number and can store as many steps (locations points along the route) as needed, as the route received from Google is basically composed from a collection of these multiple points. Each point is defined by its latitude and longitude, which makes it possible for us not only to search for correlation between two points from the table, but also to know how far are they from each other, by using a simple mathematic formula. This can be used in the case that the points are close but not the same, and we would like the program to consider them as the same point (up to a certain radius around these points). The route table also contains a reference to the driver's details in the users table and as people join the ride, a reference to their details as well (this is stored in the "route users relation" table). The users table saves all the data and status of the users, ratings, reviews, etc. The groups table can help us create matches only between users within the same group if necessary, and because each user can be a member in more than one group, we save his desired groups at a table called "groups user relation". Another tables, "route preferences" and "general preferences" deals with users preferences for each ride or for their general demands for all rides, respectively. The databases we used for implementation is MySQL (for external storage on a local server for now), and SQLite (for local storage on the device).

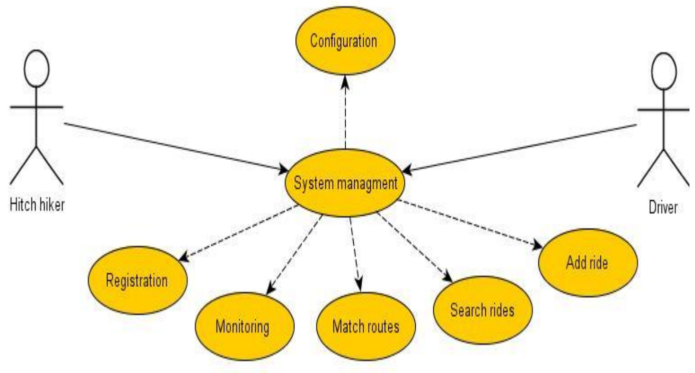

Figure 6: Usecase Diagram - The system's main functions

There are other tables apart from the ones mentioned above. The overall structure of the MySQL database is determined by these tables (see fig. 7 for details):

1. Users - to store users' details and encrypted passwords.Also contains users' application preferences and usage data. Passwords are encrypted using SSHA (Salted SHA-1) method.

2. Drivers routes - information on all rides offered by drivers. A row stores the ride's origin and destination, time, driver's id, and passengers who joined.

3. Drivers points - a point is the latitude and longitude coordinates. A route is the total amount of points that exist along the way. This table stores all the points for existing routes.

4. Passengers routes - all rides requested by people which have a valid match;

5. Passengers points - points of the above passengers' routes.

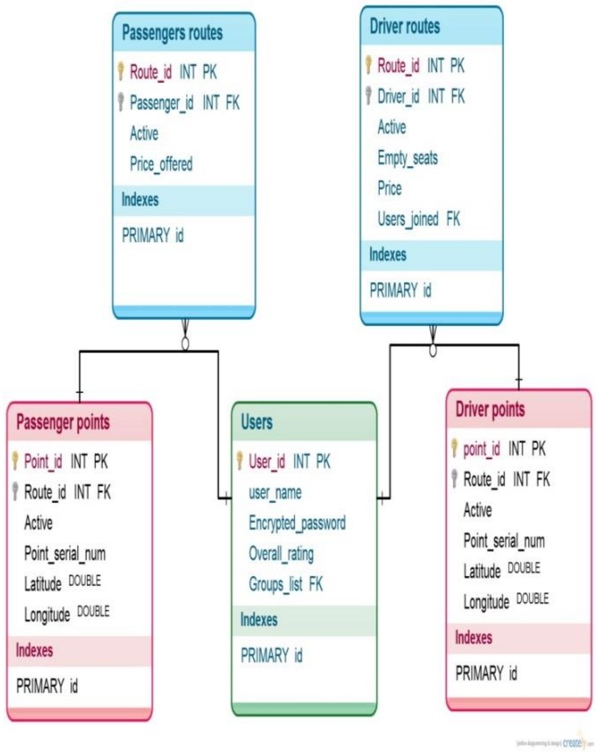

Figure 7: The database model

V. FUTURE WORK

There are some research possibilities for further work on the current application:

• Voice recognition: How convenient would it be if we could just talk to our phone and say in our own words: "I'm driving from Memphis Tennessee to Chicago Illinois at 5pm, 2 available seats". Since our application is built in a modular way, it would be very easy to add unique and special features like this. Using voice recognition will substantially increase the use of people. As technology becomes more and more advanced, we expect things to be faster and easier to handle. Talking to a phone in a free manner is, generally speaking, more user friendly, and will save him precious time.

• Account Ratings: In order to increase and attract more users the system must have a way to rate users with ratio to their use of the application [11]. The more they use the application the more benefits they should receive. Adding a system that gives points for each drive, points for good service to the passenger, come into considerations when the passenger enters his feedback from the drive. Each user will have a score that will be taken into account in the way the algorithm sorts the optimal rides for the passenger. On the other hand, if there is a driver who either does not give good service or drives in a dangerous way, the passenger can give him a low rating and that would be taken into consideration when compiling the algorithm. We hope that in the future, high ratings will enable drivers discounts and offers from businesses and coffee shops along the routes, which may increase the use of this method.

• Account Credits: The primary way to attract people to use this application must consider the financial point of view. Drivers spend a lot of money on fuel. This is one of the main reasons people refrain from taking their own car for a long ride. However, if the driver knows he will be sharing the expenses with other passengers, it will give him a good reason to take his car [12]. Seeing that the application is build in a modular way, adding the necessary code and data that will give the users an option to share the expenses of the ride is feasible.

• Group Car sharing: Besides the usual rides that individuals enter either as offers or requests, this system in the future will contain the option to see rides by category. The categories can be various occasions and special events such as weddings, sports event, concerts and graduation ceremonies. The categories could also be existing groups in our implementation such as: friends, fellow students, work associates, club members, etc.

VI. Conclusion

In this article we have presented the design and implementation of a new approach in developing mobile ride-sharing and carpooling applications. The application works in real-time and matches car rides by comparing routes to find the longest common path. Unlike ride sharing websites, which have to be updated frequently by the users, do not contain interaction between them, and do not supply enough information about exact pickup or drop-off locations, and unlike other ride sharing mobile applications that find matches only according to the users' origin and destination, in the above system there are updates at real time, exact locations based on GPS readings, interactions among the users, and algorithm that enables the system to locate and match the best suitable ride according to the users' paths and coordinates along the way, which allows users to share just parts of their ride. This project demonstrates an algorithm that builds a new route from two given routes dynamically, step-by-step, with attention to the maximum preferred deviation from the original route, and with error-correction when exceeding this limitation. We believe that with the growing attention to environmental issues, and as the density of transportation and parking places continue to increase, there will be a greater need and use in such an algorithm, to the benefit of all.

References

[1]OICA, "Automative production statistics," http://oica.net/category/production-statistics/, International Organization of Motor Vehicle Manufacturers, Tech. Rep., 2011.

[2] D. Tencer, "Number of cars worldwide surpasses 1 billion; can the world handle this many wheels? " *Huffington Post - Business*, Aug. 2011. [Online]. Available: http://www.huffingtonpost.ca/2011/08/23/car-population_n_934291.html

[3] J. Sousanis, "World vehicle population tops 1 billion units," *WARDSAUTO - The information center for and about the global auto industry*, Aug. 2011. [Online]. Available: http://wardsauto.com/ar/world_vehicle_population_110815/

[4] L. Yvkoff, "Drivers spending 1 month each year in traffic," *CNET reviews*, March 2011.

[5] "Estimating urban traffic and congestion cost trends for australian cities," Bureau of Transport and Regional Economics , Department of Transport and Regional Services, Australia, Working Paper 71.

[6] D. A. Hennessy, D. L. Wiesenthal, and P. M. Kohn, "The influence of traffic congestion, daily hassles, and trait stress susceptibility on state driver stress: An interactive perspective," *Journal of Applied Biobehavioral Research*, vol. 5, no. 2, pp. 162–179, 2000.

[7] A. M. Amey, "Real-time ridesharing : exploring the opportunities and challenges of designing a technology-based rideshare trial for the mit community," Master's thesis, Massachusetts Institute of Technology, 2010.

[8] http://www.avego.com/.

[9] http://zebigo.com/.

[10]R. Cozza, C. Milanesi, A. Zimmermann, D. Glenn, A. Gupta, H. J. D. L. Vergne, C. Lu, A. Sato, T. H. Nguyen, and S. Shen, "Market share: Mobile communication devices by region and country,3Q11" Gartner, Tech. Rep., Nov. 2011.

[11]A. S. Shirazi, T. Kubitza, F. Alt, B. Pfleging, and A. Schmidt, *WEtransport: A Context-based Ride Sharing Platform*, 2010, pp. 425–426.

[12]E. A. Deakin, K. Frick, and K. M. Shively, "Markets for dynamic ridesharing? case of berkeley, california," University of California Transportation Center, Working Papers, 2011