Smart Space Logistic Service for Real-Time Ridesharing

Alexey Kashevnik, Nikolay Teslya, Nikolay Shilov
SPIIRAS
Saint-Petersburg, Russia
{alexey, Teslya, nick}@iias.spb.su

Abstract

The paper describes a logistic service-based approach to real-time ridesharing based on smart-space concept. Smart-M3 information platform is used as smart space infrastructure for presented approach. The service is based on Smart-M3 RDF ontology which is formed by ontology slices of participants' mobile devices. The paper presents an algorithm for finding appropriate fellow-travelers for drivers as well as definition of acceptable pick-up and drop-off points for them.

Index Terms: ridesharing, ontology, smart-m3, smart space.

I. INTRODUCTION

Nowadays, a dramatic increase of the number of vehicles can be seen in many major cities and, as a result, the load on existing transport networks increases. This leads to frequent traffic jams and accidents. One of possible solutions for this problem is ridesharing. Ridesharing (also known as carpooling, lift-sharing and covoiturage), is a shared use of a car by the driver and one or more passengers, usually for commuting. Dynamic ridesharing (also known as instant ridesharing, ad-hoc ridesharing, real-time ridesharing or dynamic carpooling) denotes a special implementation of a ridesharing service which enables a dynamical formation of carpools depending on the current situation. Typical for this type of carpooling is:

- arrangement of one-time trips instead of recurrent appointments for commuters;
- the usage of mobile phones for placing carpooling requests and offers through a data service;
- automatic and instant matching of rides through a network service.

The first historical incidence of successful ridesharing was the tremendously popular yet short lived "Jitney Craze" beginning in 1914, when the US economy fell into recession with the outbreak of WWI, and some entrepreneurial vehicle owners in Los Angeles began to pickup streetcar passengers in exchange for a 'jitney' (the five cent streetcar fare).

The second major period of rideshare participation, and the period most likely to be identified as the first instance of traditional carpooling, was during the World War II (WWII). Opposite to the jitney era, the government encouraged ridesharing heavily during WWII as a method of conserving resources for the war effort. This period of ridesharing promotion was exceptionally unique, since it entailed an extensive and cooperative effort between the federal government and American oil companies.

The third period of interest in ridesharing picked up substantially with the Arab Oil Embargo in the fall of 1973 and 1979 oil crisis [8].

Nowadays, the next period of interest in ridesharing is expecting. It is associated with the intensive development of data processing, transfer technologies, and computing capacities, which can simplify the search for fellow travelers.

The following main schemes are used by people in different countries for searching for fellow-travelers:

- Search via public forums and other communities. For example: *eRideShare.com* [11], *PickupPal* [12], *Zimride* [15], *RideshareOnline* [14], *rideshare.511.org* [13], *CarJungle* [10]. The advertisements about trips are posted on a Web-site by users. This advertisement includes the start and end points, some information about people who post this ad, trip cost, time of the trip, etc;
- Search via private Web-services. People can get account in a private service only if they have an invitation. For example, *Zimride* service has a private interface for universities and companies;
- Search via special applications on mobile devices. With these applications users can edit their profiles, routes and search for fellow-travelers. The examples are *PickupPal* [12] and *Avego* [9];
- Search via agents (e.g. taxi companies);
- Pick-up points (not pre-arranged).

Software for the mobile devices uses the client-server architecture. This architecture provides for the implementation of a centralized server and clients sending data processing requests to the server. Presented in the paper approach is based on the decentralized smart space infrastructure. This approach allows increasing the stability, speed, and reduces the network's load.

II. SMART-M3 PLATFORM

For the presented ridesharing system the open source Smart-M3 platform has been used [7]. The key ideas of this platform are device, domain, and vendor independent. Another key idea is that devices and software entities can publish their embedded information for other devices and software entities through simple, shared information brokers. Information exchange in smart space is implemented via HTTP using Uniform Resource Identifier (URI) [1]. Semantic Web technologies have been applied for decentralization purposes. In particular, ontologies are used to provide for semantic interoperability.

Smart-M3 platform consists of two main parts: information agents and kernel (fig. 1) [6]. The kernel consists of two elements: Semantic Information Broker (SIB) and data storage. Information agents are software entities installed on the mobile device of the smart space user. These agents interact with SIB through the Smart Space Access Protocol (SSAP) [2]. The SIB is the access point for receiving the information to be stored, or retrieving the stored information. All this information is stored in the data storage as a graph that conforms to the rules of the Resource Description Framework (RDF) [5]. By these rules all information is described by triples "Subject - Predicate - Object".

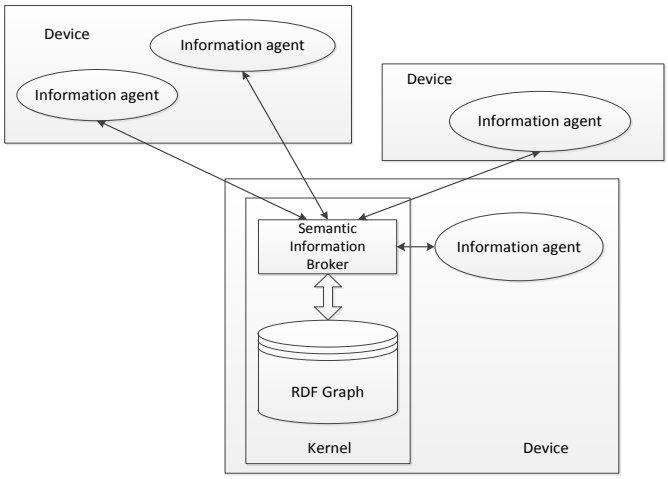


Fig. 1. Smart-M3 Platform

III. THE LOGISTIC SERVICE ONTOLOGY

The logistic service ontology describes the domain area of ridesharing at the macro level (fig. 2). The macro level ontology is based on integration of parts of the mobile devices' ontologies.

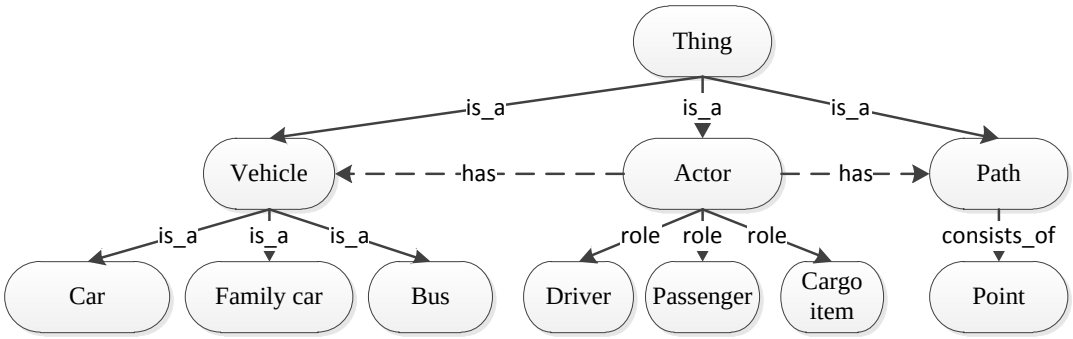


Fig. 2. Logistics service ontology on the macro level

The logistics service ontology consists of three main parts: vehicles, actors and paths.

A. Vehicles

The vehicles are:

- cars with no more than four vacant seats;
- family cars with 5 to 8 vacant seats;
- buses with 9 and more vacant seats.

B. Actors

The actors are: drivers, passengers and cargo items. All of them have vehicles and paths. For example, driver has his own car and several points defining his/her home, work and other locations. Passenger may prefer some vehicle type and has points of home, work, and other locations. Cargo items have size and vehicle type needed for its transportation.

The class actor consists of (fig. 3):

- ID. Unique ID for each user;

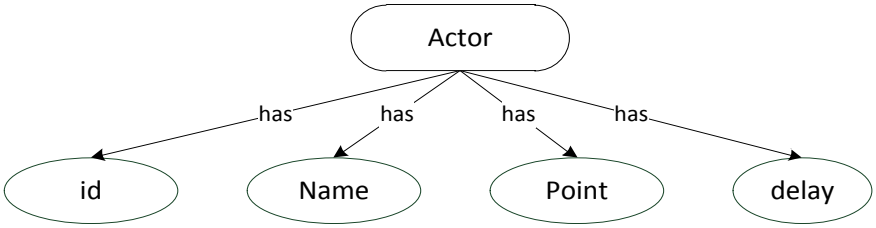


Fig. 3. Class "Actor"

- Name. First and last name of the user;
- Point. Path point belong to the user (2 minimum: the start and the end);
- Delay. Maximal possible time of waiting in the meeting point.

The class "Driver" is a subclass of the class Actor and inherits all its properties with two own properties:

- Vehicle. Vehicle type;
- Detour. Maximal detour from the shortest path.

The class "Passenger" is a subclass of the class Actor and inherits all its properties with own property "Detour" the same as in the class "Driver".

The class "Cargo item" is a subclass of the class Actor and inherits all its properties with own property "size" defining the physical size of the cargo item.

For the path definition the set of points is used. This set is an ordered list of key points obtained as result of the shortest path searching algorithm (e.g., Dijkstra or A*). The class "Point" has the following structure (fig. 4):

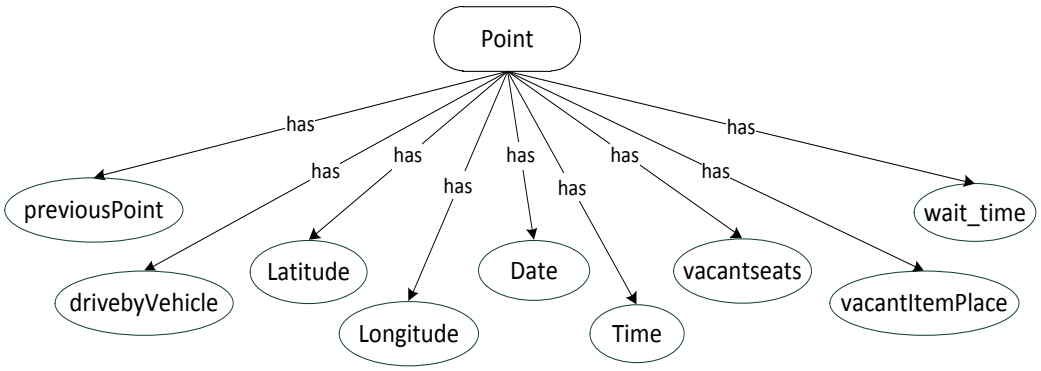


Fig. 4. Class "Point"

- previousPoint. Contains the previous path point. For the start point its value is "FALSE";
- Latitude;
- Longitude;
- driveByVehicle. If the point belongs to the passenger, it contains the driver who gives a ride to this passenger. If the passenger walks then its value is "FALSE";
- vacantseats. The number of vacant seats in vehicle in point;
- vacantItemPlace. Tne number of vacant places for cargo items;
- Date. Date, when the user will be at this point;

– Time. Time, when user will be at this point;
– Wait_time. How long the user will be waiting in this point.

Since the ontology in the smart space is represented in RDF standard, it looks like follows:

(‘user1’,	’name’,	‘Name Surname’)	- name of user1
(‘user1’,	‘is_a’,	‘Driver’)	- user1 is a driver
(‘user1’,	‘vehicle’,	‘vehicle_type’)	- user1 has this type of vehicle

etc.

In [4] the logistics service ontology is described in detail.

IV. Algorithm for finding matching driver and passenger paths

The problem of finding a matching path between the driver and the passenger in the ridesharing service can be formulated as follows: it is needed to determine the possibility of ridesharing between participants, based on the information about their routes and restrictions set by users’ services. The following algorithm describes the procedure of finding matching path acceptable for the driver and the passenger in the presented ridesharing service.

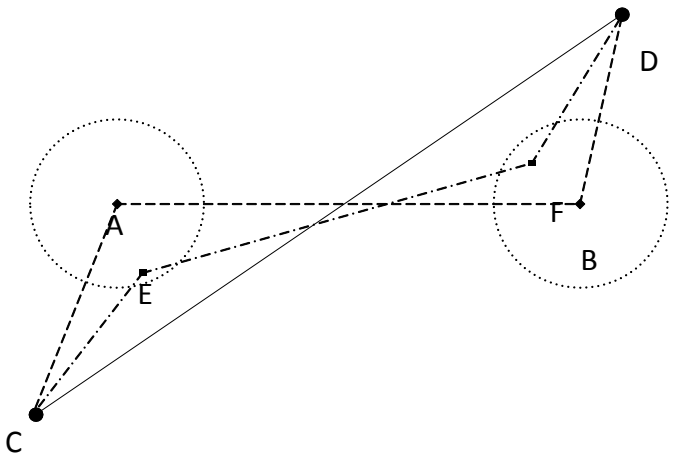


Fig. 5. The main idea of matching driver and passenger path search

Let A be the start point and B be the end point of the pedestrian's path. C is the start point and D is the end point of the driver's path. The shortest driver's path, which is found with the help of GIS, is indicated by the solid line (in generally, CD is not a straight line, it depends on the map of the region). Fig. 5 shows that the driver and pedestrian move almost in the same direction and in some parts of the routes the driver can give the pedestrian a ride. This situation is indicated in the figure by the dotted line (the CABD path) and it is the simplest situation, because the meeting points match with the start and end points of the pedestrian’s path. A more difficult situation is searching for a meeting point when it belongs neither to the driver’s shortest path nor to the pedestrian’s one, but satisfies both the driver and the passenger. One of the possible situations is indicated in the figure by the dash-dot line with the meeting points E and F (the CEFD path). These points have to meet the following restrictions:

1) The distance between the start point of the passenger and his/her meeting point should be less than the maximum allowed detour of the passenger. This area is indicated in the figure by the dotted circle.

2) The driver's detour should be less than the maximum allowed detour.

The general scheme of the matching routes searching algorithm will be follows:

```
FOR EACH driver DO
  FOR EACH passenger Do
    Find_mathing_path(driver.path,passenger.path); // according to the above scheme
    constraint_checking();
    IF ALL constraints IS performed THEN
      set_passenger_for_driver();
  ENDFOR;
ENDFOR;
```

The goal functions for finding the meeting points are:

- Shortest total path (interesting for the driver);
- Minimal waiting time (interesting for the driver and passenger);
- Shortest distance between the passenger's start and end points and meeting points (interesting for the passenger).

As a result, the general task of matching paths has the exponential complexity, therefore, it is necessary to apply heuristics to reduce the task dimension.

A. Heuristic 1

There is no need to calculate merging paths for all pairs of drivers and passengers. It can be possible, for every driver to build a set of candidate passengers:

$$(pp_1^x - dp_i^x)^2 + \left(pp_1^y - dp_i^y\right)^2 \le (PDetour + DDetour)^2, \quad (1)$$

$$(pp_2^x - dp^x{}_i)^2 + (pp_2^y - dp^y{}_i)^2 \le (PDetour + DDetour)^2, \quad (2)$$

where pp_1, pp_2 — the start and the end points of the passenger's path, dp_i — driver path point i, *PDetour*, *DDetour* — detours of the driver and the passenger.

B. Heuristic 2

There is no need to search through all possible combinations of meeting points. The following heuristics help to reduce the number of the possible combinations.

The first heuristic selects points of the sector from which the driver starts. Fig. 6 shows the situation when there is only one point ("C" point) meeting constraints (1) and (2). To determine the potential meeting points it is needed to calculate the angle (3)

$$\theta = \text{arctg}\left(\frac{C^y - A^y}{C^x - A^x}\right) \quad (3)$$

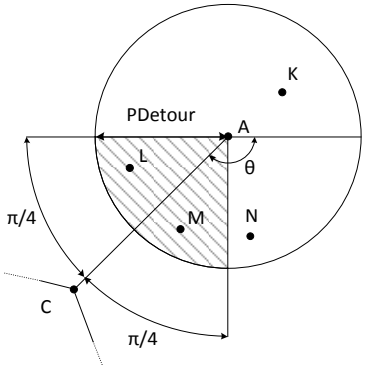


Fig. 6. The first heuristic

and select points in the area $\left[\theta\text{-}\frac{\pi}{4}, \theta + \frac{\pi}{4}\right]$ (points L and M in fig. 6). Point A will always be within the list of the possible points as the passenger's start or end point. If there are more than one point meeting constraints (1) and (2) then the search area expands. This situation is shown in fig. 7 with two points C and F meeting the constraints (1) and (2), and point N is also included in the expanded area.

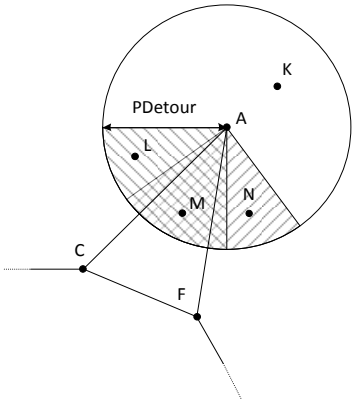


Fig. 7. The first heuristic with two driver's points

The negative sides of this heuristic are:

- selected points can be further than the driver's maximal detour;
- some of potential meeting points can be lost if the an incorrect angle is chosen.

The second heuristic (fig. 8) searches for meeting points at the intersection of the circle with radius PDetour and the circle with radius DDetour. In this way all of the selected points are potentially reachable for both the driver and the passenger, with no need to determine the angle that restricts the selection area. The selection area can be expanded by increasing the number of the driver's path points meeting constraints (1) and (2).

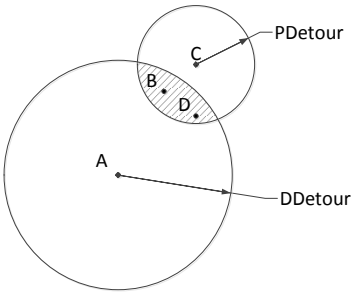


Fig. 8. The second heuristic

Both heuristics require the following constraints to work effectively:

- A lot of drivers. Heuristics have strong limitations and filter out a lot of points. If there are no many drivers, then the use of the heuristics will rarely get positive result.
- A small value of DDetour. Heuristics will not be helpful with a large value of DDetour.
- Uniform distribution of roads on the map. The uneven distribution of roads (rivers, lakes, etc) leads to a lack of roads in some sectors, which could lead to the loss of possible meeting points due to the need to detour around the obstacles and to pick up the pedestrian on the other side.

Both heuristics are used in the logistics service prototype and help to reduce the time of search in more than 1.5 times.

V. System working scenario

Common system working scenario is shown in Fig. 9.

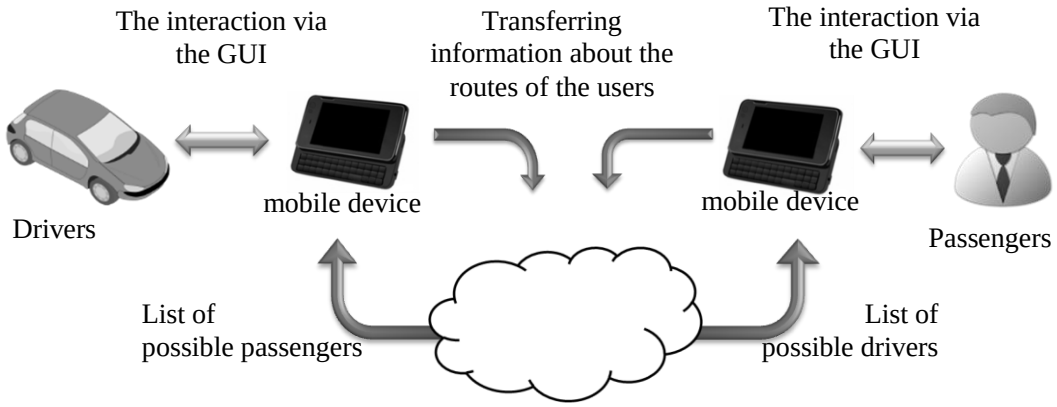
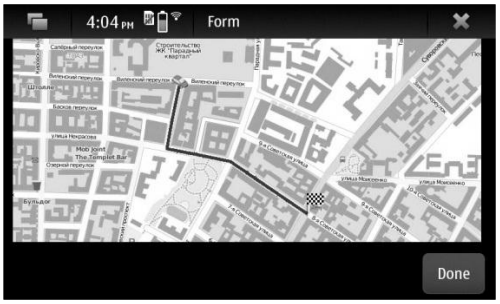
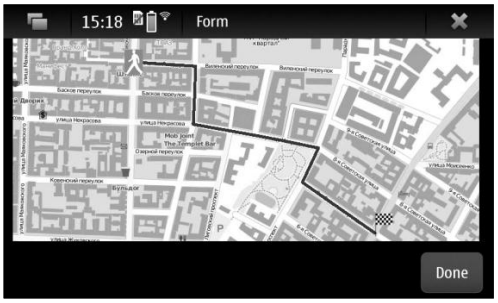


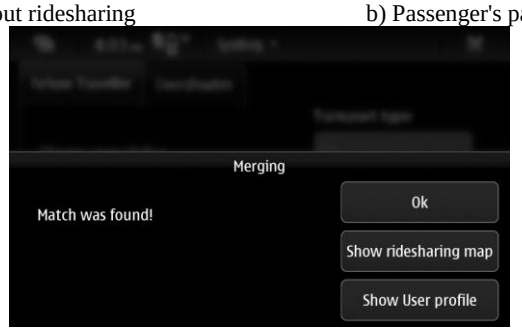
Fig. 9. System working scenario



a) Driver's path without ridesharing

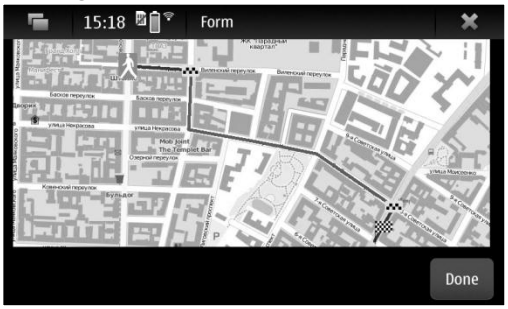


b) Passenger's path without ridesharing



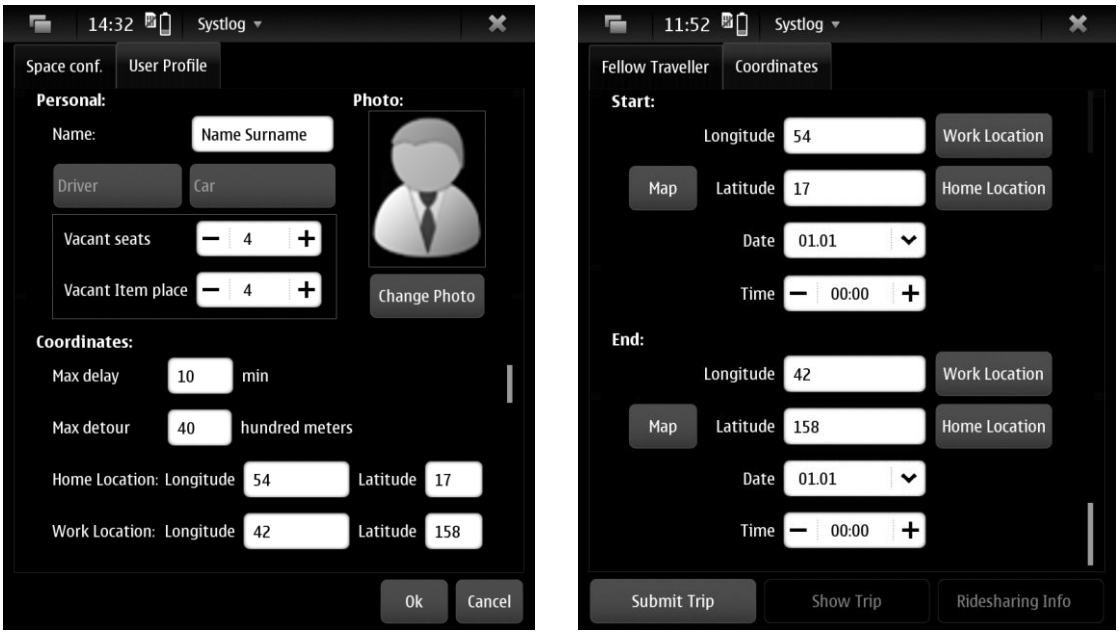
c) Notification message

d) Driver's path with ridesharing



e) Passenger's path with ridesharing

Fig. 10. Prototype screenshots (routes)



a) user's profile configuration b) user's path configuration

Fig. 11. Prototype screenshots (user's routes and preferences)

The mobile application is installed by all users of the service. This application collects the information about the user's agenda, preferences (fig. 11, a), most frequent routes (fig. 11, b), etc. with the agreement of the user. Also the user can set additional constraints, for example, max. delay, max. detour, social interests, etc. (fig. 11, b). This information is transferred into the smart space after the internal processing and depersonalization (only signs of information are transferred, not the raw information). During the execution of the logistics algorithm the groups of fellow travelers are formed. Then, users interactively get the possible fellow travelers with their profiles, meeting points, meeting time, full recommendations about the route (fig. 10, a-e) and if they have permission they can get the link to the external resources, e.g., social network page, which helps the user to make a decision. Sometimes, suggested driver can be a friend of the user friend (this information can be useful for the user in decision making stage). Also, the real-time search is supported. Users can login into and logout from the smart space, change restrictions and then receive the list of fellow travelers in real-time. So, all of work is done by the smart space and users do not need to perform any actions to find fellow travelers.

VI. Conclusion

The logistic service-based approach to real-time ridesharing. With an extensive use, ridesharing can reverse the current world trend of increasing number of cars on the streets of the cities. This will reduce the load on the transport system of the cities, reduce traffic jams and will increase parking places. Ridesharing helps to save fuel, to reduce the overall cost of parking. By the MIT research from 50% to 77% of drivers could rideshare on a daily basis without significant changes in behavior. Max effort VMT (Vehicle Miles Traveled Tax) reductions were 9% - 27% [3]. The developed algorithm can effectively find appropriate fellow-travelers for drivers. The presented heuristics help to reduce the time of search in more than 1.5 times.

ACKNOWLEDGMENT

Some parts of this work have been supported by Open Innovations Framework Program FRUCT – www.fruct.org. Some elements have been carried out under grants of the Russian Foundation of the Basic Research (#10-07-00368, #12-07-00298).

REFERENCES

[1] T. Berners-Lee, R. Fielding, L. Masinter, RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax, URL: http://tools.ietf.org/html/rfc3986

[2] J. Honkola, H. Laine, R. Brown, O. Tyrkkö, «Smart-M3 Information Sharing Platform». *7th Conference of Finnish-Russian University Cooperation in Telecommunications (FRUCT)*. URL: http://fruct.org/conf7/Honkola_Smart_M3.pdf

[3] M. Oliphant, A. Amey, Dynamic Ridesharing: Carpooling Meets the Information Age. *Webinar for the American Planning Association, co-hosted by the MIT team*,(2010), p. 63.

[4] A. Smirnov, A. Kashevnik, N. Shilov, H. Paloheimo, H. Waris, S. Balandin. Smart Space-Driven Sustainable Logistics: Ontology and Major Components, Sergey Balandin, Andrei Ovchinnikov (eds.) *Proceedings of the 8th Conference of Open Innovations Framework Program FRUCT*, Lappeenranta, Finland, 9-12 Nov., 2010, pp. 184-194.

[5] Resource Description Framework (RDF). *W3C standard.* URL: http://www.w3.org/RDF/

[6] Smart-M3 at Wikipedia. URL: http://en.wikipedia.org/wiki/Smart-M3

[7] Smart-M3 at Sourceforge, 2012. URL: http://sourceforge.net/projects/smart-m3.

[8] Rideshare History & Statistics. *MIT "Real-Time" Rideshare Research*. URL: http://ridesharechoices.scripts.mit.edu/home/histstats/

[9] Avego. URL: http://www.avego.com/

[10] CarJungle. URL: http://www.carjungle.ru/

[11] eRideShare.com. URL: http://erideshare.com/

[12] PickupPal. URL: http://www.pickuppal.com/

[13] Rideshare 511. URL: http://rideshare.511.org/

[14] RideshareOnline. URL: http://www.rideshareonline.com/

[15] Zimride. URL: http://www.zimride.com/