Software agents in support of a taxi corporation

Mikolaj Leszczynski, Marcin Niedabylski,
Radoslaw Kutkowski, Maria Ganzha
Department of Mathematics and Information Sciences
Warsaw University of Technology
Warsaw, Poland
leszczynskim,niedabylskim,kutkowskir@student.mini.pw.edu.pl

Marcin Paprzycki
Department of Intelligent Systems
Systems Research Institute
Polish Academy of Sciences
Warsaw, Poland
Marcin.Paprzycki@ibspan.waw.pl

***Abstract*—Management of a fleet of cars for hire (e.g. taxis) involves number of issues that should be solved to provide high quality service to the customers, while keeping drivers "happy". In this paper we show how a software agent system can be used to solve some of them and, potentially, improve both customer service and driver experience. We discuss the design and implementation of the system and illustrate (through a simulation) how application of the system supports fleet management and facilitates better driver workload balance / pay.**

I. Introduction

Let us consider management of a fleet of cars for hire. While, today, applications such as Uber [1] or Lyft [2] have introduced novel approaches to facilitating car hiring, let us focus on an old model of a *taxi corporation*. However, what should be kept in mind is that a number of issues that are going to be raised and addressed in what follows are applicable also to: MyTaxi [3], Uber, Lyft as well as future models of car hiring that will involve fleets of self-driving vehicles.

The oldest scenario of "acquiring a taxi" involves customers calling a central (phone) number, where someone receives details of an order and radio-broadcasts it to the drivers. The driver, who responds first, "gets the order" and proceeds to pick-up the passenger. Lack of ICT support results, among others, in: (a) unknown wait time, (b) lack of support in the case when the taxi, on the way to the customer, encountered some problems, (c) lack of knowledge how many taxis are already waiting for customers in a given area, (d) uneven distribution of rides (income) between the drivers, etc.

Currently, in majority of countries, mobile technology has been introduced to taxi corporations. As a result, customer has multiple channels to request a taxi (phone-call, SMS, web, etc.). Order is redirected to the drivers (often divided by the city section) and displayed on a screen of mobile devices (e.g. a smart phone or a phablet). Drivers can accept an order by pressing a button on the screen (action, which potentially redirects their focus from driving to the screen of the device). Drivers also know how many taxis are already awaiting orders in a given part of town. Often, drivers in a given region are divided into queues, to which they can sign into (if they know that they are going to that area). While a big improvement, such systems still do not fully address issues like (a), (b) or (d) above. Furthermore, they still require a lot of attention of the driver (taking their attention away from the driving – thus making it dangerous). Note also that, recently (May, 2016), Lyft announced that it will add information about expected wait time to their smartphone application (see, [4]).

Therefore, it would be nice to add functionality to the ICT-system of the taxi company. Some of the objectives of an "ultimate taxi corporation management system" may include (but, surely, more goals can be realized through the proposed solution): (1) Management of *which taxi should pick-up which passenger* should be semi-automatic and use multi-criteria analysis (including, among others, estimated time of arrival to the client location, current earnings distribution, etc.). In this way the driver would focus on driving, rather than on catching the next order. (2) Dealing (semi-autonomously – minimizing the involvement of the driver) with the *cases when unforeseen circumstances occur* during passenger pick-up, or driving the passenger to the destination (traffic jams, police controls, car breaking, accidents, etc.). (3) Providing extra *safety for the drivers* by informing that the car is not going to the established location (which may mean that the taxi was hijacked – in recent years, in Poland, there were multiple situations when taxis were hijacked and drivers injured or killed). (4) Reducing unnecessary disparities between payments of all drivers. While this goal may seem somewhat controversial, we believe that its realization could improve well-being of the whole corporation. Therefore, we show that, using the proposed system (even in its relatively simple form), this goal can be achieved (at least to a certain extent). However, in real-life, its use is not "mandatory". (5) Dealing with the situation when the client is "far away" and there are *no drivers willing to go there*. Here, the trust in the fact that the system will (over time) equalize earnings (and thus will make sure that the driver will not be loosing money by picking such customer) may be of value.

It should be obvious by now that what is needed is a system that involves autonomous entities representing drivers, allows for negotiations, monitoring of the situation, re-negotiations and/or re-scheduling in case of problems, etc. This seemed like a perfect fit for an agent system (see, for instance [5], [10]). Therefore, we have decided to design and implement one to see if it can actually deal with main issues involved in management of a taxi corporation.

The remaining parts of the paper are organized as follows. First, we outline the design of the *TaxiAgent* system. This includes the main use case scenarios. We follow with more

detailed description of the implemented system, while used technologies are also briefly described. Next, it is shown that the system actually works (it correctly deals with the considered scenarios). Finally, we present initial results of simulations focused on illustrating that the proposed system can have positive effect on leveling income (workload) of drivers.

II. Designing the *TaxiAgent* system – general considerations

Let us now consider what are the main issues that the design of the proposed *TaxiAgent* system should take into account. As stated above, we have decided to realize it as an agent system, in which a software agent will be "placed" in each taxi. Here, it is worthy noting, that mobile devices placed in taxis can be assumed to: (a) have an "uninterruptible" power source, and (b) if needed, be connected to the Internet (either directly, or through the GSM infrastructure). This removes some restrictions that had to be considered, for instance, in the glider pilot support system discussed in [7].

The second part of the system is going to be the central "module", which is going to possess complete knowledge about the "state of the corporation". This knowledge includes, but is not limited to: (1) taxis that are currently "in operation" (including their location, status – waiting, on the way to the client, on the way with the client, direction where they are going, estimated arrival time to the destination, earnings level – for a given day, week, month, etc.), (2) orders – waiting customers, their locations, time when they want a taxi to be available, (3) taxis that are on a break, but are likely to "be back" within limited time (possibly, expected time when a given taxi will be back servicing customers).

The third part of the system is the client "interface." Here, in a real-world system multi-channel interface would be required. As mentioned earlier, this should include, among others, WWW site, SMS, phone, and smart phone application. For the latter one, applications for Android and iOS should be created. However, for the testing purposes, we ave decided to develop only the Android-based user interface.

Let us now list scenarios that the proposed system should be capable of dealing with.

Basic scenario: Here, a customer orders a taxi. Systems establishes, which taxi should pick the customer (taking into account positions of taxis that are in the area, or that will enter the area shortly, possible arrival time, earnings distribution, etc.). The selected taxi is confirmed and the client is informed about pickup details (taxi number, car make, etc.). Order is monitored until customer is delivered to its destination.

Taxi cannot reach customer on time: In this case, the agent in the taxi informs the central system about nature of the problem (break, traffic jam, police control, accident, etc.). Note that, in this case, it is extremely likely that it will be the driver that will have to initiate the "emergency action" of the system. However, more complex scenarios, involving autonomous actions of the system (that recognizes existence of a problem), can also be envisioned. The central system finds and confirms the replacement taxi, and informs the customer. Next, basic scenario is resumed.

Taxi encounters problems when taking the customer to the destination: Here, traffic jam is a problem that cannot be solved. However, in case of other situations (e.g. taxi breaking, police control resulting in an extended delay, accident, etc.) the taxi agent arranges for another taxi to be sent to pick up the customer (from the location where the "taxi with problems" has stopped). Here, again, it is likely that the driver will initiate this scenario, but more complex behaviors of the system are also possible. After the new taxi picks the passenger, the basic scenario resumes.

Taxi monitoring: Since the system knows where the taxi is going, and where it is located, it is possible to monitor the situation and instantiate actions in a situation when the taxi is going in the "wrong direction". However, this scenario is out of scope of our current considerations.

Driver takes a break: Here, the taxi driver requests a break. During that time taxi will be skipped in the process of assigning clients. This functionality is useful in practice (taxi needs to be filled with petrol, driver needs coffee / food break, etc.). Since majority of drivers may want a break at the same time (and that it happens during the time when high demand for taxis materializes) the system should provide a possibility to "cancel" driver's break manually (by selecting appropriate option in the central module). This decision should be communicated to the driver, who should acknowledge acceptance.

Load/earnings balancing: One of the aims of the proposed system is to balance load, and thus driver earnings, across the corporation. To achieve this goal the system should select the "underutilized driver" to pick the next passenger, as long as it is reasonable. In other words, if two (or more) drivers can pick the customer within a given time, the one who earned the least should be selected.

More detailed description of implemented scenarios is enclosed in sections V and VI.

III. TaxiAgent – defining system modules

Let us now look into some more details of the three, just identified, main modules of the system. We will approach them from the point of view of the identified key scenarios.

Taxi driver application: This application is instantiated on a mobile device placed in a taxi (typically a separate – dedicated – device, other than the smart phone of the driver). It offers a map and a GPS navigation functions (delivered by the *TaxiRouteAgent*). Besides navigation, the driver can use it to report going on a break, road conditions (e.g. car break, traffic jam, police control, etc.), as well as request another taxi to be sent (see, above). An agent (called *TaxiCommunicationAgent*) that is responsible for communication with the central module is also a part of this application. Futhermore, each *TaxiCommunicationAgent* has its own status: occupied/free/on break, which is registered in the JADE's Yellow Pages service (see section IV). This tool provides an easy way to obtain a set of *TaxiCommunicationAgents* with the same status. Note that

the geopositioning can be implemented as a separate module within the *TaxiAgent* system (as done now). However, it could also be integrated with exiting driver-support software. Both approaches have advantages and disadvantages, but analyzing them is out of scope of current contribution.

In Figure 1, we see a map displayed on taxi driver application's screen, when a client is assigned. Picture of a car symbolizes current position of the taxi, icon of a human represents client's localization, and a flag – the desired destination.

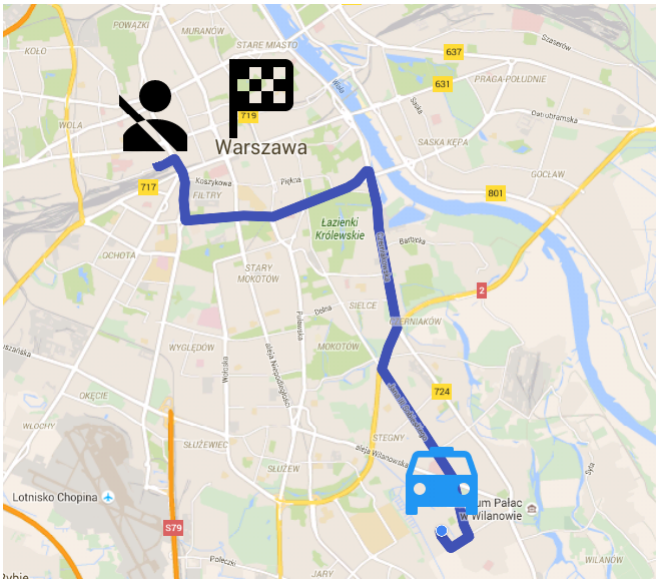

Fig. 1. Screen from taxi driver application with taxi's and client's position and client's destination marked

Central module: This core application works on a central server (consisting of one or more machines, depending on the size / needs of the company). Its main objectives are (a) choose the most suitable taxi for a client (and the driver) and (b) oversee that the order is completed. In order to achieve this, an agent (called an *AngelAgent*) is created for each order requesting a taxi. Every *AngelAgent* is responsible for one client – it is launched when a new order appears, and it works until client reaches her/his final destination. In case of an emergency, it is the *AngelAgent* that is responsible for taking all necessary steps to solve the problem. Besides the *AngelAgent*, the central module consists of a *ReceiveOrdersAgent* and a *DBAgent*. The *ReceiveOrdersAgent* receives orders from clients and creates new *AngelAgents* for each one of them, whereas the *DBAgent* is responsible for querying the database. The central module enables also monitoring of work of all elements of the system. Employees using the central module are able to read logs (which contain information about taxis and orders), manage negotiations when end of break(s) is needed, and add new taxi drivers to the system database.

Client application: This is an application, which is instantiated on a mobile device. It allows user to choose both, the start and the end location, and also the expected time of taxi's arrival. The location can be chosen by one of the following methods: (a) by using the current GPS position, (b) by selecting point on the map, or (c) by typing the address into the application. When a taxi is requested,*ClientAgent* sends message to central module and the order is farther processed. Next, customer receives information about expected time of arrival and can track the taxi on the map.

Emulator: During the process of implementation of the *TaxiAgent* system, an obvious need for testing the system arose. Thus far, it was impossible to try the system in the real-world, and thus the emulator module was created. This is a server application with a simple design. The emulator can generate clients, taxis and can simulate all scenarios described in section II except *Taxi monitoring*.

Map module: It is a website, which (as the emulator) was created to support testing of the system. All taxis and clients are displayed on the Google map. Whenever one of them changes location, the corresponding marker is moved. It is also possible to obtain more information about taxis and clients, for example: earnings, GPS location, status of the taxi (on break, occupied, free), start/end position of the client.

IV. *TaxiAgent* SYSTEM – USED TECHNOLOGIES

To achieve the requred functionalities we decided to use JADE (Java Agent Development Framework, see [8]), a multi-agent platform created by the Telecom Italia Lab. It provides agent abstraction, asynchronous communication, task execution and a Yellow Pages service. Furthermore, JADE is still being developed (version 4.4.0 released on 23.12.2015 was used for the implementation of the system reported here). Other technologies used in implementation were: (a) Android 4.0 for mobile applications, (b) Microsoft SQL Server, (c) GoogleMaps, (d) ASP.NET – only in the Map module. Note that iOS support is one of future objectives.

Android applications were equipped with the JADE platform, in order to communicate with the central module. However, due to issues with string conversion between different platforms (PC and Android) it was necessary to serialize messages' content to the JSON format.

Finally, the *TaxiRouteAgent* uses queries to GoogleMaps to obtain the best possible route.

V. CHECKING CORRECTNESS OF REALIZATION OF PROPOSED SCENARIOS

Let us now check if the implemented system works for the scenarios described in Section II.

A. Basic scenario

The messages exchange that should take place in the basic scenario, depicted as a sequence diagram, is shown in Figure 2.

Specifically, the process presented there consists of the following messages sent by participating agents (we assume that, at the start of the process, the taxi driver application is being turned on).

- During authentication of the taxi in the system, the *TaxiCommunicationAgent* sends message of type *Request* to the *DBAgent*. It contains a password provided by a taxi

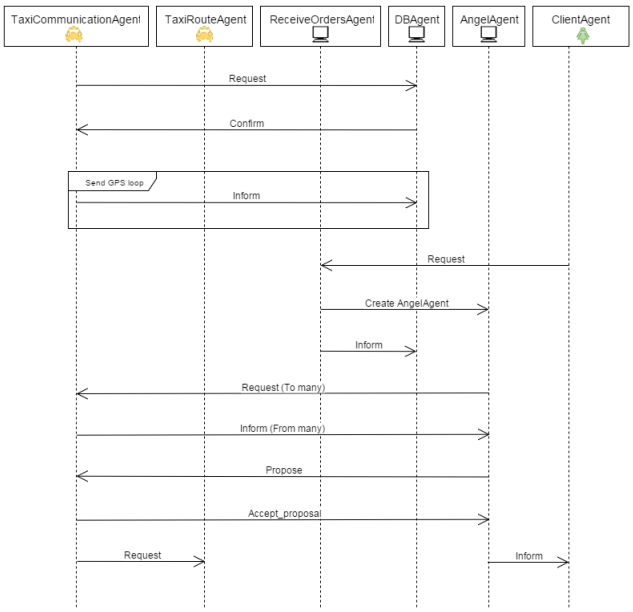

Fig. 2. Communication diagram – basic scenario

driver. The *DBAgent* checks if the password is correct and sends back message of type *Confirm*. Then, the driver starts working.

- Since the beginning of work, the *TaxiCommunicationAgent*, cyclically, sends messages of type *Inform*, which enclose taxi's location. The *DBAgent* writes this information into the database. In this way the system is aware of position of every taxi. Decision how often such message is to be sent is one of the parameters of the system.
- When a clients orders a taxi, the *ClientAgent* sends, to the *ReceiveOrdersAgent*, a *Request* message, which includes coordinates of the client and the specified destination. Then, the *ReceiveOrdersAgent*, creates a new *AngelAgent* and passes to it data sent by the client. Next, the *ReceiveOrdersAgent*, sends an *Inform* message to the *DBAgent*, which contains client's localization and destination.
- The *AngelAgent* checks the Yellow Pages service (provided by JADE), which taxi drivers are free and then sends a *Request* to all of them. The *TaxiCommunicationAgents* respond with an *Inform* message, which contains information about expected time to arrive to the client. In the future, it is planned to use more complex method of choosing taxis by the *AngelAgent* (e.g. checking occupied taxis, which are heading near client's location). However, it is out of scope of the system discussed here.
- The *AngelAgent* sends to the selected taxi the *Propose* message, which is an offer to fulfill the client's order. If the *TaxiCommunicationAgent* responds with the *Accept_proposal* message, the *AngelAgent* informs the *ClientAgent* about the estimated time of waiting for the taxi and the estimated cost.

Let us now illustrate how this scenario is realized when the system is running. To do this we have used the, JADE provided, *SnifferAgent*, which registers all messages exchanged between selected agents. The registered messages, from a sample run, are presented in Figure 3.

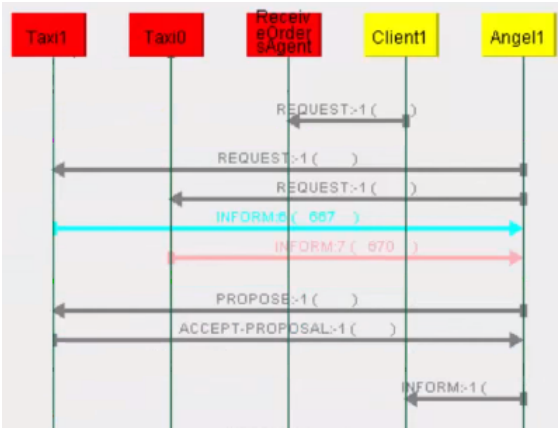

Fig. 3. Sniffer agent – basic scenario

It is easy to see that the message exchange pattern follows that, which was depicted in Figure 2. From this, and from a number of experiments that we have run, we can state that the system is capable of successfully realizing the basic scenario.

Taxi cannot reach customer on time

When a taxi cannot arrive to the client, (in the current version of the application) the driver presses the button on the screen, which invokes the *TaxiCommunicationAgent* to solve the problem. It informs the *AngelAgent* (assigned to this ride), about the need to find a replacement taxi. To avoid the situation when client would have to wait an extended period of time for a (replacement) taxi, the *AngelAgent* increases the "priority" of such customer. As a result the system will serve this client in the first place.

In Figure 4, a map-based screen-shot representing this scenario has been shown. Here, the broken-down taxi is marked in the circle, the waiting client is represented by a triangle. Finally, in a rectangle there is another, unoccupied taxi. In the Figure 5 further development of this situation may be observed. As it could be predicted, the *AngelAgent* selected the "taxi in the rectangle," and this taxi is now on the way to pick up the client.

Taxi encounters problems when driving customer to its destination: Similarly to the previous scenario, in this situation driver also informs the *AngelAgent* about the problem. Then it is responsibility of the *AngelAgent* to find new taxi, which will pick up the customer from place, where breakdown happened.

Again, to check how this scenario is realized we used the SnifferAgent. The registered messages, from a moment of sending a message to *AngelAgent* by *TaxiCommunicationAgent* are shown in Figure 3.

VI. Workload / pay balancing

As stated above, an additional feature of the proposed system is to facilitate a "fair distribution" of income between

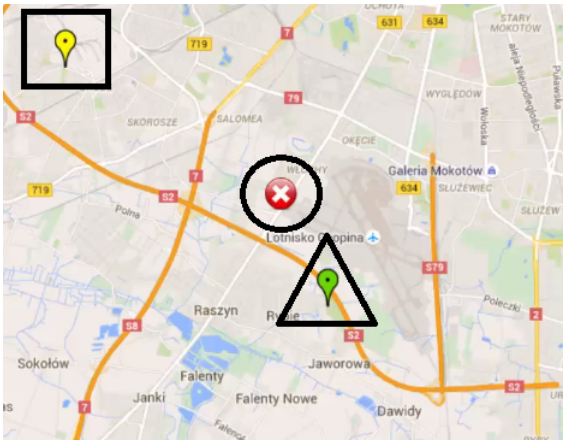

Fig. 4. Taxi cannot reach customer on time – beginning

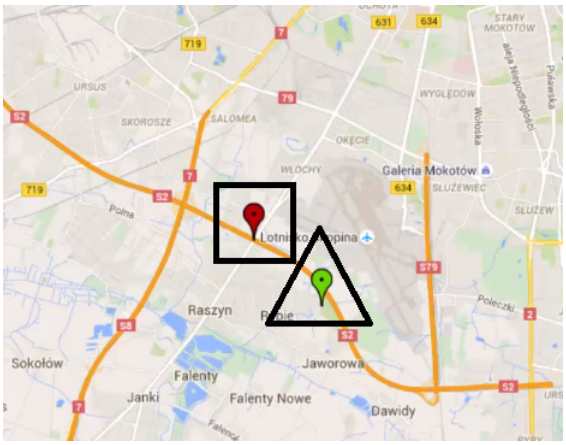

Fig. 5. Taxi cannot reach customer on time – development

drivers. While one can envision multiple approaches, of varying complexity, to reach this goal, we have decided to check ifthe agent system, described here, is capable of achieving it. Therefore, we have implemented the following algorithm.

1) Upon its creation, to oversee the order, the *AngelAgent* collects information from all unoccupied taxi drivers, about their predicted time of arrival to the client. List of free taxis is obtained from the Yellow Page service.
2) Then, it sends a query to the *DBAgent*, which provides information how much money each driver earned that day. Here, we assume that the money earned is proportional to the time that a given driver is already working. Obviously, a more complex approach is possible (in order to provide an even better balance of work-time, payment, distance driven, etc.).
3) If only one driver can reach the client in time, then it is chosen. However, if multiple taxis have an "acceptable arrival time", then the taxi with smallest income during that day is being chosen by the *AngelAgent*.

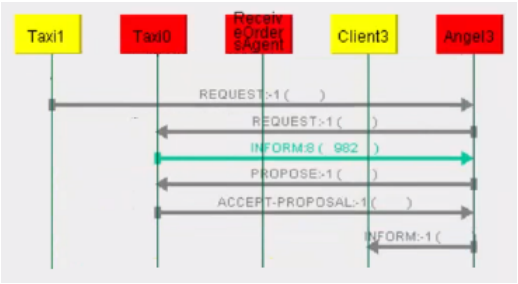

Fig. 6. Sniffer agent – basic scenario

Separately, we have considered situation of a client with an "unfavorable location" (e.g. suburbs). Such client could have to wait for a long time, or even will never be served, if all taxis were serving customers in a "central location". To deal with this situation, each client has an assigned priority. Initially, it is equal to one, but it increases in one of these cases: (a) if there is no free taxi, able to serve the client, (b) if the *AngelAgent* of that client chooses a taxi, but the taxi refuses the offer, (c) if the taxi does not reach the client due to some event (see, Section V). Here, a taxi is able to reject *AngelAgent*'s proposal, if in 10 seconds after receiving the first offer, an *AngelAgent* with higher priority of the contract, will ask the taxi to serve its client. Specifically, let us assume that the driver's *TaxiAgent* receives an offer from the *AngelAgent*. When this offer comes to the taxi, the *TaxiAgent* will wait for 10 seconds for offers, which have higher priority.

Recall that, in our approach, these two situations have to be combined with each other. Driver is more willing to go and pick up a passenger "away" if (s)he knows that the pay balancing system will help recover the lost time / income.

A. Pay balancing tests

To test our approach to pay balancing, two experiments were conducted. In the first, five hundred clients were created at random locations. Then, ten taxis were added, all in the same region. The goal of the test was to simulate work of the corporation as long as all clients would have been served. In Figure 7 and Figure 8 we depict incomes of taxi drivers with and without pay-balancing schema.

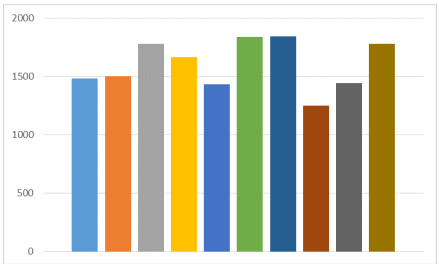

Fig. 7. Results of the first experiment without the income criterion

Here, the overall number of clients was much higher than the number of taxis. This resulted in the situation, where a taxi never has to wait for a new client. Specifically, when (in the early stage of the process) a taxi lost negotiations for a client, it (almost) immediately got involved in obtaining the next client.

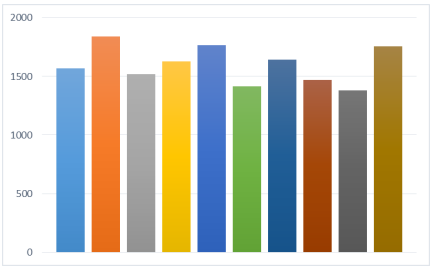

Fig. 8. Results of the first experiment with the income criterion

As the process continued, there was only one taxi left so it had to "win negotiations" and receive an order. Furthermore, as the taxi delivered the client to the destination, there were subsequent clients waiting to be served. Comparing the results in the two figures leads to a simple conclusion. When the number of taxis highly exceeds the number of clients, every driver earns similar amount of money – regardless of the used criterion for selecting the taxi.

In the second experiment we have been adding clients in a cyclic way. A new set of clients was created right after customers from previous groups were served. An important condition was to, each time, create a smaller number of clients than the total number of currently available taxis. Specifically, we have instantiated ten taxis and, in every iteration, seven new clients were added in random locations on the map. The test finished after fifteen iterations, which resulted in one hundred and five clients served. In Figure 9 and Figure 10 results of the experiment are shown.

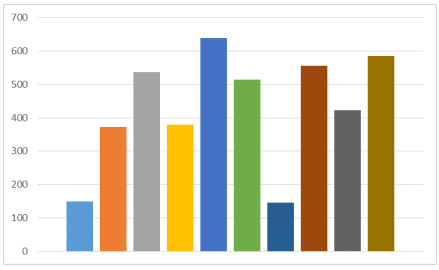

Fig. 9. Results of the second experiment without the income criterion

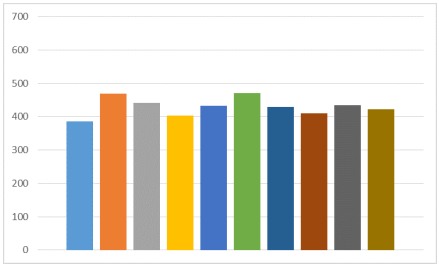

Fig. 10. Results of the second experiment with the income criterion

In the second experiment, where the number of taxi drivers was (constantly) greater than number of clients, winning negotiations was crucial to earn money. Here, without using the income criterion, the taxi which was the nearest to the client was winning the negotiations. As a result some drivers served more clients than other drivers and earned substantially more money. In the second case, with the income criterion, all drivers served a similar number of clients and their income was relatively similar. Note that similar results have been observed also in other tests run for this scenario.

VII. Concluding Remarks

In this paper, we have presented a multi-agent system that supports a TAXI corporation. The system has been implemented and tested and could be installed in an actual company. The presented solutions consists of two mobile applications – one for the driver and one for the client, a database and a server application. The server application architecture allows it to be placed on multiple computers at the same time, leading to potential performance improvement for large-scale usage. The server intermediates between the taxi driver and the system, receives orders from the clients and creates a new agent for every client. During the implementation, we have also created a website with a map and an emulating module that was used to test performance, the behavior of agents and the implement scenarios. In the proposed solution, we have included real life scenarios like taxi breakdown and taxi driver coffee-break. Furthermore, it was shown that the proposed multi-agent system, using a relatively simple workload balancing schema, may assign taxis to clients in such a way that every driver will earn similar amount of money, which could help companies to distribute income fairly. Obviously, the simplistic approach could be replaced by a more complex one, and larger scale tests run. The system is designed in such a way that the process of implementing the new features (i.e. fuel cost optimization, two-step planning, or driver rating) is simple, making its usage for the future promising.

References

[1] www.uber.com
[2] www.lyft.com
[3] www.mytaxi.com
[4] http://9to5mac.com/2016/05/23/ride-sharing-service-lyft-testing-scheduled-pickups-from-mobile-app/
[5] Perugini, D., et al. Agents in Logistics Planning-Experiences with the Coalition Agents Experiment Project. In Proceedings of workshop at the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003). Melburne, Australia, July 2003
[6] Zhu, K., A. Bos. Agent-based design of intermodal freight transportation systems. NECTAR Conference, Delft, 1999.
[7] A. Gab, P. Andreou, M. Ganzha, M. Paprzycki, GliderAgent a proposal for an agent-based glider pilot support system. In: Proceedings of the 15th International Conference on Methods and Models in Automation and Robotics (MMAR), IEEE Press, 2010, 55-60
[8] www.jade.tilab.com
[9] J. M. Salanova Grau, M. A. Estrada Romeu, Agent based modelling for simulating taxi services
[10] L. M. Martinez, G. H. A. Correia, J. M. Viegas. An agent-based simulation model to assess the impacts of introducing a shared-taxi system: an application to Lisbon (Portugal)
[11] J. M. Salanova Grau, Taxi services modeling for decision making support
[12] A. Glaschenko, A. Ivaschenko, G. Rzevski, P. Skobelev, Multi-Agent Real Time Scheduling System for Taxi Companies