Auctions Negotiation for Mobile Rideshare Service

Sameh Abdel-Naby
University of Trento - DIT
ARS-LOGICA IT Labs
Email: sameh@dit.unitn.it

Stefano Fante
University of Trento - DIT
ARS-LOGICA IT Labs
Email: stefano.fante@dit.unitn.it

Paolo Giorgini
University of Trento - DIT
Povo (TN), Italy.
Email: paolo.giorgini@unitn.it

***Abstract*—Rideshare systems allow a substantial number of people to mutually benefit from using less cars in a specific region. This would rationalize energy consumption, save money, and decrease traffic jams and pollution. However, accessibility issues have prevented these architectures from being widely spread. In this paper, we present an agent-based Rideshare system that is accessible via lightweight devices. We use auction mechanism as a method of negotiation among autonomous and proactive agents, by this we aim at accelerating agents' interactions while resolving end-user composite tasks.**

1. INTRODUCTION

Lightweight devices are increasingly showing their necessity and reliability. Cellular phones and similar devices are part of the new telecommunications era, which made it possible to virtually carry your office anywhere you go. Nowadays, people are using pocket devices that allow them to check their emails, exchange faxes, surf the internet, edit documents and do shopping. These services are provided through quite simple, user-friendly and well-developed interfaces, and are costless with respect to the value of services users are getting.

Rideshare is another service to be mobilized. The way a Rideshare system works is related to the availability of empty seats in a car and, the interest of a user to contribute in the journey cost in exchange of occupying this seat. The interactions made constitute reliable means of transportation for many people in an increasing number of countries, and it is usually provided exclusively by a third-party website that uses a web-based technique to match service requests.

In classical agent-based rideshare systems [7], users' desires are represented by delegated agents, and a super agent is available to match the service requests these agents are carrying. This earlier scenario is applying Multi-Agent System (MAS) techniques to form a Service Oriented Architecture (SOA). However, several contributions were made to the literature of Rideshare systems. These contributions has similar notions but different implementation approaches, besides, they are not reachable by holders of lightweight devices.

In this paper, we focus our research on enabling lightweight devices to smartly handle Rideshare service interactions using software agents and utilizing advanced communication capabilities. Although there are some restrictions that are given by users (e.g. time to achieve) and others given by the technology (e.g. Bluetooth data exchange rate), we apply an auction negotiation mechanism to increase the number of goals achieved and the system usability.

The paper proceeds as follows: Section II outlines our motivation. Section III explains the Rideshare framework. Section IV introduces the algorithms used to run the auction among system agents. In Section V we introduce in details the implemented system. Section VI concludes the paper.

2. MOTIVATING SCENARIO

There are some repetitive and related situations that occur in our daily life. Looking at one of them, we found that using a car to move from a place to another will increase the flexibility to schedule appointments, reliability and comfort. We also noticed that, on the long run, pollution and stress caused by traffic jams will badly affect our life.

If a tool is provided to match peoples' common interests, it would increase cooperation and simplify lots of our daily tasks. It is quite common to see a car owner that has empty seats and commuting daily between two fixed locations (e.g., house and work). It is also common to see commuters of public transportations taking the same route everyday, or people trying to move between different remote places in irregular times. In fact, average car occupancy in the UK in 2004 is reported to be 1.59 persons/car; 1.2 for commuters and 2.1 for holiday [13]; 38% of people traveling in a car in 2004 were unaccompanied drivers, 25% were drivers traveling with one or more passengers and 36% were passengers. And in Germany is even less [14].

For example, Bob and Alice are two potential system users that are located at the train station, while John is another potential user that is located at the bus stop nearby. Bob has arrived to the train station driving his car, but Alice and John do not have cars. All of them are coming from different locations and again moving to different ones, but it happens that John and Alice destinations are located on Bob's way to work. Moreover, the cost of taking public transportations to reach Alice and John destinations could be of need to Bob in exchange of offering them a ride. Bob is now able to use his cellular phone to offer a ride to John and Alice that were both using their smartphone or PDA to seek a ride. Intelligent software agents are now delegated by users to seek and offer car rides, then agree on a sharing cost, meeting points and times. According to the preconfigured level of interactivity, these agents would take decisions on behalf of users and communicate the final results.

In a classical agent-based Rideshare system, agents' behavior, level of interactivity and decision making schemes may

1-4244-0971-3/07/$25.00 ©2007 IEEE.

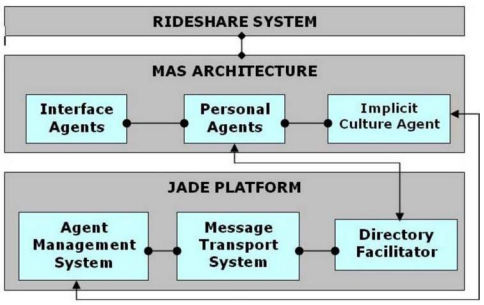

Fig. 1. The Three-layer Model

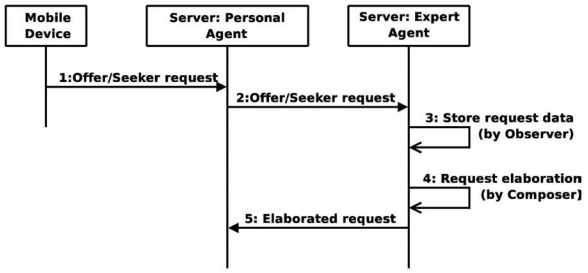

Fig. 2. User-Request Elaboration Process

lead to disagreement and imprecision. On the contrary, an auction mechanism can aid to resolve complex situations; this mechanism ensures ultimate benefits gaining for both service supplier and demander.

3. THE RIDESHARE FRAMEWORK

The architecture we implemented is offering users of lightweight devices a complete Rideshare service package, it consists of a several layers that are combined together to form the general architecture of the running MAS. This architecture is accessible via Bluetooth-enabled lightweight devices. We have chosen Bluetooth as a major communication method, yet users can use SMS, GPRS/UMTS or a Web interface to interact with the system.

Users of Andiamo can access the system through Bluetooth access points that are directly connected to the Multi-Agent platforms. For each access point we have an implemented Multi-agent system where Personal Agents (PA) of car owners and ride seekers interact and negotiate potential rides.

As shown in Fig. 1, our framework has three different layers.

1) The agent platform, which is based on JADE framework.

2) The Multi-Agent architecture including the Implicit Culture (IC) part.

3) The top layer is the handler of user requests and the interface line, which is the Rideshare service management method. Further on, we explain them in details.

The ***MAS Architecture Layer*** is implemented using JADE (Java Agent Development framework) [1], which is a FIPA-compliant [2] framework for MASs development. JADE provides: (1) an Agent Management System that allow us to create agent containers in distributed hosts, (2) a Directory Facilitator (DF) that provides a yellow-pages service, and (3) a Message Transport System that handles agents communications.

This architecture layer is responsible of receiving and processing users' requests. A one-to-one correspondence between agents and mobile devices is established throughout this phase. Each agent is identified by a unique Bluetooth address of the corresponding mobile device. It is possible for a single device to have multiple PA that are initiated through diverse access points. When a user request is received, the platform checks whether the PA of this specific device exists. If not, a new PA is created. Each PA communicates and interacts with other agents in the system in order to find "partners", the PA remains wandering until at least one user request is fulfilled.

*1) **Interface Agents**:* Interface Agents (IA) employ certain techniques to provide assistance to a user dealing with a particular computer application [6]. In Andiamo, each IA has its main features, which are: knowledge acquisition, autonomy and collaboration. At this phase, IAs are managing the creation of new service requests. These agents are recognized by the system as 'AddNewServiceAgent' and they receive from the preceding layer the service request including user details parameters (Bluetooth address, user information, and service request details). Later on, they transmit this data to the corresponding PA and they remain in the system only to achieve this action and then vanish.

A new agent is created for every service request. The transmission protocol between a PA and an IA is summarized in four main steps. 1) The NewService Request that is issued by the IA and directed to the PA. 2) The acknowledgment response from the PA. 3) NewUserInfo that is sent again from the IA to the PA. 4) The final "accept" message from the PA to the IA.

*2) **Personal Agent**:* These agents represent the system users and the work done on their behalf. The interactions of these agents include two main parts: (1) the elaboration of users' requests, and (2) the negotiation among agents. The PA that elaborates a request for a ride is called **Seeker Agent** (SA while a PA that elaborates a ride offer is called **Offerer Agent** (OA). The mechanism used for agents' negotiation will be explicitly illustrated in Section IV.

*3) **Agents Interactions**:* In Fig. 2 we present the interaction protocol used by agents during the request elaboration phase. On each platform there is a dedicated agent, called Expert Agent (EA), which contains the System for Implicit Culture Support (SICS) [5][8]. The SICS consists of three components, 1) the Observer, which uses a database of observations to store information about actions performed by users in different situations, 2) the Inductive Module, which analyzes the stored observations and applies data mining techniques to find a theory about the community culture, 3) the Composer, which exploits the observations and the theory in order to suggest actions in a given situation. In Andiamo, the use of the IC framework (Implicit Culture Agent, Fig. 1) is to let the system suggests the meeting points that are frequently used by other system users and observed by the system. More details about

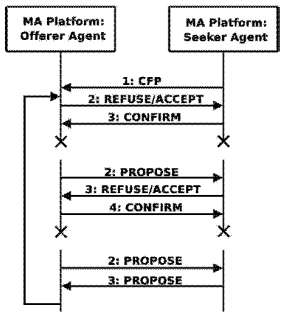

Fig. 3. A Typical Rideshare Transmission Protocol

the IC framework are available in [9].

After a PA receives its user's request (step 1), it sends it to the EA (step 2). On the EA side, an observer component of the SICS extracts data from the request and stores it in the database of user's observed behaviors (step 3). Composer component estimates the real value for parameters and if the input is incomplete or wrong (step 4). For the elaboration process, the Composer uses the information about the past user's actions, obtained from Observer and analyzed by Inductive module. Finally, the user's PA receives back the elaborated request (step 5), which it processes during the second phase.

SICS needs to gather information about users behavior. To observe user's behavior, EA extracts data from the requests it gets from the PA. Two other additional sources of observation could be added. The first is the database where the results of agent negotiations are stored. This storing takes place every time two PAs agree on sharing a trip and send their proposals to the database. The EA extracts necessary information (e.g., departure, arrival place and meeting points) from the proposals and stores them in its internal database. The second source is the user's feedback repository. When the ride is finished, the user gives feedback to the other user(s). His evaluation is stored in the mobile phone/PDA and is sent to the EA as soon as the user establishes connection with the corresponding server via his mobile device.

The interaction mechanism used in Andiamo is based on the following parameters of the trip: *Request Type, Passenger Type. Departure Time, Departure Date, Departure Place, Departure Meeting Points. Arrival Place, Arrival Meeting Points. Offset, User Feedback, minRequested/maxOffered Money (contribution for the ride) and Number of Seats.* This applies to OA as much as to a SA.

In Fig. 3 a typical Rideshare service transmission protocol is demonstrated. The Multi-Agent platform is located to serve the interactions between an OA and a SA. A sequence of steps is taken between both, offerer and seeker, in order to achieve a successful Rideshare agreement. In addition, the possibility to apply an automatic or a semi-automatic service mode implies that the interaction between two agents can be, either interrupted to prompt an inquiry to the user, or self-decision making. Following to that, we describe the significance of negotiation and we consider its automatic service model.

*4) **Service Publication**:* Within the JADE DF, every PA publishes its carried service requests. If these requests are recognized by the system and that PA is identified, then the service will be registered. The initial interaction starts when a SA tries to find a OA with similar destinations, day and time of departure and with a feedback greater than or equal to the desired value. Then, the SA will contact every OA found by the system and consequently, communicates with end-user the retrieved data. Notably, if the value of the feedback is less than the requested value, it is possible to contact back the user asking a permission to decrease the requested feedback value so an agreement could be reached. The same thing happens for the time and other similar parameters.

4. THE AUCTION-BASED NEGOTIATION

Given a set of lightweight devices that are capable of communicating a service request with central **Service Oriented Architecture** (SOA) via Distributed Access Points (DAPs), and given that these DAPs and central servers are providing mobile users with location-based service. Here, lightweight devices are tools to clarify users' preferences. When user inserts his offer/request, a particular configuration file is automatically created on the device side. Subsequently, when a Bluetooth connection is established with a server, the lightweight device links with the SOA and the file is transferred.

Eventually, a phase where system verification occurs is placed. The arrival of a new agent to the server side requires the running MAS to verify whether this agent is new and to be bootstrapped or it already exists and it means to update the behavior of a previously running agent. A group of autonomous agents that are delegated by several users to achieve varied tasks in different times is formed at the server side of the architecture. Given that some of the tasks to be achieved are complex and require agents coordination, thus a negotiation scenario that requires agent-to-many is established.

Algorithm 1 is the algorithm used on the OA side to invoke and manage a specific auctioning situation. From `line 1` to `line 3`, both OA variables, `bestValue` and `numLoop`, are initially set to `'0'`. In `line 4`, the OA requests the SA to start the auction by sending the value of the best offer previously obtained during the pre-offer session. From `line 5` to `line 7`, the OA waits to receive new offers from all involved SAs, and a `'val'` is created as a function to calculate the currently obtained best-offer-value.

From `line 8` to `line 10`, if the algorithm had its first round and a `'val'` is gained, the `'bestvalue'` in `line 1` is now updated with the value of `'val'` and the number of loops `'numLoop'` is incremented. Otherwise, since it is not the first loop, from `line 11` down to `line 19`, the OA checks whether the `'val'` function is increasing in comparison with the previously obtained best value or not. If `'val'` is greater, the value obtained from the concerned SA is communicated with other SAs and, they are asked to communicate new offer if applicable, then the algorithm is restarted, `line 14` and `line 15`. If the `'val'` is less or equal to the best value previously obtained, the auction is

```
Offerer_Agent_procedure()
1:  bestValue = 0;
2:  numLoop = 0;
3:  auctionIsOpen = true;
4:  askSAToStartAuction(bestPreOffer);
5:  while (auctionIsOpen) do
6:    waitForOffers();
7:    val = calculateBestValueOfFunction();
8:    if (numLoop == 0) then
9:      bestValue = val;
10:     numLoop++;
11:   else
12:     if (val > bestValue) then
13:       bestValue = val;
14:       requireNewOfferToSeekers(bestValue);
15:       numLoop++;
16:     else
17:       if (val <= bestValue) then
18:       quitAuction();
19:       informWinners();
20: end while
21: quitAuction();
```

Algorithm 1 The procedure taken by the Offerer Agents.

```
Seeker_Agent_procedure()
1:  SABestOffer = 0;
2:  sent = false;
3:  while(auctionIsOpen) do
4:    sent = false;
5:    bestOffer = waitForRequest(bestPreOffer);
6:    decision = decideIfAcceptOrRefuse();
7:    if (decision == accept) then
8:      while(modificationsArePossible && !sent) do
9:        newVal = reviewParameters();
10:       if (newVal>SABestOffer&&newVal>bestOffer) then
11:         SABestOffer = newVal;
12:         sendOffer(SABestOffer);
13:         sent = true;
14:       if (!sent && !modificationsArePossible) then
15:         sendOffer(SABestOffer);
16:     end while
17:   else
18:     if (decision == refuse) then
19:       quitAuction();
20: end while
21: quitAuction();
```

Algorithm 2 The procedure taken by the Seeker Agents.

suspended and the best-bid SA wins (the `'bestValue'`), `line 17` to `line 19`. Finally, the algorithm terminates and the auction scenario is ended, `line 20` and `21`. Later to that, we explain the SA behavior in response to OA.

Algorithm 2 is used on the SA side to determine the significance of its role in the impending auction. In `line 1` and `line 2`, a variable `'SABestOffer'` that carries the SA best offer value is created and set to `'0'`. A variable `'sent'` is initially set to `'false'` and it changes to `'true'` only after a SA has communicated his offer. From `line 3` to `line 6`, SA holds its offer transfer until a communication was received from the OA asking for an auction participation. The SA puts the results from the evaluation function into the `'decision'` variable. From `line 7` to `line 9`, if the SA accepts the call for auction, a self-revision for the holding parameters is made. This revision refers to the SA insistent to obtain the auctioned item; therefore, it is made with the intention to show extra negotiation flexibility. The part from `line 10` and down to `line 13` refers to the comparison made by the SA to put together the newly obtained value and the existing one. If the new value obtained is greater than the previous one and greater than the `'bestOffer'`, the future offered value `'SABestOffer'` is set to be new one `'newVal'`, and the offer is sent to the concerned OA.
`Line 14` to `line 16`, if the self-revision made by the SA has yielded a disappointing result and the value gained is the same as the previous one, this specific SA does not send the previous value if `'modificationArePossible'` is `'true'`. The SA continues to review the carried parameters until `'modificationArePossible'` becomes `'false'` or it communicates new best offer. If `'modificationArePossible'` stays on `'false'` and parameters are not sent, SA communicates same offer. However, from `line 17` to `19`, if SA refuses the auction call, the algorithm terminates. If the user has an inflexible behavior, the algorithm passes the first condition on `if (decision == accept)` but the successive `while (modificationArePossible && !sent)` return `'false'`. The method `'decideIfAcceptOrRefuse'` return `'refuse'` if for instance, a SA has a lot of time before the deadline to achieve the task; therefore, it postpones auction participation. Finally, the algorithm is terminated and the auction scenario is ended, `line 20` and `21`.

Auctioning among agents requires high level of agent-to-user interactivity and increased level of network resources consumption; therefore, agents' intelligence appears when a repetitive scenario occurs. If system user is configuring the mobile-based application to repeat the same service request on daily or weekly basis (e.g., common in mobile news exchange service or carpooling), the created demanding agent would participate in system auctions only if needed. Once an agreement is settled between a specific supplier and a demander at a certain price, the next time this demander agent will first look-up the very exact supplier agent, which has potential agreement than others in early agreement. This is due to learning agent behavior that maintains an array that saves last successful agreement details.

5. RIDESHARE SYSTEM LAYER

In this section we describe the general architecture used for our service delivery. We start from system requirements to the various sub-components and their interaction. The architecture is obtained by extending and customizing ToothAgent [3] [4] Used-Books offering system that is able to communicate with mobile users through a Bluetooth connection and exchange useful information corresponding to a student's interests located in a university. Applying the ToothAgent architecture in our service model makes the centralized servers offer the

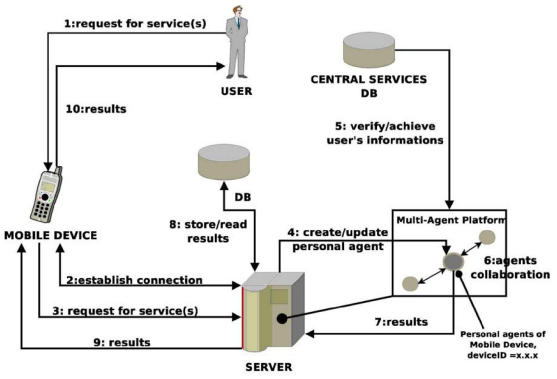

Fig. 4. Mobile-to-Service Accessibility Scenario

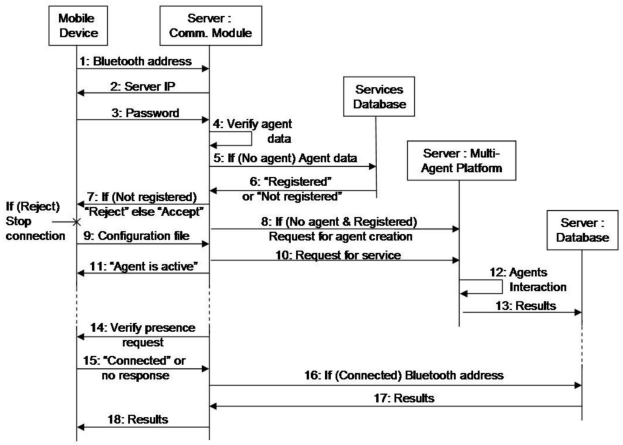

Fig. 5. Service Accessibility

Rideshare service instead.

5.1 SystemComponents

- **The mobile device** communicates the user's requests with the servers and receives the results.
- **The distributed servers**. Each of them contains: 1) a Multi-Agent platform with PAs each of which is representing a single user, 2) a database where results are archived, 3) an interface responsible for establishing connections with mobile devices and for redirecting the users' requests to the corresponding personal agents.
- **The central services database**, accessible via web. It contains information about all the servers and their properties, such as name, location, etc. The DB also stores the information about users registered to the system.

Fig. 4 illustrates the general architecture of the system and the interaction among its components. The connection between the mobile device and the server is established through Bluetooth but, as we say in Section II, it can be established also through SMS or GPRS/UMTS. In particular, a user's cellular phone communicates to the server all the requests, and then receives back the results. The cellular phone may also receive inquiries about a possible modification in the decisions taken by system users and re-communicate the reply with server. Moreover, cellular phone can be used to send the partner evaluation score, which will be reflected in the future feedback value for whoever will offer/seek a ride. From the server side, a contact is made to the central service DB to check the user's information (age, feedback, etc). Later on, the server updates the offerers and seekers reputation value. The central DB is responsible for storing all the information related to specific service request and the interactions made by its two PAs.

5.2 ServiceAccessibility

Three steps for a user to access the services: (1) to complete a mobile-based identification form; (2) to run the Bluetooth application on the mobile device, and (3) to operate a certain function to activate the required service. The application is written in Java and uses JSR-82 [11] and JSR-120 [12] which are the Bluetooth and the Wireless Messaging API for Java. The application starts a continuous search for Bluetooth-enabled devices in the neighborhood and whenever it finds a server, the software on the device establishes a connection with the server (step 2) and sends the requests related to the Rideshare services (step 3). The request is then processed by the server and the results are sent back to the user (step 4 -10). The mobile device stores the server's address to keep track of the contacted servers (see Fig. 4).

Fig. 5 shows the protocol we use for the interaction between different components. A specific communication module on the server is responsible for managing the interaction with the mobile device. This module receives the Bluetooth address and the password from the mobile device (steps 1 and 3) and checks in the platform running on the server whether a PA is assigned to that mobile device (step 4). The module employs the Bluetooth address and the password to map the mobile device with a specific PA. If there is no PA previously assigned to this user, the communication module connects to the central services database and verifies whether the user is registered to the system (steps 5–6). In case of a positive response, it creates a new agent and assigns it to the mobile device user (step 8). Then, the mobile device sends the configuration file to the communication module (step 9), which forwards all the user requests to the appropriate PA (step 10).
The PA then starts interacting with other agents on the platform trying to satisfy all the user requests (step 12). In our example a PA receives one or more requests for finding or asking rides. If the agent reaches an agreement with another agent about their users requests, it stores the results locally in the server database (step 13). Later the results could be sent back to the user (steps 14–18).

5.3 PendingResultsRetrieval

When a connection between a server and a mobile device is established, the communication module sends to the mobile device the IP-address of the server (step 2 in Fig. 6). The mobile device stores the IP addresses of all the visited servers in an XML file that is used later on to retrieve all pending

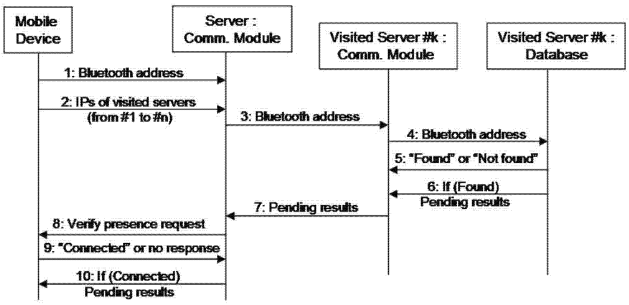

Fig. 6. Pending Results Retrieval

results. The format of the results produced by the PA may contain the request identifier, contacts (e.g. phone number) of the users interested to share the ride, the departure time, etc. If the user doesn't want to use the SMS service, he/she receives the results immediately in his/her mobile device, but only if he/she is still within the Bluetooth server range. In this case, the communication module checks the availability of the mobile device and sends across the results stored in the internal server database by the corresponding PA.

Fig. 6 shows the interaction protocol of retrieving pending results via mobile device. Considering our running example, a situation in which a user is close to the server of the train station. After establishing the connection, the mobile device sends the list of IP-addresses of all the previously visited servers (e.g. university servers, city center servers, etc.) to the train station server. The communication module of the server sends the Bluetooth MAC address of the mobile device to all listed servers (step 3). In turn, the communication module of each server extracts from the internal database all the stored results related to that user and sends them back to the requester server (steps 4–7). All the results are collected by the communication module and finally sent to the mobile device (steps 8–10). If the mobile device is no longer connected to the server (e.g., the user has left the train station), the retrieval process will fail and the results will be cancelled. Yet these results will still be accessible via the original servers. Therefore, as we already said, a possibility for the server to communicate with the user through SMSs is achievable.

5.4 ExperimentFacts

We tested the system using Nokia 6630, N73, N70, 6600, Motorola v3 and Sony-Ericsson P910 mobile phones and PC/Server equipped with generic Bluetooth adapter. Bluetooth communications have been implemented using BlueCove [10] which is an open source implementation of the JSR-82 Bluetooth API for Java. We have tested the system on different scenarios, and in those circumstances we have involved a number of people (students of the university, workers and citizen). From these tests, we have obtained significant results which were stored as reference. We notice the time to obtain an agreement between two agents is the same for every situation. In the cases in which there are more seekers than the seats offered by the offerers, the agents winning the auction are always the stronger agents (i.e. the agents that offers much money, that have an higher feedback, etc.). A limitation of this model, however, is the lack of a monitoring process of the number of active agents in single MAS.

6. CONCLUSION

Negotiation among agents that are serving computer based applications differs from this used for lightweight devices. We are rapidly approaching the era of lightweight services and a great focus and an immediate redirection is realized towards the achievement of cooperative agents in mobile-based service architectures. In this paper we presented an implemented application of a Mobile-based Rideshare service application where Multi-Agent system and Bluetooth and other wireless communication technologies are combined to support co-localized communities of users. We discussed the architecture of the Multi-Agent platform applied for our system, the specific protocols used and the algorithms that have been implemented to realize the Agents interaction. We recommend a verification process of system scalability before real-life use and a deep test of its performance with a considerably high number of users and for a long period of time.

ACKNOWLEDGMENT

Partially involved projects are: PAT (UNIQUIQUE SUUM, MOSTRO, STAMPS), EU-SERENITY and PRIN-MEnSA.

REFERENCES

[1] Fabio Bellifemine and Giovanni Caire and Dominic Greenwood, *Developing Multi-Agent Systems with JADE.* Wiley, February 2007.
[2] OBrien, P and Nicol, R, *FIPA - Towards a Standard for Software Agents.* In: BT Technology Journal, Vol.16:3, pages 51-59, 1998.
[3] Bryl Volha, Giorgini Paolo, Fante Stefano, *Toothagent: a Multi-Agent System for Virtual Communities Support*. In the Proceedings of the eighth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2006), 2006.
[4] Bryl Volha, Giorgini Paolo, Fante Stefano, *An Implemented Prototype of Bluetooth-based Multi-Agent System*. In the Proceedings of WOA 2005: 6th AI*IA/TABOO Joint Workshop "From Objects to Agents", 2005
[5] Alexander Birukov and Enrico Blanzieri and Paolo Giorgini, *Implicit: An Agent-Based Recommendation System for Web Search*, In the Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems, 2005.
[6] P. Maes and R. Kozierok, *Learning Interface Agent*, The Eleventh National Conference on Artificial Intelligence, Washington D.C., MIT Press, 1993.
[7] Amit B. Kothari, *Genghis - A Multiagent Carpooling System*, B.Sc. Dissertation work, submitted to the University of Bath, May 11, 2004.
[8] E. Blanzieri and P. Giorgini and P.Massa and S. Recla, *Information access in Implicit Culture Framework*, In the Proceedings (on line) of the ACM SIGIR Workshop on Recommender Systems, 2001.
[9] A. Birukou and E. Blanzieri and V. D'Andrea and P. Giorgini and N. Kokash and A. Modena, *IC-Service: A Service-Oriented Approach to the Development of Recommendation*, In the Proceedings of the 22nd Annual ACM Symposium on Applied Computing ACM Press, 2007.
[10] http://sourceforge.net/projects/bluecove/
[11] http://www.jcp.org/en/jsr/detail?id=82
[12] http://www.jcp.org/en/jsr/detail?id=120
[13] *Other factors affecting travel*, UK department of transport, http://www.dft.gov.uk/pgr/statistics/datatablespublications/personal/mainresults/nts2004/sectionsixotherfactorsaffect5253
[14] *Dynamishe Fahrgemeinschaften*. eNotions, http://www.m21-portal.de/Verkehrsbereiche/detail.php?detail=/projekte/2005_08_16_15_28.php