UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

ANDIAMO: A Multiagent System to Provide a Mobile-based Rideshare Service.

Francesco Sottini and Sameh Abdel-Naby and Paolo Giorgini

December 2006

Technical Report # DIT-06-097

ANDIAMO: A Multiagent System to Provide a Mobile-based Rideshare Service

Francesco Sottini, Sameh Abdel-Naby, and Paolo Giorgini

DIT - University of Trento
{francesco.sottini, sameh, paolo.giorgini}@dit.unitn.it

Abstract. A diverse range of architectures and concepts has been proposed by scholars within the theme of Car Pooling. Most of these studies have attempted to tie together two major elements: the need for people to move from a place to another, and the resources used to accomplish this action. Based on the use of location and available car seats, Rideshare systems allow a substantial number of people to share car rides. These systems would, among other advantages, rationalize energy consumption, save money, and decrease traffic jams and human stress, and eventually make a significant improvement in human life. However, system accessibility has prevented these architectures from being widely reached. In this paper, we present an agent-based Rideshare system that is accessible via lightweight devices (mobile phones and PDAs), where Bluetooth technology is adopted to reflect users' locality. The paper illustrates the overall infrastructure of the system, the specific protocols it uses, and the algorithms that it implements to recognize the Multiagent System (MAS) negotiations.

Keywords: Multi-agent Systems, Carpooling, Mobile Services, Bluetooth, Lightweight Devices.

1 INTRODUCTION

Rideshare is a method to reduce the use of cars in a specific town or area. Reducing car usage helps in turn to decrease pollution and prevent some other related problems. This usually takes place by having a car owner who uses his/her car to move from a place to another, and another person who is interested to go somewhere along the car owner's way to destination, and at the same time the ride seeker is willing to share the ride cost with the car owner. This Rideshare interactions constitute a reliable means of transportation for many people in an increasing number of countries, and it is usually managed and organized through a third party website that uses a web-based technique to match requests. Consequently, different research activities were carried out by governments in different parts of the world to encourage mobility in general and Rideshare services in particular. The key motivations behind providing this service is usually saving the cost of transportation, reducing the pollution coming out of cars, and avoiding the formation of traffic jams and the waste of time.

In addition to the necessity to be always connected to the internet, web-based Rideshare applications have other several disadvantages (e.g., privacy and preferences matching). Therefore, ease-of-use is only realized in using mobile based applications. Lightweight devices such as PDA and Cellular phones are ubiquitous and their uses has recently extended traditional communications to the so-called Mobile Virtual Communities [1] which facilitate the collaboration and the information exchange among mobile users that are physically located in one environment or remotely connected. Moreover, such communities are flexible to allow new users to join and existing ones to leave.

A number of mobile-based Multiagent applications have been proposed in the literature. MobiAgent [2] is a framework that uses Agents to allow users to access various types of services, ranging from web search to remote applications control, directly from their mobile phones or PDAs. Once the user sends a request for a specific service, a Personal Agent (PA) is created in a centralized server, then the user can disconnect from the network while the agent continues to work on his/her behalf. Afterward, the user is notified via SMS that some results are available and consequently the user reconnects to the network and downloads the results. The theory of Multiagent systems (MASs) has already been applied in the area of transports management and traffic reduction.

Remarkably, the areas of major influence for MAS are in the analysis and description of traffic systems, which increase the autonomy of traffic components and facilitate the integration of several frameworks [3], (e.g., an emergency rescue management centre that links up accident, pollution and decision support modules). In classical web-based Rideshare management systems [4], users are represented by agents and a super-agent is responsible to match user's requests, but the final decision is yet taken by users.

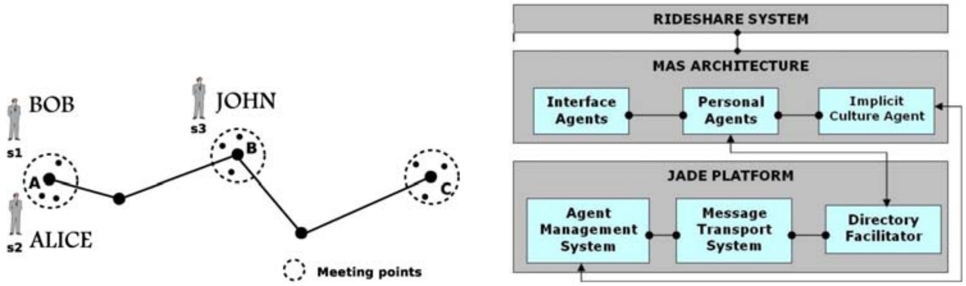
This paper presents an agent-based framework that implements a Rideshare service and uses the ToothAgent [5] architecture, where an infrastructure of a Multiagent application is accessible via lightweight devices that are Bluetooth-enabled.

This paper proceeds as follows: Section 2 outlines our motivation by explaining a running example. Section 3 explains the general framework of applying a Rideshare service using MAS architecture. Section 4 introduces the implemented system architecture and explains it in detail. Section 5 concludes the paper.

2 A RUNNING EXAMPLE

We illustrate our motivation through a scenario that briefly outlines the importance of Rideshare service for lightweight devices users. In Figure.1/a; There are three different locations (or meeting points) that are connected and easily reachable by cars. Passengers and car drivers are randomly distributed along these locations, and both parties share common interests (i.e., desired destinations, common commuting times and locations).

In this scenario, The first location is the **Train Station** (A), second one is the **Bus Stop** (B) and the **University** (C). **BOB** and **ALICE** are two



(a) First Figure

(b) Second Figure

Fig. 1. These two figures show the Rideshare Motivation Scenario and The Three-layer Model

persons who are located in **Train Station**, while **JOHN** is another person located in **Bus Stop**. These persons have several interests that lead them to end at the same final destination, **University**. If **BOB** is the only car-owner, thus using **BOB**'s car might be the appropriate means of transportation for the three of them, especially if a mobile-based proper Rideshare system is used that allows all of them to reach an agreement on-the-fly. Accordingly, **BOB** may put some conditions on sharing his/her ride with others, in addition to the cost that he/she will share with the others, while **ALICE** and **JOHN** can simply reach their destination.

According to the above scenario, several problems are practically critical: the computational determination of the destinations and the ways taken to reach them, distances and their estimated instances, time-to-wait for the ride to be achieved, and time-to-spend on the ride itself.

3 THE RIDESHARE FRAMEWORK

Within a Rideshare management system, major parts are achieved through the negotiation process of agents. Autonomous agents that take part of a Multi-agent system, which is serving a mobile-based application, are well-known for their capabilities to achieve complex organization and negotiation tasks within a particular community. Therefore, the application of autonomous agents in our system is essential.

The general architecture of a Rideshare management system consists of a number of multi-agent platforms accessible via Bluetooth-enabled lightweight devices. The user can access the system from a number of Bluetooth access points directly connected to the Multiagent platforms. For each access point we have an implemented Multiagent system where *Personal Agents (PA)* of car owners and ride seekers interact and negotiate potential rides. Moreover, the Rideshare framework is conceived as a layered approach that allows the implementation of a tangible Rideshare system. As shown in Figure.1/b the model has three layers:

(1) an agent platform (JADE); (2) Multiagent architecture with the Implicit Culture module; and (3) the superior layer that includes the Rideshare system.

3.1 THE MAS ARCHITECTURE LAYER

The MAS is implemented in JADE (Java Agent Development framework) [6] a FIPA-compliant [7] framework for multi-agent systems development. JADE provides: (1) an Agent Management System (AMS) that allows for having agent containers in different hosts (distribution), (2) a Directory Facilitator (DF) that provides a yellow-page service, and (3) a Message Transport System (MTS) that supports the communication among agents. This part of the architecture is responsible for the lightweight devices, receive and process a user's requests. There is a one-to-one correspondence between agents and mobile devices (users). An agent is identified by a unique Bluetooth address of the corresponding mobile device. The same device may have many PAs within different platforms and through diverse access points. When the user request is received, the platform checks whether the PA of the device exists. If not, a new PA is created. Each PA communicates and interacts with other agents in order to find "partners" and the PA remains until at least one request to satisfy is found.

INTERFACE AGENTS (IA). Interface Agents are computer programs that employ certain techniques to provide assistance to a user dealing with a particular computer application [8]. In our case, each of the Interface Agents (IA) that is implemented in our framework has its main three features: knowledge acquisition, autonomy and collaboration. That will make IA within this platform phase manage the creation of new services. These agents are named "AddNewServiceAgent", and they receive from the preceding layer the service request and user details parameters regarding a requested trip (Bluetooth address, user information, request parameters), then transmit this data to the corresponding PA. This IA remains in the overall system only to achieve this action and then vanishes. Shortly, a new agent is created for every service request. At this point, we consider that an Interface Agent will be responsible of adding a new service request to the overall system, the transmission protocol between a Personal Agent and an Interface Agent is summarized in four main steps. 1) The NewService Request that is issued by the Interface Agent and directed to the personal agent. 2) The acknowledgment response from the Personal Agent. 3) NewUserInfo that is sent again from the Interface Agent to the Personal Agent. 4) The final "accept" message from the Personal Agent to the Interface Agent.

PERSONAL AGENTS (PA). This Agents type is considered to be the kernel of the Rideshare service. These Agents represent the system users and the work done on their behalf. The interaction among these agents includes two main parts: (1) the elaboration of user's requests, and (2) the negotiation among agents. Afterwards, a PA that elaborates a request for a ride is called a **Passenger** while a PA that elaborates a ride offer is called a **Driver**.

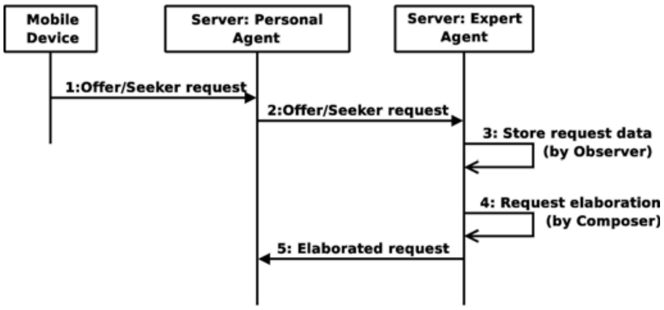


Fig. 2. User-Request Elaboration Process

The initial step in this phase of the architecture ensures that the request is elaborated and detailed. Looking back to our motivation scenario (Figure.1/a), the user **ALICE** can initiate a request *Ï’m asking for a ride from Train Station to Bus Stop*¨ without indicating any exact meeting point that could be close to the Train Station, which make the request incomplete. Thus, the PA makes the request clearer by using the information about what meeting points other users have recently indicated for the same location. Another scenario may occur: a request by user **BOB** could be *”I would like to serve passengers of a feedback value of 100”* and, at the same time, no users in the system have a feedback value of 100. In this case, the PA should reduce the feedback value, asking for the system user permission.

In Figure.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) [9]. The SICS consists of three components: the Observer, which uses a database of observations to store information about actions performed by users in different situations; the Inductive Module, which analyzes the stored observations and applies data mining techniques to find a theory about the community culture; the Composer, which exploits the observations and the theory in order to suggest actions in a given situation (*Explained more further ahead*). After a PA receives its user’s request (step 1), it sends it to the Expert Agent (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 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. In the end the user’s PA receives back the elaborated request (step 5), which it processes during the second phase.

Finally, the SICS needs to gather information about the users behavior. In the described system to observe user’s behavior, Expert Agent (EA) extracts data from the requests it gets from the Personal Agent (PA). Two other additional sources of observation could be added. The first is the database where the results

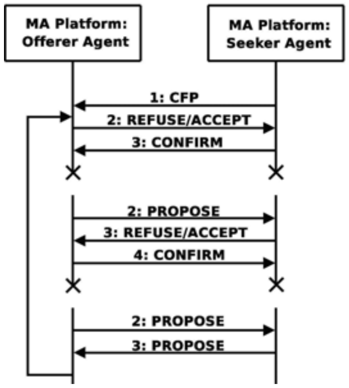


Fig. 3. A Typical Rideshare Transmission Protocol

of agent negotiations are stored. This storing takes place every time two personal agents agree on sharing a trip and send their proposals to the database. The Expert Agents 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. More to the point, the system assumes that only if the potential partner is contacted the feedback is positive, otherwise it is negative. The feedback information is sent to the Expert Agent as soon as the user establishes connection with the corresponding server via his mobile device.

The second step taken in this phase is concerned with the interaction mechanism that is based on the following parameters of the trip, which applies to a **Driver** as much as to a **Passenger**: *Request Type, Departure Time, Departure Date. Departure Place, Passenger Type, System Type. Arrival Place, Offset, Departure Meeting Point. User Feedback, Persons Number, Arrival Meeting Point.*

In Figure.3 we demonstrate the typical transmission protocol used for the Rideshare service, where the Multiaget platform is located to server the interactions between an **`Offerer Agent`** (i.e. Ride Giver), and a **`Seeker Agent`** (i.e. Ride Seeker). A sequence of steps are taken between both, giver and seeker, in order to achieve a successful Rideshare (e.g. Refuse/Accept, Propose, Re-Propose and Confirm).

The next part illustrates the interaction process phases made between two agents to reach a trip/ride agreement, taking into consideration that not only one linear decisional process but also a sequence of negotiation phases is available, as if each sequence is related to a group of parameters. 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-organized. Following that, we describe the significance of negotiations and we consider its automatic service model.

- **Service Publication.** In the JADE Directory Facilitator (DF), every PA publishes its carried service requests. If these requests are recognized by the system and that PA is identified with a particular ride giver, then the service will be registered as follow:

KIND =	Offer-ride
ID Combination =	Driver-departurePlace-arrivalPlace-dayOfMonth
EXTRAS =	Name = Feedback
	Value = valueOfFeedback

While in the passenger side, and during the creation of system pairs, the service registration will take this shape:

KIND =	Request-ride
ID Combination =	Passenger-departurePlace-arrivalPlace-dayOfMonth

The initial interaction phase starts from a **Passenger** who is trying to find, within the DF, a **Driver** with the same way on the same day and with a feedback greater than or equal to the requested value. Then, the **Passenger** will contact every **Driver** that is found by the system and, consequently, communicates the available feedback. 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.

- **Negotiation for Departure Time.** This negotiation phase tries to find a common departure time between a **Driver** and a **Passenger**. Figure.4/a shows the algorithm used to form the ride giver interactions, considering that the ride seeker part is just different in the steps taken to send and receive commands. Starting from row 8, **BOB** checks if the associated ride seeker have fulfilled his predefined requirements, if the checking results is "Yes", row 9 takes place by letting **BOB** check if **ALICE** is fitting somewhere in his range, if "Yes" Acceptance is initiated at this time; otherwise **BOB** stops the interaction because **ALICE** is not compatible. Row 16, **BOB** checks whether his proposal is accepted by **ALICE**. Row 19, **BOB** checks if his time value is greater or less than the proposed time; if it is greater, (20-24) it has to increase its value with an arbitrary step otherwise decrease (30-34). Row 25 as well as row 35, if **BOB** proposes a time that is close to **ALICE**'s time value, **BOB** accepts the proposed time, otherwise creates a new proposal. The algorithm has to terminate when the same value is sent or received for two times.
- **User Type Negotiation.** Every user has indicated a list of preferred partners types into the service request; this interaction tries to verify whether the located partner is compatible with the pre-defined list. Figure.4/b shows the interaction algorithm used in the ride giver side. The algorithm is divided in to two main parts, from line 4 to 10, the ride giver (**BOB**) tries to check if the partner's types are in his preference list. If the located partner is satisfactory (line 11), then it automatically starts the second part of the

```
1: temp ←proposal {departure time proposed by the offerer}
2: response ←0 {departure time responded by the seeker}
3: find ←false
4: repeat
5:    $send\_proposal(temp)$
6:    old_resp ←response
7:    response ←$receive\_response()$
8:    if $old\_resp = response$ then {check if the seeker has already reached the limit}
9:       if $(proposal - offset) \leq response \leq (proposal + offset)$ then
10:         $accept(response)$
11:         find ←true
12:      else
13:         return NULL
14:      end if
15:   else
16:      if $temp = response$ then
17:         $accept(response)$
18:         find ←true
19:      else if $response > temp$ then
20:         if $(temp + step) \geq (proposal + offset)$ then
21:            temp ←proposal + offset
22:         else
23:            temp ←temp + step
24:         end if
25:         if $(temp - step) \leq response \leq temp$ then
26:            $accept(response)$
27:            find ←true
28:         end if
29:      else {$response < temp$}
30:         if $(temp - step) \leq (proposal - offset)$ then
31:            temp ←proposal - offset
32:         else
33:            temp ←temp - step
34:         end if
35:         if $temp \leq response \leq (temp + step)$ then
36:            $accept(response)$
37:            find ←true
38:         end if
39:      end if
40:   end if
41: until find
42: return response;
```

a) Departure Time Interaction Algorithm

```
Require: preference {array of preference of the offerer}
         profile {array of profile of the offerer}
1: OK-preference ←false
2: match ←true
3: agreement ←false
4: for $i = 0$ to $preference.length - 1$ do
5:    $send\_pref(preference[i])$
6:    OK-preference ←$rec\_pref()$
7:    if $OK - preference$ then
8:       break
9:    end if
10: end for
11: if $\neg OK - preference$ then {the seeker profile is not in the preference}
12:    $send\_pref(\mathsf{NOT - MATCH})$
13: else
14:    repeat
15:       str ←$rec\_profile()$
16:       if $str \neq \mathsf{NOT - MATCH}$ then {check whether the seeker informs that the offerer profile is not in its preference array}
17:          if $is\_member(str, profile)$ then
18:             $send\_profile(true)$
19:             agreement ←true
20:          else
21:             $send\_profile(false)$
22:          end if
23:       end if
24:    until $(str = \mathsf{NOT - MATCH}) \vee agreement$
25: end if
26: return agreement
```

b) Users' Type Matching Algorithm

Fig. 4. Main Interaction Algorithms

algorithm where the ride giver (**BOB**) is checked if his type of people is what the ride seekers want. Considering the fact that ride seeker part is inverted in respect to the ride giver part, therefore, the algorithm from this point, row 14 to 24 and part from row 4 to 10, is reversed. The preference here refers to an array with the requested types of people that would be accepted by the system/user, while profile here means an array that holds the user's type.

- **Departure / Arrival Points Negotiation.** The interaction, between a **Driver** and a **Passenger** is concerned with finding common meeting points for departure and arrival. The exchanged message is: Departure-Place_Name, Arrival-Place_Name.

In our example, **BOB** provides the following meeting points of departure (in order of preference): car parking, university hall, time square. On the other hand, user **ALICE** provides the following points: research centre, library hall, university hall. The interaction between these two PAs suggests the "`university hall`" as the meeting point of departure. Essentially, it is possible to contact the user, only a passenger type, in case of a mis-match of meeting points to ask him/her to reconsider the previously given list.

- **Final Trip Agreement.** Reaching this decision-making phase indicates that both the driver and the passenger are sure of the compatibility of preferences and the matching of requests regarding a specific trip. The two PAs do not achieve an immediate agreement, instead they wait for five hours (arbitrary choice) prior to the departure time. This intends to increase the chance for every PA to interact with other agents to find better resolutions.

  The main goal of this phase is to get the best solution that fits into the user's preferences. When the negotiation parameters of the decision-making process are more similar to the parameters in the service request, a user's request will be properly fulfilled. So at the end of this phase, the PA partner will get a score based on the value of parameters taken from the sequence of negotiations: the partner name, score and parameters are stored in a list.

  Finally, the PA will contact every compatible partner found, from best to worst, until the number of people indicated in the service request is fulfilled. If two PAs have the same score, the system takes the one that is stored first on the list. Afterward, the PA produces an answer with its personal information and sends it to every founded partner giving them a positive feedback. The user will have 24 hours to confirm or disconfirm the positive feedback for his/her partner. Similar to the reputation techniques in web-based auction systems, the generated feedback will be significant in recommending or avoiding the future matching between a certain ride giver and a ride seeker.

THE IMPLICIT CULTURE MODEL. The applied Multiagent platforms are all based on the Implicit Culture framework [9]. Implicit culture is a generalization of collaborative filtering [10], where data mining techniques are used to extract knowledge about users' behaviors. In our system Implicit Culture results particularly useful to support the PAs interaction. More details about the Implicit Culture framework are available at [11].

For example, a student **X** may need to know the most frequently used meeting point within a university campus. The idea of the Implicit Culture framework is to let the system suggests the meeting points that are frequently used by other university students (i.e., members of that community). In this case, the system may suggest to student **X** to move to the most frequently used meeting point within the university campus, which is the Car Parking.

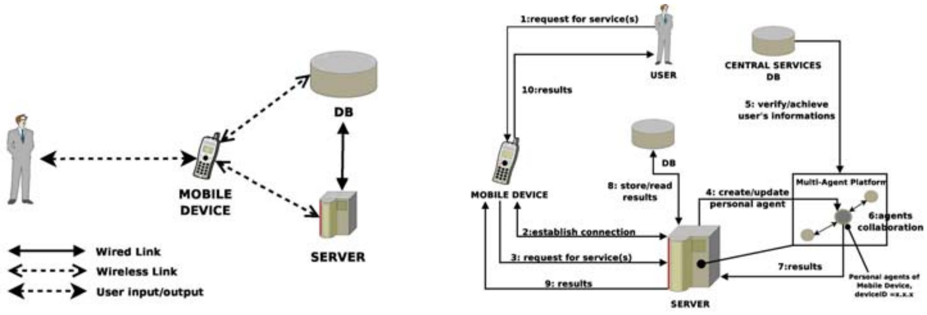
4 RIDESHARE SYSTEM ARCHITECTURE LAYER

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 [5] 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 Rideshare service instead.

4.1 SYSTEM COMPONENTS

The architecture of the system includes three main components:

- **The mobile device** communicates the user's requests with the servers and receives the results.
- **The distributed servers** within our system are mainly responsible for the Rideshare service. Each of the servers contains: 1) a multi-agent platform with Personal Agents 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 database stores also the information about users registered to the system.



(a) First Figure

(b) Second Figure

Fig. 5. The two figures above describe The System components Interactions and The Mobile to Service Accessibility Scenario

Figure.6/a 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 wireless communication technology. 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 Recommunicate 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 was offering 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 ride giver reputation value. The central server DB is responsible for storing all the information about a specific request and the interactions made between its two PAs.

4.2 SERVICE ACCESSABILITY

For a user to access the services, he/she needs to make three steps: (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 [12] which is a Bluetooth 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 Figure.6/b).

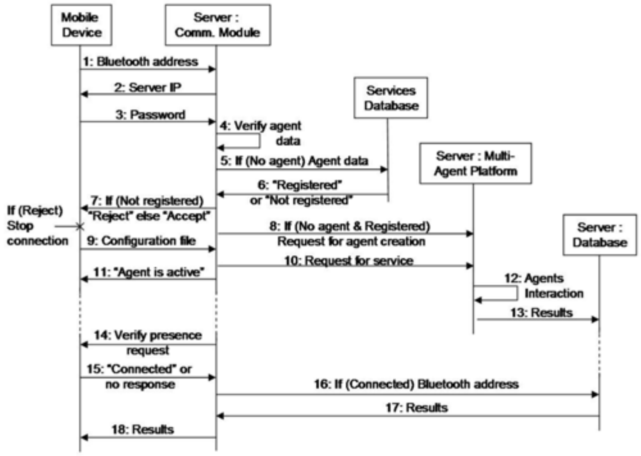

Mobile Device
Server : Comm. Module
Services Database
Server : Multi-Agent Platform
Server : Database
1: Bluetooth address
2: Server IP
3: Password
4: Verify agent data
5: If (No agent) Agent data
6: "Registered" or "Not registered"
7: If (Not registered) "Reject" else "Accept"
If (Reject) Stop connection
8: If (No agent & Registered) Request for agent creation
9: Configuration file
10: Request for service
11: "Agent is active"
12: Agents Interaction
13: Results
14: Verify presence request
15: "Connected" or no response
16: If (Connected) Bluetooth address
17: Results
18: Results

Fig. 6. The Service Accessibility

Figure.7 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 user-ID 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). That is by matching the Bluetooth address of the device with the password. 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 string to the communication module (step 9), which forwards all the user requests to the appropriate PA (step 10). The Personal Agent 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).

Additionally, the possibility to apply different telecommunication technologies is considered. For instance, using an advanced wireless communication method such as Wi-Fi or GPRS can be applied to facilitate the communication between the server side and the cellular phone side. The user may be not available within the communication range of the Bluetooth; therefore, an alternative solution may be the user-communication techniques through a wireless messages exchange platform, which can be easily integrated to the core architecture.

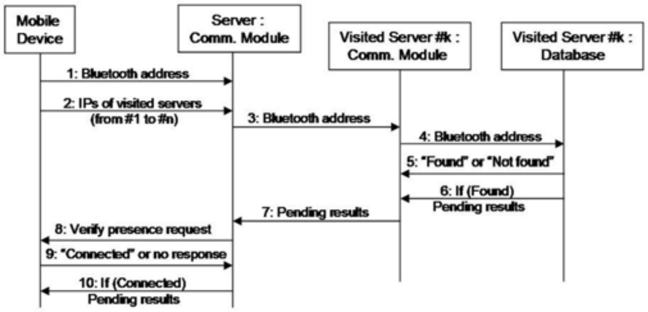

Mobile Device
Server : Comm. Module
Visited Server #k : Comm. Module
Visited Server #k : Database
1: Bluetooth address
2: IPs of visited servers (from #1 to #n)
3: Bluetooth address
4: Bluetooth address
5: "Found" or "Not found"
6: If (Found) Pending results
7: Pending results
8: Verify presence request
9: "Connected" or no response
10: If (Connected) Pending results

Fig. 7. The Pending Results Retrieval

4.3 PENDING RESULTS RETRIEVAL

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 Figure.8) The mobile device stores the IP addresses of all the visited servers in an XML list that is used later on to retrieve all pending results. The format of the results produced by the personal agent may contain the request identifier, contacts (e.g. phone number) of the user interested to share the ride, the departure time, etc. The user may receive the results immediately in his mobile device, this happens if and only if he/she is still within the Bluetooth / server coverage. 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.

Figure.8 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 library), the retrieval process will fail and the results will be cancelled. Yet these results will still be accessible via the original servers. Therefore, a possibility for the server to communicate with the user through SMSs is achievable.

4.4 EXPERIMENT FACTS

Issues pertaining to the usability of users, cellular phones and their interaction are multiple. What characterize this interaction are the features of contextual awareness, task hierarchy, visual attention, hand manipulation and mobility, which are sensitive to scenario change [13].

We tested the system using Nokia 6630 mobile phones and PC/Server equipped with generic BlueTooth adapter. Bluetooth communications have been implemented using BlueCove [14] which is an open source implementation of the JSR-82 Bluetooth API for Java. We have tested the system on different scenarios, and obtained significant results which were stored as reference. The time to obtain an agreement between two agents is the same for every situation. A limitation of this model, however, is the lack of a monitoring process of the number of active agents in single MAS.

CONCLUSIONS AND FUTURE WORK

In this paper we presented an implemented application of a Mobile-based Rideshare service application where Multiagent system and Bluetooth wireless communications technology are combined together to support co-localized communities of users. We discussed the architecture of the Multiagent platform applied for our system, the specific protocols used and the algorithms that have been implemented to realize the Agents interaction. Then we presented some implementation issues related to the system we have built. We recommend a verification process of system scalability before Real-life use and, testing its performance for a considerably high number of users.

There were, and still, big interests that are growing and well recognized in the direction of Rideshare systems. In 1995, Edward Walbridge [15] estimated that energy savings from ride sharing would be 48 million barrels of crude oil per year. The associated reduction in congestion is estimated to save driver time worth $6.2 billion annually. Walbridge attempted to explore the possibility of controlling the car owner's free seats, sharing rides modules and the movements of people in a particular area. That is through the use of a computer in each urban area that matches riders and drivers in real time and is accessed by lightweight pocket-sized ones. In 1999, D.J. Dailey and D. Meyers [16] presented the Seattle Smart Traveler (SST) which is an application of World Wide Web (WWW) technology to test the concept of automated dynamic rideshare matching. Dailey and Meyers could demonstrate through their model that car-pooling process is a quadratic function of the number of users participating.

Recently, Claudio Cubillos, Claudio Demartini and Franco Guidi-Polanco [17] have taken a different approach. Based on the Contract-Net Protocol, bids-filtering process and the use of agent framework, they presented a mediated planning model for the scheduling of trip requests under a passenger transportation system. This approach correlates with our future focus, which we expect that it will enhance ANDIAMO on the Rideshare offers/requests matching phase. Moreover, there is a great potential in integrating ANDIAMO with other services, such as Public Transportations info portals and online maps viewers, which would increase reliability and usability. That is because the system would be giving alternative transportation solutions based on the received ride details. We also propose the system integration with spatial databases, for managing the geographical information used in the system. Another potential direction is the integration of web service systems to add the value seen out of any Internet-enabled application.

References

1. Rakotonirainy, A., Loke, S.W., Zaslavsky, A.: Multi-agent support for open mobile virtual communities. In: Proceedings of the International Conference on Artificial Intelligence (IC-AI 2000) (Vol I), Las Vegas, Nevada, USA. (2000) 127–133

2. Carabelea, C., Berger, M.: Agent negotiation in ad-hoc networks. In: Proceedings of the Ambient Intelligence Workshop at AAMAS'05 Conference, Utrecht, The Netherlands. (2005) 5–16
3. G., M., B., B., A., H.: Applications of multi agent systems in traffic and transportation. In: IEE Transactions on Software Engineering. (1997) 144(1):51 60
4. Kothari, A.B.: Genghis - a multiagent carpooling system. B.Sc. Dissertation work, submitted to the University of Bath (May 11, 2004)
5. Volha, B., Paolo, G., Stefano, F.: Toothagent: a multi-agent system for virtual communities support. In: Technical Report DIT-05-064, Informatica e Telecomunicazioni, University of Trento (2005)
6. JADE: Java Agent DEvelopment Framework website — http://jade.tilab.com/.
7. FIPA: Foundation for Intelligent Physical Agents — http://www.fipa.org/.
8. Maes, P., Kozierok, R.: Learning interface agent. In: Eleventh National Conference on Artificial Intelligence, Washington D.C., MIT Press (1993) 459465
9. Birukov, A., Blanzieri, E., Giorgini, P.: Implicit: An agent-based recommendation system for web search. In: Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems, ACM Press (2005) 618–624
10. Blanzieri, E., Giorgini, P., P.Massa, Recla, S.: Information access in implicit culture framework. In: Proceedings (on line) of the ACM SIGIR Workshop on Recommender Systems, ACM (2001)
11. Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N., Modena, A.: Ic-service: A service-oriented approach to the development of recommendation. In: Proceedings of the 22nd Annual ACM Symposium on Applied Computing ACM Press, ACM (2007)
12. JSR-82: Java APIs for Bluetooth — http://www.jcp.org/en/jsr/detail?id=82.
13. Henry Been-Lirn Duh, Gerald C. B. Tan, V.H.h.C.: Mobile usability: Usability evaluation for mobile device: a comparison of laboratory and field tests. In: 8th conference on Human-computer interaction with mobile devices and services, MobileHCI'06. (September, 2006)
14. Blue Cove project — http://sourceforge.net/projects/bluecove/.
15. Walbridge, E.W.: Real time ridesharing using wireless pocket phones to access the ride matching computer. In: Vehicle Navigation and Information Systems Conference Proceedings/6th International VNIS. (July, 1995) 486 – 492
16. D.J.Dailey, D.Meyers: A statistical model for dynamic ridematching on the world wide web. ITSC 99 Tokio, Japan (5-8 October, 1999)
17. Claudio Cubillos, C.D., Guidi-Polanco, F.: Passengers trips planning using contract-net with filters. 8th International IEEE Conference on Intelligent Transportation Systems Vienna, Austria (13-15 September 2005)