ANDIAMO: A MAS to Provide a Rideshare Service

Stefano Fante

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 brief paper, we present an agent-based rideshare system that is accessible via lightweight devices (mobile phones and PDAs). The paper (as [1] does) illustrates the overall infrastructure of the system and the specific protocols it uses.

1 Inroduction

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 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. 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. In classical web-based rideshare management systems, 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 [2] architecture, where an infrastructure of a Multiagent application is accessible via lightweight devices that are Java and Bluetooth enabled.

This paper proceeds as follows: Section 2 explains

the general framework of applying a rideshare service using MAS architecture. Section 3 introduces the implemented system architecture and explains it. In the end, Section 4 presents some concepts seen during the doctoral course on Advanced Topics in Agent-Oriented Computing and try to map them in the Andiamo system.

2 A rideshare framework

Within a rideshare management system, major parts are achieved through the negotiation process of agents. Autonomous agents that take part of a Multiagent 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 Multiagent platforms accessible via lightweight devices. The user can access the system from a number of Bluetooth access points directly connected to the Multiagent platforms or simply by sending an SMS. We have a central 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. The model has three layers: (1) an agent platform (JADE); (2) A Multiagent architecture with the Implicit Culture module; and (3) a superior layer that includes the rideshare system.

2.1 The MAS architectural layer

The MAS is implemented in JADE (Java Agent Development framework) a FIPA-compliant framework for Multiagent 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, receives and processes a user's requests. An agent is identified by a unique name and there is a one-to-one correspondence between agents and mobile devices (users). 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, otherwise its behavior is updated with the new request. 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.
Personal Agent (PA). This Agent 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**.
On each platform there is a dedicated agent, called Expert Agent (EA), which contains the System for Implicit Culture Support(SICS) [3]. 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. After a PA receives its user's request, it sends it to the Expert Agent. 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. Composer component estimates the real value for parameters if the input is incomplete or wrong. 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, which it processes during the second phase. Finally, the SICS needs to gather information about the users behavior. In the described system, in order 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 database, where the results 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 EA extracts necessary information (e.g., departure, arrival place and meeting points) from the proposals and stores them in its internal database.
- 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, passenger type, departure date, departure time, offset, departure place and departure meeting point. Arrival place, arrival meeting point, user feedback, number of people.

The Implicit Culture Model. The applied Multiagent platforms are all based on the Implicit Culture framework [3]. Implicit culture is a generalization of collaborative filtering, 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. 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.

3 Rideshare system architectural 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 [2] 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.

3.1 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 Multiagent 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, 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. The connection between the mobile device and the server is established through SMS/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 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 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 The system... after the doctoral course

So far, we have briefly described Andiamo without considering the concepts demonstrated during the doctoral course on Advanced Topics in Agent-Oriented Computing.
In Andiamo, we can identify a number of artifacts, such as the number of computational entities seeking the utilization of software agents. Looking at the scenario where central database, which can be also viewed as an artifact used by agents to store and recover their information. This kind of artifacts mediates between agents and the adjacent environment and also offers to agents a fundamental support. Consequently, when a database responds to agents, these agents change their behavior according to the state of the database and the surroundings. Involved artifacts increase agent's ability to manipulate and transform different objects besides; all concerned agents use the available databases operations. These operations would change the artifact's state and, as a result, they would also leave an impact on the overall environment. Eventually, the changes occurred in the environment will influence the behavior of other partaking agents.
Prior to this course, we used to implement and use artifacts but, we did not properly realize their exact role in the system or, we perceived their roles differently. Artifacts are neither autonomous nor proactive, they are transparent and predictable, and they have a specific function but not a goal. They are simply designed to serve and get utilized by system agents.
In this context, we can mainly distinguish three types of artifact:

- **Individual artifacts**, which handle a single agent interactions.
- **Social artifacts**, which handle the interactions of several agents and artifacts.
- **Environmental artifacts**, which handle the interactions between a MAS and its environment.

Applying the above classification, we became able to locate our central database within the environmental artifacts, because when agents modify their internal state, these modifications influence both, the environment and agents' behavior. In addition, system agents compete and collaborate to obtain and offer a car ride, and this is mostly driven by applying a function to calculate the value of the ride proposal, and this function depends on agents' parameters and their ability to modify them complying with user's desires. This particular value computation mechanism can be seen as a social artifact because it handles all the interactions among agents. When an agent uses this function, it modifies the behavior of other agents that are forced to recalculate their parameters to obtain a new competing value.
Examples of individual artifact implemented in our system are the functions used by agents to verify their state, their position in the riders ranking list, and their situation in the overall system. The EA's database too can be seen as an individual artifact. It is used only by the EA for the computation of the historical data and, it is also used to give proper suggestions about meeting points and other important information to the PAs.
In general, concepts about agents discussed during the course are similar to those adopted by us. Our system agents are active entities that encapsulate control and the rules to govern them. They are autonomous and therefore pro-active; they are strictly coupled with the environment where they live and act; they act to change the state of both, other agents and environment; moreover, they operate to achieve all their goals (e.g., they work on defeating other agents and win the running ride auction).

5 Conclusions

The brief description we give on Andiamo showed how Personal Agents (PAs) are created, functioning, compete and collaborate to achieve their goals. We also underlined the concept of artifact and we identi-

fied some different layers of artifacts in the Andiamo platform.

References

[1] Sottini F., Abdel-Naby S. and Giorgini P.. Andiamo: A Multiagent System to Provide a Mobile-based Rideshare Service. University of Trento, DIT. Technical Report 06 -097.

[2] Bryl V., Giorgini P., Fante S. : Toothagent: a multi-agent system for virtual communities support. In: Technical Report DIT-05-064, Informatica e Telecomunicazioni, University of Trento (2005)

[3] 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) 618624