Increasing Interactivity in Agent-based Advanced Pocket-Device Service Application

Sameh Abdel-Naby, Paolo Giorgini and Stefano Fante

Department of Information and Communication Technology (DIT)
University of Trento, Povo 38100, Italy.
{sameh, paolo.giorgini, stefano.fante}@dit.unitn.it

Abstract. Independence, intelligence and interactiveness are making software agents strongly approach the development of advanced service applications for both, pocket and fixed computing devices. In this paper we present an interactions protocol that is used by intelligent agents operating in a dynamic environment. In particular, we focus our research on the situation where a multi-agent system is serving lightweight devices through advanced communication methods (e.g., Bluetooth). Like similar contributions, our interactions protocol provides agents with a monetary system and a mechanism for feedback calculation. The goal of our research was to accelerate efficient agents interactions while resolving end-user composite tasks.

1 Introduction

Lightweight devices such as cellular phones and PDAs are increasingly involved in most of our daily life duties. Nowadays, people can go anywhere carrying their pocket devices which allow them to check their emails, surf the internet and do shopping. Services are provided through user-friendly and well-developed interfaces, and almost costless in regard to the value of services users are getting. Currently, standard mobile services that never existed before are becoming a must (e.g., SMS and MMS), and advanced ones that newly existed are now highly desired (e.g., Service Guides and Group Gaming).

Several efforts in literature, for example [3], tackled the scenario where the cellular phone of a visiting scholar establishes a connection with a localized Multi-Agent System (MAS) and, a synchronization is made to finally come up with meetings agenda. Participants of such scenario are moving within a university carrying their lightweight devices or, they have previously delegated an agent to act on their behalf. Automated system agents cooperate and negotiate available times to create a suitable agenda for everybody. Accordingly, the visiting scholar and meeting requesters are forming together a Mobile Virtual Community [7] that is location-based.

Another approached scenario is about tourists that turn to be MAS users after enabling the Bluetooth functionality of their pocket devices and, using a preinstalled application, their devices communicate with distributed servers and retrieve useful information related to the places they are visiting (e.g., [1]). In different approaches, every single item available at a museum can be represented by

its own software agent, and this agent can cooperate with others to fulfill certain complex user desires (e.g., relevant places opening times and transportations).

Through lightweight devices, users in previous scenarios are performing a set of actions that are driven by application instructions to finally create a delegative agent. Eventually, this agent will be searching for methods to fulfill user desires and, a matchmaking process will take place; an agent that carries specific information will look for another agent that is willing to give extra data so a task gets completed. Still, sometimes an agent will never find a single completer and thus, there are complex one-to-many scenarios where group of agents cooperate.

Unless software agents learn to properly interact there will not be an extra capability for people to cooperate. A negotiation language that is applied among distributed agents is helping them to understand each other, discuss their desires and finally achieve their objectives. Several of the negotiation protocols proposed by scholars are inspired from sociological, political and psychological studies about human negotiation in real-life situations such as auctions, peace agreements and biddings.

We focus on multi-agent systems that deliver location-based services to users of lightweight devices through advanced communication capabilities. We contribute to existing literature by presenting a negotiation algorithm we adopted in a rideshare application, which increased the interactivity level among involved agents. Although there are some restrictions given by users (e.g. time to achieve) and others given by involved technologies (e.g. Bluetooth data exchange rate), still the architecture we developed is reliable and increases system usability.

The remainder of this paper is structured as follows. Next section emphasizes our research motivation. Section 3 looks at the building blocks of the proposed interactions protocol. Section 4 applies the presented negotiation algorithms to a testbed application. Section 5 highlights the related work. Section 6 demonstrates our future work and concludes the paper.

2 Motivating Scenario

In a MAS delivering service content to lightweight devices, a set of uncooperative agents that are self-interested and benefits maximizers are located. It is more often that this set contains two different types of agents, BUYER AGENT (BA) and SELLER AGENT (SA). Each of them holds information related to its role in the system and, a BA keeps data that helps SA increases his profit, and the data SA keeps helps BA to achieve the overall objectives of the system. If agents predefined behavior is strict and intelligent "usual case in MAS" this will lead both agents to - *sometimes* - reach a situation of disagreement.

The fact of having two successfully matched agents and yet no useful results are gained is quite challenging. The existence of autonomous agents in MAS is necessary to increase system reliability and, interactivity between all application entities is still highly desired, but an unfruitful negotiation process among involved software agents is what a complete application should avoid, and this is what we precisely try to address.

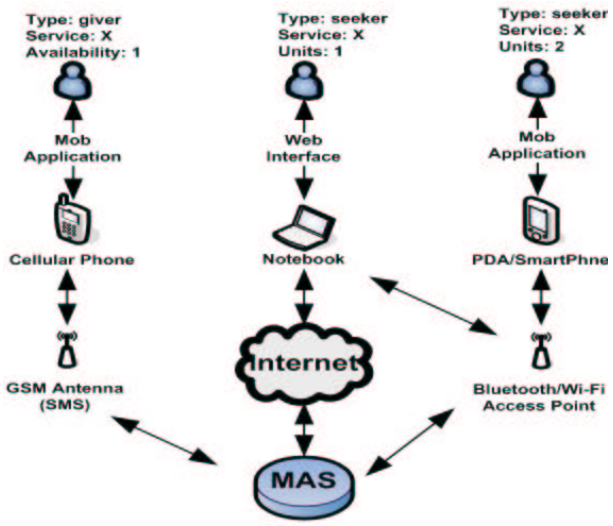

Fig. 1. Different devices use different communication methods to interact.

In figure 1, we assume that three different users are interested in using the same multi-agents architecture to obtain a certain item or service. This service is limited to the demand and supply of a specific product among system users (e.g., available care seats in a carpooling system or used books in trade environment). These users are using their lightweight devices to communicate with the service architecture and, each is adjusted to the use its pocket application. Actors differ, one can be a service giver and the others are the requesters. If we move to the point where the number of acquisition requests is greater than the number of offered items, insufficient matches occur.

Each of the involved lightweight devices is configured to utilize specific communication method to access the service, a cellular phone or a notebook may exchange service requests using SMSs, The Web or a distributed Bluetooth or even Wi-Fi access points. If a user is offering a single item that more than a single requester is interested to have, the managing MAS will drive these three - *or more* - users to a complex situation where the ownership of the offered item is not determined. In this case, the system hangs at the pre-agreement point where the service preferences are matching, items are available, but conflict is located and no actions are taken.

An auction mechanism can be invoked to resolve any complex situation that may occur among several competing agents. This mechanism can be restricted to different conditions, such as time and location constraints, and it can be wisely adapted to ensure ultimate benefits gaining for both, the supplier and demander agents. The invoked mechanism can also be heuristic by storing auctions results that involve same software agents - *representing same users* - more than a certain number of same scenario participation.

3 The Auctioning Interactions Protocol

In this section we present the negotiation scenario involving concerned agents to finally establish proper communication channels, achieve better results, and increase the level of efficient interactivity.

Given a set of lightweight devices that are capable of communicating specific data with central MAS servers via distributed access points and, given that the overall architecture is providing end-users with a predefined location-based service. The lightweight devices here are used to clarify users preferences and consequently, a Mobile-to-Server Link Agent (MSLA) is created. This particular agent carries specific user desires details and, it is capable of transferring from a lightweight device through the nearest access point to reach server side. When the MSLA arrives to one of the central service servers, its carried desires are forming an autonomous software agent that reflects certain user characteristics. This MSLA is basically a configuration file that is produced by each lightweight device participating in a certain trade scenario.

Eventually, 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 meant to update the behavior of a running agent.

A group of delegative autonomous agents that are seeking to achieve different tasks in different times is located at the server side of the architecture. When some of the tasks to be achieved are complex and require high level coordination, a negotiation scenario that requires a single agent to deal with several service requests coming from different greedy agents is situated. However, in agent-to-agent situations, the negotiation protocol applied is simple and efficient; it is the same as market demand and supply equality. When the supplier and the demander are matched, a mutual benefits exchange is achieved. This usually occurs because only one demander and one supplier are located within the service range of each other. Unsurprisingly, in agent-to-many it is more complex.

In figure 2, Algorithm 1, we show the algorithm used by Seller Agent (SA) to invoke an auctioning situation that is expected to resolve a complex negotiation situation. From line 1 to line 3, both SA variables, `bestValue` and `numLoop`, are initially set to `'0'`. In line 4, the seller agent requests the Buyer Agent (BA) 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 SA waits to receive new offers from all involved BAs, 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 gradually incremented.

Otherwise, since it is not the first loop, from line 11 down to line 19, the SA checks whether the `val` function is increasing with respect to last obtained best value. At this point, two scenarios may occur, if `val` is greater, the value obtained from the concerned BA is communicated with other BAs and, they are asked to Re-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 suspended and the BA currently had the `bestValue` wins - *line 17 to line 19*.

```
Seller_Agent_procedure()
1:  bestValue = 0;
2:  numLoop = 0;
3:  auctionIsOpen = true;
4:  askBAToStartAuction(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:       requireNewOfferToBuyers(bestValue);
15:       numLoop++;
16:     else
17:       if (val <= bestValue) then
18:       quitAuction();
19:       informWinners();
20: end while
21: quitAuction();
```

Algorithm: 1 The procedures taken by the Seller Agent.

```
Buyer_Agent_procedure()
1:  BABestOffer = 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>BABestOffer&&newVal>bestOffer) then
11:         BABestOffer = newVal;
12:         sendOffer(BABestOffer);
13:         sent = true;
14:       if (!sent && !modificationsArePossible) then
15:         sendOffer(BABestOffer);
16:     end while
17:   else
18:     if (decision == refuse) then
19:       quitAuction();
20: end while
21: quitAuction();
```

Algorithm: 2 The procedures taken by the Buyer Agent.

Fig. 2. The negotiation protocol assisting agents to establish proper communications.

Finally, the algorithm is terminated and the auction scenario is ended - *line 20 and 21*. Following to that, we explain the BA responses with respect to SA.

In the same figure but Algorithm 2, from line 1 and line 2, a variable `BABestOffer` that carries the buyer agent 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 BA has communicated his offer. From line 3 to line 6, while the auction is open, BA holds its offer transfer until a communication was received from the Seller Agent (SA) asking for an auction participation decision. The BA puts the results from the evaluation function into the `decision` variable.

From line 7 to line 9, if the BA accepts the SA call for auction participation a self-revision for its parameters is made. This revision indicates BA's insistence to obtain the auctioned item and its intentions to show extra negotiation flexibility. The part from line 10 down to line 13 refers to the comparison made by the BA 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 `BABestOffer` is set to new one and stored in `newVal` and the offer is sent to the corresponding SA.

Line 14 to line 16, after the BA self-revision, if the value gained is the same as the previous one, this specific BA do not send the previous value if `modificationArePossible` is true. The BA continues to review the carried parameters until `modificationArePossible` becomes false or it communicates new

better offer. If `modificationArePossible` stays on `false` and parameters are not sent, the BA communicates same previous offer.

From line 17 to 19, if BA refuses the auction call the algorithm terminates and the auction involves this particular agent ends. Eventually, the algorithm passes on the first condition if `decision == accept` but the condition of the successive while `modificationArePossible&&!sen`t return `false`. The method `decideIfAcceptOrRefuse` return `refuse` if for instance, a BA has enough time before the auction deadline and it decides to refuse present participation. Finally, the algorithm is terminated - *line 20 and 21.*

Auctioning among agents requires high level agent-to-user interactivity and network resources consumption. Therefore, agents' intelligence may appear when a repetitive scenario occurs. The algorithms presented can be further adapted to maintain system and participants history.

Once the pocket-device application is configured to repeat the same service request on daily or weekly basis, and similar agreements are achieved between a specific supplier and a demander at a certain price, the next time this demander agent will be first looking-up the very exact supplier agent that has potential agreement than others. This can be simply added through a learning agent behavior that maintains an array that saves last successful agreement details.

4 A Case Study

ToothAgent [2] is an example of a Multi-agent system (MAS) that allows university frequenters to use any of their computing devices to exchange used books requests and offers. Once an agreement is reached, the system helps students to agree on meeting places and times. This is all done through normal Bluetooth communications that take place between both, user and distributed servers. Agent-oriented programming techniques are used to form a Mobile Virtual Community [7]. This helps the system, including its Intelligent, Mobile and Autonomous agents to go through the matchmaking and exchange of requests processes and, support the price negotiation phase.

Andiamo [11], is an example of a MAS implementation that provides its users with a possibility to utilize their lightweight devices to offer/look-up shared car rides. To understand the *Rideshare* service or *Carpooling* as stated in literature; it is a method to reduce the use of cars in a specific town or area, and it take 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 system 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.

In this section, we further elaborate on the mechanism we proposed through the demonstration of a pocket-device rideshare service architecture we developed. In our example, we primarily assume that a car ride giver (Seller Agent) - `SA Started` - has communicated and submitted the offer details to the Multi-Agent

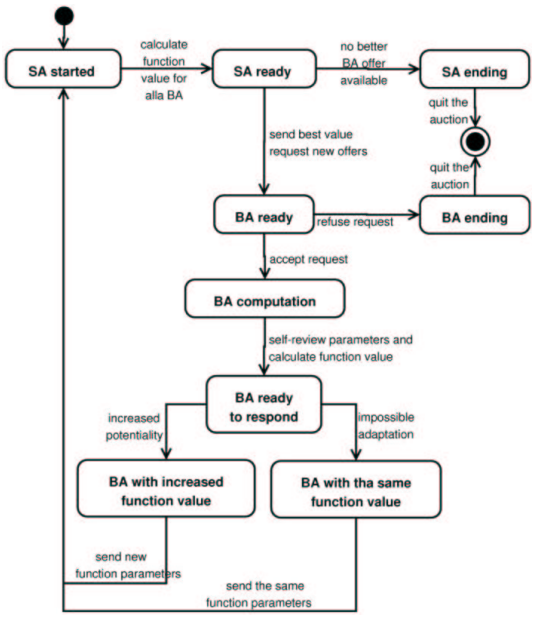

Fig. 3. Auction call and termination in a Rideshare MAS service architecture.

System, this MAS is managing the exchange of service requests among connected computing devices, and we assume that only one available car seat is given by the car owner, and a matching phase has resulted three or more interested ride seekers that are all willing to share the given ride cost.

As shown in figure 3, from this point on SA Ready, which is the agent acting on behalf of the ride-giver, will be responsible of resolving this complex situation by: 1) according to the parameters given by agents of the ride seekers, a calculation process is performed and each agent is assigned a value, 2) a comparison between the yielded values is made and sent to interested ride seekers, then a call for auction is made, 3) a request to all interested agents to send new offers is communicated. Each agent acting on behalf of a ride seeker is free to choose to participate in the auction, but hence an agent has decided to skip an auction, the negotiation process involving both parties is ended - `SA Ending` - `BA Ending`.

This was considered because end-users may put more rigid or loose behavior on the representing agent at anytime. But once a seeker agent has decided to go through the auction - `BA Ready`, a self-revision process for the carried parameters is made - `BA Computation`, and then a value calculation is made and compared to the previous one obtained, then results are communicated with the ride giver agent originally invoked the mechanism. The seeker agent keeps trying to compromise in accordance with interests so a new agreement is reached.

Results reached after parameters modification - `BA Ready to Respond` - will indicate if a new auction winning potential is found. Depending on the value obtained in earlier step, the same old parameters or new ones will be communicated

back with the ride giver agent (SA) - `BA with Increased Function Value` or `BA with Same Function Value`. The ride giver agent re-evaluates the received parameters including those received from newly joined agents, if any. Then, an announcement is made for the only available car seat winner. Accordingly, the auction terminates and the entire negotiation process ends. The mechanism can be repetitive only if no agreement situation was found and the time to achieve the actual ride is yet far.

Auction invocation and the exchange of messages among involved agents consume time and network resources. Therefore, the operating MAS can further store the auction initiator and winner to rapidly resolve future complications. This can only happen if the algorithms used were adapted to observe certain auction results that are identically repeated, so the system would automatically consider this winning ride seeker agent as high-potential future auctions winner, or one of the winners - *if more available seats were given*.

5 Related Work

Part of the research conducted in Distributed Artificial Intelligence (DAI) focuses on the coordination among objects located in distributed environments. Thus far, a research topic under DAI that is Distributed Problem Solving (DPS) proposes negotiation strategies that mostly seek the construction of what we call Distributed Objects Communication Language (DOCL). Among other advantages, negotiation languages are helping the establishment of cooperative environment that successfully achieve multipart tasks and deliver refined services.

Scholars have also attempted to address the problem of enhancing multi-agent coordination by reflecting real human negotiation skills inside a computing environment [4, 5, 6]. These studies were mainly carried out because, 1) the need to construct a computing program that entirely acts on behalf of its operator has imposed the need to apply Agent-oriented concepts and theories. 2) The need to construct a cooperative computing program that automatically interacts with other entities to achieve complex tasks has raised the need for a proper negotiation language. These two reasons are forming together the need to design the negotiating agents that are able to meaningfully interact, talkatively negotiate and mutually maximize their benefits.

In their book [9], J. S. Rosenschein and G. Zlotkin are doing what they call Social Engineering; they have dedicated part of their research on how designers of software agents would react to the development process of Multi-agent systems and, the use of certain design steps regarding the accomplishment of suitable negotiation protocol, which in return will lead to appropriate interactions among several MASs. They emphasized the urgent need to look at agents as the new era of human *surrogates*, and this is because of the nowadays speed taken to approach full system and machines delegation.

In Game Theory [12], a clear approach was taken to study the rational behavior among self-interested agents. Different software designers are working on the development of several software agents; these development processes will

only produce an agent that is reflecting designer's personal behavior. Although the agents produced are self-interested and autonomous, they are going to interact with different agents that are designed differently and contain different level of autonomous performance and complexity. An agent that is rationally driven within a system entities will make goals and procedures to achieve them clear for all system actors, but it will apply an atmosphere of firmness and inflexibility in formed interactions. This earlier discussion has raised the confrontation of two important design aspects, would it be more appropriate to design an agent that is deterministic or an agent that is flexible?

When Distributed Artificial Intelligence (DAI) started to have its own structure as independent research area, Reid G. Smith has contributed significantly to this structure formation by having his PhD thesis defense, in 1978, discussing a new perspective in achieving proper negotiation and interactivity among multiple automated network-nodes. Later to that, an important contribution was added to literature regarding the same topic, which is Contract Net [10]. When applied to multi-agent systems, the Contract Net protocol assumes that each node in the network is an agent that is seeking another completer-agent that may, together, form a coalition to resolve a complex task. This coalition can yield some results that can not be achieved if each agent is operating separately.

When the exact rare resources are to be used by several agents, an Auction [8] is formed between these agents so that system resources are utilized at the highest possible value, and certain negotiation language that perfectly applies in this situation is used. Due to issues related to equality, ordering and planning, Auctions have gained a wide range of applications in multi-agent systems. Four major auction types that are widely recognized: 1) English, 2) Dutch 3) First-Price Sealed-bid, 4) Vickrey's Mechanism or Second-price Sealed-bid. These auctions are reflecting real human behavior in different auction styles and similarly apply it to agents.

6 Conclusions and Future Work

Negotiation protocols used among agents that are serving computer based applications differs from those used for computing pocket devices. We are rapidly approaching the era of lightweight device services, and as a result a great focus and immediate redirection is realized towards the achievement of cooperative agents in mobile-based service architectures. In this paper we explained the motivation behind our interests to find an appropriate agents' negotiation protocol that serves pocket-oriented applications. We demonstrated the research conducted in reaching cooperative MAS architectures, and the negotiation protocols previously proposed by scholars that mostly targeted fixed computing devices environments. We proposed our negotiation protocol, and we applied it on Rideshare service architecture.

Our future research will focus on increasing the usability of agent-based pocket service application, and accelerating the efficient delivery process of any mobile service content. We will work on integrating the newly proposed negoti-

ation protocols to architectures that support lightweight devices. Eventually, we will simulate agents behavior in achieving complex tasks while applying different adaptive negotiation mechanism. This will help us observe differences in application performance and refine the proposed protocols. We also intend to study the possibility to make developers of pocket software agents able to a standardized but customizable negotiation protocol.

Acknowledgement

This work partially involves EU-SERENITY, PRIN-MEnSA, PAT-MOSTRO, PAT-STAMPS, and PAT UNIQUIQUE SUUM. We also thank ArsLogica for the unabated cooperation and support given to innovation.

References

1. M. Bombara, D. Cali, and C. Santoro. Kore: A multi-agent system to assist museum visitors. In Proceedings of the Workshop on Objects and Agents (WOA2003), Pp 175-178, Cagliari, Italy.
2. V. Bryl, P. Giorgini, and S. Fante. Toothagent: A multi-agent system for virtual communities support. In Proceedings of The Eighth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS), Hakotade, Japan, May'06.
3. O. Bucur, P. Beaune, and O. Boissier. Representing context in an agent architecture for context-based decision making. In Proceedings of the Workshop on Context Representation and Reasoning (CRR'05), Paris, France, 2005.
4. K.-M. Chao, R. Anane, J.-H. Chen, and R. Gatward. Negotiating agents in a market oriented grid. In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 436-437, IEEE Computer Society, 2002.
5. Jennings, N. R., Parsons, S., Noriega, P. and Sierra, C. On argumentation-based negotiation. In Proceedings of the IWMAS, MIT Endicott House, Massachusetts, USA, October'98.
6. S. Kraus. Negotiation and cooperation in multi-agent environments. Artificial Intelligence journal, Special Issue on Economic Principles of Multi-Agent Systems, 94(1-2):79-98, 1997.
7. A. Rakotonirainy, S. W. Loke, and A. Zaslavsky. Multi-agent support for open mobile virtual communities. In Proceedings of the International Conference on Articial Intelligence (IC-AI 2000), Las Vegas, Nevada, USA, pages 127-133, 2000.
8. Kate Reynolds. A survey of auction types. Agorics, Inc., 1996.
9. J. S. Rosenschein and G. Zlotkin. Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. The MIT Press, 1994.
10. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29(12):1104-1113, December, 1980.
11. A. Sameh. F. Stefano and G. Paolo. Auctions Negotiation for Mobile Rideshare Service. In the Proceeding of the Second International Conference on Pervasive Computing and Applications (ICPCA07), July'07, Birmingham, UK.
12. J. von Neumann and O. Morgenstern. Theory of Games and Economic Behavior. Princeton University Press, 1980.