UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38123 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

BarterCell: an Agent-based Bartering Service for Users of Pocket Computing Devices.

SAMEH ABDEL-NABY, OLEKSIY CHAYKA, PAOLO GIORGINI.

September 2009

Technical Report # DISI-09-053

BarterCell: Agent-based Bartering Service for Users of Pocket Computing Devices.

Sameh Abdel-Naby
Dipartimento di Ingegneria e Scienza dell'Informazione
Via Sommarive, 14
Trento, Italy
sameh@disi.unitn.it

Oleksiy Chayka
Dipartimento di Ingegneria e Scienza dell'Informazione
Via Sommarive, 14
Trento, Italy
oleksiy.chayka@dit.unitn.it

Paolo Giorgini
Dipartimento di Ingegneria e Scienza dell'Informazione
Via Sommarive, 14
Trento, Italy
paolo.giorgini@unitn.it

ABSTRACT

On behalf of nomadic users and through the use of computing pocket devices, agents can efficiently operate in wireless networks, cooperate to resolve complex tasks and negotiate to reach agreements while attempting to maximize their utilities. However, the negotiation protocols intelligent agents use are hardly considering several of the requirements evolved after the increasing reliance on mobility. In this paper we present a new negotiation protocol that avoids the use of mediating agents and applies a voting-like mechanism to handle service requests of nomadic users in wireless networks. We examine our approach in a scenario where it is essential for a multi-agent system to establish a chain of mutually attracted agents seeking to fulfill different bartering desires. We compare the results obtained with those produced after using an adjusted version of the Strategic Negotiation Model.

1. INTRODUCTION

The existence of heavyweight computing devices has once encouraged different organizations to address their necessity to process and manage their data. Efforts on that direction were concluded in Data Processing Systems (DPS) that are enhancing the way organizations interact with any of their focal information. Eventually, these applications were called Information Systems [11], and it started to play a major role in the way an organization emerges to an industry, sees the achievement of its goals, or studies the improvement of its future state.

Intelligent information agents [8] are those capable of interacting with several distributed or heterogeneous information systems representing end-users in obtaining data and overcoming information overload. In [7] authors have relied on information agents, data acquisition agents and rule-based reasoning agents to build a multi-agent system (MAS) [21] capable of receiving data from a legacy information system - Enterprise Resource Planning (ERP) - and control the extracted information using AI techniques. That effort has added the possibility of an existing information system to be customized according to the new preferences of end-users without any re-engineering processes.

Using ubiquitous agents, another approach was taken in [3] to allow mobile devices to access Web Information Systems (WIS) depending on their location. Authors have used agents to represent the goals nomadic users would like to achieve, store the exact location and connection features of each user and then migrate these agents to different information systems (or other mobile devices) to find relevant data or another information agent capable of answering user's requests. In PUMAS, the agents negotiation was implemented using standard distributed systems technique - message passing and recommendations - in spite of the dynamicity of mobile users and the limited resources of a mobile network.

Negotiation protocols are easing the tasks distributed information systems are concerned with, such as reaching agreements and sharing resources. Assuming that each independent information system is an autonomous agent, scholars have attempted to address the significance of a proper negotiation model in multi-agent systems using different approaches (see [15] for an overview). For example, some of the negotiation protocols used in e-commerce or data allocation applications are inspired from actual auctioning [12], others by the notion of contracting [17] and even politics and economics in [6].

Applications that consider agent-oriented approaches to offer mobile services [10] are likely to guarantee a level of reliability that is not achieved alternatively. The ongoing research on MAS negotiation protocols is considering the partial physical linkage of users to large number of ISs [13]. However, none is broadly considering active Mobile Information Systems (MIS) [19] that are delivering services to nomadic users and, the differences it would make on the negotiation scenario occurring among cooperative, intelligent and autonomous agents. For example, in order to avoid the unavailability of users within a specific service coverage area and, apart from the little attention paid to limited resources and data traffic rate in mobile networks, the speed of reaction a modern information system should address varies according to the area size it serves.

In order to minimize the amount of interactions between system components and adjust the use of resources, we present a negotiation protocol that manages the interactions of agents performing in dynamic mobile networks and, it avoids the

central management of any service request and instead; it establishes an election process among concerned agents to finally come up with a mutually benefiting agent - that varies - to match pending requests.

We use an agent-based mobile bartering application to examine the performance of our protocol. We simulate the behavior of agents in strict situations where time and network resources are limited. We adjust another negotiation protocol from existing literature, simulate the behavior of agents in the same bartering application, and we compare the results obtained in both scenarios.

The paper is organized as follows. Section 2 gives an overview of the related literature. Section 3 introduces our negotiation protocol. In section 4 we present BarterCell, which is a case study that applies our protocol. Section 5 introduces a customized version of the Strategic Negotiation Model [14] that we adapted and simulated to evaluate ours. Section 6 concludes.

2. RELATED WORK

In [16] they define a negotiation protocol in multi-agent systems as "the public rules by which agents will come to agreements, including the kinds of deals agents can make and the sequence of offers/counter-offers that are allowed". Same authors have distinguished between high-level negotiation protocol like those in multi-agent systems and, low-level negotiation protocols like those in networks (e.g., AppleTalk). In fact, high-level protocols are concerned with the content of the communication and not the mechanism that these content use to transfer.

R. G. Smith [17] has stressed the same distinction showing ARPANET and other similar protocols at this time as examples of high-level negotiations. He showed his standpoint by considering the high-level protocols as methods that lead system designers to decide "what [agents] should say to each other". And low-level protocols make system designers decide "how [agents] should talk to each other". The Contract-Net protocol Smith presented assumes the simultaneous operation of both; agents asking to execute tasks and agents ready to handle it. The asking agents broadcast a call for proposals, and the helping agents submit their offers and then one is granted the pending task.

Eventually, four different types of auctions [4] were widely considered in the literature of agents; 1) English, 2) Dutch 3) First-price Sealed-bid, 4) Second-price Sealed-bid. These auction types share the same goal, which is granting a single item (sometimes combinatorial) to a single agent (sometime a coalition) in a limited resources environment. An agent may participate in an auction so one of the carried "personal" tasks can be accomplished, or - like in cooperative systems- a learning behavior can be implemented so agents are able to predict the future importance of this item to another agent, which is known as commonvalue.

In [20] proposed model of a coalition formation enables each agent to select individually its allies or coalition. Therefore, the model supports the formation of any coalition structure, and it does not require any extra communication or central coordination entity or agent. However, the model results in increased processing for each of the negotiating agents. Moreover it does not consider overall coalition interest during its creation.

Definition of an optimal coalition in [18] is based on Pareto dominance and distance weighting algorithm. Instance-based learning algorithm is used to select an optimal coalition from set of coalitions. Optimal coalition determination became a multi-objective optimization problem because they take each agent in the coalition with abilities represented by multiple metrics while we use coalition formation mechanism based on resources available to each agent.

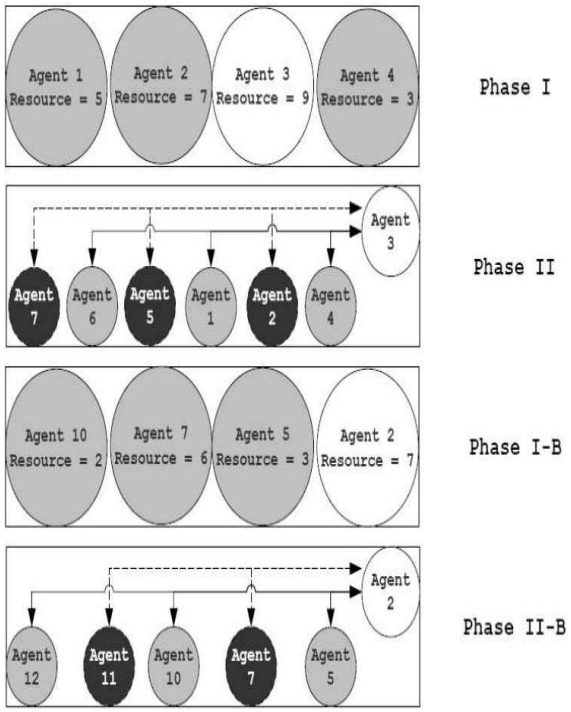

Figure 1: Electing a manager agent in diverse scenarios.

In [2] it concentrates on trusted kernel-based mechanism of making a coalition rather than on efficiency of a common task solving. Common task is then distributed among agents of a formed coalition. The most complex task in our work lies on top-view agents' organizational level rather than on task-solving abilities of all agents in a coalition. One central agent that will manage other agents from top-view level can find a way to best organize agents of a given community and satisfy their needs in a best possible way.

3. THE WISE NEGOTIATION

For one computational unit it is sometimes impossible to achieve a given task due to many obstacles, such as lack of resources (e.g., information, hardware power, etc.). These complex situations can be resolved if cooperation among system entities is achieved so that each is able to solve a part of the overall goal. Solving a joint problem means that there must be correct negotiation mechanism between all of the units involved into the same process.

In a decentralized environment it is hard for all peer units to organize themselves in a way to effectively solve a given problem. It is even harder when those units are asked to find definite structure (sequence of their cooperation). In order to choose the optimal cooperation scheme in a given set of peer units at particular time, we made agents orga-

nize themselves to choose the group manager that will "see" the overall picture and decides the best possible cooperation scheme.

We assume that each of those peers trusts each other and has the same criteria of solution optimality and group manager selection. The only difference it can be found in the resources available to each unit separately. Among agents in a group, the initial phase of cooperation includes the discovery of all available peers. Then, an exchange of information on available resources will take place between all of them. Based on this information, each unit will compute to find the agent eligible to manage and further define the optimal cooperation scheme.

The managing agent is the one involved into the next optimal solution processes with the highest probability to achieve the foreseen task. This can be defined according to the resources it has in that exact moment. The manager agent first notifies all other agents of its role acceptance then it implements an algorithm to find an optimal solution for group activities.

This algorithm relies on resources of the group and is not requiring any extra communication between group members. Once a solution is found, the manager agent notices all listed units whether they are involved in the optimal solution or not. If an agent must take part in a foreseen solution, it will "know" the overall cooperation mechanism between agents in that scenario (set of units of the system involved, sequence of their cooperation and resources that they should give to others or get from them). Group manager will have a short-term list of solutions that were proposed and refused by other units. Newly selected manager unit will start tracking such list from scratch.

To update the list of available resources, group manager will search for optimal solution until it can find any. As far as a manager agent is having its interests in group cooperation and it will be able to carry out its role, so it will keep its main task. As soon as a group manager refuses its role or due to a sudden occurrence of malfunction, all other agents choose another manager by restarting the negotiation process with discovery and information exchange phase.

Other units can join existing group at run-time by notifying the manager agent. Eventually, after the manager agent finishes its current cycle for optimal solution discovery, it will report the solution to involved agents and then it will inform all agents of negotiation process to restart. From the discovery phase, recently arrived agents will take part of the new cycle.

As shown in figure 1, depending on the resources available for each involved agent in phase I, agent (3) was elected to match the service requests among interested agents. In phase II, and after the involvement of new comers, agent (5-7), the manager agent has decided that agents in light-gray circles (4, 1 and 6) including himself can all build a chain of mutual interest. Then, the mismatching agents have decided to restart, and then the same initial process was repeated in phase I-B where agent (2) with resources = 7 was elected to manage the matching process that involved new comers too, and resulted in phase II-B.

4. A CASE STUDY

Introducing the application we developed to examine our negotiation protocol relies on defining first Bartering as well as Swapping. Barter is a disappearing type of trade where

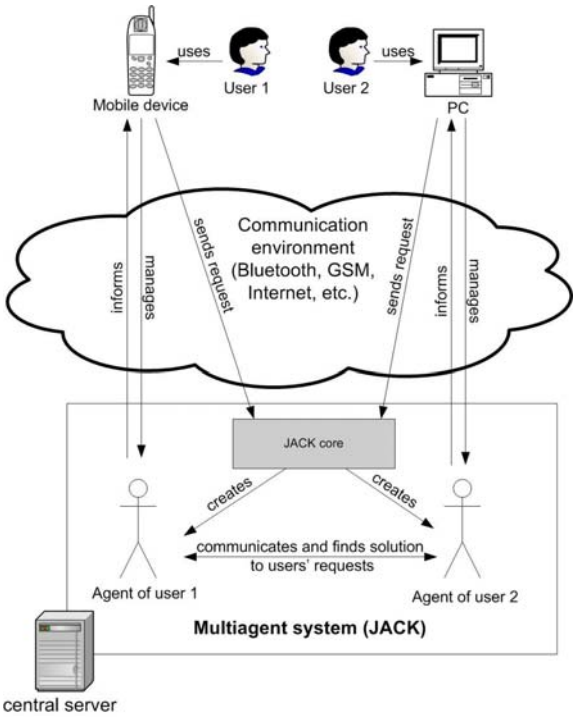

Figure 2: The architecture of BarterCell.

none of the recognized monetary systems are used in exchange of products or services, and only items of similar value are exchanged. Swapping is the modern approach to replace the ancient bartering services with websites that encourage users of computing devices to build virtual communities and share similar interests. BarterCell is our approach to provide users of recent and portable computing devices a barter service on the go. Based on the location and characteristics of a specific community, BarterCell would use agents to build the chain of exchange connecting several frequenters of the same area.

4.1 Motivation

The three key motives behind the development of BarterCell are: 1) reviving the idea of bartering within members of a specific community and promoting the benefits of location-based services, 2) to test the new negotiation protocol we introduce later on in this paper, 3) motivating existing users of pocket devices, and attracting new ones to benefit from recent advanced technologies by widening the range of services that can be offered to them on the go.

4.2 Architecture

The architecture of BarterCell, as shown in figure 2, relies on the user's capable pocket devices or PC to accomplish a successful bartering task. Via the preinstalled java application, users start by creating their own profiles using simple and user-friendly interface, insert their preferences, and add details related to the kind of items they are exchanging. If a PC is used, the user will be asked to directly upload the saved data to a central server which, in return, make it available to agents running on Jack [5], which is an interactive

Algorithm 1: BarteringService Builder

```
1  currentAgent = agent.ID
2  while (currentAgent.isAlive) do
3    cDList = cOList = currentMDItem = MOItem = ChainMaker = NIL
4    currentChainDecision = optimalChain = NIL
5    agentsList = get_available_agents()
6    for all a_i ∈ {agentsList - currentAgent} do
7      send(a_i, currentAgent.offers, currentAgent.demands)
8      cDList = updateCommonDemandsList(a_i.demands, cDList)
9      cOList = updateCommonOffersList(a_i.offers, cOList)
10   end for
11   currentMDItem = findMostDemandedItem(cDList)
12   dG = findMDGivers(currentMDItem, cOList)
13   MOItem = findMostOfferedItem(cOList)
14   oS = findMOSeekers(MOItem, cDList)
15   ChainMaker = ChainMaker(cDList, cOList, currentMDItem, dG, oS)
16   if (ChainMaker == currentAgent) then
17     runChainMakerService(agentsList, cDList, cOList, currentMDItem, dG, oS)
18   else
19     T = initTimer()
20     while (agentProvidesService(ChainMaker, T))
21       optimalChain = getResults(ChainMaker, T)
22       if (currentAgent ∈ optimalChain) then
23         userCurrentDecision = sendResults(optimalChain, currentAgent.user)
24         currentChainDecision = getCommonDecision(optimalChain, agentsList,currentAgent)
25         if (currentChainDecision == "Yes") then
26           updateAgentODLists(optimalChain, currentAgent.offers, currentAgent.demands)
27           sendChainContactsToUser(optimalChain, agentsList, currentAgent.user)
28         end if
29         sendOptChainDecision(ChainMaker, currentChainDecision)
30       end if
31     end while
32   end if
33 end while
```

Figure 3: BarteringService Builder

platform for creating and executing multi-agent systems using a component-based approach.

Currently, our system is deployed using distributed Bluetooth access points that are all located within a specific environment (e.g., university). Due to some technology limitations, users are asked to be present within the coverage of any connecting spot in order to transmit their data files to the central server. Regardless of the methods used to transmit the data, the processing and sequence of system instructions from this point on will be the same.

Once received from a user, the message or file content is made available to the multi-agent system, thus it can create a delegated agent that carries the particular characteristics of each system user. This agent is identified using the Media Access Control (MAC) address of the device used to communicate user's data with the server. On behalf of users, agents start to interact, cooperate and negotiation with other system actors in order to achieve the predefined objectives in the given time frame. These objectives are related to particular bartering services, which make them complicated and hardly realized in real life scenario without involving sophisticated technology.

Among other benefits, JACK was chosen to handle all of the agent's interactions because of its ability to meet the requirements of large dynamic environments, which allow programmers of agents to enrich their implementations with the possibility to compatibly access several of the system resources. JACK has also made the communication language applied among involved agents with no restrictions, which made any high-level communication protocol such as KQML [9] or FIPA ACL easily accepted by the running architecture.

4.3 The algorithm

Upon user request, a personal agent will be activated and algorithm 1 is starts to execute. Agent will be registered in available multiagent system and will stay active until a suspension by its user.

The agent created uses these variables: list of demands for all agents of a given system (cDList), list of offers for all agents of a given system (cOList), ID of an agent that will make bartering chains (ChainMaker), most demanded item in a system at a given time (currentMDItem), ID of an agent running (currentAgent), list of all available agents (agentsList), set of agents which offers currently most demanded item (dG) and set of agents which seeks for currently most offered item (oS), set of agents that are able to make an optimal bartering chain in a given system at particular time (optimalChain).

Being activated, agent will try to get information of other agents in the system (Line 5). If all other agents will be already involved into process of chain creation, the newly arrived agent will get ID of the system's ChainMaker and notify of its desire to join to established group of agents. ChainMaker will finalize its ongoing computational cycle and inform all agents of service finish (see "ChainMaker Operation"). After this point all agents will start new cycle for search of optimal bartering chain.

The new cycle of bartering chain creation will start from discovery of all available agents in the system (Line 5) and creation of list of those agents. For each of those agents every other agent will send its demanded and offered items. Thus all agents of the system will have common demands list (cDList) and common offers list (cOList) (Lines 6-10).

Based on the list of common demands each agent finds the most demanded item in a given group of agents at given time (currentMDItem) and the corresponding set of agents that proposes that item (Line 12). Most offered item and agents seeking for it will be selected to further define ChainMaker (Line 15).

If an agent finds out that it must be the ChainMaker at that time (Line 16), then it runs "algorithm 2", accepting the role of ChainMaker and thus providing other agents with corresponding service. If the role must belong to another agent of the system, current agent will track responses from ChainMaker (Lines 19-31).

Tracking of ChainMaker's responses in addition to getting results will also include checking for service availability (Line 20). This function is designed for both parsing of messages from ChainMaker and checking whether it can carry out its role. Every time a ChainMaker finishes creating an optimal chain, it notifies both agents involved into it and those agents that will be out of it (in order to let all agents know the state of ChainMaker).

Timer initialization (Line 19) is done to check ChainMaker's availability by any agent that is not interacted for a definite period of time. If an agent will find out from response of ChainMaker that it belongs to an optimal chain (Line 22), it will ask its user to accept or reject given chain (Line 23). If proposed chain will be rejected, it will not be selected any more until current ChainMaker carries out its role. Newly selected ChainMaker will start building its own list of rejected bartering chains. Having a positive decision as for proposed optimal chain, agents will send to their users contacts of other users whom they should contact in order to make barter (Line 27).

Figure 4: ChainMaker Operation

There is a little algorithm - not showed here - that is used in-between for 2 purposes: for every agent it helps to define which agent is ChainMaker in a given system (and thus to wait for informing from it of an optimal bartering chain in a system) and for ChainMaker it helps to define the root node of bartering trees. As a result, this algorithm will give ID of an agent that must be ChainMaker in the system at time of running the algorithm.

In the second algorithm, the ChainMaker can start giving its service if it has non-empty list of own demands (Line 3). Provided that it has the list, Chain-Maker starts new computational cycle (Lines 3-56). The cycle starts with a search for new agents (Line 5) that might wait to join existing group of agents (that are in agentsList). If there will be at least one agent waiting to join, ChainMaker will inform all known agents of service finish (Lines 7-9). All agents, including new, will start negotiation process from the beginning ("BarteringService Builder").

If there are no new agents, ChainMaker will inform all agents of a new computational cycle, and then checks for optimal chains in queuedChains[] (Line 16) that it has proposed during previous computational cycles (if there were any). If at least one of user in some queued optimal chain has refused to barter in it, the whole chain will be considered as refused and it will never be proposed again by current ChainMaker as long as it will carry out its role. Refused chains will be stored in a refusedChains[] set that will be updated along with queued-Chains[] every time ChainMaker gets information of refused chain (Lines 18-22).

Accepted bartering chains will simply removed from queued-Chains[] (Line 23). After ChainMaker will have a list of available agents and a treeRootAgent, it will start building bartering trees. Each tree will begin from treeRootAgent with every child, representing agent that demands at least one item from list of its parent's offers. While analyzing every path on such tree the ChainMaker will find repetitions of agents, it will create a complete set of agents that can barter between them. The shortest possible chain will be recorded to chains[] that will consist of shortest bartering chains of 3 types (combinations of demand types): 1) Strict; 2) Strict + Flexible; 3) Strict + Flexible + Potential. The shortest chain selected is built for each corresponding combination (Lines 26-28).

If ChainMaker will succeed to find one shortest chain at least, it will select the optimal from chains[] (Line 30). Optimal bartering chain will consider its length and combination of demand types it's based on. Considering chain of equal length, the highest priority is given to a chain that will be based on Strict demands while the least priority is given to a chain that will be based on Strict + Flexible + Potential demands.

After selecting an optimal bartering chain (at particular period of time), the ChainMaker will inform all involved agents at (Line32). Each agent in the chain will have information such as which other agents are involved into proposed optimal chain, which items should be exchanged and corresponding contact information of users. ChainMaker will remove from common demands list and common offers list those items that will be in proposed optimal chain (and will be potentially exchanged later) (Lines 33-34). If one of optimal chains will be refused to be executed, ChainMaker will restore items that were involved into it (Lines 20-21). Every proposed optimal chain will be placed into queuedChains[] (Line 36) to further track whether it will be accepted by users or not. Every agent that will wait for results from ChainMaker and will not be involved into optimal chain will get a message "cycle finished" (Lines 37-39). This will be indicator that ChainMaker has finished computing optimal chain, during previous computational cycle that agent wasn't into it and new computational cycle will be started by the same ChainMaker. This message will cause every agent's timer restart to check chain making service availability.

If ChainMaker fails to achieve a goal, it will notify all involved agents (Lines 41-43). This message will cause the restart of negotiation process "BarteringService Builder". If optimal chain will consist not only of Strict demands items then the ChainMaker tries to make it so by changing treeRootAgent to the next most appropriate agent (Lines 47-48). In the rest of the algorithm, if there will not be any agent for current most demanded item, the next most demanded item and corresponding treeRootAgent will be chosen. If finished with the list of demands or a suspension message received from its user, the ChainMaker will inform all agents of service termination. Agents still interested in a bartering service will restart a negotiation process.

4.4 Testing BarterCell

To test our architecture we used a D-Link DBT-900AP Bluetooth Access Point that is connected to the university LAN through a standard 10/100 Mbit Ethernet interface. This device offers a maximum of 20 meters connectivity range with the maximal bit rate support of 723Kbps, and the possibility to concurrently connect up to seven Bluetooth-enabled devices. The same access point is authenticating pocket devices that have BarterCell previously installed in

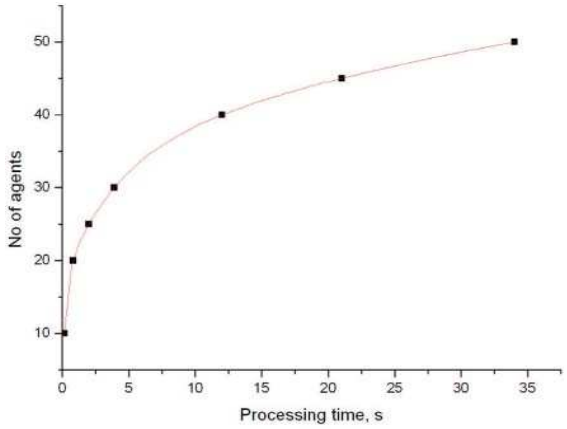

Figure 5: Simulating the number of Agents in BarterCell

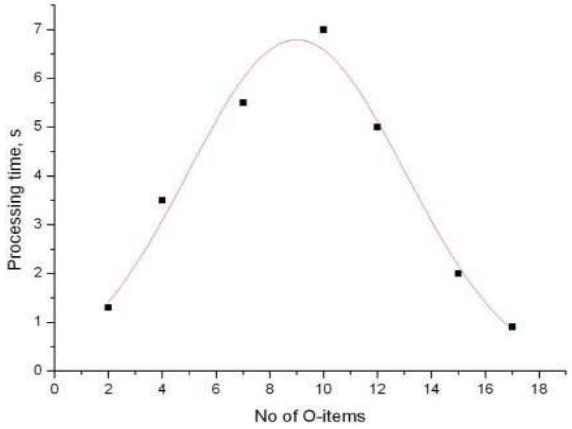

Figure 6: System Load Distribution

it and, it works as a deliverer of the service requests and responses from and to the central servers. On the end-user side, four competent cell phones were used to communicate semi-adjusted bartering interests with central servers. These de- vices are Nokia 6600, 6260, 6630 and XDA Mini. On the server side, a capable PC was used with JACK 5.0 and BlueCove installed in it.

5. EXPERIMENTAL RESULTS

In setting up our simulation, we chose to compare our protocol with the Strategic Negotiation Model [14] because of its approaches to address problems encountered in distributed data networks that are likely to occur in dynamic mobile environments. The model uses Rubinstein's approach of alternating offers [1].

In the strategic model, there are number of agents N = A1 An, and they are supposed to reach an accepted outcome on a delegated task within certain predefined times that are located in a set T = 0, 1, 2. At each time slot t of the overall process, the algorithm considers the results previously obtained to decide whether to allow another involved agent, at period t+1, to make a new offer. The protocol keeps

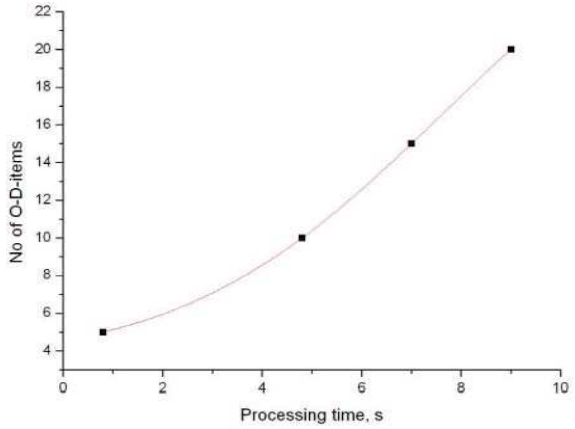

Figure 7: Simulating the number of items at each agent level

on looping until an offer is accepted by all agents and the proposed solution is then put into practice.

In our simulation environment, we have adopted and simplified the protocol in the following way: each agent searches for match of its every Ordered-item with every Demanded-item of every other agent. Adopted algorithm finishes working after each agent is able to define the other agent(s) that it can exchange with (a chain of length max. 3 agents).

As part of our simulation setup, we also used JDots for tree building that is object oriented software component. Each node of a tree was built with JDots is representing an object with its own fields and methods. Our algorithm works much slower with a huge amount of agents (e.g., >300) because the main agent needs to build three trees. Nevertheless, while testing with less than 200 agents, both algorithms are giving similar results in time, having variations in quality of results and further potentialities (e.g., work with object trees vs. dataset of matching agents pairs).

To obtain the results showed in figure 5-10, we have compared results of this protocol implementation with those received after implementing our (tree-based) algorithm that searched for first optimal chain of length 2. During the simulation we have put number of D-items to 15 and number of O-items to 5.

Figure 5 shows how fast the main agent finishes searching for possible optimal chains depending on the total number of known agents. Here, we assumed that the number of O-items is 5 and the number of D-items is 15.

Figure 6 shows how fast the main agent will finish searching for all possible optimal chains depending on number of items that each agent proposes. Total number of offered + desired items is constant (20). Peak of the graph represents the most time consuming state when number of offered items is equal to number of desired items. In this state the main agent has the biggest number of possible exchange combinations. Assumptions used, Max number of items: 20, Max number of D-items: 15, and Number of agents: 30.

Figure 7 shows how fast main agent will finish searching for all possible optimal chains depending on number of items at each known agent. In Fig. 4, comparison Results particular case, we assumed that the number of D-items = number of O-items, the number of D-items are only of "Strict" type

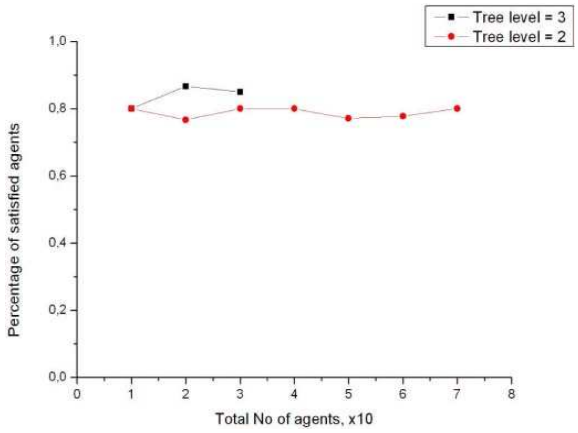

Figure 8: Agent Satisfaction Level

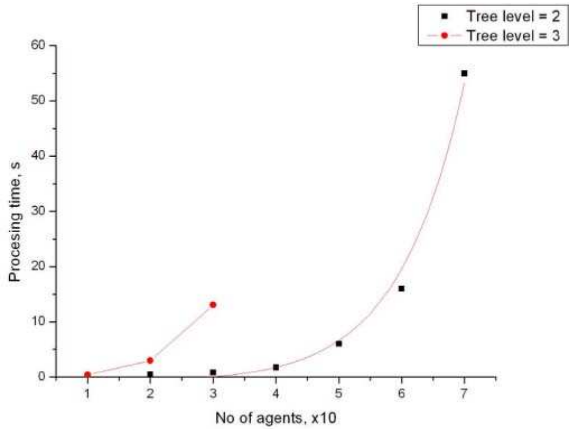

Figure 9: Processing time representation of agent satisfaction level

and the number of agents is 30.

Figure 8 represents how many agents would be satisfied (i.e. involved in one of optimal chains produced by main agent) until the main agent finishes all possible chain-building processes. Depending on the trees level (depth) the percentage of satisfaction will vary. Here, we assumed that the number of D-items is 20 and the number of O-items is 5.

In Figure 9, we show the processing time representation of the simulated agent satisfaction level of figure 8.

Once more, since we have chosen the Strategic Negotiation protocol [14] as a benchmark to evaluate and measure the performance of our algorithm with respect to existing ones, we have made some slight customizations for it in order to be fulfilling the minimum requirements of our application and thus comparable to our protocol. We first made each involved agent seeks to match all of its O-items with other agent's D-items.

The adopted algorithm finishes when each concerned agent has defined its completing agent(s), which it can make the bartering with (a chain of services exchange with a maximum length of 2 agents). Searching for the first optimal chain of length two, in figure 10 we simulated agent's satisfaction level by comparing the results obtained after implementing the strategic negotiation protocol to agents of BarterCell with our tree-based algorithm. We assumed that the number of D-items is 15 and the number of O-items is 5.

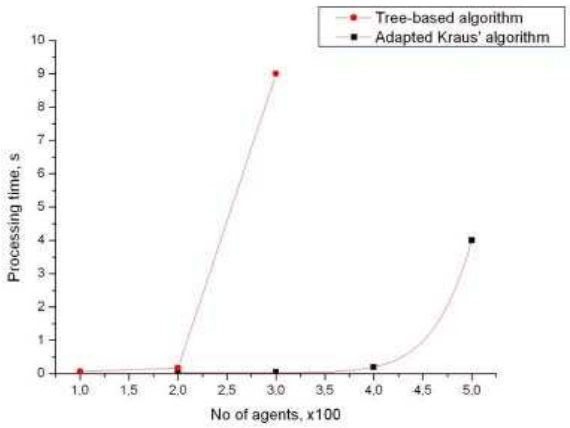

Figure 10: Abstract Comparison of the Different Negotiation Protocols

No of agents in a chain	Strategic Negotiation algorithm	Our Protocol
2	$O(n^4)$	$O(n^2)$
3	$O(n^7)$	$O(n^3)$
4	$O(n^8)$	$O(n^4)$
5	$O(n^9)$	$O(n^5)$

The table here compares our solution with the Strategic Negotiation model which made it easy to see how great difference in number of required interaction between agents is there. In our approach, interactions between agents are virtual, the same as we showed previously in BarterCell, we use one agent that builds chains of mutual interest agents. For example, for creation of a chain of length 3 it must make O(n7) interactions between agents of a given system, (n2) (n-2) (n-2) (n-1) (n-1) (n) where: O(n2) agents must communicate with each other to exchange the lists of resources. O(n-2) agents will communicate with their succeeding peers regarding resources that they can obtain from their preceding peers. O(n-2) if there will be at least one chain of length 3, then during this communication agent that initiated chain of resources giving will get back information of how its chain can be executed. O(n-1) that agent will report to every agent in the chain of coalition formation availability. O(n-1) every agent received message of the coalition formation availability will decide whether it will be in the coalition or not; decision will be sent to every agent of prospective coalition. O(n) all agents will send message that will initiate their chain.

6. CONCLUSIONS

The increasing efforts and interests in the development of Agent Oriented Software (AOS), and the autonomous behavior and level of intelligence Agents are now able to represent have made it clear that a great potential exists in using AI techniques to deliver mobile services. Negotiation protocols used among agents that are representing users of PC applications differ from those used for users of modern computing pocket devices. The changes observed on the behav-

ior of nomadic users, the level of dynamicity and the number of constraints applied on the hardly available resources on-the-move have led us to differently consider architectures of advanced mobile services.

We introduced a negotiation protocol for agents representing nomadic users and interacting in dynamic mobile information systems. We described the state-of-art of agent's negotiation in general and the attempts to enhance them in specific situations. We propose a new and wiser negotiation protocol. Then we use BarterCell that is an agent-based mobile bartering service application to examine the performance of this protocol. We simulate the behavior of agents in strict situations where time and network resources are limited. We adjust another negotiation protocol from existing literature, which is the Strategic Negotiation Model [14]. At last, we simulate the behavior of agents in the same bartering application, and we compare the results obtained in both scenarios

7. REFERENCES

[1] R. A. Perfect equilibrium in a bargaining model. *Econometrica*, pages 97–109.

[2] e. a. B. Blankenburg. Trusted kernel-based coalition formation. In *AAMAS'05*, July 2005.

[3] Carrillo-Ramos, Angela, and et al. Pumas: a framework based on ubiquitous agents for accessing web information systems through mobile devices. In *the 20th Annual ACM Symposium on Applied Computing (SAC2005)*, New Mexico, 2005.

[4] A. Chavez and P. Maes. Kabash: An agent marketplace for buying and selling goods. In *1st Int. Conference on Electronic Commerce, IECE98*, Soeul, Korea, 1998.

[5] M. Clark. Jacktm intelligent agents: An industrial strength platform. In *Multi-Agent Programming, ed. Rafael H. Bordini.*, pages 175–193, Soeul, Korea, 2005.

[6] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions.* The MIT Press, January 2006.

[7] K. D., C. Chatzidimitriou, A. Symeonidis, and P. Mitkas. Minformation agents cooperating with heterogeneous data sources for customer-order management. In *the 19th Annual ACM Symposium on Applied Computing (AIMS 2004)*, March.

[8] M. K. (Ed.). *Intelligent Information Agents.* Springer, 1999.

[9] T. Finin and R. Fritzson. Mckay. a language and protocol to support intelligent agent interoperability. In *The Proceedings of the CE&CALS Conference*, Washington, USA, June 1992.

[10] A. Inc. Going going gone! In *A Survey of Auction Types.*

[11] J. L. King and K. Lyytinen. *Information Systems: The State of the Field.* Wiley, 2006.

[12] P. Klemperer. Auction theory : A guide to the literature. *Journal of Economic Surveys*, 3(13).

[13] 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.

[14] S. Kraus. *Strategic Negotiation in Multiagent Environments.* MIT Press, 2001.

[15] H. Raiffa. *The Art and Science of Negotiation.* Belknap Press, 1982.

[16] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.* The MIT Press, 1994.

[17] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 12(C-29):1104–1113, December 1980.

[18] e. a. T. Scully. Coalition calculation in a dynamic agent environment. In *the 21st International Conference on Machine Learning*, Canada, 2004.

[19] J. Walker. *Mobile Information Systems.* Artech House Publishers, June 1990.

[20] T. Wanyama and B. H. Far. Negotiation coalitions in group-choice multi-agent systems. In *AAMAS'06*, May 2006.

[21] M. Wooldridge. *An Introduction to MultiAgent Systems.* John Wiley & Sons, 2002.