PhD Dissertation

International Doctorate School in Information and Communication Technologies

University of Trento

Department of Information Engineering and Computer Science

A Nomadicity-driven Negotiation Protocol, Tactics and Strategies for Interacting Software Agents

SAMEH ABDELNABY

Advisor:

Prof. Paolo Giorgini

Università degli Studi di Trento

January 2010

Abstract

The rising integration of pocket computing devices in our daily life duties has taken the attention of researchers from different scientific backgrounds. Today's amount of software applications that bring together advanced mobile services and literature of Artificial Intelligence (AI) is quite remarkable and worth investigating. Cooperation, coordination and negotiation are some of AI's focal points wherein many of its related research efforts are strengthening the join between sophisticated research outcomes and modern life requirements, such as serviceability on the move.

In Distributed Artificial Intelligence (DAI), several of the research conducted in Multi-Agent Systems (MASs) addresses the mutually beneficial agreements that a group of interacting autonomous agents are expected to reach. In our research, we look at agents as the transportable software packets that each represents a set of needs a user of a pocket computing device demands from a remote service acquisition platform. However, when a set of software agents attempt to reach an agreement, a certain level of cooperation must be reached first, then, a negotiation process is carried out. Depending on each agent's negotiation skills and considerations, the returns of each accomplished agreement can either be maximized or minimized.

In this thesis, we introduce a new negotiation model, (i.e., protocol, set of tactics, strategy), for software agents to employ while attempting to acquire a service on behalf of users of pocket computing devices. The purpose of our model is to maximize the benefits of the interacting agents while considering the limitations of the communication technologies involved and, the nomadic nature of the users they represent. We show how our model can be generically implemented. Then, we introduce two case-studies that we have been working on with our industrial partner and, we demonstrate these cases' experimental results before and after applying our negotiation model.

Keywords
[Multi-Agent Systems, Negotiation, Nomadicity, Pocket Computing Devices]

Acknowledgments

Authoring a doctoral dissertation is a long journey, longer than I expected, and I am in debt to many who have supported me along this journey.

It is with great respect that I thank Prof. Paolo Giorgini for his help and insight. I also would like to thank ArsLogica for sponsoring part of my PhD.

I would like to also thank members of my PhD committee for the time and effort they spent on reviewing my work and accepting to participate in the final ceremony: Prof. Marie-Pierre Gleizes, Prof. Andrea Omicini, and Prof. Aldo Franco Dragoni. Thank you for everything.

I am also grateful to Prof. Bruno Beaufils and all the members of the group of Systémes Multi-Agents et Comportements (SMAC) at the University of Lille 1, (a.k.a., University of Science and Technology of Lille), for the massive amount of understanding and support during the last and most critical phase of my PhD.

Throughout the journey, I have been fortunate to enjoy the compassion and backing of several friends: Olga Zlydareva, Rasha Nasr, Amr Youssef, Raian Ali, Mahmoud El-Gayyar, Sameh Ibrahim, Nasser Afify, Abdelhakim Fraihat, and my brothers Moataz & Amr. I am grateful to all of them.

Finally, I would like to acknowledge the indispensable contribution of my parents for without their love, sacrifices, and prayers, this research would have never been completed: Mohamed Abdelnaby Abdalla & Ahlam Ibrhaim Youssef, Thank You.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pocket computing devices (PCDs), such as Smartphones and PDAs, are increasingly showing the efficiency of relying on them and the importance of obtaining them. Recent advances on pocket computing devices are part of the communications' revolution that made it possible to virtually carry our offices anywhere we go. Nowadays, people are using different types of lightweight devices that allow them to check their emails, exchange faxes, surf the Internet, edit documents, do shopping, and play a role in a social network. Some of the responsibilities scholars of this time are taking into account to advance this revolution are summarized in making all these services interoperable, secure, robust, interactive, smarter, scalable, autonomous, and lighter.

Several recent market studies, such as [1, 2], and older ones such as [3], have showed that after the success of the Global System for Mobile communication (GSM) in late 90s, enabling cellular phones to access "*services*" of the Internet was the groundbreaking idea that positively affected the world's mobile phone penetration rate. Besides, particularly in [3], studies have anticipated that latest market statistics - *at that time* - indicates that unvoiced mobile phone services will approximately be worth USD 200b by 2010.

It is now clear that - *in general* - any service that can be electronically represented to end-users will also have to be also *mobilized* in the very near future. Therefore, the mobile services era that is ahead of us has made it clear that a new age of further innovative, intelligent and sophisticated service development techniques is required.

Agents' deployments in industrial and profit-making applications are continually growing and, related research are relatively expanding, for an overview: [4], [5], and [6]. Literature of Multi-Agent Systems (MAS) is witnessing the success of delivering advanced mobile services to users of computing pocket devices, (e.g., Kore [7], mySAM [8]). These applications apply several of DAI's approaches and take advantage of agent-oriented software engineering

methodologies to build goal and service-driven architectures that assist users on-the-go.

In the context of this thesis, when we mention the phrase "*on-the-go*" we tend to refer to the active style of users of pocket computing devices that make them wander from a place to another while always seeking a connectivity to interact with a certain remote service acquisition platform.

The ability for an agent to reflect the preference and intelligence of its user in virtual communities is making agents techniques better in increasing the portability of nomadic users. A number of agent development frameworks have recently included an extra feature to enable the development of mobile applications. For example, a recent release of JADE [9] - *one of the most used Agent frameworks* - has introduced limited APIs for mobile applications.

Scholars' efforts were also directed toward the integration of agent-based systems and peer-to-peer computing, and the returns both areas would gain from such integration. A survey on that direction can be found in [10], wherein the author is pointing out the frontier of these two research areas, MAS and P2P systems, and surveying the research efforts that have been made so far to address this issue. Moreover, in [11] an extension to that was recently presented to address the same potential integration but throughout lightweight computing instead. With Android [12] - *that is an open source mobile operating system running on the Linux kernel* - scholars of [11] have raised the significance of integrating JADE [9] and ANDROID [12].

From our experience at The University of Trento, we could also state that creating a range of interactive services that are further intelligent, independent and accessible by mobile and computer users is suitably realized through the use of autonomous agents and Multi-Agent Systems. However, extra enhancements, (i.e., new negotiation protocols and tactics), were required and will continue to change in order to meet the world's latest technology trends and requirements, plus, users' behaviors.

1.1 Our Research Context

In Distributed Artificial Intelligence (DAI), a significant part of the research conducted is focusing on increasing the level of cooperation achieved between agents that are located in distributed environments [13]. These agents can be perceived as the operating entities, (e.g., servers), that are randomly located in different spaces, or the platforms running on these machines, and recently, in our research it can also be perceived as the tiny descriptors that are produced by computing pocket devices to wander in virtual environments reflecting nomadic users' preferences and needs.

In an intersection between two of DAI's subtopics, Distributed Problem Solving (DPS) [14, 15] and Multi-agent Systems (MAS) [16, 17], our main focus comes in a place related to the quality, complexity, speed and amount of data or services that can be offered by these distributed agents with respect to the amount of resources utilized. Within this theme, several negotiation strategies were proposed by scholars that mostly aimed at the construction of a proper negotiation protocols, mechanisms, or tactics that agents may use to reach mutually beneficial agreement. An example of that can be the Strategic Negotiation in Multiagent Environments of Sarit Kraus [18].

The art of negotiation [19], and its attractive research arguments are always of great scholarly interest. Different science branches, such as political sciences and sociology, are analyzing and studying the best way humans and countries are discussing their opinions and exchanging their ideas with intentions to reach agreements. Several of the research activities in various sciences have intuitively demonstrated that different situations require different negotiation approaches to increase the potential for better overall discussion outcomes. Similarly, computer scientists are also alarmed with the importance of negotiation studies and explorations.

From the literature of Multi-Agent Systems (MAS), many research efforts have been approaching differently the problem of resolving complex situations among interacting agents by means of self-organization as presented in [20], and others by means of argumentation [21], and also by means of cooperation as presented in [22]. However, negotiation: another alternative for resolving complex situations among agents, is the focus of our research work, nevertheless, evaluating different approaches in a Nomadicity setting such that one we address is of great interest to us in the near future.

In multi-agent systems, several research efforts are addressing the negotiation of agents in different contexts and, for different purposes. Literature contributions, such as those of [23, 24, 18], are presenting negotiation models; (i.e., Protocol, Tactics, Strategies), that address specific situations wherein it is likely to have several heavyweight computing machines interacting with each others. It is then assumed that by choosing and applying one of these negotiation models, the interactions among all involved interacting components will be driven to simplicity and clarity in resolving complex situations.

Existing development approaches and implementation frameworks, which are agents-driven, are taking existing negotiation models into account they are not necessarily addressing the current changes of light computability and, they are likely to miss the support of Nomadicity [25], and its emerging requirements. Identifying the new requirements that take the nomadic character of nowadays users into account is an essential task, especially after the advances made on lightweight pocket devices and communication tech-

nologies. Considering these new requirements while designing a negotiation model that controls the interactions of these nomadic entities is another challenging phase on the direction of reaching Nomadicity.

1.1.1 Communications & Nomadicity

Having a reliable means of communication are costless with respect to the real value of the services offered to PCDs on-the-go, which encourages more people every day to demand more of the on-the-run kind of services. It is now evident that changes are about to happen in the way mobile phone services are perceived and contents are delivered. Consequently, changes have to be made on the protocols and techniques used to facilitate these foreseen interactions of software and people.

Several of nowadays technologies are connecting a large number of users located in various places to different services that are available in remote locations. However, the production of advanced communication architectures and the development of service applications are two twisted industries that are correspondingly advancing but, lately, in different speeds. The capabilities of nowadays cellular phones are greater than the application utilizing them, and limited to the interaction protocols currently available.

The vision of nomadicity [25] is increasingly bringing people's attention to the upcoming era of new scientific challenges. This vision has stressed the necessity to ensure anytime anywhere access to computing and communications and, the fact that we need to look at the disconnected state of a data demander as a common one and not an exceptional or failure. Same effort has also highlighted some of the technical challenges that we may encounter while architecting a nomadic-aware system. They have also approached the understanding of nomadicity by identifying a three-element checklist. Among others, this checklist has given special emphasis on the development of network protocols specialized in nomadic interactions, which come up to our scientific interests and to the scope of this thesis.

According to Kleinrock, [25], we are nomads by nature, and still we do need to be always connected, reachable and reaching. The information world is no longer seen through the traditional client / server eyes where wired PCs are exchanging data packets with fixed servers. Tolerable computing devices are spread in the pockets of almost every walkers of this universe. People move from an environment to another very frequently, and at each location they may move within a range of different spaces, and at every different end the means of computation and communications may vary. Consequently, technologies embedded in today's pocket devices are encouraging the realization of visionary goals and assisting the development of their supporting

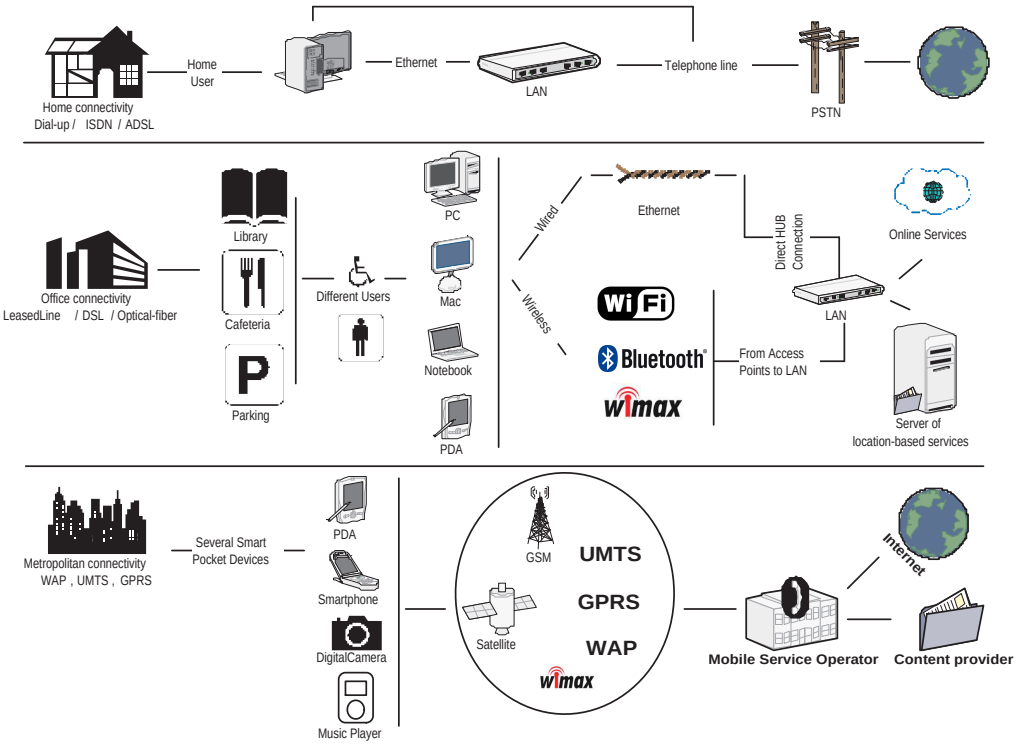


Figure 1.1: Nomadicity, nomadic devices, nomadic environments and nomads

applications.

The places we stopover, the pocket devices we use, and the connections we exploit, they all vary with respect to our schedules, motives, and interests. In figure 1.1, we show three different settings where possibly a user of a pocket computing device may daily experience. These scenarios or settings reflect the diverse range of connectivity and technological solutions that might be encountered to every one of us, and how significant it is to always use the right protocol and techniques that fit to each of these settings separately.

For instance, at home we might use one of the commonly found Internet connections, (i.e., Dial-up, ISDN), which are in turn rely on telephone lines and the Public Switched Telephone Network (PSTN) to finally reach the Internet and make use of several online services. In different time frame, we might also be located at a university's library and using different computing devices, which may also have a different operating systems and computing capability. This highlighted diversity might be normal if the available services are to be acquired from a fixed service provider. Meaning, it does not require major user-to-user interactions or any social interactivity.

Moreover, since a university is a wide open location with different cam-

puses, several technologies might be used to cover and connect different of its locations (e.g., Bluetooth or Wi-Fi). It is also possible to find users with special needs, (i.e., handicaps), which require particular connectivity methods and unusual service handling approaches. Still, all these users may choose to use a service that is location-based wherein the interaction of this location's members is an essential necessity to operate this offered service. A proper software model to the implementation of such an application will be required to capture and consider the requirements of each user and their different means of communications.

Home users and university students may also be considered as shoppers while walking around the city malls and shops. It is now very likely to see shoppers that are having pocket computing devices that are capable of establishing a connection with different networks to retrieve lots of information and make use of many available services. Therefore, a larger range of technologies are then used and an extended set of requirements shall be considered while designing the communication protocols linking all there users and services together.

User of pocket computing devices are also willing to interact, cooperate and also build their own social networks on the go. From agents perspective, it is then expected that each software agent representing a user of PCD is supposed to interact and cooperate. Therefore, the more agents interact the higher is the level of community's cooperation. Relatively, a significant role of the negotiation protocol applied among interacting software agents is realized.

1.1.2 Serviceability & Agents Negotiation

The negotiation language or protocol applied among software agents make them able to understand each other and discuss their needs and eventually achieve their objectives; interact properly. Therefore, when the interactions of a number of software agents are at a certain level of efficiency, the chances of their delegating users are higher to establish a desired connection or cooperation. As a result, different negotiation protocols have been proposed by scholars, these protocols are mostly inspired by sociological, political and psychological studies about human negotiation in real-life situations such as Auctions, Peace agreements and Bidding theories.

In [26] a definition was given to 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, [24], 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.

In [27], a service-oriented negotiation model was presented to handle the interactions of autonomous agents operating in a business process management application, which is a client / server communications. However, Research efforts to come up with a negotiation protocol that increases the serviceability of agents in highly dynamic environments are few. That shall make our research effort as one of the early steps taken on the way to tackle such an interesting and critical topic.

Negotiation protocols, from a technical perspective, are invisible for nomadic users; nevertheless, they play a significant role in elevating users' desires to continue relying on a particular service application. Negotiation, and accordingly coordination, among agents differs when the assembled service is to be delivered to computing pocket devices, (e.g., cellular phones, PDA). Among several constraints, time, network traffic and connectivity in mobile environments are hardly accepting the existing approaches to system entities negotiation.

Different circumstances accompany different MAS environments, and - *depending on the situation* - a specific negotiation protocol is chosen and applied on all system agents. Time, data transfer rate, general bandwidth constrains, bridge connection stability and security might not form great obstacles in computer based MAS implementations, as much as it may cause a failure for a certain mobile-based MAS application. Up to date, all of the influential and advanced negotiation protocols that are used in the development of agent-based mobile service applications are coming from computers or servers' environments or, databases and information systems applications.

1.2 Problem & Solution Statement

The problem we are addressing in this thesis can be summarized in the introduction of a new agent-based negotiation model, (i.e., Protocol, Strategies, Tactics), that considers the recently emerging nature of lightweight computing. While employed by a set of interacting software agents, our negotiation model is expected to facilitate the agents' mission of acquiring a service on behalf of the specific nomadic users they represent

The idea of our negotiation model is to: 1) reduce the number of messages exchanged among software entities that share common interest and interact to fulfill each other's demands; 2) increase these entities' confidence in the decisions they take prior to their service-driven actions; 3) advance the managing application's reliability, as well as users' credibility, by adding a sense of realism to the interactions occurring among the negotiating software entities; and, 4) bind these interactions' outcome to a tolerable responsiveness time. By putting together these motivating ideas we aim for:

I. Optimizing the utilization of resources, (e.g., bandwidth), in nomadicity-oriented settings.
II. Defining the circumstances associated with each of the decisions that software entities are entitled to take in order to meet users' preferences while acquiring a service.
III. Reflecting the level of flexibility a real-life trading situations are having on the interactions occurring among service-driven software entities.
IV. Making available a channel of feedbacks between the service benefiters and the software entities acting on their behalves.

We approach what we aim for by means of:

STEP 1: We give the negotiating software entities a service-aware space to state themselves. We do that by expanding the dimension of the expressive terminology our negotiation protocol supports, (i.e., accept, reject, pend, deposit).

STEP 2: We identify a range of decision-making strategies that the negotiating entities employ before considering one of the expressions provided by the applied negotiation protocol.

STEP 3: We provide to the negotiating entities the possibility to follow two extra service acquisition procedures that are inspired by the transactions' suppleness of the *Fleamarkets*. Particularly, the situations in which traders attempt to decrease their decisions' risk by pending or depositing a trade for a certain period of time.

STEP 4: We link all of the interactions happening among the negotiating software entities to two different timelines in which one imposes the times for actions, (e.g., terminate negotiation), and the other imposes the times for decisions, (e.g., pend).

1.3 Early Highlights

In this section, we briefly link between some of the Nomadicity requirements we discussed in section 1.1.1, our model's objectives we mentioned earlier in section 1.2, and how we tackled these objectives along our research work.

We would like to also stress on the fact that the number of emerging requirements *Nomadicity* is brining up to our concerns is certainly higher than what we outlined or mentioned in earlier sections. Therefore, with the presentation of our negotiation model we did not intend to address all of the requirements the nomadic nature of nowadays users is conveying, however, we did intend to cover few of them.

1.3.1 Reducing Exchange of Messages

We reduced the number of exchanged messages among interacting software agents and better utilized the resources of nomadic environment by giving all agents the possibility to better express themselves throughout a single action, and also defining the subsequent circumstances yielding from the use of each action separately.

Therefore, instead of agents *accepting* or *rejecting* an offer and then preparing counter-offers and, re-attach into a negotiation loop using the same negotiation fashion, agents employing our model will now be able *pend* or *deposit* a negotiation process reflecting how significant is the negotiation process for them, how large is their time-frames, and their desire to find better alternatives.

1.3.2 Reflecting Real-life Behaviors

In the negotiation model we propose, which we introduce further on in chapter 3 and chapter 4, we detached tactics from strategies to lead agents into the reflection of users' real-life behavior. We perceived the set of strategies a software agent is capable of employing as the global behaviors that each can give this agent a specific characteristic, (i.e., Friendly, or Loyal). On the other hand, we perceived the set of tactics we made available for an agent as

the number of attitudes this agent can imitate in order to support its global behavior, (i.e., become a connection-driven or time-oriented).

1.3.3 Building Channels of Feedback

As we explained earlier in section 1.1.1, and referred to in [25]: being nomadic require users to be faster in taking decisions because of the frequent change of locations they go through, and the different circumstances encountered at each location, (e.g., connectivity). The faster nomadic users are required to decide upon a certain issue the more supporting information they demand to ensure the correctness of their decisions, (i.e., a feedback).

In our research, we handled the high level of *responsiveness* nomadic users require from their service application on-the-go by ensuring the constant availability of a feedback that an agent is always ready to communicate with the user it represents. We did that by linking the negotiation session an agent is involved in with predefined timelines that requires the frequent reporting of an agent's current state.

Then, we made these timelines address two different situation: a) the situation where the number of nomadic users within a service application is *tolerable* and, b) the situation where the service application is *jammed*. Moreover, we linked these timelines with two different tactics that agents are permitted to employ while *time* is a main concern for the users they represent while *feedbacks* is also addressed.

1.4 Publications

1.4.1 International Journals

- Abdel-Naby, Sameh and Giorgini, Paolo. Negotiating Service Acquisition for Users of Pocket Computing Devices. **Submitted to** The *IEEE Pervasive Computing*.

- Abdel-Naby, Sameh and Giorgini, Paolo. A Nomadicity-driven Negotiation Protocol, Tactics and Strategies for Agents' Interactions. **Submitted to** The *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*.

1.4.2 International Conferences & Workshops

- Alexis Morris, Paolo Giorgini, and Sameh Abdel-Naby. Simulating BDI-based Wireless Sensor Networks. In the Proceedings of *The 2009*

IEEE / WIC / ACM International Conferences on Intelligent Agent Technology (IAT'09), September 15-18, Milano, Italy.

- Sameh Abdel-Naby, Paolo Giorgini, and Raian Ali. Towards Integrating Agents with Objects Tracing Systems in AmI. In the Proceedings of *the fifth European Workshop on Multi-Agent Systems (EUMAS'07)*, Hammamet, Tunisia. December 13-14, 2007.

- Raian Ali, Sameh Abdel-Naby, Antonio Mana, Antonio Munoz and Paolo Giorgini. Agent Oriented AmI Engineering. In the Proceedings of *the Ambient Intelligence Developments Conference (AmI.d 2007)*, Sophia Antipolis, French Riviera, France. September 17-19, 2007.

- Abdel-Naby, Sameh. Giorgini, Paolo. and Weiss, Michael. Design Patterns for Multiagent Systems to Elevate Pocket Device Applications. In *the 8th Annual International Workshop on Engineering Societies in the Agents World, ESAW'07*. NCSR "Demokritos", Athens, Greece. October 22-24, 2007.

- Abdel-Naby, Sameh. Giorgini, Paolo. and Fante, Stefano. Increasing Interactivity in Agent-based Advanced Pocket Device Service Application. In the Proceedings of *the Ambient Intelligence Developments Conference (AmI.d 2007)*, Sophia Antipolis, French Riviera, France. September 17-19, 2007.

- Abdel-Naby, Sameh. Giorgini, Paolo. and Fante, Stefano. Increasing Interactivity in Agent-based Advanced Pocket Device Service Application. In the Proceedings of *the Ambient Intelligence Developments Conference (AmI.d 2007)*, Sophia Antipolis, French Riviera, France. September 17-19, 2007.

- Abdel-Naby, Sameh. Fante, Stefano and Giorgini, Paolo. Auctions Negotiation for Mobile Rideshare Service. In the Proceeding of *the IEEE Second International Conference on Pervasive Computing and Applications (ICPCA07)*, July 26-27, 2007, Birmingham, UK.

- Abdel-Naby, Sameh and Giorgini, Paolo. Sweeper-Agent Recommendations Tree Early Scheme. In the Proceedings of *the Ambient Intelligence Developments Conference (AmI.d06)*, Sophia Antipolis, France. September 20-22, 2006. Pp 147-155. (Best paper award).

- Abdel-Naby, Sameh and Giorgini, Paolo. Smart Ride Seeker Introductory Plan. In the Proceedings of *the Third Starting AI Researchers' Symposium*, Riva del Garda, Italy. August 28-29, 2006. Pp 247-248.

- Abdel-Naby, Sameh and Giorgini, Paolo. ToothAgent: Brushing on your Behalf. In the Proceedings of *the 4th Industrial Simulation Conference (ISC2006)*, Palermo, Italy. June 5-7, 2006. Pages 49-51.

1.4.3 Technical Reports

- Sameh Abdel-Naby, Oleksiy Chayka, and Paolo Giorgini. BarterCell: an Agent-based Bartering Service for Users of Pocket Computing Devices. September 2009. Technical Report # DISI-09-053. University of Trento, Italy.

- Sameh Abdel-Naby and Paolo Giorgini. Locating Agents in RFID Architectures. Technical Reports # DIT-06-095. University of Trento, Italy.

- Sameh Abdel-Naby and Paolo Giorgini. Semi-Heuristic Negotiation Protocol for Agent-based Mobile Service Application. Technical Reports # DIT- 07-004. University of Trento, Italy.

1.5 Structure of the Thesis

CHAPTER 2: explains the related state-of-the-art. Since three different literatures are intersecting in order to construct negotiation models similar to the one we propose, in this chapter we give a literature review about the, 1) advanced Agents Negotiation in Common Settings, 2) Agents' Negotiation for Service Acquisition, and 3) Agents' Negotiation in Wireless Networks. In each literature, we give a comparison between the research effort of three different groups of scholars that are most related to our research.

CHAPTER 3: introduces the first part of our negotiation model. It starts by formalizing the general setting wherein we expect our model to be applied, then a description of the negotiation issue that any two interacting agents are expecting to resolve. We then formally introduce our negotiation protocol, negotiation sessions, and the negotiation timelines.

CHAPTER 4: introduces the second part of our negotiation model. The set of tactics and strategies, and their sub-categories, which we believe to have a great impact on the act of agents' negotiation in a modern service-acquisition environment.

CHAPTER 5: explains how our negotiation model can be implemented within any agent-based service acquisition software platform. However, within this chapter we do not restrict the implementation of our model to any particular application; instead, we attempt to prove its broad applicability.

CHAPTER 6: presents two case-studies we were working on with our industrial partner to provide a ridesharing and bartering services to users of Pocket Computing Devices.

CHAPTER 7: explicitly go through the experimental results we have obtained and we also emphasize the advantages of employing our model.

We conclude this thesis by outlining our future work and give a summary of all the research and development efforts we went through during the past four years.

Chapter 2

State of the art

Finding a proper negotiation protocol to be applied within autonomous agents representing users of pocket computing devices in service acquisition scenarios and domains is a new research direction. Several of the well-built approaches already presented by scholars of Distributed Artificial Intelligence (DAI) are not addressing the realization of advanced mobility requirements.

For instance, a commonly found consideration is always taking into account the communities of agents that perform a set of actions in situations where their communication channels are predefined and static. However, the literature of Multi-agent Systems (MAS) is starting to include approaches to the design of negotiation protocols that pay particular attention to the provision and acquisition of services on-the-go. In this chapter, we discuss the general perspective of Agent-based computing and Agent-oriented Software Engineering, then we emphasize the relationship between Ambient Intelligence and Agents Negotiation. Later in this chapter, we survey the literature of agents negotiation from three different research perspectives.

2.1 The Agent Paradigm

The Agent Paradigm was firstly dealt with within the literature of the AI community. Recently, and after the long hard experience of artificial intelligence, researchers could find other areas to exploit with great gains and returns the agent paradigm. Agent paradigm has received a special interest in software engineering community as a paradigm shift from the object-oriented one [28, 29].

The shift is based on seeing the world as a society of distributed intelligence units, called agents, that have characters and can decide. This way of viewing the world differentiates itself from the object oriented one that

conceptually view the world as a collection of objects. Objects provide encapsulation of data together with the procedures related, they are used by main well defined central control, and do not have their own autonomy.

Agent-based computing [30] is currently becoming an important research area. This increased interest is motivated by the need for software can act on behalf of its user, software that is able to realize the concept of agency. Giving a definition for agent is not straightforward; there is no consensus about the main characteristics an agent should have to deserve this name. A well accepted definition of software agent is found in [28]:

> *"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives."*

An agent is supposed to have its own control over its state and behavior, to percept the environment around and to affect it in turn. Being in an environment and sensing it implies the necessity that agent can react to environmental context changes. Moreover, agent is supposed to activate goals without external prompt and to tailor suitable plans to achieve them. The key characteristics an agent must have that are highly agreed upon, see [31] for an overview, are: autonomy, proactiveity, reactivity, situatedness, directedness, and social ability.

Being autonomous, an agent behaves independently according to the state it encapsulates. For example, an agent, by contrast to an object, can decide the way of how to respond to the incoming messages from other agents. Agents interact with each other without losing control if they do not allow that. Proactivity means that agent is able to take the initiative without external order. Agents have goals and act in order to achieve them. This is more complicated than reacting in timely fashion to direct environment stimulus.

Situatedness means the ability of agent to settle in an environment that might contain other agents, to perceive it, and to respond to changes that happen in it. An agent might make changes and effect this environment in turn. Directedness means that agent has a goal, this goal represents the reason of the actions an agent has to take. An agent does not exist in vacuum; instead it lives in a society of other collaborative or possibly competitive groups of agents. Agents have the social ability to interact with other agents. This interaction might be motivated by collaborative problem solving.

A long discussion can be found in the literature about what formulates an agent and what differentiates it from an object: see [32] for an overview. However, here, we are not concerned about such discussion, rather we believe that using *Agent* as a kind of abstraction approach enables us to view the

world as a large organization of autonomous entities, directed by goals, able to sense environment changes and can learn by time. In addition, agent-driven abstraction might help us also to analyze and design complex open systems, and presents more natural way to start with, and hopefully this will lead to more robust and flexible software systems in correspondence.

An agent is supposed to live in a society of agents; multi-agent system (MAS) is known as a system composed of several agents collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system. The relation can be alternatively competitive one, like for example multiple agents responsible for advertising products in an open market on behalf of different producers, or a society of agents in an e-auction. Again, defining MAS is not that straightforward. MAS might help us decomposing the problem into components that are able to interact and deal with unpredictable situations that can happen in complex systems such as those required to reach Ambient Intelligence (AmI) [33] and [34].

A MAS represents a natural way of decentralization, where there are autonomous agents working as peers, or in teams, with their own behavior and control. Each of these agents looks to the world from its own perspectives and has its own goals and intentions. Such MAS is expected to work well with open complex systems, and to scale well by time. It is one promising computing paradigm for implementing many application domains such as e-commerce, enterprise resource planning, and traffic control, and so on [35]. We consider AmI as a system that fits by its nature to agent and multi-agent system paradigm as we are going to discuss later.

2.2 Agent-Oriented Software Engineering (AOSE)

Software engineering is different from other engineering disciplines in its dependability on engineer skills of analyzing the problem, designing a suitable solution, and coming up with the final system [36]. Although software engineering is qualitative in nature, a serious research is being done to find more and more scientific methods, models, and criteria that assist developing the intended software. Problems are everywhere in software development process, engineering a software is an engineering for abstraction. For example, understanding precisely what a software is supposed to do and transforming this knowledge into abstract models readable by both of engineers and stakeholders is far of being easy as it seems.

According to [37, 38], large industrial software development projects has been encountering several obstacles due to the fact that the final outcome does not precisely meet expectations. The models used to describe software

requirements and design need to be compact and expressive enough to replace usefully the natural language. The models need therefore to be precise enough to not lose the real concepts they are supposed to represent. The models might be formal or transformable into formal ones, so reasoning can be done over them with the purpose of discovering any anomalies, incompleteness, or inconsistencies. Software engineering methodology is concerned not only about inventing and using modeling languages that can express what the system has to fulfill, and the software design, but rather it has to provide a process model for creating such models in turn.

Software agent that persistently observes the environment, interprets it, acts upon it, and communicate with other agents with other agent to resolve a task, all together are the building blocks of a promising computing paradigm for implementing open complex systems, such as Autonomic Computing [39]. AOSE methodologies, that is an influential factor in such visionary roadmap, tend to analyze and design such kinds of complex systems in order to finally arrive to an agent-based implementation.

There are several research groups working in developing their own AOSE methodologies, see [29] for an overview. The orientation towards agent does not mean that these methodologies use agency concepts and agent mentalistic notions along with all phases of developing software, rather the goal is to analyze and design in a way that leads to multi agent system. Only Tropos [40], as an AOSE methodology, uses the notions of agent and the related mentalistic notions from the early analysis down to the actual implementation.

For instance, the Gaia methodology presented in [41] takes agents' roles as the main issue of design and, with each role there is a number of linked responsibilities, permissions, activities, and protocols that define any role-to-role interactions. Another example for an AOSE methodology could be the SODA methodology, which is presented in [42], where the notion of *task* and the separation between *individual* and *social* issues are together playing a fundamental role in designing agent-based systems.

As the use of computing is becoming an essential part of individuals' daily life together with business and organizations, and as we increasingly need to combine between different computing ends and parties, the need for software that is dynamic, flexible, adaptable, situated is more critical. The need for software evolution is becoming faster than software development process itself. Solving these challenges is based to a large extent on the way such software has to be engineered. Agent oriented software engineering is attempting to generate methods, such as those mentioned in [43], that enable developing a software which can resist against evolving requirements, a software that is flexible enough to adapt and change according to the new

environments and requirements, the same concept as those explained in [44] and [45].

Agent oriented software engineering (AOSE), by contrast to object oriented software engineering and structured analysis and design, is not restricted or deeply influenced by some existing programming paradigm, [46] and [47] for argument discussion. Agent oriented software engineering research is now taking the initiative towards programming languages and infrastructures that serve the concepts suitable for software development instead of using those of existing programming languages in reverse unnatural way.

Being limited to programming languages has enforced those previous software engineering practices to focus on the solution domain, since the concepts used are not those describing naturally requirements and problem domain. Agent oriented software engineering is growing together with agent oriented programming languages, (e.g., AgentSpeak [48] and Jason [49]), and also with agent-based infrastructures, such as this one for Mobile Workforce Management in a Service Oriented Enterprise presented in [50]Dickson. This growth might fill the gap between problem and solution domains. Hopefully such consistency will make software development process faster, and lead to software can be easily evolved and maintained, and can adapt to different environments and requirements.

2.3 Ambient Intelligence (AmI) & Agents Negotiation

It is well known that agent paradigm is a promising paradigm for implementing complex open systems like e-commerce, air-traffic, enterprise resource planning, and so on [35]. The characteristics of these domains fit well to what agent and multi-agent systems can do. Software Agent is a software element that realizes the concept of agency, and acts on behalf of people or other agents.

One of the challenges that face building an AmI is the lack of models and software engineering practice that help analyzing system requirements, designing the system to be built, verifying and testing the implemented one, see [51] for an overview. Until now the research is in its first stages, and the need for suitable development methodologies has been already recognized. For engineering AmI, we might need different software engineering methods from those that are suitable for developing request/response systems, where system behavior is well known and determined strictly, and where human-computer interaction is desktop driven one. AmI shifts this way of interaction into contextual, direct, and invisible human environment interaction, hiding

the computers in the background of this environment.

The disappearance of computers and coupling environment appliances with computing devices will arise like any new technology a variety of challenges. The system domain is no longer some sort of business or organization has a clear business process and tasks. Users are no longer those clerks or students in a library system; instead users are now those normal people in houses, offices, campus and other daily life environment. The request/response scenario is replaced here by continuous sensitive, reactive, intelligence surrounding computing.

2.3.1 The Multidisciplinary AmI

Approaching an ambient that is perceptive, intelligent, and active will involve multiple disciplines to contribute creating the final scene. Several researches are being done in AmI area, with some differences in emphasis and direction. Multiple terminologies are being used as this research is in its first steps. In the rest of this section, we will investigate the vision of AmI, and try to capture a variety of disciplines that need to meet in order to achieve this vision.

Philips vision of AmI [52] is based on shifting computers into the background, and supporting the ubiquitous computing with more awareness capabilities. The vision is based on three elements, 1) the ubiquity, which refers to those computing devices intertwined with human environment anywhere, and functioning anytime, 2) the transparency of such computing systems, so they are hidden in the background, 3) and the intelligence; they should act instead of being only responsive to human commands. Such system relieves people of thinking about many repetitive needs and takes the initiative of doing what should be done in the correct moment and approach.

MIT vision of AmI [53] similarly views it as an unobtrusive integration of computing with our daily life. Such computing provides humans with relevant information and performs necessary tasks when needed on their behalf. Such ambient will be continuously careful, doing the suitable tasks in a transparent, invisible and intelligent way. Traditionally, computers work as an apparent messenger or mediator between humans and environment. In AmI, this relation is replaced by direct non-disruptive relation between humans and the environment they are located within. In short, AmI computing is no longer visible.

The vision of invisible disappearing computers was addressed by Weiser [54]. The vision expected ubiquitous existence of computing and communication capabilities anytime and anywhere. AmI focuses on assisting the intelligence and awareness of this ubiquity of interconnected computing de-

vices, so computing starts to take the initiative on behalf of human. AmI is meant to orchestrate the variety of environment objects in a way they might interoperate to do more complex tasks as well. Ubiquity of computing is the basis an AmI is built on.

However, the terms ubiquitous computing, nomadicity, pervasive computing, ambient computing, ambient intelligence are now used interchangeably with some differences in the context and emphasis.

AmI is now about integrating computing devices with the environment we all live in; it is then sitting on the opposite side of virtual reality which brings world inside computers [54]. This makes computers invisible and relieves people mind of even knowing about their existence. To arrive this point, computers has to adapt to user needs and character by contrast of the traditional scene in which user is supposed to adapt to computer systems. This is now of great importance because people spend increasingly more time to interact with computing systems.

To people, it is becoming a source of stress being obligated to remember when and what and how to do tasks. With AmI, artefacts encapsulate implicitly the role of computer mediation. Artefacts will look as they have their own character, autonomy, and intelligence, they are more agents than normal objects.

Consequently, AmI is by nature a multidisciplinary paradigm [55]. Distributed intelligence is needed to cover this intelligent ambient, it is now composed of distributed intelligence units that we might call Agents. New hardware design is needed for embedding computing devices invisibly inside the surrounding physical environment. AmI system is situated within a highly dynamic environment that is open for changes, these changes need to be sensed and interpreted in a way that is timely fashion and relevant to what might serve user needs. The input now is coming implicitly, and continuously from a variety of sensors, cameras, and other kind of peripherals. Such environmental information need to be modeled and reasoned about in order to take the correct contextual decision.

Computer disappearance was considered by Weiser as one of the most profound technology features [54]. Apart from the physical disappearance of computing devices, there is that mental disappearance toward peace of mind in human life. To achieve such peace of mind, the interaction between human and computer is updated to direct interaction between human and environment [56]. New novel ideas of interaction design have to be invented to move from the explicit interaction to an implicit one [57]. The implicit interaction includes the notion of implicit input known more commonly as Context [58].

Context awareness [59, 60] is an essential feature an AmI system has to

tackle in order to act in adaptive and intelligent way. This context, that might be spatio-temporal, environmental, personal, social, and so on, needs to be modeled, captured, analyzed and reasoned about [61]. Reasoning about context needs a model and formalization acts as a knowledge base, and enables inferring more high level knowledge. For example blood pressure and body temperature besides user current activity and location might reveal user current mood, this mood can be provided implicitly as an input, so AmI might take some actions as a response.

AmI is expected also to have the ability of learning and keeping track of human historical behavior. AmI embodies a high degree of personalization to human profiles and life styles. Software personalization is a standalone research now, but we might hardly consider AmI as a useful system if it behaves in the same way with different kind of people and characters. The social mobility of humans is another important issue an AmI application needs to consider. People normally play more than one social role; they should be accordingly supplied by tailored services and information considering their social context [62].

AmI arises many social issues that need to be studied and analyzed before AmI can get acceptance in practice. The ubiquity of computing might relieve people mind in one hand and might have negative impacts as well. People will feel that they lost control, and might not trust technology. People have already lost some privacy providing that cellular phones enable other party of at least knowing their location, and the same for using credit cards. Instead of commanding computing, computing in AmI is supposed to control several aspects of people everyday life.

An essential principle in this regard is that human do not feel that they lost control, and to enable them configuring their needs in a simple way, may be through some privacy patterns. However, we see many interesting practical domains that can benefit from AmI scenarios, such as the health care domain, in particular those specialized of caring old people, and supporting persons with dementia problems, where AmI might play the role of caregiver.

2.3.2 Agent-oriented AmI development

Employing the agent paradigm in implementing AmI scenarios is increasing due to the characteristics of agents and how well they fit in the context of AmI, as it was referred to in [63]. Examples to Agent-oriented approaches to the development of Ambient Intelligence applications can be found in [64, 65, 66]. Agent paradigm as a kind of abstraction is also capable of giving a good contribution with regards to AmI systems development, including analysis and design phases, besides the security issues. In this section we will

state our initial view of the agent oriented AmI engineering and securing.

As we explained previously, AmI shows a degree of complexity and multiple inter-related disciplines that require using special engineering paradigm. This need is coming from the new nature of such systems, where behavior is not known in details, or adequately controllable. AmI is distinguished by its dynamicity, openness, and complex inter-relations amongst environment components.

The agent paradigm, with respect to practices from object oriented software engineering, offers a higher level of abstraction suitable for engineering complex systems [67]. Agent paradigm enables engineering software at the knowledge level; at this level we talk of mental states, of beliefs instead of machine states, of plans and actions instead of programs, of communication, negotiation and social ability instead of direct interaction and I/O functionalities, of goals, desires, and so on [68].

Tackling the complexity of developing complex software can be done through some techniques such as 1) Decomposing the problem into smaller sub-problems that can be managed more easily. 2) Using abstract models to represent system focusing on some concepts and relations, and omitting others unrelated. Such models should be compact and expressive in order to usefully summarize and even formalize what can be alternatively expressed by the natural languages. 3) Defining and managing the inter-relationships between problem solving components as they were an organization of some hierarchy [69].

As shown in [36], agent paradigm is not only useful as software construct but rather it can be used as a new way for analyzing and designing complex systems. Using the decomposition, abstraction and organization techniques to tackle the complexity of such systems can be done following agent paradigm from the early phases. Decomposing complex systems into related subsystems, each with its own thread of control, and own objectives to be achieved autonomously can be seen as a society of interacting agents. Agent paradigm provides a sort of abstraction to model problem domain in terms that are too consistent with solution domain. Subsystems are viewed as autonomous agents, agent social ability implies the interrelation at high level amongst those autonomous subsystems.

This interaction might model cooperation, coordination, or negotiation amongst agents. The evolution of inter-relations between components of complex systems and the different aggregation these components can be classified at different levels of abstraction match closely to agent and multi-agent system paradigm. As for the dynamic organization structure, agent paradigm has the expressivity to represent these concepts due to its explicit structure and flexible mechanisms. A methodology called Gaia [41] was developed to

reflect such ideas providing a methodological way for engineering some kinds of complex systems, which is Similar to many AOSE methodologies currently available.

Another attempt for using agent paradigm as conceptualization construct is based on BDI agent architecture, the world is viewed as a society of actors each has its own autonomy, and might depend on each others for task to be performed, goal to be achieved or resource to be provided [67].

Agent beliefs are the world model at the conceptual level, agent desires are translated into goals to be achieved, while the intention an agent might commit is considered as a plan. The multiple plans an agent might follow to achieve the same goal give some degree of flexibility for dealing with different contexts. Goals are analyzed through means-end analysis to conclude the actual actions by which goals are achieved. These actions are the actual requirements of the intended final software [70]. Tropos is another methodology was developed on the basis of these ideas, it uses agent mentalistic notion along all the phases of software development [40].

For engineering AmI, like for example smart campus, we need to decompose it into autonomous subsystems, and to abstract using knowledge level conceptualization rather than the fine grained one used by OO which is useful for predicted behavior and relatively static systems. With AmI we are not talking about an organization with one well defined behavior, business process, and straight control. Here the ambient is always changing and in an unpredictable way sometimes, so we need high degree of adaptability to cope with AmI going to serve everyday life scenarios with a lot of alternatives. Considering AmI as complex open system, we believe that agent paradigm and agent mentalisitc notions can contribute well for analyzing, and designing AmI scenarios rather than only implementing them.

2.4 Agents Negotiation in Different Contexts

Getting back to the main contributing literature of this thesis: since three different literatures are intersecting in order to construct negotiation models similar to the one we propose, in the following subsections we give a literature review about the, 1) advanced Agents Negotiation in Common Settings, 2) Agents' Negotiation for Service Acquisition, and 3) Agents' Negotiation in Wireless Networks. In each literature, we give a comparison between the research effort of three different groups of scholars that are most related to our research.

2.4.1 Advanced Agents Negotiation in Common Settings

In this section, we go through and also put into comparison some scholars' contributions on the direction of introducing a proper negotiation models for agents interacting within an environment that is static and computationally potent.

In [26], agent's negotiation protocols are addressed with respect to a hierarchy of three abstract domains; 1) Task Oriented Domains, where agent's activity is a set of tasks to be achieved, 2) State Oriented Domains, where an agent is moving from an initial state to a set of goal states, 3) Worth Oriented Domains, where agents evaluate each potential state to identify its level of desirability. Several examples were given to further illustrate the different negotiation types that agents may encounter within these domains. Even though the three domains are likely to cover all common scenarios, still all of the protocols presented, the assumptions made, considerations and the given examples are outlying from mobility and service oriented domains.

In [18], a particular focus was given to agents interacting in modern distributed information retrieval systems arguing that the cooperation of information servers relatively increases with the advances made on agents negotiation. Two scenarios of agent's negotiation are considered; 1) Negotiation about data allocation, where autonomous agents / servers are sharing documents and they need to decide how the best they could locate them. 2) Negotiation about resource allocation, where the main focus is given to domains of limited resources as well as those of unlimited ones and, agents are bilaterally negotiating to share expensive or common resources.

Based on Rubinstein's model for alternating offers [71], the negotiation protocol presented in [18] is straightforward. An offer is made by one agent to another that has to choose between accepting, rejecting or opting out of the negotiation process. Each agent has its own utility function that evaluates all possible negotiation results, and a strategy to decide what actions to perform at every expected situation. Although the negotiation of agents about the allocation of limited resources is similar to agents interacting within a low capacity communication network (e.g., Bluetooth Network), the limited options each agent has in response for an offer, limiting the negotiation process to bilateral situations and, avoiding the role of end-users in all scenarios will make this approach inspiring but not principal to our research.

The Contract-Net protocol presented in [24] is a high-level negotiation protocol for communicating service requests among distributed agents. R. G. Smith considers the high-level negotiation 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 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, or session is closed.

Three slight drawbacks in the earlier approach;

1. Linking between high-level and low-level negotiation protocols is essential when it comes to agents interacting in limited and variable resources environment. For example, when users of pocket computing devices delegate software agents to exchange and accomplish service requests on-the-go, the efficiency of the negotiation protocol agents will employ is relatively increasing with the size of bandwidth a network utilizes, and the time it takes to transfer agent's requests / messages from location to another.

2. A central decision making situation may easily occur when a service seeker initiates the call for proposals and, it receives back all of the prospects offers and, the same gent is the only one who decides upon the termination of the negotiation process.

3. In the Contract-Net it is always assumed that two different types of agents are interacting (e.g., buyer and seller agents), which is not right for us. The agent representing a user of a pocket device is taking the selling role when there are services to be offered for others and, the buying one when the end-user is searching for something to acquire.

With table 2.1 we conclude this subsection showing that; 1) the assumptions scholars are making while designing wide-ranging negotiation protocols are coming out of particular domains influence, which make them appropriate only in similar settings. 2) The examples and case studies authors used are addressing familiar but not explicit problems when it comes to the serviceability of pocket computing devices and its requirements. 3) Although it is important for autonomous agents to maintain a level of heuristic and collaborative performance, still the three efforts have a common acceptance on the isolation of each negotiation process. 4) The idea of designing a new negotiation protocol for agents representing users of pocket devices in acquiring and providing services on-the-go can be considered as an expansion to the above mentioned efforts.

2.4.2 Agents Negotiation for Service Acquisition

In this section we further tighten our literature analysis to scholars' efforts that are made to introduce a new negotiation models for agents interacting

Table 2.1: Agents Negotiation in Common Settings: A Comparison

	[Smith, 1981]	[Rosenschein and Zlotkin, 1994]	[Kraus, 2001]
Agents	Loosely coupled asynchronous nodes containing a number of distinct knowledge-sources.	A program that electronically represent a person or a machine in different encounters.	Intelligent computer systems with automated behavior.
Environment	Task-sharing (cooperative task execution)	Task-oriented domains, State-oriented domains, worth-oriented domains.	Data & Resource Allocation
Objective	Coming up with the best way to allocate tasks among distributed nodes that have enough processing capabilities.	a) Allow machines able to make constructive agreements, b) make computers interact flexibly, represent our interests, and compromise when needed.	a) Increasing the cooperation of distributed information servers so that data are better stored. b) Efficiently managing the use of limited resources among agents with common interests.
Assumptions	Nodes are interconnected. They communicate by sending messages. No memory is shared. A low-level communication protocol preexists to support efficient and reliable communication of bit streams between nodes.	Designers are building their agents to maximize expected utility. Each negotiation process is isolated. Agents may commit to a common utility. The cost of each operation an agent takes is independent from the agent carrying it out.	Agents always try to maximize their utility. Agents don't prefer to opt out of a negotiation. Agents are committed to whatever results a negotiation process will lead to. Negotiation processes are neither related nor connected.
Protocol	a) Managers (nodes) make a task announcement. b) Contractors (nodes) select those they like, evaluate and submit bids. c) Managers evaluate bids and make contracts with the best fitting.	Each agent has its own space of possible agreements that they simultaneously propose. Agents agree if they match utilities or exceed it, and negotiation moves to another round if not, forbidding agents to offer lesser utility.	An offer is made by one agent to another that has to choose between accepting, rejecting or opting out of the negotiation process. The negotiation ends if all the agents accept the offer or if one decides to opt out.
Example	A network of sensor and data processing nodes spread in a large geographic area. Nodes with high processing capabilities are looking for sensor nodes that provide signal features, and the vice-versa, until a task called "signal" in a distributed sensing system is achieved.	Two researchers collaborate to photocopy chapters out of a book taken by one of them. Group of neighbors agree to organize a carpool. Two agents that are trying to organize a meeting.	a) Data and information system component of the earth observing system (EOSDIS) of NASA. b) Mobile Robots sent to Mars by NASA (i.e., sharing limited resources).

to acquire one or more predefined services. However, these efforts were yet considering the traditional wired environments and their requirements.

In [72] a service-oriented negotiation model for autonomous agents was presented. Following the traditional client/server approach scholars assumed

that an autonomous agent is a "client" to another serving agent "server" that is in turn delegated to achieve a certain goal, which is acquiring or selling a service. It was also assumed that all agents are operating from servers with outstanding computing capabilities and, no considerations were given to the connection linking them. Authors have focused their research on the reasoning model an agent will employ to identify its prospective servers, deciding about whether to perform parallel or sequential negotiations, making or accepting an offer and, abandoning a process.

Same authors have weighted their approach with respect to the British Telecom (BT) business process of providing a quotation for designing a network to provide a service to a customer. Accordingly, times to reach and execute an agreement were both considered while identifying their negotiation model. Based on a model for bilateral negotiation that was presented in [19], authors of the service-oriented negotiation model proposed a multilateral variation of it to satisfy the application domain they are interested in, which bring it up to our curiosity as well. However, the requirements they attempted to satisfy (i.e., privacy of information, privacy of models, value restrictions, time restrictions and, resources restrictions) are not covering something like the end-user responsiveness time and end-user dynamicity that are vital for agents representing users of pocket computing devices and operating within variable connectivities.

In [73], an agent-based architecture for service discovery and negotiation was presented. The realization of three novel requirements have motivated the work of these authors, these requirements are; 1) Agents interactions are not necessarily executed in one network and not only involving two types of agents. 2) Diverse connection technologies can be utilized at different costs, which increases the complexity of a system and enable a higher level of end-user dynamicity. 3) A service application should automatically react to changes as long as it is of end-user benefits. The scenario they used to motivate their work involves three different agents. The user agent, which is located on the portable device of the user and, it contacts a marketplace agent that is installed at wireless hotspot location and it is responsible of maintaining a list of available Internet Services Providers (ISPs) that each has a representing agent called ISP agent.

An agreement is reached when the user agent succeeds to make a contract with one of the ISP agents and retrieves a configuration file that, eventually, the end-user installs on its pocket device to get an Internet access through the best available ISP. The sequence of interactions among involved agents was described but, authors did not impose any additions or intelligence to the already existing negotiation mechanisms and, instead, this issue was left very generic by means of FIPA Contract-Net [74] or a FIPA-EnglishAuction-

Protocol [75]. Therefore, agents in this situation are still using the normal propose, refuse, and accept model. The time an agent takes to decide upon an action, the reasoning functions it employs before a decision is taken and, the communication channel inconsistency are all unconsidered elements throughout this research effort.

It is also worth highlighting here that four different types of auctions are widely considered in the literature of agents; 1) English, 2) Dutch 3) First-price Sealed-bid, 4) Second-price Sealed-bid, (e.g., in [76]). 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.

A multi-agent negotiation method for on-demand service composition is presented in [77]. Agents here are expected to negotiate in order to reach agreements about combining different services from different providers to finally meet a consumer's expectations, which is not a handheld device user. The negotiation process is functioning by means of messages exchanging. When an invocation for service acquisition occurs it is assumed that all available agents are representing specific services in a network and, they receive a message that contains a set of requirements to fulfill. If a single agent is capable of providing this service on its own it broadcasts a OK message, if not, it transmits a Nogood message to the others asking for help. Other agents receive this help request and review there capabilities and give a response and so on.

Using messages exchange to negotiate the acquisition of a service between entities located in a fixed wireline network is likely to be satisfactory. However, in a wireless network, where bandwidth and resources are expensive and limited, this approach will add a considerable amount of traffic and, it will increase the time a service application takes to act in response to an end-user request. In addition, it was not observed in same author's work any level of end-user interactivity, which drift this effort on the machines orientation side of negotiation's research.

From table 2.2 we observe the following: 1) Agents are not robots in a warehouse, distributed servers in information systems or even large network nodes; they are a number of interacting encapsulated programs that represent an end-user interests or an organization goals. 2) Scholars approaches to the negotiation of software agents in service-oriented domains changes from these of resources sharing and data allocation. The focus is more into making agents self-interested and benefits maximizers than being cooperative -

so that service requests are perfectly satisfied. 3) Efforts that may have indirectly touched the requirements evolved along with the new era of mobile serviceability are rare and not dedicated, and if any, a maximum of one or two common issues are addressed (e.g., time and resources utilization).

2.4.3 Agents Negotiation in Wireless Networks

In this section we go through the few research efforts that were made to improve the negotiation of agents interacting through wireless networks to achieve delegated goals. These research efforts are the least far from the research issues we tackle since we also consider agents negotiations across unwired limited resources networks but for service-driven interactions.

In [78], an architecture for pervasive negotiation that uses multi-agent and context-aware technologies is proposed. The main focus of this research effort is to consider clients profiles, their preferences and locations so that personalized services, promotions and offers are transmitted to their handheld device. In their article, authors have looked at pervasive negotiation considering it as the "negotiation that is conducted everywhere, at anytime, by any devices, and by anyone" and, an example for a car driver that uses a PDA to find the best price a nearby petrol station can offer is given. Authors have also assumed that all of the involved devices are constantly capable of establishing a wireless connection.

In this article, three different agents are involved in any negotiation scenario, these agents are; 1) User agents that announce the preferences of the end-users they represent and wait for responses. 2) Supplier agents that take into consideration the preferences and context provided by a particular user agent to compete for service provision. 3) Negotiation agents that maintain all of the allowed negotiation strategies and mediate between the earlier two types of agents. To allow context-aware negotiations, a Global Positioning System (GPS) to track end-user moves is operating on top of the multi-agent platform.

Therefore, it was also assumed by authors that all of the handheld devices are GPS enabled and, a web browser is available on the client side of the application to facilitate the task of profile editing and parameters change.

The negotiation mechanism proposed by these authors is mostly influenced by semantic web approaches. The negotiation agent (i.e., the mediating agent) is coming up with the best fitting strategy in particular situation by parsing a negotiation ontology. These ontology is flexible and interactive so that end-users and service providers are able to present their own negotiation conditions.

Intelligent information agents [79] are those capable of interacting with

Table 2.2: Agents Negotiation for Service Acquisition: A Comparison

	[Faratin et al., 1998]	[Bircher and Braun, 2004]	[Cao et al., 2005]
Agents	Client/server software representing an entity in acquiring a service	Software that represents an end-user interacting with another representing a company's service through a third mediating agent.	A software entity responsible for a choice of a service and its invocation rather than actually offering it.
Environment	Business Process Management Application.	Interoperable Heterogeneous systems.	On-demand web services composition.
Objective	Guiding an agent through the initiation of an offer, evaluating incoming proposals, and generating counter proposals by means of a comprehensive reasoning model for service oriented negotiation.	Realizing a marketplace for temporary Internet service provision through a mediating agent and service agent's interactions.	Adjusting Agents negotiation techniques to bring together different service sources and finally respond to complex service requests made by end-users.
Assumptions	A single service is available from multiple sources. Each agent can be either a client or server at a time. Agents may be required to make trade-offs to reach agreements. The social context influence agent's behavior. Agents do not know anything about each other (e.g., utility function).	FIPA OS is installed on each participating device. A new agent has to log its profile in the management system and directory facilitator of FIPA OS. At least two wireless service providers are available at each hotspot.	Each agent keeps information about a set of services. Each service process is an independent workflow. Service agents have to cooperate to achieve any given task. The knowledge of each service agent is divided into three parts; basic, constraints and social knowledge. Message exchanging is the language agents talk using 1st sending 1st arriving.
Protocol	Agents define the set of variables they will negotiate about. The negotiation between two agents starts as series of alternate succession of offers and counter offers relating to the values of the predefined variables. The negotiation continues until an agent terminates the process or an offer is accepted by one of them.	A user agent contacts the local marketplace agent asking for a list of available service providers. It uses the Contract-Net protocol to make a contract with one of the suitable service agents. If agreed, the service agent sends the configuration file of the negotiated service to the user agent.	Each agent sets priorities and values for the service variables it operates according to local constraints. Each agent uses an OK message to broadcast the new values of local variables to other agents. If another agent agrees on these values it establishes cooperation. An agent broadcasts a *Nogood* message if it cannot satisfy a constraint.
Considered Requirements	Privacy of information, privacy of models, value restrictions, time restrictions & resources restrictions.	The interacting entities are running on different computers and connected over the Internet. The seller entity supports service discovery, negotiation & manage different network connectivity.	Services composition to increase web knowledge and information sharing. Constraints that web services may have are all of equal importance.
Example	After receiving a service request, six agent types representing six different divisions in British Telecom interact to produce a price quotation	A user intends to move to a place where several Internet access providers may grant his computing device different services at different costs.	Online travel planner that integrates services provided by different entities such as airline companies, hotels and credit card companies, and that requires several agents' cooperation.

several distributed or heterogeneous information systems representing end-users in obtaining data and overcoming information overload. In [80] authors have relied on information agents, data acquisition agents and rule-based reasoning agents to build a multi-agent system (MAS) 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 [81] 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.

2.5 Related Work

While considering the Internet as a highly dynamic environment, negotiation protocols used in e-commerce or data allocation applications are inspired by actual auctioning mechanisms [82], others by the notion of contracting [24] and even politics and economics [83].

In [73], an agent-based architecture for service discovery and negotiation was presented. The realization of three novel requirements have motivated the work of these authors, these requirements are; 1) Agents interactions are not necessarily executed in one network and not only involving two types of agents. 2) Diverse connection technologies can be utilized at different costs, which increases the complexity of a system and enable a higher level of end-user dynamicity. 3) A service application should automatically react to changes as long as it is of end-user benefits.

A multi-agent negotiation method for on-demand service composition is presented in [77]. Agents here are expected to negotiate in order to reach agreements about combining different services from different providers to finally meet a consumer's expectations, which is not a handheld device user. While in [78], an architecture for pervasive negotiation that uses multi-agent and context-aware technologies is proposed. The main focus of this research

effort is to consider clients profiles, their preferences and locations so that personalized services, promotions and offers are transmitted to their hand-held device.

Coalition formation is another approach to Agents Unioning. In [84] model for coalition formation was proposed to enable each agent to select individually its allies. The model supports the formation of any coalition structure, and it does not require any extra communication or central co-ordination entity. Similarly, the definition of an optimal coalition in [85] is based on Pareto dominance and distance weighting algorithm. Besides, in [86] trusted kernel-based mechanism for making a coalition rather than on efficiency of a common task solving.

In [87], a model for automated negotiation is proposed for mobile agents to achieve complex tasks in mobile web commerce, which is repeatedly a client to server approach. They introduced a definition of a user and purchase profile that are used mainly in supporting the buying decisions prior to actual product purchase. However, they have assumed that all the heavy computation is performed in the fixed network, by the *CallApplication*, (i.e., one of the assumed linked applications that is responsible of communicating product info with end users), and the agent platform. Similar to our approach, authors of this paper have considered the fact that users are always required to be connected to the managing platform while the negotiation taking place.

Coming from a Wireless Sensors Network (WSN) background, in [88], they proposed an approach for developing autonomous agents that has an economic behavior that make them able to negotiate and independently take rational economic-driven decisions. Two sceneries for an autonomous agent to negotiate a service were presented: 1) presents the negotiation involving the transportation and communication services with the Adhoc network, 2) represents the usage of the WSN to the proposed agents. However, it is clear that no users or pocket computing devices are involved at any point of their description and, their agent is not a software, it is a WSN node with limited batter and computing capabilities within a network.

From Electronics, another approach to automated agents negotiation and decision making in resources-limited environment was presented in [89]. Scholars here are perceiving a multi-agent system as a set of distributed radio antennas of a cellular network and, by means of agents' *contracting*, these agents would share the use of resources, (e.g., bandwidth). The approach is quite remarkable from *Antennas* negotiation perspective, yet, it does not address our focal interactions of nomadic users, their delegated software agents, pocket computing devices, communication channels, and the agents platform.

In [90], authors argue that by using reinforcement learning an agent will

then be capable of employing a set of strategies that respond to the opponents' with various concessions and preferences. The protocol authors propose is of two actions, either and agent will respond giving a deadline for reaching an agreement, or a complete rejection. The main focus of the two experiments they explained is how an agent learn a behavior while still keeping its functionalities at the same level of performance, which the call it: "*not breaking down*".

Here, it is also worth highlighting that: 1) most of the research efforts we referred to in this section are of recent years, which is reflecting the novelty of this research field - *Agents Negotiation in Unwired Service Acquisition Networks*. 2) Since the deployment of software agents in mobile service applications is increasing slower than those of Web applications, it is then expected to have bigger range of improvements for online agents negotiation than those we tackled.

2.6 Chapter's Summary

It is clear that an increasing consideration for approaches, theories, implementation techniques, and development methodologies from the literature of multi-agent systems in deploying practical applications can be observed. It is also clear that the world's focus now is shifting towards lightweight applications, mobility, and services that can be of use to users on the go. Therefore, it is intuitively foreseen that in order for agents' contributions to keep on influencing the implementation of modern service applications a shift towards mobility and its supporting instruments is required.

In this chapter, we have surveyed in details the current state of the art of agents' negotiation. We have started from the wide notion of ubiquity and everywhere computing while showing the significance of the agent paradigm in between. Then we have showed and compared the most influential research efforts on agents negotiation in ordinary computing networks, (i.e., wired computing). Then approached agents' literature from another angel, which is related to the scenarios where a specific service acquisition is required. From that aspect, after not finding much in that field, we have picked and compared three different research contributions while highlighting the case-studies they have relied on to address their motivations, which had nothing to do with lightweight computing or unwired networks.

The last part of this chapter went through the state of the art of what we believe to be very close to our research contribution; Agents negotiation in wireless networks. Scholars' effort on that direction are still hardly observed even though the overall actual trend of developing multi-agent systems are

driving concerned developers toward mobile services development and integrations.

It is also worth highlighting here the reasons we decided to produce this thesis, which are:

1. Introduce a survey of the current state-of-the-art that links between agents' negotiation, service acquisition, wireless communications, and AmI. By introducing this survey we aim at: a) highlighting the importance of considering a dedicated negotiation models for agents representing users of pocket computing devices, b) the distinction between negotiations in wired networks and negotiations in unwired networks.

2. Explain our various practical experiences on designing, developing, and deploying service acquisition multi-agent systems for users of pocket computing devices.

3. Summarize the lessons learned from these earlier experiences by introducing a new negotiation model for agents that represent users of pocket computing devices. By this model we address and overcome the obstacles encountered while putting agents' approaches into modern practices.

4. Referencing the results we obtained with respect to the lessons we learned, the negotiation model we came up with, and also with respect to the existing negotiation models. By doing that we aim at clarifying the advantages and disadvantages of studying our overall research efforts and outcomes.

Chapter 3

The Nomadicity-driven Negotiation Model

In this chapter we introduce our negotiation model. It starts by defining the general abstract setting wherein we expect our model to be applied then a clear description of the negotiation issue that any two interacting agents in the context of our research are expected to resolve. Then we characterize the negotiation parties - *Agents*. We then formally introduce the negotiation protocol and its related time management approach.

3.1 The Model's Abstract Setting

Different communities of agents may address different concerns in which some of these concerns may be the result of combining two or more sub-concerns. In our research, a concern is the abstract concept a community is continually supporting as long as certain objectives are better accomplished. Besides, a sub-concern is a generically narrower concept of a community's abstract concern, which is satisfied whenever a specific set of objectives is being achieved.

For instance, a group of robots in a warehouse might be concerned with placing all of the received objects in dedicated spaces, but a sub-concern emerges when a subgroup of these robots is concerned with organizing - *only* - the north part of this warehouse. A possible sub-subconcern occurs while two robots of the north-part subgroup are concerned with organizing the red objects only. However, robots operating in a warehouse together with the robots operating in a nearby automobile manufacturer are forming a *society* of robots.

Definition 1. ***A Agents Society****: is a set of agents located in a space wherein different interests' agents are encountering.*

As definition 1 outlines, and figure 3.1 depicts, when a group of agents come into a common space and, within this group; a number of agents are assigned to completing different abstract concerns, together they form what we call an agents' society. To better elaborate on this, we should think of an agents' society the same as we think of all robots in factories of a specific industrial zone. For example, the industrial zone in Milan has different factories that each has a number of operating robot agents, therefore, all agents in all factories of Milan are forming the Milan's society of industrial robot agents, even though each of these robots is having tasks with different natures to achieve.

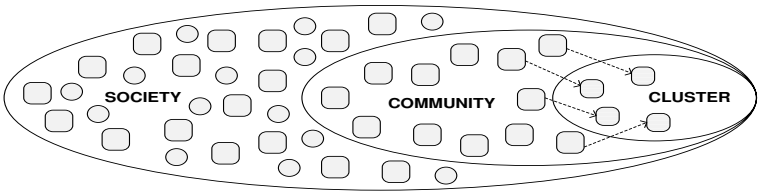


Figure 3.1: Agents' Society, Community, and Cluster.

We breakdown a society into sub-societies in definition 2. So, within all factories of Milan, robots involved in car manufacturing, and those of washing machines production, together they are forming two different communities of robot agents, but yet they both belong to the Milan's society of industrial agents. However, the classification of societies and their communities are affected by the perspective a problem is tackled from. For instance, from a different perspective, industrial agents of north Milan can also be considered as a society by itself, and every set of similar robots can form a society's possible community. This classification can also be made according to robots colors, types, or names, and so on.

Definition 2. ***A Agents Community****: is a subset of an agents' society where a common interest is shared among all of its participants.*

In a community of agents, if a group of agents come into agreement about completing a sub-concern of their community's abstract concern, then we call this group an *agents' cluster*, (Definition 3). For an agents' cluster to be formed, the achievement of a common task must be shared among this cluster's parties. Meaning, this cluster's parties are uniting to achieve a task, (e.g., two agents in a warehouse: organizing boxes). However, for a union to occur, prospective agents must first agree on forming this union, and for agents to agree they must negotiate. Negotiation between agents in order to unite and become one of the possible clusters in a community is the broad scope of our model.

Definition 3. ***An Agents Cluster****: is a subset of an agents' community wherein all parties have came into a mutually beneficial agreement that satisfies their predefined needs.*

The total number of concerns a specific society addresses can be descendingly placed on a pyramid of concerns. In figure 3.2, this concerns' pyramid has a society's very abstract concern on top of it, then this concern's sub-concerns in less-abstract levels. However, these sub-concerns can also be considered as abstract concerns for sub-communities that play different roles inside the larger community. Depending on the size and number of responsibilities a community has, breaking down the concerns into sub-concerns can be carried on within N levels of descending abstraction, until the least community of a society is defined.

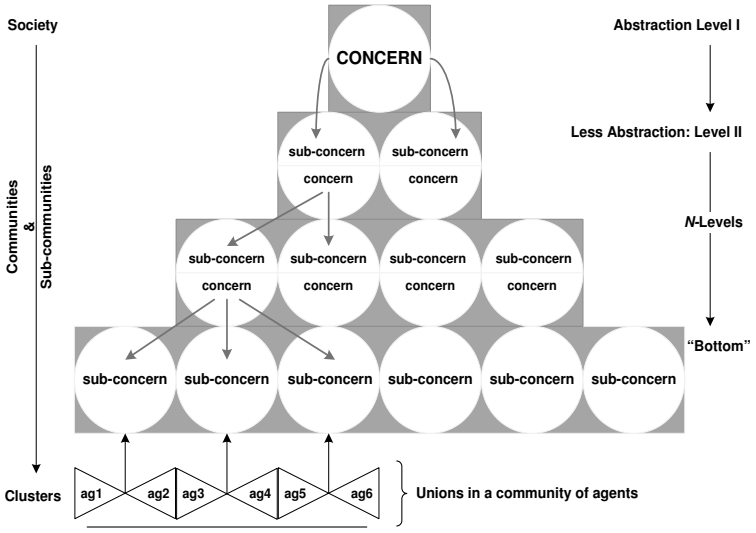


Figure 3.2: Community's concerns, sub-concerns and agents' unions

Since the least sub-community in a chain of a larger communities is the one that has an abstract concern and a set of indivisible sub-concerns. Then - *at the bottom of our pyramid* - any set of sub-concerns that is linked to an earlier level concern are, together, reflecting a specific community where agents' clusters may exist. Depending on the number of involved agents, different sizes of clusters may exist within a community. However, in our model we consider a specific type of clustering in which only two agents are involved, which we call it a *Union*, which we define later in this section.

Example 1. A University, A Nearby Factory, and The Province.

People working for the same university may each have a different role, but

together they form one community that satisfies the university's main concern, which is providing education. On a higher level of abstraction, the university's community and the community of a nearby factory, together, they form a new community of workers, which satisfies a different concern, such as developing the province. However, a professor or more of the same university may have their own community-related concerns, (e.g., become a dean, get funds). Therefore, in a community, each played role can be associated with new concerns that are different from the community's concern, but yet are subs of it, *sub-concerns*.

In order for this professor to satisfy his personal concern - *community's sub-concern* - a set of actions needs to be taken, (e.g., write proposals, dine with key contacts). However, each of these actions would cost its taker something in return. For instance, a professor would tradeoff some of his time with the action of writing a project proposal. Since different actions are likely required to satisfy a single concern, then the value of the tradeoff associated with this concern is expected to increase, (e.g., lots of time), or a number of different tradeoffs would emerge, (e.g., time and money). Therefore, a community's sub-concern is associated with one or more action(s), and a cost that is equal to one or more tradeoff(s). Thus far, we could possibly say that the set of sub-concerns a community searches to satisfy corresponds to a set of tradeoffs members of this community are ready to do.

At a certain time, a professor's insistence to submit a project proposal, and his lack of tradeoffs (e.g., no enough time), may push him to negotiate with his colleagues the idea of establishing a union. Then, the professor might have a chance to tradeoff something that he does not lack at this particular moment, (e.g., fund sharing). In this situation, the negotiation process a professor would carry out is affected by a set of subsequent requests that he previously prepared to eventually ensure the satisfaction of his personal concern. For example, do you have a spare time? When? Do you work in groups? Do you currently face funds problems? And so on.

Example's Conclusions: for every community of people there is a main concern. For every member of this community there is a number of personal concerns, which are also sub-concerns of a community's main concern. There is a cost for every action taken by a member to satisfy a community's sub-concern. This cost is a set of tradeoffs a member will do in order to perform this concern's related actions. Consequently, there are two possibilities for a community member to satisfy his concern, either 1) all tradeoffs are available and given by this member unaccompanied, or 2) a mutually beneficial union is established between a group of community members in order to make all

concern's tradeoffs available.

Making the later possibility occur will require the existence of a negotiation process between a member and a potential union partner. A member's decision whether to accept a union formation will depend on the responses he receives to the requests he asks to the same union's potential partners. Therefore, a set of requests corresponds to the tradeoffs a member is willing to do while taking place in a union.

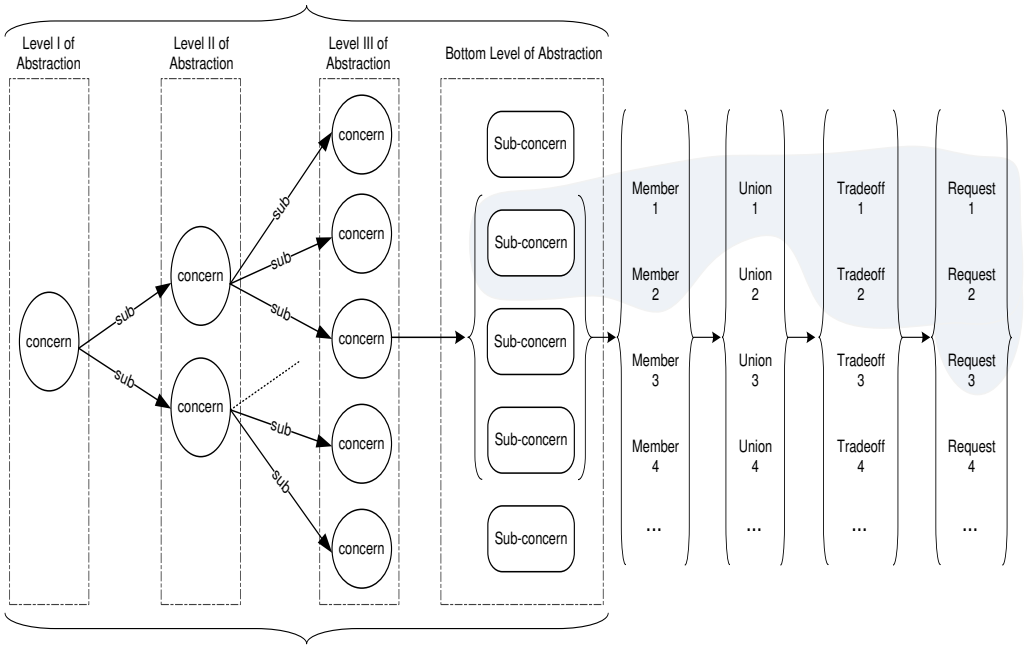


Figure 3.3: Community's concern, subconcerns, members, unions, requests & tradeoffs

3.1.1 A Community, its Concern, and Sub-concerns

In this section we further elaborate on the relationship between a community's concern, its sub-concerns, members, unions, and correlated requests and tradeoffs.

As figure 3.3 depicts, every community's concern can be described by means of a number of of smaller sub-concerns wherein each of these sub-concerns can be either directly assigned to an entity that is committed to satisfying it or, each sub-concern is in turn divided into a less-abstract set of sub-concerns. Therefore, assuming that there are N concerns, $\mathbf{Concerns} = \{C_1, ..., C_N\}$, that are distributed among an M levels of abstraction, $\mathbf{Levels} = \{L_1, ..., L_M\}$ in which every $l \in Levels$ represents a class of concerns that is

more abstract than its subsequent one, then, elements of the **Concerns** are distributed among all **Levels**. Therefore, the number of concerns at L_1 is less than, or equal to, the number of concerns at L_2, and the number of concerns at L_2 is less than, or equal to, the number of concerns at L_M.

Consequently, the top level of abstraction, $L_1 \in$ **Levels**, contains one main concern. The bottom level of abstraction, $L_b \in$ **Levels**, contains the set of concerns, (i.e., sub-concerns), that cannot be broken-down. As a result, as shown in figure 3.3, every subset of concerns are associated with one concern of its earlier level of concerns, which is one possible bottom edge of a concerns' hierarchy. Hence, this bottom edge represents also a society's specific community, its main concern, and its members' sub-concerns.

3.1.2 Members, Unions and Tradeoffs

Definition 4. ***A Union****: is the agreement of two agents of the same community to fulfill part or all of each others requests.*

In every community there is set of members, **Members** $= \{M_1, ..., M_s\}$. Every subset of **Members** represents a community's possible *union* - Definition 4. Since in this thesis we assume that a community's sub-concern can be achieved by means of members' unioning, then, all of the community's sub-concerns are associated with all possible unions that can are likely to be established in a community, **Unions**.

Therefore, a $u \in$ **Unions** contains a subset of two **Members**. Members of every possible union will be trading off something that they have in order to eventually satisfy the sub-concern they are assigned to. Consequently, a set of tradeoffs is associated with all possible unions within a community.

Definition 5. ***A Tradeoff****: is a part, or the whole, of what an agent is ready to give in exchange of fulfilling one or more of service requirements.*

In order for a member to agree about trading off one or more of his belongings, and then establish a union with other members, he needs to guarantee his share of the union benefits. Therefore, a potential union partner will attempt to maximize his union benefits by negotiating the fulfillment of a set of requests with the other potential partner of the same union. The requests of this member, the requests of other potential union partners, and all requests of a community's possible unions' partner, together, they will form a community's set of requests, **Requests**.

Definition 6. ***A Request****: is one of the characteristics describing the general demand an agent foresees fulfilled in a prospective union.*

We conclude the depictions of figure 3.3 by locating a possible relationship between a community's sub-concern, members of this community, the union these members may form, the tradeoffs they will make, and the requests each member will impose in order to ensure that a union's benefits is worth giving a specific tradeoff. This possible set of relationships are the highlighted area in figure 3.3.

3.1.3 Tradeoffs vs. Requests

Following the example we mentioned earlier in this chapter, (i.e., A University, A Nearby Factory, and The Province), we could possibly assume here that "*providing education*" is a main concern for a University. Therefore, a set of sub-concerns can also be taking into account, (e.g., hiring enough lecturers). The University, its main concern, and its sub-concerns, are all of a *community* that is part of a bigger picture, which is a *society*.

Formally, this can possibly look like that: *providing education* can be positioned at $C_3 \in$ **Concerns** referring to the fact that this concern is at the third level of abstraction within a *society*, meaning at $(L_3) \in$ **Levels** of abstraction. Therefore, the same university's sub-concerns, (e.g., increase research funds, increase classrooms), are located on the subsequent level of abstraction, $L_3 + 1$.

As described earlier while formalizing the model's abstract setting, in each community there is a set of members who share one common concern and, also unite to resolve this concern's sub-concerns. Then, within the community of a university, $M_1 \in$ **Members** can be a Professor that was assigned to satisfying the sub-concern "Increase research funds".

However, to better elaborate on this interconnected relations between Members, Concerns, Requests and Tradeoffs, we will assume that M_1 are lacking the time to achieve the delegated task. Consequently, M_1 attempts to negotiate the idea of establishing a union, $U_1 \in$ **Unions**, with M_2, so that the time they both have will be enough to write a project proposal and increase research funds. In this situation, time is tradeoff, $F_1, F_2 \in$ **Tradeoffs**, both members will have to exchange with the fact of writing a project proposal.

On the other hand, M_1 and M_2 cannot tradeoff their time unless union benefits are guaranteed for each, (e.g., fund commission, promotion, PhD students). Therefore, a community's sub-concern is completed if $M_1, M_2 \in$ **Members** fulfill their requests, $\{R_1, R_2, R_3\} \in$ **Requests**, by means of joining possibly $U_1 \in$ **Unions** while $F_1, F_2 \in$ **Tradeoffs** are the union conditions.

Definition 7. ***An Instance:*** *is the specific concern an agent attempts to complete by means of negotiating the establishment of a union with one of*

the same community's agents.

In order for us to keep our model focused on negotiation about a certain issue, we conclude this subsection by highlighting the distinction between any of the community's sub-concerns and the sub-concern that a specific agent addresses. We do that by using the word "*instance*" - *Definition 7* - to refer to the single concern a specific agent attempts to complete while playing a certain role in a specific community, and "*instances*" to describe all the same agent's concerns. Consequently, an "*instance*" is a concern for an agent but a sub-concern for its community.

3.1.4 Agents Societies and AOSE

The concept of representing agents as societies is also addressed in the literature of Agent-Oriented Software Engineering (AOSE). In [91], authors have addressed the fact that the correct representation of multi-agent systems should be done through the perception of agents' societies wherein global laws are followed to achieve proper interactions. Same scholars have also stressed the fact that this law-driven interactions will lead to improving the global behavior of the overall multi-agent system. Yet, the later research effort did not address the service-centric society of nomads we focus on in our research and, consequently, the motivating examples given are all of the client/server field of application, (e.g., Internet Services).

Different approaches were considered to model agents societies in a given environment. For example, in [92], relying on the Interactivist-Expectative Theory of Agency and Learning (IETAL), authors have proposed a model for a multiagent society based on expectancy and interaction. However, this later approach did consider neither the notion of Service-driven interactions nor the idea of agents Unions to fulfill mutually beneficial objectives, and scholars here assumed that all agents of a society are equipped with sensors that detect similar interest agents without getting into negotiations.

In [93], scholars have presented what they call the vision of open agents societies. They perceived the notion of agents societies as "*Flexible network of heterogeneous software processes, each individually aware of the opportunities available to them, capable of autonomous decision making to take advantage of them, and co-operating to meet transient needs and conditions*". Although in addressing their vision scholars of the later article have focused on wireless communications and technology, yet, they have highlighted a number of risks in their approach, such as: 1) the fact that the entire vision is built upon the existence of mutual trust between all of the society members, 2) the level of autonomy an agent will have is complete, therefor, it is assumed that no

upper agents interactions, (i.e., users involvement).

Last but not least, we would like to highlight the fact that our aim from presenting the notion of agents' *society*, *community*, *cluster*, *unions*, and service-centric communities is to facilitate the representation of our negotiation model, and precisely reflect the mental method we followed to prepare for our model formalization and introduction.

3.2 The Negotiation Issue

In this section, we use the broad description of the model's setting presented in the earlier section to identify the specific community wherein our negotiation model can be applied. We define the issue which two agents of this community are going address in their negotiation, and probably agree on its realization. Depending on the sets of requests each negotiating agent is seeking to fulfill, a deal between two agents can be reached under different conditions. Therefore, we then link between an agent's possible situations of acceptance with its requests. We conclude this section by putting together the sets of requests of an agent and the tradeoffs it is ready to offer and, linking them with the different cases wherein a successful negotiation may occur.

3.2.1 The Service-Centric Community

Definition 8. ***A Service-Centric Community****: is the set of agents interacting with the intention to fulfill the abstract objective of acquiring a predefined service.*

In our model, we consider the negotiation between two agents that are members of the same service-centric community, which we outline in Definition 8. Therefore, all agents are aware of the community's abstract concern / provided service, (e.g., dating, or ridesharing, or bartering). Depending on the kind of service a community is concerned with, a *union* between two agents reflects the completeness of a unique community's sub-concern, (e.g., date(john,sara)). Therefore, a community's sub-concern is created once an agent is searching for a union partner so that together they fulfill each others' requests.

In a service-centric community, we assume the existence of a central agent that we call it a *head-agent*, Definition 9. This *head-agent* is the managing authority of a community, (e.g., a multi-agent platform).

Definition 9. ***A Head-Agent****: is the central and managing member of a community, which is responsible of applying a community's common regulations.*

For example, within a community of dating service, although all agents are seeking to get a date and somehow pay for it, yet; the *head-agent* will be responsible of putting all male agents in one category and doing the same for all female agents. Besides, the *head-agent* will ensure that any male agent that searches for a union partner is actually looking into the category of female agents, and the vice versa. The same applies for a ridesharing community. All ride-giver agents will be separated from ride-seeker agents, even though both categories contain agents of the same type. By "*same type*" we mean; all agents are searching to acquire a service and give something in return.

It is worth highlighting here that the reason in our model's setting we drifted toward the existence of an agents' managing entity that is centralized, (i.e., a head-agent (e.g., agents platform)), goes back to the fact that the service-centric community we address in our research is more of an industry-driven provisioning of a commercial service. Therefore, a complete control and monitoring over the behavior and types of interactions software agents are performing is always necessary.

Assumptions

In traditional service provisioning approach, there has been always an entity that demands and another that supplies. These two interacting entities can be intuitively perceived as a *User* and a *Content Server*, or a set of *Supplying Machines* interacting with a set of *Demanding Machines*, and so on and so forth as long as the predefined objectives of both entities are achieved throughout their interactions.

However, to address the social ingredient of a service provisioning approach, the entity that demands and the entity that supplies should be of the same nature to become capable of establishing mutually beneficial relationships, (e.g., a DB server & a linked Web server).

Therefore, since in our model we are focusing on the interactions of two software agents that are virtually representing the social interactions of nomadic users to acquire a service on their behalf. Then, the demander and supplier here are of the same type - *users*, plus, the demander is also a supplier and the vice versa, so the foreseen relationship is mutually beneficial. As a result, in general, our model assumes that any possible objective of a service-centric community can be achieved by means members getting together - *unioning*.

Moreover, in this thesis, we consider a type of unioning that involves the agreement of only two software agents. The reasons behind taking that decision are: 1) it is more feasible to partition a service demand into a set of smaller services than a user into smaller ones, (e.g., searching for three dates can be easily divided into three similar but separate agents that seek three different unions). 2) We are also looking forward for the *one issue many parties* extension of our model instead of the current *one issue two parties* approach, and the transition is doable that way than others. 3) The case-studies that we worked on with our industrial partner are mostly of this nature - *P2P interactions*.

To summarize, in a service-centric community, we assume the following:

1. A sub-concern cannot be completed by one agent. Therefore, in order for each agent to realize any of its service *instances* a union with another agent, of the same community, is required. As a result, all of the sub-concerns a service-centric community has are completed only by means of unions between two agents.

2. Every unioning agent is not involved in completing more than a sub-concern at a time so that a full commitment to the current negotiation session is achieved. Besides, since our negotiation protocol permits the negotiation parties to *pend* or *deposit* their current negotiation session and look for alternatives, so concurrent negotiation is anyway achieved but in an indirect way.

3. All agents are of the same type; therefore, every agent that is willing to provide a service is also acquiring one, which is reflecting real-life situations between humans, (i.e., mutual beneficial agreements).

4. In order for two agents to complete a sub-concern of a service-centric community, they both will have to consume some elements that make them capable of completing these sub-concerns - *Tradeoffs*. For example, when two robots form a union to handle the red objects; they both accept to tradeoff their time and electrical power with completing this sub-concern. In a ridesharing community, when agentD and agentS form a union to complete a sub-concern, which is a car-ride; the driver trades off the place available in his car with the money a ride seeker is ready to pay / tradeoff.

3.2.2 The Service-Centric Issue

In a service-centric community, two negotiating agents will discuss the formation of a mutually beneficial *union*. Meaning, a successful negotiation should

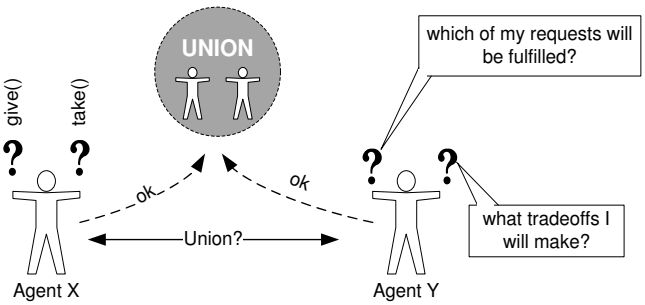


Figure 3.4: The negotiation *Issue* of a service-centric community.

lead agentX to employ a number of its capabilities (*Tradeoffs*) in order to satisfy a set of needs (*Requests*) agentY has, while agentY is doing the same for agentX.

Therefore, as figure 3.4 depicts; for two agents to complete their *instances*, they must unite. For a *union* to occur, agents must tradeoff something that they are capable of providing with the completeness of these *instances*. In order for two agents to agree to tradeoff something they have, they must first be persuaded with the benefits of this prospective union while being involved in a negotiation session - *Definition 10*.

Definition 10. ***A Negotiation Session*** *is the time space in which two agents are negotiating the formation of a union.*

An agent gets into a *negotiation session* following the *head-agent*'s task of applying the community's common rules. However, all agents of a specific community are having their own description of the service they are searching to acquire and what they are willing to give in return, (e.g., if I get a blond or curly female from 20 to 25 I would give either a dinner or flower).

For every agent, this general description is broken down into a set of requests and tradeoffs (Definition 6 & Definition 5), (e.g., hair = blond, hair.alternative = curly, age 20, age.alternative = 25, tradeoff1=dinner, tradeoff2=flower). Eventually, an agent negotiates with its potential union partner the possibility to satisfy a set of requests with respect to the associated tradeoffs.

3.2.3 Agent's satisfaction and its Instances

Depending on the nature of requests each agent in a service-centric community is searching to fulfill, the service a community makes available to its members may have different forms. From an agent perspective, the different

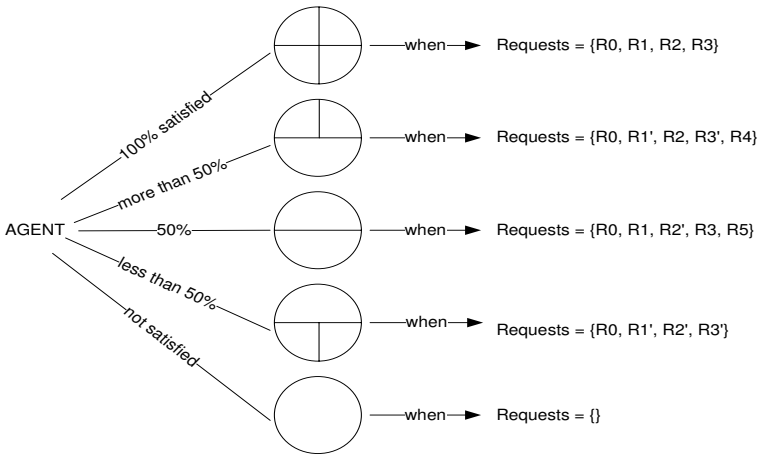


Figure 3.5: Levels of satisfaction Vs. Sets of Requests

forms a service takes correspond to different levels of satisfaction an agent may attain while acquiring a service.

The highest level of satisfaction an agent could possibly attain is associated with the fulfillment of a specific set of requests. The lowest level of satisfaction an agent may reach to - *not satisfied* - corresponds to the situation where none of the requests' subsets can possibility be fulfilled.

Example 2. Levels of satisfaction in a ridesharing service

In a community wherein a ridesharing service is made available throughout its members' interactions, we assume that there is a ride seeker agent that is called **AgentS** and, the highest level of satisfaction **AgentS** may attain is when a negotiation with **AgentG.1** - *that is a ride giver agent* - leads to forming a union in which **AgentS** will be: 1) picked from home, 2) at 14:30, 3) dropped by the post office, 4) with no stops in-between and, 5) the cost is $5.

The lowest level of satisfaction **AgentS** may attain is when none of the negotiations performed within the available time has led to any union formation and, consequently, **AgentS**'s *instance* was not completed. **AgentS** may also be 50% satisfied if a negotiation with **AgentG.2** has led to union in which a car ride with couple of stops are made in-between the departure and the arrival points.

In our model, an agent's *instance* can take different forms wherein each reflects a different level of satisfaction an agent may attain. Reaching each of these forms is associated with the fulfillment of a different subset of agent's requests. Therefore, the total number of forms and agent *instance* may take correspond to a large set of different requests. The optimal form of an agent's

instance, and its highest level of satisfaction, is obtained when a specific set of requests is fulfilled; a *key-set*.

Definition 11. ***A Key-Set****: is a specific subset of requests that an agent attempts to fulfill in order to obtain the optimal form of its instance.*

In figure 3.5, we use different types of circles to simplify our notion of an *instance*'s different forms. The highest level of satisfaction an agent may attain is the optimal case of an agent's *instance*, which is represented by means of a crossed-circle. For an agent's *instance* to become a crossed-circle, a set of specific requests must be fulfilled, **Requests** = {R0, R1, R2, R3}. Therefore, the set of requests that leads to this particular shape is the *key-set*.

Following the depictions of the same figure, a number of less optimal forms of the same agent's *instance* can be obtained when different sets of requests are relatively fulfilled. These emerging sets of requests may contain an agent's new types of requests or variants of the *key-set* requests. We use the blank circle to symbolize the case when none of the requests are fulfilled.

However, we conclude the depictions of figure 3.5 by highlighting the fact that any agent in a service-centric community can be either completely satisfied, not satisfied at all, or having a level of satisfaction that is neither optimal nor insufficient.

3.2.4 Agents' reactions to different Instance's forms

One or more of the items available in an agent's list of tradeoffs is associated with the realization of its *key-set*, which is in return associated with an agent's optimal *instance*'s shape. However, since an agent's *instance* may take different forms in which one is optimal, then, different sets of requests may emerge in order to define the other instance's forms. Plus, each of the emerging sets of requests become linked to different tradeoffs or, variations of the *key-set*'s item of an agent's list of tradeoffs or, combinations between all possible tradeoffs.

Figure 3.6 depicts the relationship between each possible instance form (agent's levels of satisfaction), **Requests** subsets, and **Tradeoffs**. We use the crossed-circle, again, to symbolize the optimal form of an agent's instance, the centrally divided circle refer to 50% fulfillment of the same agent's instance, the circle with its bottom side vertically divided refer to a level of fulfillment that is greater than 50% but less than 100%, and the circle with its upper part vertically divided refer to a level of fulfillment that is greater than 0 but less than 50%.

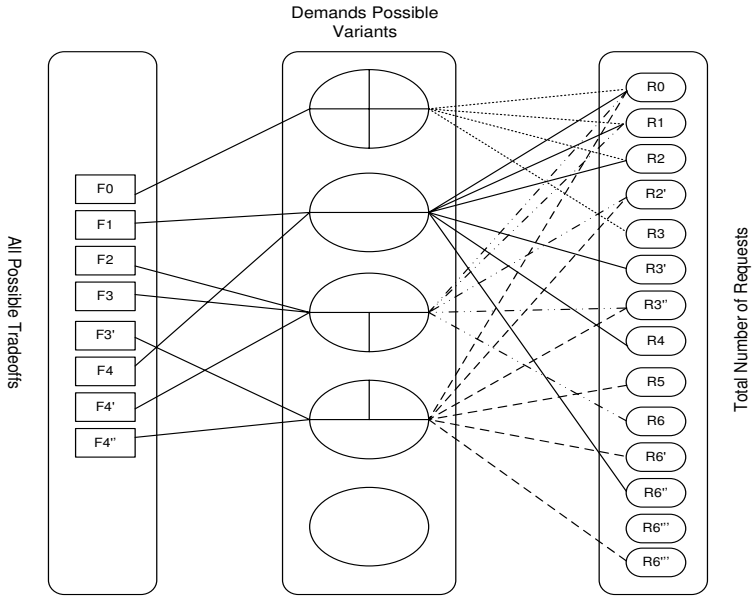


Figure 3.6: Satisfaction Levels, Sets of Requests, and Sets of Tradeoffs

As figure 3.6 shows, the *key-set* = {R0, R1, R2, R3} is the one associated with the crossed-circle (optimal instance shape), which is in turn linked to the uppermost item of the tradeoffs - $F0 \in$ **Tradeoffs**.

Following the depictions of figure 3.6, we then show the case in which a less optimal condition of the same agent's instance is linked to a subset of requests that is not the *key-set*, but yet it includes some of its elements. In addition, we show the case in which a possible instance's form is associated with more than one tradeoff, and the resulted set of tradeoffs includes a variant of an already included tradeoff, (e.g., a book's soft or hard copy).

Among other several scenarios that figure 3.6 may show, we would like to highlight the existence of an empty circle that we involve to symbolize the case of total incompletion of an agent's *instance*, and consequently it is not connected to any requests' subset or associated with any tradeoffs, but yet it is likely occurring.

Definition 12. *A possible agent's* ***View*** *is the combination of an agent's possible subset of requests plus its associated tradeoffs.*

Since in our research every agent may have a number of requests and a number of tradeoffs and, since all available requests are typically associated with one or more tradeoff. Then, an agent may have an unspecific number of *instances* in which all of them are located in-between the optimal and the minimal forms of satisfaction.

Putting together any satisfaction instance, its specific set of requests, plus their associated set of tradeoffs, we reach a particular combination that we call it an agent's *View*. According to definition 12, an agent may have more than a single *View*, and these *Views* vary according to agent's interests. Consequently, an agent may succeed to find another agent that is capable of fulfilling one of its Views.

For example, a possible *View* of an agent in a community where rideshare service is applicable could be the combination of 1) a set of requests,

$$\textbf{Requests} = \{start.trento, end.povo, route.nostops, time.1530\}$$

and, 2) a set of tradeoffs,

$$\textbf{Tradeoffs} = \{euro.10\},$$

and, 3) an instance of this agents satisfaction

$$\textbf{Satisfaction} = 100\%$$

3.3 The Negotiating Agents

In our negotiation model, there are N autonomous agents, $\textbf{Agents} = \{A_1, ..., A_N\}$. These agents are bilaterally negotiating to resolve the issue of establishing a mutually beneficial union in order to fulfill each other's service requests, which we mentioned in section 3.2.2.

We reflect an agent's requests, its *instance*'s forms (levels of satisfaction), and list of tradeoffs using a matrix. This matrix's horizontal edge represents an agent's requests, which are in turn describe the characteristics of the service sought, $\textbf{Requests} = \{R_1, ..., R_M\}$. The same matrix's vertical edge represents an agent's set of tradeoffs, which is in turn describe the characteristics of the payment, $\textbf{Tradeoffs} = \{F_1, ..., F_K\}$. The relations between the elements of these two sets are identified through the existence of an intersection between one another.

In this matrix, a cell wherein the initial point of its vertical column and the initial point of its horizontal row meet exists, and we refer to it as the *set-cell*, definition 13. Since $r_0 \in \textbf{Requests}$ contains the uppermost element of an agent's set of requests, and $f_0 \in \textbf{Tradeoffs}$ contains the uppermost element an agent's list of tradeoffs, then, a matrix's *set-cell* is also the starting point of the *key-set* a specific matrix represents.

Definition 13. *A **Set-Cell** of an agent's matrix is the intersection between its $r_0 \in$ **Requests** and its $f_0 \in$ **Tradeoffs**.*

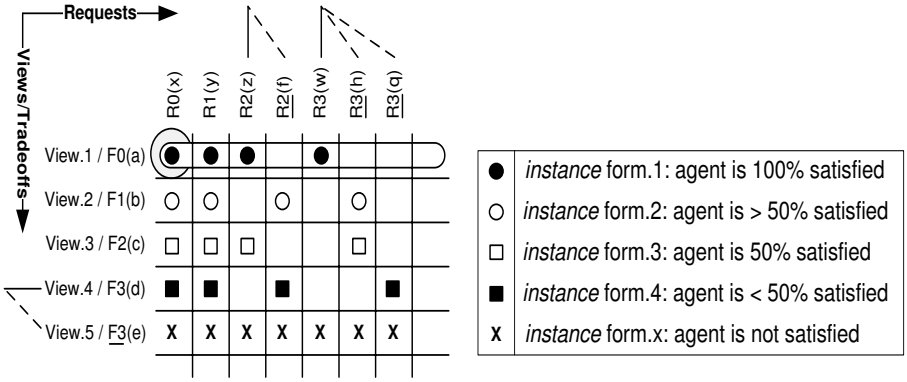


Figure 3.7: Key-sets Vs. Agent's Requests Vs. Agent's Tradeoffs: Matrix

In figure 3.7, we show an example of how the set of requests and the list of tradeoffs can be reflected on a matrix. The *key-set* of this matrix starts from the *set-cell* located within the intersection of (R0(x), F0(a)). The *key-set* and all of the matrix horizontal lines are also possible *Views* of an agent, Definition 12. Therefore, each horizontal line in an agent matrix reflects a possible tradeoff, its associated requests, and one possible negotiation *View*.

Following the depictions of the same figure, in this matrix, satisfying the elements available in R0, R1, R2, and R3 are associated with the action of trading off the value of its F0 element. Another View of the same matrix includes a variant of the R2 element together with a variant of R3, which are represented in the figure as repetitive R2 and R3. From the same figure we could observe that also tradeoffs could have variants, which is the case of F3, (e.g., 20$ or 0$). From the perspective of a service-centric community, each of the *Views* and agent has is a possible *instance* of the community's concern, Definition 7.

Example 3. Bob & Alice

In a dating community of agents, agent Bob represents a male that is searching for a female and, agent Alice represents a female that is searching for a male.

In figure 3.8, agent Bob and agent Alice are both searching for dates. Bob's initial request is indicating the fact that he is searching for a female date, R0(female). Bob would like his date to be either 20 years old - R1(20), or 25 years old - $\underline{\text{R1}}$(25). Since Bob is an understanding average height person, he prefers his date to be either Tall, Average, or Short - R2(tall), $\underline{\text{R2}}$(average), or $\underline{\text{R2}}$(short), but he likes only blonds - R3(blond).

However, Bob is willing to offer different tradeoffs in exchange of the fulfillment of every different combination of his service requests. Meaning,

for every satisfied *View* there is a tradeoff item(s) associated with it.

A carRide - `F0(carRide)` - is assigned by Bob to the *key-set*. A dinner - `F1(Dinner)` - is assigned to his date if she is 25 years old, has an average height, and blond. A free drink - `F2(freedrink)` - if she is 20 years old, has an average height, and yet a blond. Eventually, Bob will give nothing - `F4(0)` - if none of his service requests are fulfilled.

On the other hand, similar to the matrix notion of agent Bob; agent Alice has associated the fulfillment of diverse combinations of her service requests - *Views* - with a list of tradeoffs that she is capable of offering. This list contains, 1) different amounts of money, 2) tickets to watch a movie, or 3) a free drink. Therefore, if her prospect date is a male that is between 30 and 25 years old, has either short or average height, and slim, he is most likely to fit.

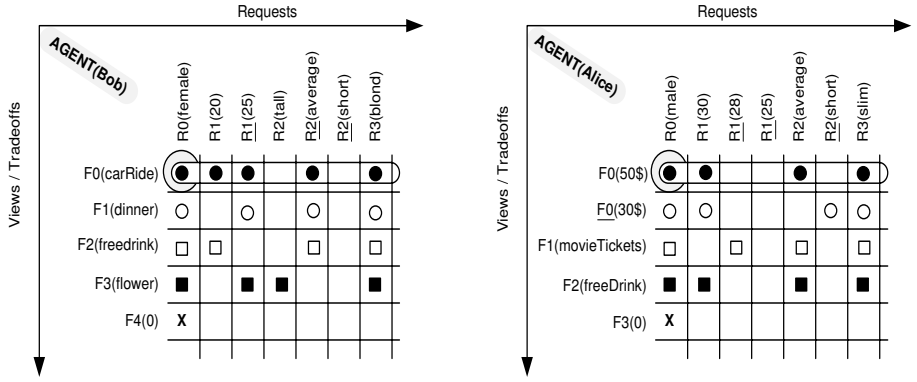


Figure 3.8: An example of two agents' service matrixes

In this example, if Bob's age, height, and weight are similar to those of Alice's preferences, and Alice's characteristics are matching those Bob is searching to fulfill, then; a negotiation between these two agents may lead to establishing a *Union*. This *Union* is expected to satisfy a possible *View* of the requests and tradeoffs that each of them has previously combined.

3.4 The Negotiation Protocol

A model for bilateral negotiations about a set of variables that is measurable on a numeric scale - *quantitative variables* - was presented in [19]. One of the main focuses of Raiffa's model is related to *two parties & multi issue* states of negotiation and their applicable value scoring system. Among several variations of Raiffa's model, scholars in [72] have relied on the *many par-*

ties & many issues instantiation to present a multilateral negotiation model between autonomous agents in a service-oriented domain.

The negotiation model we present is based on another variation of Raiffa's model. This variation deals with the negotiation situations of *two parties one issue*, which we believe to be similar to situations wherein two software agents are negotiating the conditions of provisioning a service. According to the framework of automated negotiation presented by scholars of [94]; fewer are the complexes in the reasoning behind the decision making, greater is the time saved while concluding the overall negotiation process.

The negotiation protocol we propose is straightforward. In a *Negotiation Session*, an agent proposes a union formation and the prospective agent replies using one of the following expressions: *accept*, *reject*, *pend*, or *deposit*. Once an agent decides to propose a union formation to another agent it is always assumed that the proposing agent fulfills - *accepts* - the terms of its prospect partner.

Since time is essential to the application domains we address, in our protocol, we attempt to minimize the agent's reasoning time by widening the often applied approach of alternating sequential offers presented in [71]. We expect a software agent to take less time to reason if more information is available about its potential union partner. Equally, agents reason more about their decisions if they observe less feedback from the objects they interact with.

In short, rather than using just the "*agree & disagree*" fashion of negotiation, we give agents additional space to express themselves. Similar to our methodology, in [18] an agent proposes a data-allocation plan to others and they pick *Yes*, *No*, or *Opt-out*; then a relative reasoning strategy is executed.

Why would an agent agree to fulfill the service request of another agent? This is what we will come across in the strategies and tactics subsections. For now it should be clear to us that:

1. For two negotiating agents to ***Accept*** a union formation it means that both have agreed to fulfill a possible *View* of each other.

2. It is possible for the negotiation session to encounter a different scenario where a proposal for union formation gets rejected - One agent picks ***Reject***. In different words, it may happen that the proposing agent is certain about the benefits yielding of this potential union while the rejecter is not persuaded.

3. Another occurring situation is related to the fact of an agent deciding to ***Pend*** the negotiation process. In brief, pending a union forma-

tion occurs when the proposing agent is fulfilling a view of the union necessities that is currently of no precedence to the pender agent.

Once a proposal to Pend a negotiation session is agreed upon by a session's parties, it is then assumed that the agent who proposed the Pend, (i.e., the pender agent), will then start to look for better deals elsewhere for no longer than a timeframe that is calculated in particular way, which we explain later in Chapter 5. However, it is also worth highlighting here, briefly, that the agent who have accepted the Pend offer, (i.e., pended agent), will then be suspending any external negotiation activities until further feedback is received from the pender agent or, the permitted timeframe expires.

4. As of real world situations; there is an associated cost with every action of depositing an object into the safe-box - *Or any depositing space.* This cost increases relatively with the time an object is being deposited. However, it is not likely true for this safe-box to be always available. There is a time where the depositing service closes and the deposited object is nearly "lost". Thus far, the Deposit service still has the advantage of allowing the depositor to weightlessly wander within an environment.

   In our negotiation protocol, we introduce a depositing-*like* ability for each negotiating agent. An agent will be able to ***Deposit*** a single view of the negotiated issue for a certain period of time. Once a proposal to Deposit a negotiation session is agreed upon by a session's parties, it is then assumed that the agent who proposed the Deposit will then start to look for better deals elsewhere under two conditions:

   - The specific combination of requests and their associated tradeoffs, (i.e., an agent *view* or *instance*), that were under negotiation at that time, are then excluded while this depositor agent is searching for better deals,
   - A time limitation is calculated and imposed on the process of searching for better deals, if expires without a feedback, then Deposit is considered as Accept and a union is made.

   However, here the agent depositing the negotiation session will also be entitled to look for alternative deals.

For example, in a community of agents wherein interactions are made to acquire a service that is *Dating*. The acceptance of agent **Bill** to union with agent **Monica** refers to the fact that both have agreed to go out on a date.

If Monica would have decided to reject Bill's proposal to form a union that would have referred to the fact that Bill was foreseeing a level of satisfaction from the prospective union while Monica is not fully persuaded.

However, **Bill** is also allowed to Pend the formation of a union with **Monica** and search for alternatives. This may occur when Bill's optimal level of satisfaction is only achieved if his date is Blond and, since Monica is not, and yet time is not a constrain, then a Pend might be an option. Another option might be taken into account by Bill is to look for alternatives while excluding the deal reached so far with Monica from his prospective search process - *Depositing a view at Monica's*.

3.5 Negotiation Timelines

In our model, every two negotiating agents, in a community of common-interest agents, can take decisions in certain times of the set $DTime = \{d_{t1}, ..., d_{tn}\}$. We assume that another time set, $RTime = \{r_{t1}, ..., r_{tn}\}$ is imposed to enforce every two negotiating agents to release their partners when the concerned community of agents exceeds a certain number of members. These timelines are discretely linked to each negotiation session wherein two negotiating agents are involved.

As shown in figure 3.9, the **DTime** set builds a timeline of checkpoints in which the negotiating agents are permitted to give decisions; **Decision-Timeline**. In addition, the **RTime** set builds a timeline of other types of checkpoint in which the negotiating agents are obliged to let their partners; **ReleaseTimeline**.

Following the depictions of figure3.9, we introduce the **Initializing Timeslot** (ITS), which is the time where two negotiating agents are granted an initial reasoning space to freely examine the conditions of each other, and make a decision if doable. We represent the **Termination Timeslot** (TTS), which is equal to the interval between two subsequent time checkpoints located on the **ReleaseTimeline**. We also show the **Decision Timeslot** (DTS), which is simply equal to the interval between two subsequent checkpoints located on the **DecisionTimeline**.

For example, assuming that agent **Alex** and agent **Zak** are representing users of pocket computing devices in a community where interactions are made in order to acquire a specific service, for instance Meeting Organization. Therefore, from time perspective, Alex and Zak are assigned - *by the Head-Agent* - to a negotiation session wherein no decision will be required from both within the first 5 minutes. Then, the Initializing Timeslot (ITS) = 5 minutes.

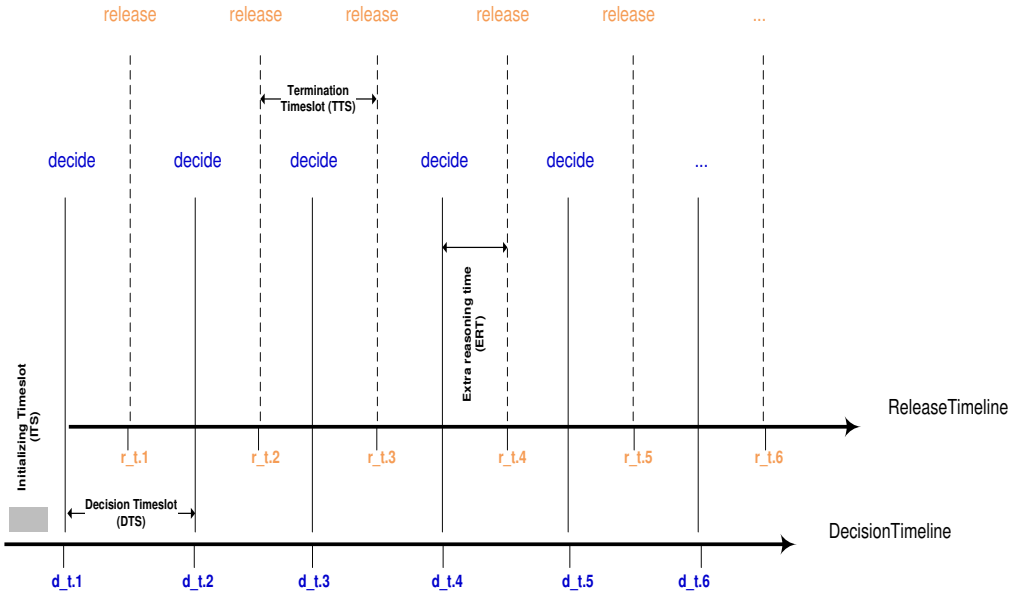


Figure 3.9: Protocol's decisions and sessions control timelines

When ITS expires without Alex and Zak having made any early decisions, then the head-agent will impose a request for decision to Alex and Zak at certain sequence of times that is predefined by the managing entity of the service, (e.g., service provider). This sequence of times is called a decision timeline, and the interval between two points on this timeline is called Decision Timeslot (DTS).

However, it is also expected that sometimes Alex and Zak are interacting within a community of colleagues and friends that is expanding at random times. Therefore, another timeline can be intersecting with the decision timeline to enforce Alex and Zak to release the assigned negotiation session. We call that later timeline the Release Timeline.

The intuition behind the introduction of the decision timeline is to increase the possibility of generating a negotiation process feedback and, consequently, the users these interacting agents are representing will be fed with some useful information on-the-run. Besides, the intuition behind the introduction of the release timeline - *when imposed* - is twofold: 1) to increase the agents' chances to interact with as many available entities as possible and take the right decision accordingly, 2) to ensure that the biggest number of agents have actually gone through negotiation sessions even though the service provided is overloaded.

3.6 Chapter's Summary

In this chapter, we started by introducing our view of an agents' *Society*, *Community*, and a *Cluster*. Then we introduced our notion of *Concerns* and the fact that a concern can also be perceived as a sub-concern from less abstract level of the society tree, and so on until a sub-concern is linked to a member of a *society* that ends at that level of abstraction: then it is that member's concern and, therefore, his responsibility to complete. Then we gave an example of mapping concerns onto society's levels.

Eventually, within the context of our negotiation model, we introduced our approach of defining a *Request*, a *Tradeoff*, and a *Union* between two agents, which we linked to a *Society*'s concerns and levels of abstraction afterwards. Then, we distinguished between any *Society*'s *Community* and a *Service-Centric Community* such as the one we address in this thesis. Then, we defined what we call the *Head-Agent*, which we assume to be responsible of putting together members of a service-centric community in order to get their service requests fulfilled.

Within a *service-centric community*, each member has a set of requests to be fulfilled and in return a set of *tradeoffs* must be made. Therefore, we then linked a members' different satisfaction levels, (i.e., *views*), with the types of tradeoffs he/she ready to make and, the different subset of requests that will consequently be fulfilled. A member attempts to fulfill any of his *views* by negotiating the establishment of a *Union* with one of the same community's members.

At the end of this chapter, we introduced the negotiation protocol, (i.e., *accept, reject, pend, deposit*), a member, (i.e., an agent), of this community will be using to interact with other members of the same community in order to reach to an agreements about a specific union establishment.

We ended the introduction of the first part of our negotiation model by explaining the link between a community's total number of members and the limitations we impose on the negotiation sessions' length. We did that by introducing the *DecisionTimeline* and the *ReleaseTimeline*, which control the times an agent must give a decision and the times an agent will be obliged to release the attached negotiation session.

Chapter 4

The Negotiation Tactics & Strategies

In this chapter we further continue the introduction of our negotiation model. It starts by formalizing the set of negotiation tactics then the strategies that we believe to have a great impact on agents interacting in a modern service-acquisition environment.

4.1 The Negotiation Tactics

In the context of the nomadicity-oriented setting we focus on, a tactic is the technique an agent adopts to achieve the eventual task of acquiring a service while matching the preferences of the end-user it represents. However, for an agent to acquire a service a mutually beneficial union formation that involves another service provisioning agent is needed.

Since it is expected that a number of agents located in the same service platform can provide comparable services but under different conditions, here; a tactic would allow an agent to pick up the right union partner that fulfills the most of an end-user's needs in that specific time.

The time wherein a certain tactic is adopted by an agent comes after a negotiation session has been assigned to it by the managing MAS and, before this agent gives a decision regarding a specific union proposal. Therefore, a decision that an agent communicates with its prospective union partner is affected by two factors. These factors are: 1) the circumstances a deciding agent went through prior to the start of the current negotiation session and, 2) the union conditions that the partner of the current negotiation session agrees upon.

In this section, we tackle the first influential factor by giving the negotiating agent the possibility to select from different tactics. Each of these tactics is an instantiation of a group of tactics that is oriented towards a particular

nomadicity-driven requirement.

We hereafter introduce three different groups of tactics in which each of them can be of use to the negotiating agent in a number of situations. However, we do acknowledge the fact that these groups and their instances can be extended to cover additional requirements. However, with this work we attempted to deal with as many useful scenarios as possible while considering the common observations we collected from similarly developed nomadicity-oriented service applications.

4.1.1 Time-based Tactics

The first group of tactics contains two main approaches that every negotiating agent may employ one of them while deciding upon participating in forming a *union* with another agent. Both tactics are related to situations where *time* is an essential factor that affects the process of making a decision about a certain *union*.

With the introduction of Time-based tactics we intend to emphasize the importance of having the interactions happening among involved agents oriented toward the fact that users of pocket computing devices are also time limited. Since a transfer from one location to another is frequently occurring, and easily made, while we are not attached to any wires, then it is highly desirable in these situations to link the negotiation agents are carrying out with the time limitations of the user himself. We approach this notion from two sides, which are: 1) when the environment of interactions, (i.e., the community), is relaxed from the perspective of participants numbers, and 2) when this environment is overloaded.

In our model, the *head-agent* (Definition 9), is responsible of imposing the ReleaseTimeline that we described in section 3.5. The existence of this timeline is enforcing the negotiating agents to let go the negotiation session they are involved in at specific times that are known to agents before they start negotiating, RTime = {r_t.1, ..., r_t.n}. Whether to impose this ReleaseTimeline or not, it is a decision that the *head-agent* takes according to the number of agents available in a community at a specific time.

These two approaches are generally targeting two abstract situations in which one of them is involving the imposition of the ReleaseTimeline and the other is not. These approaches are: a) common system state tactic, and b) system overload tactic.

Common System State Tactic

In a service-centric community, N agents are attempting to negotiate in order to acquire a service in exchange of providing another, then a undetermined number of *negotiation sessions* - *Definition 10* - are accordingly existing.

Similar to the formalization approach used in [26] to represent agents' encounters in worth-Oriented domains (WOD), we hereafter, formally, represent a service-centric community (SCC) by means of a tuple,

$$\langle C, Agents, Unions, Actions \rangle$$

where

- C is the set of possible community conditions, {Jammed, Tolerable}, which vary according to the number of operating agents in a community and reflected through the existence of the **ReleaseTimeline**;
- $Agents$ is the set of available agents;
- $Unions$ is the set of possible unions in which each contain a set of possibly satisfied two agents;
- $Actions$ is the set of possible actions that an $a_i \in Agents$ may perform, (i.e., accept, reject, pend, deposit).

Then, we represent any possible *negotiation session*, $s \in Sessions$, of a service-centric community, SCC $\langle C, Agents, Unions, Actions \rangle$ as a tuple,

$$\langle c, Ag, DTime, ITS \rangle$$

where

- $c \in C$ is reflecting one of the conditions a community could possibly experience while this *negotiation session* is existent;
- Ag is the set wherein the two negotiating agents and the related *head-agent* of this specific session are listed;
- $DTime$ is the set of times wherein the negotiating agents are required to perform one of the available $Actions$. In our mode, this set is also known as the DecisionTimeline, $\mathbf{DTime} = \{d\_t.1, ..., d\_t.n\}$;
- ITS is the *Initializing Timeslot* of this specific session in which the starting time of the negotiation is registered.

Since with this tactic we intend to address the situation where the number of agents in a community is not causing the **ReleaseTimeline** to be imposed by the *head-agent*, then the condition of the community - $c \in C$ - is always set to $Tolerable$. Therefore, the sequence of times located on a certain sessions's **DecisionTimeline** are the only external obligations considered by agents while reasoning about a specific union formation.

In our model, when two agents, $A_i, A_j \in Agents$, are discussing the formation of $u \in Unions$ by occupying a negotiation session $s \in Sessions$ while $c \in C = Tolerable$, and one of these negotiating agents decides to follow a time-driven style of interactions, every item of the set **Actions** will be performed according to the following guidelines:

- **Accept:** a negotiating agent instantly accepts a union formation when its potential union partner is by default satisfying its *key-set*. Therefore, if $A_i(key\text{-}set) = A_j(key\text{-}set)$ then A_i communicates "*accept*" with A_j and waits for response.

- **Reject:** a negotiating agent instantly rejects a union formation when none of the *Requests* that it has is matching the *Requests* that a potential union partner is having. Therefor, $A_i \in Agents$ rejects the formations of a $U \in Unions$ with $A_j \in Agents$ if $A_i(requests) \neq A_j(requests)$.

- **Pend:** situations may exist where an agent's *key-set* is not precisely fulfilled by its negotiation partner but, yet, a request or more are likely to be in common. In such cases, depending on the time available for an agent to acquire a service, an agent may *Pend* or *Deposit* its current negotiation process.

  Therefore, A_i may decide to *Pend* a union, $u \in Unions$, if the time space - t - given to this agent to achieve its goal is equal to, or less than the sum of **ITS** plus the approaching **DTS**; $A_i(t) =< (ITS + DTS)$.

- **Deposit:** within the boundaries of the same tactic, a negotiating agent would *deposit* a union when the conditions to Accept or Reject are not valid and, the time available for this agent to acquire a service is greater than the sum of **ITS** plus **DTS**.

System Overload Tactic

As we described earlier in section 3.5, in a service-centric community of agents - SCC $\langle C, Agents, Unions, Actions \rangle$, the existence of the **ReleaseTimeline** reflects a community's overload - *the number of operating agents has exceeded the predefined standards of tolerability.* Consequently, following the same

formal approach of the common system state tactic described in the earlier subsection, we hereafter introduce a time-driven tactic for an agent to employ while the community is overloaded.

Therefore, a negotiation session in an overloaded community will look like this: $\langle c, Ag, DTime, RTime, ITS \rangle$, where RTime is is the set of times imposed by the *head-agent* wherein the negotiating agents are required to release their negotiation partner. Accordingly, we assume that the condition of the community, $c \in C$, is always set to $Jammed$.

As we showed earlier, within this family of tactics, time is considered only while an agent is choosing whether to *Pend* or *Deposit* a union. Therefore, in this specific tactic the *Accept* and the *Reject* actions are similar to those of the common system state tactic. Since this tactic addresses the situation when the **ReleaseTimeline** is assumed to be always imposed, then, different from the earlier tactic, two emerging time considerations must be taking into account while the agent is deciding whether to *Pend* or *Deposit* a *Union*.

These considerations are; 1) the Extra Reasoning Time (ERT) that is equal to the gap between a checkpoint of a **DecisionTimeline** and its approaching checkpoint of a **ReleaseTimeline** and, 2) the Termination Timeslot (TTS), which is equal to the interval between two subsequent checkpoints located on the **ReleaseTimeline**.

- **Pend:** while $A_i, A_j \in Agents$ are negotiating the formation of a union, $u \in Unions$, A_i may decide to *Pend* this negotiation if the time space, t, given to this agent to acquire the desired service is equal to, or less than the sum of **ITS** plus the approaching Extra Reasoning Time **ERT**; A_i's time space $=< (ITS + ERT)$.

- **Deposit:** a negotiating agent may decide to *Deposit* a negotiation session if t is greater than the sum of the Initializing Timeslot (ITS) plus the Extra Reasoning Time (ERT) and, the Termination Timeslot (TTS); A_i's time space $> (ITS + ERT + TTS)$.

4.1.2 Connectivity Related Tactics

In this group of tactics, we present two different approaches for negotiating agents to follow while considering the nature of the communication channel they rely on to communicate feedbacks with the entities they operate on behalf. These two approaches are aimed at addressing two situations, one in which a group of highly reliable communication technologies is used and, the other addresses the situation when influential limitations are observed on the behavior of the employed technology. Namely, these approaches are:

1) Prompt communications channel, and 2) less responsive communications channel.

By "*prompt communication channels*" we refer to wireless connectivity that allow end-users to send and receive service requests within a large coverage, and with acceptable data transmission rate, (e.g., Wi-Fi, WiMAX). Besides, by "*less responsive communication channels*" we refer to wireless connectivity with a number of technological obstacles, such as limited number of concurrent connections, short-range coverage, and low-speed data transmission, (e.g., IrDA, Bluetooth).

A Tactic for prompt communication channels

In a service-centric community, a set of different communication channels can be employed by negotiating agents to communicate negotiation feedbacks between one another, $\mathbf{Channels} = \{ch_1, ch_2, ..., ch_w\}$. The reason agents may need to give a feedback regarding the status of a negotiation process is due to the fact that a specific agent might be delegated by another to acquire a service, (e.g., a user is delegating a software agent to acquire a service).

In our model, all communication channels are divided into two subsets of **Channels**. One, we call is $StChannels \in \mathbf{Channels}$, which includes all the communication channels that are of strong reliability - *prompt communication channels*. And the other, we call it $LimChannels \in \mathbf{Channels}$, which includes all of the limited reliability communication channels - *less responsive communication channels*.

In this tactic, we look at a negotiation session, $s \in Sessions$, from the perspective of a negotiating agent, $a_i \in Agents$, that is relying on one of the communication channels of the subset $StChannels \in \mathbf{Channels}$ to communicate a feedback with an agent/user that is on higher level of authority - *Delegating Agent*.

- **Accept:** understandably, and in spite of an agent's available negotiation time or type of communication channel; within a negotiation session, $s \in Sessions$, in which a *head-agent* - a_h - is managing, agent $a_i \in Agents$ instantly accepts the formation of union $u \in Unions$ with agent a_j when its *key-set* = a_j'*key-set*.

- **Pend & Reject:** in this tactic, we consider the action of rejecting a union formation by a_i as a sub-action of *Pend*. The reason we do that goes back to the fact that the availability of a reliable, and therefore communicative, channel of communication may lead the *Delegating Agent* of a_i to reconsider its possible views of the needed service. Consequently,

by this procedure we intend to expand the probability of increasing the overall number of unions that can be formed within a community.

Therefore, within this tactic, $a_i \in$ **Agents** pends $u \in$ **Unions** when a_i(**Request**) $\neq a_j$(**Requests**). Then a phase, which we call it the **Post-Pend Procedure**, starts in which a_i calculates the pending time and, communicate it with the *head-agent*. The intention here is to let the *head-agent* communicate the status of this negotiation session and the pending time with the *Delegating Agent* of a_i hoping for a reconsideration.

Two possible approaches are available for a_i to calculate a session's pending time, these approaches are:

1. When the ReleaseTimeline is imposed, meaning that the society's is under a **Jammed** condition, a_i's pending time will be equal to the interval between the $dt \in DecisionTimeline$ wherein the *Pend* action was taken and, the first session releasing time, $r_t \in ReleaseTimeline$.
2. When the ReleaseTimeline is not imposed, meaning that the society's current number of agents is **Tolerable**, a_i's pending time will equal to the interval between the $dt \in DecisionTimeline$ wherein a_i decided to Pend the session and, the subsequent decision time $dt + 1 \in DecisionTimeline$.

Further to a_i's situation, there are two circumstances in which a *Reject* can be communicated with a_j. These circumstances are: 1) when the pending time expires without receiving any reconsiderations from the *Delegating Agent* and, 2) when the reconsiderations (Views Modifications) are received but with no changes included.

However, when the *Delegating Agent* of a_i reconsiders the *Views* describing the different states of the negotiated service - *within the permitted pending time*; then, the *head-agent* of the same community resets the negotiation session that involves $a_i, a_j \in$ **Agents**.

- Deposit: commonly, the *Deposit* option is used by a negotiating agent if neither instant *Accept* nor *Reject* or *Pend* are of any use. Therefore, $a_i, a_j \in Agents$ are having some common requests that are likely to take place in forming *Views* that are different from those forming the *Key-Set*.

Since in this tactic we assume that a_i is relying on a highly reliable communication channel, (i.e., $ch \in StChannels$), then, a feedback containing the status of the current negotiation session is communicated

with a_i's *Delegating Agent*. The intention behind updating the *Delegating Agent* of a_i's status of the current negotiation session is to give a space in which modifications on the concerned service requests can be imposed so that agreements are reached.

Simultaneously, a_i starts to follow the regular steps taken after depositing a negotiation session, which are: 1) excluding the deposited view, $a_i(views - 1)$, and then 2) engaging for other potential unions within the depositing time. Once an alternative union is found, a_i ends its deposited negotiation session with a_j and the newly emerged session is then addressed.

However, if no new potential unions are found, a_i waits until the depositing time expires, if no modifications were imposed by the *Delegating Agent*, then the deposited session is concluded by forming a union between $a_i \& a_j$ based on the previously deposited view.

A tactic for less responsive communication channels

In general, the existence of a less responsive channel of communication between a negotiating agent and its *Delegating Agent* affects the quality of interactions among them. As a result, the level of autonomy of the negotiating agent must be increased to meet the lack of feedbacks from agent's higher-authority. Therefore, in this tactic we minimize the number of feedbacks sent by a_i to its *Delegating Agent* and, we reduce the number of situations in which a feedback from a higher authority is urgently required in order for a_i to proceed with the negotiation.

In a negotiation session, $s \in Sessions$, of a service-centric community, SCC $\langle C, Agents, Unions, Actions \rangle$, $a_i, a_j \in$ **Agents** are interacting to reach an agreement about forming a $u \in Unions$. Assuming that a_i and its Delegating Agent are relying on a $ch \in LimChannels$ to exchange feedbacks.

The following part shows the scenarios wherein a_i *Accepts*, *Rejects*, *Pends* and *Deposits* a negotiation session. However, here we exclude the $pend \in Actions$ because, as showed in the earlier section, it gives less time to the negotiating agent to search for better unions and it is useful only in case the Delegating Agent reconsiders its service requests.

- **Accept:** In this tactic, a_i tends to instantly *accept* the formation of a union, $u \in Unions$, with a_j if both are fulling each other's optimum *View*, which is the *key-set*. Formally, the condition of accepting a union formation can be described as follows; a_i's *key-set* = a_j's *key-set* $\rightarrow a_i \cup a_j$.

- **Reject:** Once the *head-agent* puts a_i & a_j in a negotiation session, and in spite of what tactic is adopted by both agents to handle this negotiation session, if none of the requests a_i holds are matching any of a_j's, then a_i *rejects* the session without going into extra reasoning.

  Formally, this condition of rejecting a union formation can be described as follows: $a_i(\textbf{Requests}) \neq a_j(\textbf{Requests}) \rightarrow \neg(a_i \cup a_j)$

- **Pend:** while adopting this tactic, a_i can possibly *pend* its ongoing negotiation session - $s(a_i, a_j) \in Sessions$, if at least one of its subset of **Requests** can be satisfied by forming a union with a_j, plus, the tradeoffs a_i is willing to give in return are not what a_j is seeking for.

  Following to the action of pending a negotiation session, a new **Post-Pend** phase starts. In this phase, and all through the session's *head-agent*, a_i communicates a feedback about its current status with a higher authority agent - *The Delegating Agent*. Different from the earlier tactic, whether the feedback have reached to its Delegating Agent or not, a_i immediately starts to search for alternative unions without waiting for modifications imposed by higher-authority.

  Similar to the tactic of prompt communication channels, two different methods are likely to be employed by a_i to calculate the time available to search for alternative unions / session's pending time. Depending on whether a community is Jammed or Tolerable, these methods are:

  1. When the ReleaseTimeline is imposed, a_i's pending time will be equal to the interval between the $dt \in DecisionTimeline$ wherein the *Pend* action was taken and, the first session releasing time, $r_t \in ReleaseTimeline$.
  2. When the ReleaseTimeline is not imposed, a_i's pending time will equal to the interval between the $dt \in DecisionTimeline$ wherein a_i decided to Pend the session and, the subsequent decision time $dt + 1 \in DecisionTimeline$.

  Thus far, we intend to let a_i proceed with searching for alternatives normally without expecting a modification from its Delegating Agent, even though a status feedback was sent through. Since this tactic tackles the situation wherein a_i is relying on $ch \in LimChannels$ to communicate feedbacks; therefore, it is not always true that the feedback reaches the Delegating Agent. However, if reached, the Delegating Agent is free to whether reconsider the combination of **Tradeoffs** and **Requests** / Views, or ignore it.

At this point, the current situation may lead a_i to go through one of three different scenarios, these scenarios are:

1. The Delegating Agent receives the feedback but decides to ignore it, or act upon it but the associated $ch \in LimChannels$ was not reliable enough. Consequently, a_i will *reject* the prospective $u(a_i, a_j) \in Unions$ when the previously calculated pending time expires.
2. The Delegating Agent succeeds to communicate new *Views* to a_i before the expiration of the pending time, and accordingly, a_i is updated by the community's *head-agent* and, the same session is then restarted.
3. The Delegating Agent does not receive the feedback and no actions are accordingly communicated back to a_i. However, a_i finds a potentially higher union formation while the pending time is valid. Therefore, $s(a_i, a_j) \in Sessions$ is terminated and new setting is respectively imposed.

- Deposit: in this tactic, the difference between the *pend* and the *deposit* options of agent's **Actions** is summarized in the increasing number of feedbacks a_i is ready to communicate with its Delegating Agent.

In this tactic, we introduce a new notation that reflects the relationship between a negotiating agent and the higher-authority agent that it represents. Precisely, $a_x \hookrightarrow a_i$ means that a_x is the Delegating Agent of a_i, which makes a_i the representative of a_x in a specific community. In a community, when $a_i, a_j \in Agents$ gets into a negotiation ($s \in Sessions$), a_i may *deposit* this session if at least one of its *Views*, that is not the **key-set**, can be fulfilled by forming a union, $u \in Unions$, with a_j.

The first feedback a_i communicates with a_x occurs when a *deposit* action is performed. Whether this feedback reaches a_x or not, a_i excludes the deposited *view* while searching for alternatives. The deposited *view* is basically a subset of a_i's **Requests** and their associated **Tradeoffs** - *Definition 12*.

Eventually, the concerned ***head-agent*** attempts to re-involve a_i in a new negotiation session as long as the depositing time is valid. Recalling the fact that the $deposit \in Actions$ gives more time to an agent to search for alternatives than *pend*, then, two methods are available for a_i to calculate the depositing time, these methods are:

1. When the ReleaseTimeline is imposed, a_i's depositing time will be equal to the interval between the $dt \in DecisionTimeline$ wherein the *Deposit* action was taken and, the secondly encountered releasing time, $r_t + 1 \in ReleaseTimeline$.
2. When the ReleaseTimeline is not imposed, a_i's depositing time will equal to the interval between the $dt \in DecisionTimeline$ wherein a_i decided to *Deposit* the session and, the secondly encountered decision time $dt + 1 \in DecisionTimeline$.

Within the depositing time, if a_i gets into a new negotiation session, and the potential union of this new negotiation fulfills a_i's `key-set`, then the previously deposited session is terminated and a_i instantly accepts the *union* formation of the newly evolving agent. However, if a_i gets into a negotiation session that satisfies a *View* which is different from the deposited one and yet not the ***key-set***, a feedback is made available for a_x to retrieve.

Since in this tactic we focus on less responsive communication channels, $ch \in LimChannels$, so if no reconsiderations are received from a_x within the depositing time, then a_i picks the union that will satisfy the *view* that is associated with less-valued tradeoffs, (e.g., located within the bottom levels of a descending list of tradeoffs).

However, a possibility that a_x employs the received feedbacks, modify one or two *Views*, and communicate its reconsiderations back to a_i. Accordingly, this will make the ***head-agent*** terminate the current activities of both, $a_i, a_j \in Agents$, and reassign both them to the same negotiation session.

4.2 The Negotiation Strategies

Following the formalization approach taken in [27], we hereafter introduce the two strategies negotiating agents may employ while reasoning about the establishment of a union with one another. As mentioned in earlier sections, when a union is established it indicates the fact that two negotiating agents have come into agreement about fulfilling each other's requests.

In our model, the goal of employing one of the proposed strategies by any negotiation $a \in Agents$, of the service-centric environment we address, is to maximize this agent's benefits. In our research, maximizing an agent's benefit of a union is done through the confirmation of the inexistence of better unions at the time of negotiating another potential union.

Formally, we further explain our model's strategies from one agent side, which is $a_i \in Agents$. Therefore, after the introduction of negotiation tactics in the previous section, a service-centric community (SCC) can be newly represented in this section by means of a larger tuple,

$$\langle C, Agents, Unions, Actions, Tactics \rangle$$

where

- C is the set of possible community conditions, {Jammed, Tolerable}, which vary according to the number of operating agents in a community and reflected through the existence of the **ReleaseTimeline**;
- $Agents$ is the set of available agents;
- $Unions$ is the set of possible unions in which each contain a set of possibly satisfied two agents;
- $Actions$ is the set of possible actions that an $a_i \in Agents$ may perform, $\mathbf{Actions} = \{Accept, Reject, Pend, Deposit\}$.
- $Tactics$ is the set of all possible tactics that an agent may employ while negotiating a potential union formation.

Whether a community is **Jammed** or **Tolerable**, as explained in section 3.5, there is always a time wherein the two negotiating agents are forced to give a decision about the currently negotiated union formation.

In SCC, and further to being assigned to a $s \in Sessions$ by the community's *head-agent*, when $a_i, a_j \in Agents$ are negotiating a potential union formation, $u \in Unions$, we refer to the time in which a_i employs one of the proposed strategies using ST. The Strategy Time (ST) of a_i is the period

of time that starts and ends with the occurrence and the termination of the first decision-making time of the **DecisionTimeline**.

Therefore, a_i's strategy regarding a specific union formation is also depending on a_j's initial feedback about the same union, which a_j shows by means of **Actions**; accepting, rejecting, pending or depositing a negotiation session.

In [18], a formal approach to negotiation strategies was presented wherein a strategy is expected to help the agent, whose turn is to react on a specific offer from a negotiating partner, to define its subsequent action. We adopt and add on Kraus's idea of looking at agents' reactions to the proposed offers as a set of *functions*, which made her Strategic Negotiation model capable of maintaining each agent's negotiation history, (i.e., $Length(f_1, ..., f_N)$), along with the offers' record that each agent receives while being operational, (i.e., $Last(f_1, ..., f_N)$).

In our model, negotiating agents are meant to represent a set of *higher-authority agents* that are located within an abstract phase of the model hierarchy, (e.g., users of pocket computing devices).

Since this abstract set of agents are - *by nature* - more intelligent and having the ability to socialize more than the delegated software agents, we hereafter tend to increase the negotiating agents' benefits by means of two social elements. These elements are:

1. **Cooperation**: a negotiating agent is considered as a social entity within its community if it tends to exploit the data collected along the negotiation time, (e.g., agents' IDs, their views, timestamps), to help encountered agents reach their goals. Therefore, agents socialize by means of cooperating with their encountered parties, which is considered as a strategy only if it brings back some returns / maximizes agent's benefits.

   In real world situations, people of a specific community may be motivated to cooperate with each other in order reach common goals, fulfill their job requirements, or even leave a good impression on others so that their reputation grow bigger. The last cause is what our model will be adopting afterwards to formally present one side of an agent's strategy.

2. **Fidelity**: a negotiating agent is also considered as a semi-social, but better protected, entity within a community if it tends to form unions with agent-friends rather than unknown entities.

   Taking into consideration the application domain we address in our model, mobile services vary and, in some scenarios, upper agents would

prefer to get their delegated agents into forming unions with well-recognized partners. For example, in carpooling services, our studies have been always proving that end-user would prefer to bear extra little cost, (e.g., time, money, or even farer drop point), in favour of getting into union with people that they previously know.

4.2.1 Enabling Socialability through Cooperation

In a negotiation session, $s \in Sessions$, when two negotiating agents, $a_i, a_j \in Agents$, are discussing the formation of a possible union, $u \in Unions$, the strategies both agents are allowed to adopt vary depending on the requirements that the designers are attempting to fulfill while outlining the community's core intuition and foreseen outcomes.

However, we refer to the set of strategies **Agents** are allowed to make use of within any service-centric community falling within the application domain we address - *mobile services* - as $\textbf{Strategies} = \{st_1, ..., st_g\}$.

In our model, in order for a negotiating agent to be a cooperative entity within a service-centric community three main considerations must be taken into account. These considerations are:

1. **Information Acquisition**: For an a_i to be useful to others, $(Agents - a_i) \in SCC$, and therefore cooperative, agents within a service-centric community must observe a benefit that is retrieved only through communicating with this agent. Consequently, a negotiating agent that is in favor of this strategy must be collecting the products in which others might be interested to buy - *the information wherein other agents might see useful.*

   Within a community of negotiating agents, useful information are those used by any agent to shorten its mission of successfully forming a mutually beneficial union. Thus far, we can basically observe that one of the key factors a negotiating agent must obtain in order to be cooperative is the fact of being able to acquire informations from negotiation partners.

   Similar to Kraus's approach that we mentioned earlier; in this model's strategy, $Length(f_1, ..., f_N)$ of a_i stores its negotiation history, (i.e., negotiators' names, durations, ... etc). Moreover, $Last(f_1, ..., f_N)$ of a_i stores the offers it receives at each time a negotiation occurs.

2. **Information Distribution**: The second phase of this strategy addresses the situation wherein the previously acquired information is made useful to the acquirer. In our model, $a_i \in Agents$ uses the data

collected out of its previous encounters, which is a result of combining $A_iLength(f_1, ..., f_N)$ and $Last(f_1, ..., f_N)$, by making it available to its current session's partner.

From the four of our model's **Actions**, which made available for any of the negotiation agents in $s \in Sessions$ while negotiating a potential union, $u \in unions$, **Reject** is the only action in which its occurrence enables an agent to distribute, and make use of, the previously acquired data. Reasonably, a negotiating agent will not be willing to help its negotiation party unless a specific foreseen potential in establishing a union vanishes.

Since our model assumes that the negotiation issue is a fixed / service, (e.g., Dating), but has different shapes, (e.g., Young Male or Mature Female), therefore, an agent picks the data to be passed to its negotiation party by means of matching the conditions the current negotiator attempted to fulfill with the previous negotiations it went through.

To better illustrate this point, we assume that the issue, (i.e., service), which a_i and a_j are negotiating is referred to as *SER*, while $SER_{shapes} = \{SER_{sh_1}, ..., SER_{sh_x}\}$ is the set of all possible shapes that this service may take. Then, assuming that $a_i, a_j \in Agents$ are occupying $s \in Sessions$ to negotiate the possibility of establishing $u \in Unions$. And, a_i has been already attempting to achieve its objective of acquiring a service for a while now but, for a_j; this is its first negotiation attempt.

Therefore, a_j's *Length* and *Last* are containing one negotiation attempt only, which is the one involving a_i itself. Accordingly, when a_j rejects a_i's acceptance to establish a union, a possibility occurs for a_i to pass on to a_j some useful data that might lead to faster establishment of a union for a_j, (i.e., addresses of alternative similar agents) and, a chance to prove cooperation for a_i.

If both, $a_i \& a_j$, are qualified to pass data to each other based on their previous experiences, then both are cooperative.

3. **Benefits Gaining**: In our model's strategies, we introduce the notion of agents' ***Reputation Score***, which is directly connected to the position an agent takes on the waiting-to-be-served list of the community's *Head-agent*. Every time an agent decides to pass on useful data to another agent after having its prospective union rejected, a point is added to its ***Reputation Score*** and, consequently, this agent is placed in one position higher than the expected.

4.2.2 Fidelity-driven Strategies

The second strategy of our model tackles the situations that may occur in which agents are performing in an environment that better requires the establishment of unions among agents of common interests, common preferences, or even of previous history.

In the application domain we address, some of the services that can be provided within a community are encouraged to be exchanged among people of the same geographical territory, such as CarPooling. In addition, some services are encouraged to be achieved among agents that represent users, (i.e., higher-authority agents), of the same sex, or different sex. Besides, some services are better acquired among agents that previously succeeded to establish a union.

In this subsection, among other several approaches, we introduce two possible orientations of the Fidelity Driven Strategy of our model. There orientations are: 1) friend-based orientation and, 2) zone-based orientation.

1. **Friend-based**: In a friend-based strategy, the use of the previously mentioned $Length(f_1, ..., f_N)$ and $Last(f_1, ..., f_N)$ are also considered. While $a_i, a_j \in Agents$ are assigned to an $s \in Sessions$ by the community's *head-agent* to negotiate the possible establishment of $u \in Unions$, a score function that we refer to as ***Recommend Score*** is then considered by both session's partners.

   The **Recommend Score** returns to its performer either *zero* or *one*, (i.e., a friend or not), according to a set of subsequent steps, which are 1) examining the $Length(f_1, ..., f_N)$ for a previous record involving the session's party, and if a record exists then 2) examining the $Last(f_1, ..., f_N)$ to see whether this previous negotiation has ended up with a successful establishment of a union. While both steps, 1 and 2, are true, then the score function performer / agent considers its session's partner a *friend*.

   Therefore, in our strategy we consider A_i as a friend of A_j if they have previously succeeded to establish a union together. However, situations wherein an actual *agent-friend* is negotiating the establishment of a union with another *agent-friend* although there are no signs of previous encounters. Sense in the application domain we address agents are likely to address end-users, therefore; With *actual agent-friend* we tend to refer to the relationship that may exist between agents of higher-authority.

   In this strategy we simply consider the actual friendship among *higher-authority* agents, which are the delegates of the negotiating

agents, all through the assumption that a field of the negotiating agents preferences may include a predefined list of preferred union partners / friends. Consequently, a community's *head-agent* would attempt to connect a specific negotiating agent to any of those listed in its preferences prior to the standard matching approach.

In a SCC $\langle C, Agents, Unions, Actions, Tactics \rangle$, whether the friendship between two negotiating agents in $s \in Sessions$ is yielding out of the **Recommend Score** or a **Preference-driven**, an agent would benefit from this strategy through; 1) fastening its decision if a situation in which two similar unions occur, therefore, in a ***Pend*** position, or 2) a guarantee is required as the service agents are negotiating is critical and a union among friends is highly recommended, (e.g., Car-Pooling).

2. **Zone-based**:

In our model, we consider the virtual as well as the physical presence of a specific community. The virtual presence of a community is reflected through the service in which all of its members are members of its operation, which we referred to earlier as the Service-Centric Community (SCC). The physical presence of the same community is identified by means of the geographical area in which this common service is made available.

In-between the physical presence of a community and its virtual space there are a set of connectors, (i.e., terminals). These terminals helps connecting / bridging the two sides of the overall scenario; physical and virtual.

Looking at a SCC from its physical existence standpoint, there is $Terminals = \{Ter_1, ..., Ter_u\}$, in which each covers a piece or more of the overall geographic space of a community. In our model, we break-down this geographic space into zones, $Zones = \{Zo_1, ..., Zo_v\}$. Therefore, a $ter \in Terminals$ can be associated with one or more $zo \in Zones$ and the vice versa.

In order for agents to interact, a transfer from the physical presence of a community to its virtual one is required. Therefore, in order for $a \in Agents$ to start interacting an action by the higher-authority agents must be performed wherein a specific $ter\ Terminals$ is used.

For example, from the set of higher-authority agents, (e.g., service users), *agentX* is delegating *agentB* of the lower-level set of agents to achieve a specific goal. *AgentX* then utilizes one of the community's two

presences connectors, such as ter_1, to move *agentB* from the physical layer to the virtual one.

Therefore, currently it became recognizable by the entire community, including the *head-agent*, that *agentB* was introduced to the virtual space of the concerned service through ter_1, which is associated with, for instance, $zo_1, zo_3 \in Zones$.

Employing a zone-based strategy while $a_i, a_j \in Agents$ are assigned to $s \in Sessions$ by their community's head-agent to negotiate the possibility of establishing a mutually beneficial $u \in Union$, a_i can understand how likely a_j is far by means of a simple **Vicinity Score Function (VSF)** that brings the difference between a_i & $a_j's$ zones as a result. The higher is the score the farer is the distance between a_i and a_j, and consequently any time-dependent actions, or geographically oriented decisions can be tackled.

4.3 Chapter's Summary

Several requirements may be observed in different, or even similar situations to those we addressed so far. The nomadic behavior of users of computing devices is newly emerging and rapidly advancing. Therefore, we do no claim to have covered all of the Nomadicity requirements but we do claim to have covered a critical part of it by means of tactics and strategies that interacting software agents may employ if considering the nomadic nature of users of pocket computing devices is required.

In the earlier two chapters, we introduced the the negotiation model we believe to have a positive impact on agents that represent users of pocket computing devices and interact to acquire a service. Earlier to that we formalized the general abstract setting wherein we expect our model to be applied then we also gave a clear description of the negotiation issue that any two interacting agents in the context of our research are expected to resolve. Then we characterized the negotiation parties - *Agents*.

We then formally introduced the negotiation protocol and the decision making time-lines. Then, we continued the introduction of our negotiation model by formalizing the set of negotiation tactics and strategies that we foresee them enhancing the communication between users of pocket computing devices and the agents representing them.

Chapter 5

Model Implementation

In this section, we describe the nomadicity-oriented negotiation model we propose for the interacting service-driven software agents. The goal of this model is to reach a mutually beneficial union by one agent to another while considering the nomadic nature of the end-users they both represent. Our model consists of three different stages that negotiating software agents consider throughout their interactions.

These stages are: the negotiation protocol agents apply while performing service-driven interactions, the tactics agents employ to plan their unions, and the strategies they adopt to maximize their benefits. In early parts of this chapter, we realize our model through an implementation example in which each part of our model is abstractly explained from a service application perspective.

5.1 An Overview

In figure 5.1, we summarize the overall negotiation model we presented in earlier chapters with respect to users of Pocket Computing Devices (PCDs), software agents that represent these users in acquiring a service for each other, the *head-agent* that assigns these software agents to different negotiation sessions, and the Wireless Access Points users are supposed to utilize in order to transmit their agents to the central multi-agent platform - *wherein the head-agent is located*.

Looking at the bottom part of figure 5.1, we can see a dashed line circle where a number of users are placed. These users are all assumed to be interested in acquiring a commonly known service one-the-go throughout their PCDs. The number of users and their representing agents, plus the number of wireless access points, are all extendable depending on the addressed sit-

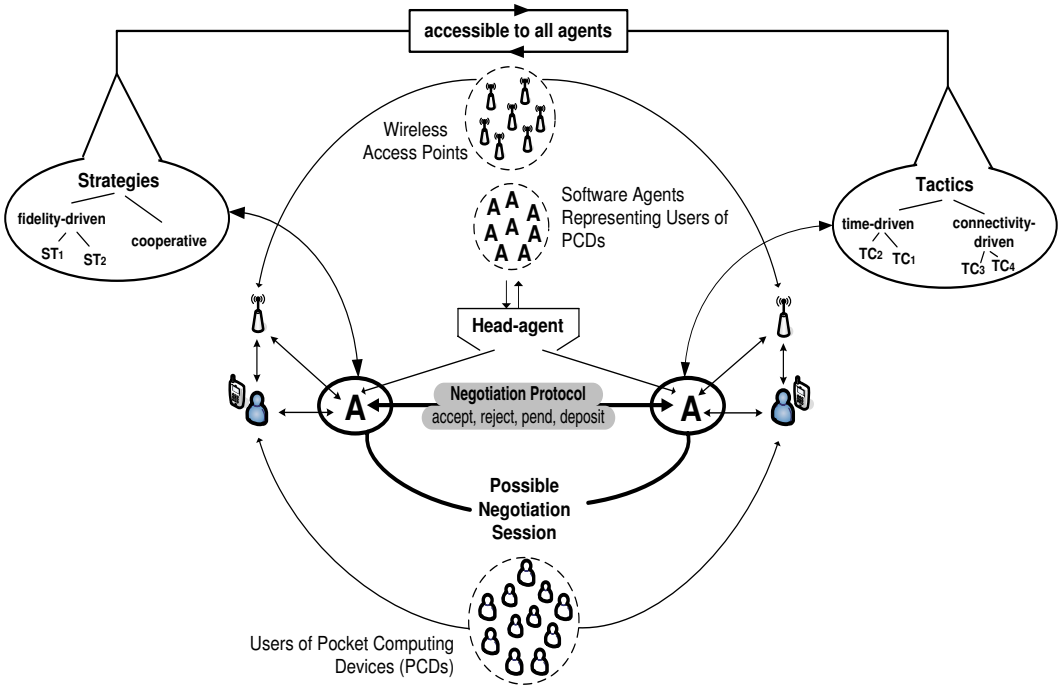


Figure 5.1: Model's Strategies, Tactics, Protocol, Users, Agents, PCDs, Wireless Access Points, Negotiation Sessions, and the *Head-agent*.

uation, therefore, we refer to any possibly to expand entity of the scenario with a dashed line circle.

Looking at center of the same figure, we assume that two of these users have interacted with their client applications of their PCDs, entered their service requests and preferences, and created an agent that will then act on behalf of these users in the virtual space wherein all other users' agents are supposed to meet, (i.e., the central multi-agent platform). It is now the role of a user to communicate this agent, (i.e., click *send*), to the central server by building a transmission chancel with one of the available Wireless Access Points (WAPs).

Following the depictions of the same figure's central part: after the transmission of agents to the managing entity, (i.e., multi-agent platform), the role of the *head-agent* is now to assign a negotiation session to every two accessible agents. The method the *head-agent* follows to select which agent is assigned to which session, (e.g., FIFO), can be manually designed by the provider of the service application. Once an agent is assigned to a session, it is then known to all session's parties that the protocol used to interact with each other encompasses only the following elements: *accept, reject, pend, deposit*, which is our model's negotiation protocol.

The upper part of figure 5.1 has on its right side the set of tactics available for every interacting agent to utilize when a communication with the user it represents is required. The set of strategies an agent can employ to maximize its user's benefits of the currently active negotiation are placed on the left side of the same figure. Both, tactics and strategies, are accessible by all agents while a negotiation is taking place. The central part of the upper section of the figure is showing the set of access points and agents that can be possible found within a context similar to that we address.

5.2 Model's Implementation Circumstances & Conditions

Before we introduce our negotiation model from an implementation perspective we would like to bring to the reader's attention the exact characteristics of the environment we address.

We start by assuming the existence of a user that is interested in a mobile service. This user reflects the service's preferences using the pre-installed client application available on his **Pocket Computing Device** (PCD). User's preferences are then used by the same client application to build what we call a **Service Request File** (SRF). An SRF is a data file that contains information which correlates a user to a specific service conditions and requirements. A unique identification number is then given to the produced SRF in order to distinguish it from others. Later to that, this SRF is made available to any of the **Wireless Access Points** (WAPs) spread in the addressed service-oriented environment.

Once a communication channel is established between the user's PCD and a WAP, the client application uses any of the supported communication technologies, (e.g., Bluetooth or Wi-Fi), to communicate all of the pending SRFs to a service's central server. The server-side application is an agent-oriented platform that offers a virtual space for a number of software agents to interact according to predefined rules - *a Multi-Agent System (MAS).* These rules are known to, and considered by all agents interested to benefit from the service this MAS offers. The data listed in an SRF is later used by the MAS to create a new, or update an existing software agent. This agent is named after the ID of the SRF created by the user's PCD.

In figure 5.2, we summarize the sequence of service invocation steps that exist prior to the time of agents' negotiation. In this figure we use the term *invokeService* to refer to the set of actions taken by a user to start and interact with the client-side application. We also use the term *processSRF* to refer to both, creating an SRF and assigning to it an ID. Since it is technically possible to use part of the user's information while constructing the ID (e.g.,

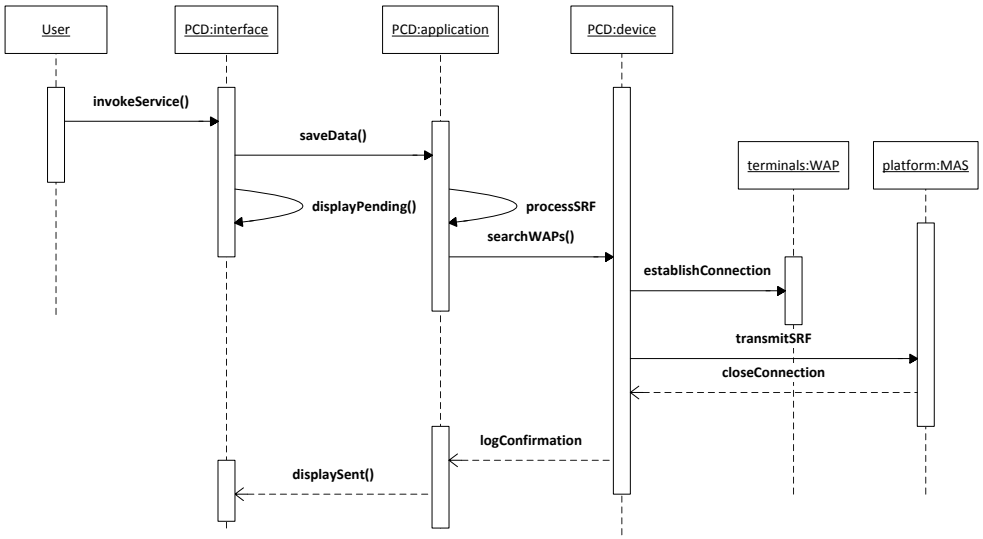


Figure 5.2: A sequence diagram explaining the phases existing prior to the negotiation time

surname), it is then expected that this ID is unique for both the client and server-side applications.

Any user's SRF, in concept, does not differ from the SRFs other users have transmitted to the central MAS. All SRFs are divided into two main categories. These categories are: 1) the *Profile*: it includes the data needed to explicitly describe a user, (e.g., age, sex, and telephone number); 2) The *Service*: it includes three subsections; Desires, Givings and Combinations. The Desires part includes a list of items a user is searching for. The Givings part includes a list of items that a user is willing to offer in return. The Combinations part includes the relationships connecting each of the desired items to the givings ones.

An example of an SRF structure is shown in figure 5.3. In this example we assume that a user "Bob" is searching for a car-ride. Bob uses the client application to put all of the service preferences and then clicks "`save`". The application starts to process the saved data and puts them in the structure of an SRF - showed in figure 5.3. From the Desires subsection, a dedicated MAS can identify the required service and its description from the *serviceName* field and its subsequent fields.

Throughout the GUI of the client application, and in the "Budget" field, we assume that Bob has listed three different givings that each corresponds to a variation of the desired service. These variations are the result of putting together the alternatives a user may give to any of the desired service parameters and, the alternatives a user may assign to the main giving item.

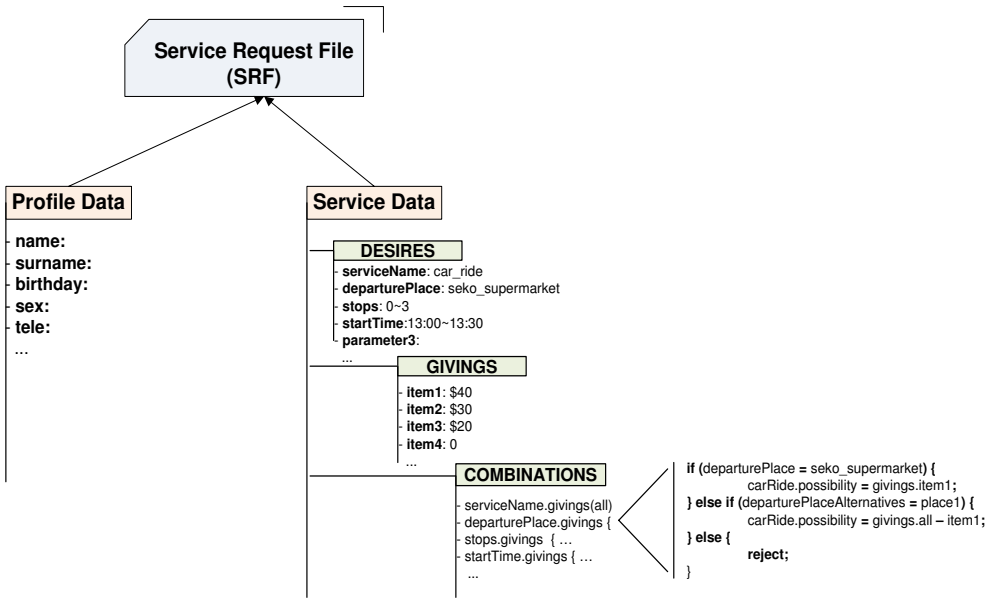


Figure 5.3: An example of an SRF structure for a car-ride service.

For example, given that Bob is also ready to tradeoff his exact departure point with some cost decrease; then, the *departurePlaceAlternatives* is set to another value. As shown in figure 5.3, a relationship that combines both less money and the alternative departure place is made available in the Combinations subsection.

```
.....

if (stops = 0) {
    carRide = '40';
} else if (stops = 1) {
    carRide = '30';
} else if (stops = 2) {
    carRide = '10';
} else if (stops >= 3) {
    reject.Offer;
...
```

or

```
.....

if (stops = 0) {
    carRide = '40';
} else if (stops = 1) {
    carRide = '30';
} else if (stops = 2) {
    carRide = '10';
} else {
    carRide = '0';
...
```

Figure 5.4: Two possible samples of a combinations subsection in a car-ride SRF

Another example of COMBINATIONS occurs when Bob cares to find a car-ride from home to office at 13:30. In return, Bob's givings vary from "40", to "30" and also "10", (USD or whatever imposed currency), depending on the conditions accompanying the offered service. A nonstop car-ride is worth "40" for Bob, one or two stops rides are worth "30" and "10". The *Service* section of Bob's SRF will have "car-ride" in its DESIRES subsection, "40" and "30" and "10" in its GIVINGS subsection, and the COMBINATIONS will have something similar to what figure 5.4 depicts.

Eventually, a number of software agents that represent different SRFs are becoming available on the server side of the service architecture. These agents are expected to achieve one goal each and wirelessly feedback the users they represent. For these agents to achieve their goals they have to rationally interact by means of a predefined procedure that is understandable to all of them - *A Negotiation Protocol*.

Once agents start to engage in a conversation and apply this procedure they are then required to examine the negotiation's possible outcomes, and do the same for all other potential conversations - *Negotiation Tactics*.

At last, an agent needs to take a decision so that goals could possibly be achieved and service requests are fulfilled. An agent performs its final action in a specific process of negotiation with respect to a set of predefined policies. These policies link together each of the negotiation protocol options to the positivity or negativity of all tactics - *Negotiation Strategies*.

I is also worth highlighting here that: at certain points of our model implementation, we had to distinguish between the phase wherein an agent is being created and then transferred from the client side and, the time an agent has actually arrived to the space wherein service-driven interactions are permitted. Therefore, software agents that represent users of pocket computing devices are not playing a role in the concerned multi-agent system unless they have been already formed and sent by users, passed the connecting terminals, and enrolled themselves for upcoming interactions. The reason we do this distinction refers to the fact that we are addressing a nomadic service environment wherein big part of its functionality relies on external entities, such as users, lightweight devices, and connecting terminals.

5.3 The Negotiation Protocol

Accepting a union formation means that both the proposer and the accepter agents will avoid additional system interactions, communicate the reached results with the users they represent, and wait for further instructions. Rejecting a union formation means that both the proposer and rejecter agents will terminate their negotiation process and search for alternatives.

Pending a union formation means that the proposer agent suspends its search activity, and stay committed to the union the pender agent may accept at a certain time, if it did not, the proposer agent then restarts searching for alternatives. In the same time, the pender agent carries on its search for potential union partners. Depositing a union formation will make the proposer and depositor agents temporarily terminate their negotiation, search for better negotiation outcomes, and keep committed to accepting the previously

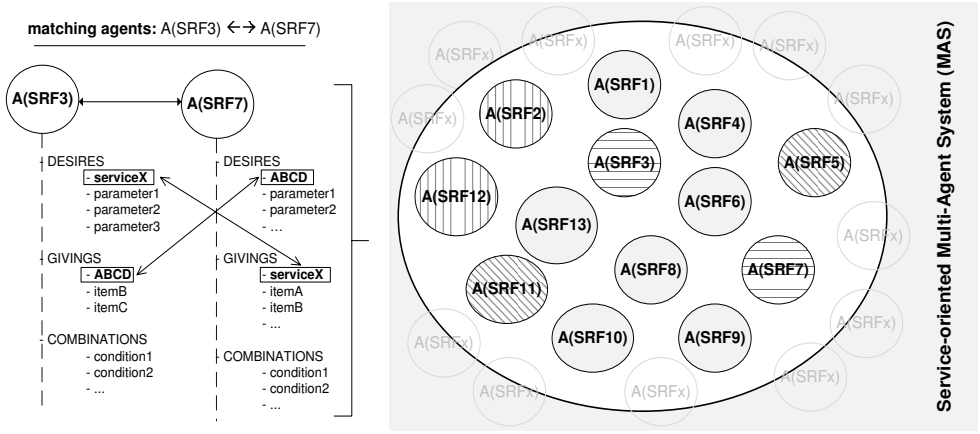


Figure 5.5: A possible view of a service-oriented MAS with matching agents.

proposed union formation till a certain decision-making time occurs.

Later in this chapter, we demonstrate the range of tactics and strategies that software agents employ ahead of choosing any of the four governing expressions.

At present, in figure 5.5, we show a view of a service-oriented MAS where software agents are being associated by matching the substances of their Desires and Givings. MASs, in our context, play a single role before a negotiation process starts, which is matching the desires of the newly arriving software agents' with the givings of other accessible agents. Once a connection between two agents is established by a MAS, their negotiation process is invoked and remains viable till a time for feedback occurs. The MAS requests agents' feedback according to two different predefined timelines that we presented in 3.5.

5.3.1 Accepting a Union Formation

For two negotiating agents to accept a union formation it means that both have agreed to fulfill the service requests of each other. Why would an agent agree to fulfill the service request of another agent? This is what we will come across in the strategies and tactics subsections. Now, in figure 5.6 we show the procedure taken by two negotiating agents as they both agree on a union formation. Once a MAS assigns a negotiation session for two negotiating agents this session is then given a unique number and a system record that relates this session with the participating agents, (e.g., the header of figure 5.6).

As we explained earlier, any negotiating agents will be granted an Ini-

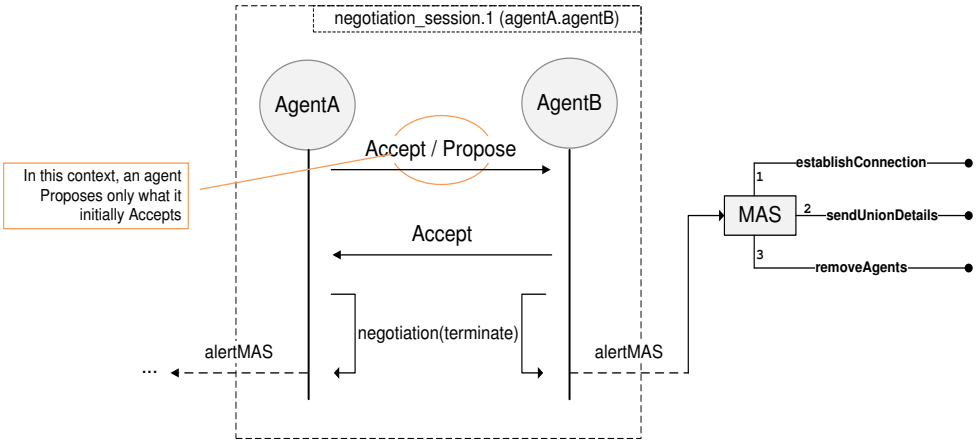


Figure 5.6: A situation where two negotiating agents accept the union formation.

tializing Timeslot (ITS) by the managing MAS to reason about each other's service combinations. In spite of their location within the **DecisionTimeline** we proposed earlier in this section, we assume that agents in figure 5.6 have reasoned about their decision and decided to exchange *accept* messages. Then both agents terminate their current negotiation process and confirm the union formation to the concerned MAS. Three different actions a MAS will perform when it realizes that a negotiation session was terminated. These steps are:

1. **establishConnection**: the MAS communicates with all of its connected WAPs in order to establish a communication channel with the users AgentA and AgentB are representing.
2. **sendUnionDetails**: the MAS communicates the union terms and details to users of AgentA to AgentB.
3. **removeAgents**: the MAS removes AgentA and AgentB from the list of accessible agents.

5.3.2 Rejecting a Union Formation

According to our proposed protocol it is possible for the negotiation session to encounter a different scenario where a proposal for union formation gets rejected. In different words, it may happen that the proposing agent is certain about the benefits yielding of this potential union while the rejecter is not persuaded. In this situation, both, the proposer and the rejecter agents are immediately enforced to start searching for alternative unions.

Unlike the ContractNet protocol [24], in our protocol if an agent receives

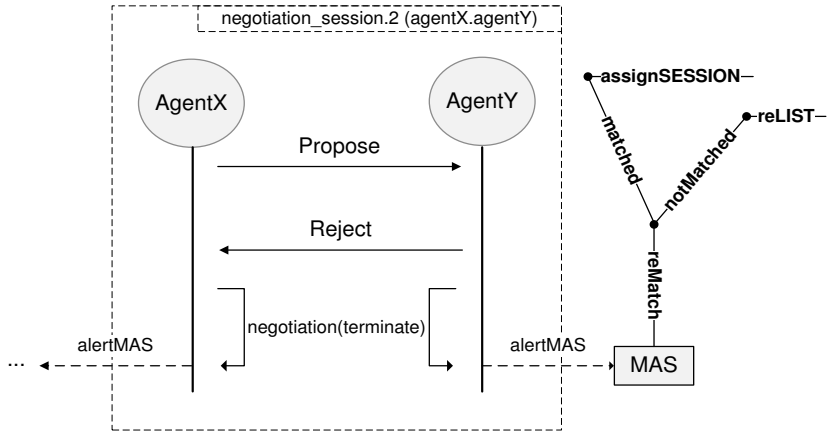


Figure 5.7: A situation where one of the negotiating agents rejects the union formation.

a *reject* to its proposal it cannot modify the proposal's conditions and resend it to the same agent again. Since a user feedback will be required to modify the conditions of an agent's proposal; applying this one-shot rule will reduce the number of exchanged messages between an agent and its user. Besides, proper bandwidth utilization is expected to be achieved.

In figure 5.7, we show the procedure carried out by the MAS in case one of the two negotiating agents *rejects* the other. Once a MAS recognizes that a negotiation session has been ineffectively terminated between two agents, it tries to re-match these agents with all accessible agents. As mentioned earlier in this section, the matching happens by means of fitting the desires one software agents with the givings of other existing ones. If a match is found then a new negotiation session is assigned; otherwise, this agent is then listed in the list of union seekers.

In figure 5.7 we did not show the situation where no *accept* is given by both agents. Since we have mentioned earlier in this section that the negotiation session terminates immediately if one of the two agents *rejects* the other, therefore, if AgentX initially rejects AgentY, there will be no need for AgentY to react and the step where the MAS is alerted will follow instantly.

5.3.3 Pending a Union Formation

Another occurring situation is related to the fact of an agent deciding to pend the negotiation process. In brief, pending a union formation occurs when the proposing agent is fulfilling a view of the union necessities that is

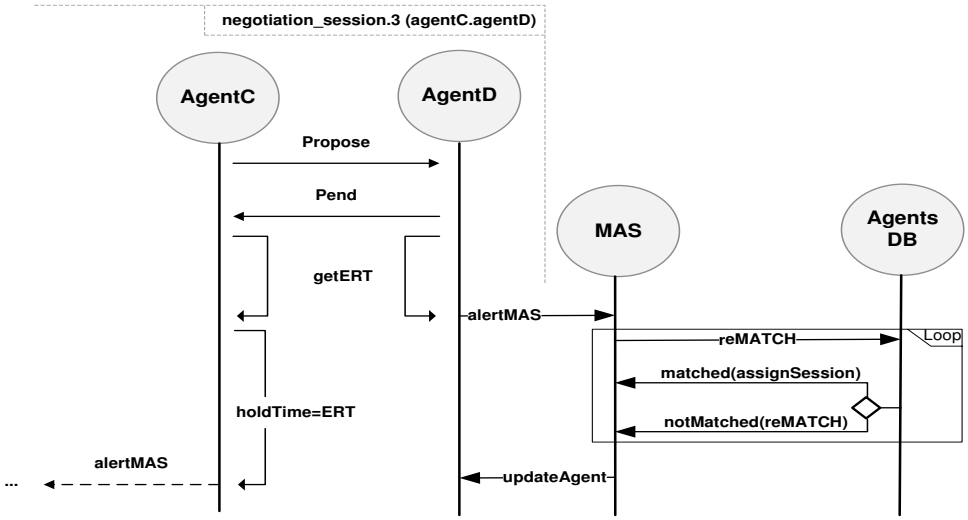


Figure 5.8: A situation where one of the negotiating agents rejects the union formation.

currently of no precedence to the pender agent.

Figure 5.8 depicts the situation where an agent decides to pend the union formation. We use the term *getERT* to refer to the action taken by an agent to calculate the new decision reasoning space. As mentioned earlier in this section, the Extra Reasoning Time (ERT) equals to the gap between a checkpoint of a **DecisionTimeline** and its approaching checkpoint of a **ReleaseTimeline**.

If an agent decides to *pend* a potential union partner within the Initializing Timeslot (ITS), then the *getERT* will return a total reasoning space that is equal to the remaining time of the ITS plus the coming ERT. If an agent decides to *pend* the negotiation process in a time of *decisionRequest* checkpoint then the ERT will be calculated normally.

At anytime of the re-matching process, the pender agent can decide to accept the pending union proposal and terminate the re-matching action of the MAS. However, once the MAS is alerted with the agent's pend decision a particular procedure for re-matching this agent is executed accordingly. The previously calculated ERT applies a limitation to the time a MAS spent on re-matching. Two possible scenarios may occur here:

- **New Match Found**: The pender agent will be assigned to a new negotiation session and the MAS will wait for the agent's feedback as long as the ERT is valid.

- **No New Match Found**: The MAS keeps on searching for a match

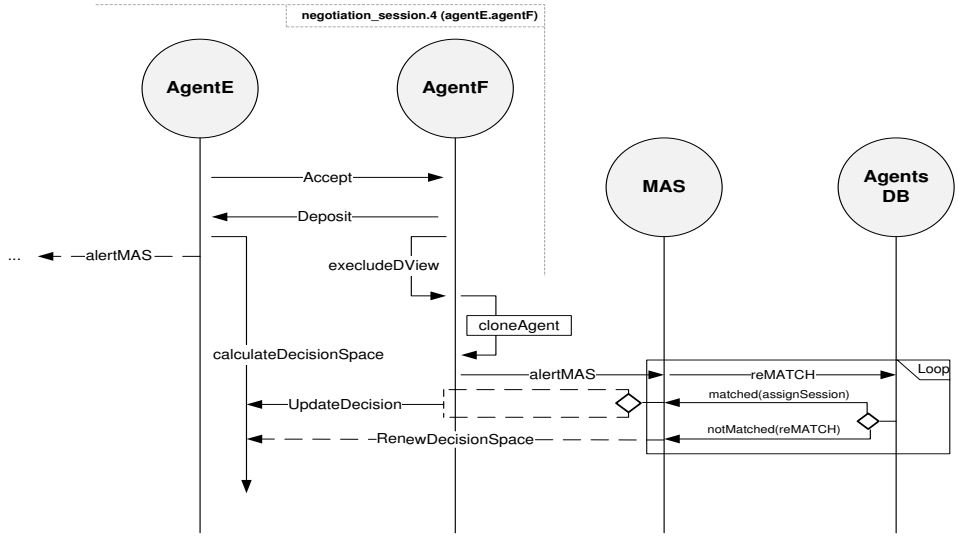


Figure 5.9: The sequence of actions taken by a depositor agent and its managing MAS

as long as the ERT is valid, if no success it returns the pender agent to the pending negotiation session.

Following the depictions of figure 5.8, while pended, the proposer agent is supposed to suspend its external negotiation activities until further updates from the pender agent side, or the expiration of the ERT. We decided to impose this rule as a method for the proposing agent to reduce its search activities when a high probability for union formation is found, and as an availability assurance for pender agents.

5.3.4 Depositing a Union Formation

Different from what happens in an accept-pend situation; depositing a negotiation session here will grant the depositor agent all the required time to ensure the unavailability of better unions. This time will completely depend on the depositor agent willingness to stay searching for alternatives. Therefore, both of the negotiating agents under a *deposit* circumstances are allowed to start searching for alternative unions once their negotiation session has been deposited.

If an agent does not find a new union and the depositing time expires, then the deposited view will be taken as an agreed upon union proposal by the proposer agent, and the earlier negotiation session is then concluded successfully. On the other hand, if the depositor agent succeeds to engage in

a more beneficial union then; the managing MAS will impose a termination action to the deposited negotiation session.

For example, an agent may hold a desired service **X** that is associated with two different items of the Givings; **G1** and **G2**. These two items are the result of two different time evaluations that a user has specified in earlier phases; **T1** and **T2**. A possible view that an agent can deposit could be "if service **X** gets fulfilled in **T2** then **G2** is valid". Recalling the Combinations examples we have given in section 5.2, it should be clear for us the place where views are found and their representation method.

In figure 5.9, we use the term *excludeDView* to refer to the action taken by the AgentF to capture the items and conditions AgentE has previously accepted. Since the nonexistence of any of the two agents in a negotiation session will make this session terminated, we hereafter adopt - *partially* - the idea of agent cloning presented in [95]. Partially because; the agent cloning approach addresses situations where processing, memory and communication loads are of agent's interest to reduce overloading in a data allocation setting. In our case, we do not intend to reduce an agent's overload as much as we are interested in the recreation of the same agent.

Eventually, a duplicate of **AgentF** without the captured view will be created by the MAS and it is then entered to a new matching loop. Whether this matching loop have resulted a successful union or not, or no matching was actually found, the decision space of **AgentF** will have to be always pushed to either termination or renewal.

5.3.5 The Protocol in a Nomadicity-Oriented Setting

We conclude this subsection by locating the negotiation protocol we introduced on the environment we address. To do that, we use the Prometheus Design Tool (PDT) introduced in [96], which is based on *Prometheus*; the Agent-oriented methodology of [97].

In figure 5.10, three main actors are involved in the process of shaping and delivering an agent to its space of interactions. These actors are; User, Pocket Computing Device (PCD), and WAP (Wireless Access Point). The space of interactions is the central Multi-Agent System (MAS) where all of the WAPs are connected to. This MAS is also considered as another system actor with respect to its role in the overall scenario. Agents that represent certain goals attempt to reach the MAS with an objective to interact with one another in order to form a service partnership - ***Union***.

As we mentioned earlier in this section, every user searching to fulfill a specific service request is also willing to give something in return. Therefore, the only agent we refer to in figure 5.10 is of one type - *SeekerAgent* - but

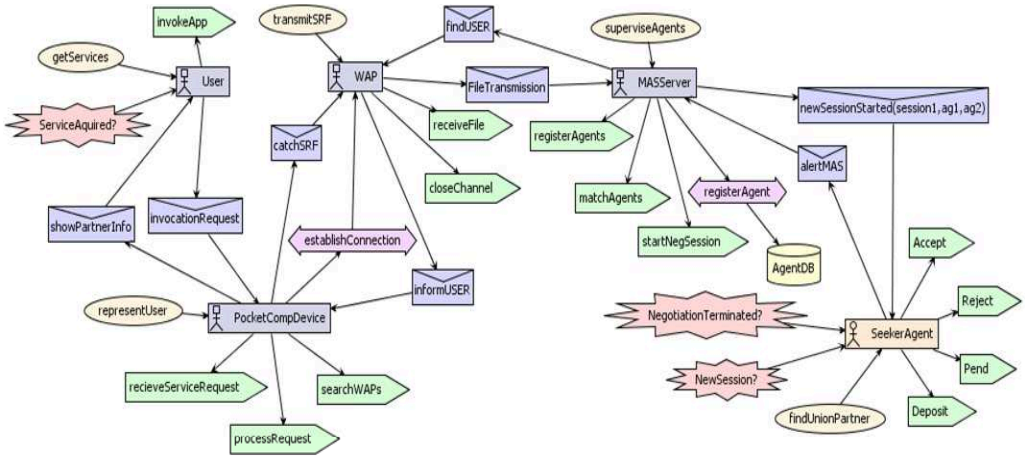


Figure 5.10: A *Prometheus* diagram to locate the negotiation protocol we introduce on the environment we address

yet it signifies both; service seeker agent and service giver agent. An agent is a service giver if its union partner is seeking the same service this giver agent offers. The same agent is also a service seeker if its union partner is giving the same service this seeker agent searches for.

Since *Prometheus* uses the message passing approach to express the type of interactions occurring among system actors, we employ this approach to abstractly describe in figure 5.10 the relations linking together all of the environment's actors. This sequence of messaging starts and ends at the user side of the architecture.

The first message sent by a user to its PCD asking to invoke the client application of the service - *invocationRequest*. Then the PCD, using a specific connection establishment procedure, sends a *catchSRF* (Service Request File) to a WAP. In turn, The WAP transmits the user's SRF to the central managing MAS server using the *FileTransmission* message.

The MAS server communicates with the agent the number of the negotiation session assigned to it and the ID of its potential union partner using a *newSessionStarted(session1,ag1,ag2)* message. Each time any of the negotiating agents takes an action that requires the MAS involvement, or the negotiation terminates, an *alertMAS* message is sent to the MAS. Messages keep on taking the same way back in order to finally feedback the user with his/her partner info.

Following the depictions of figure 5.10, a goal "*oval circle*" is associated with each actor of the system. The abstract goal of a User actor is to get a service request satisfied. The goal of the PCD is to act on behalf of the user when interacting with the service architecture is needed, which we call it *rep-*

resentUser. The MAS actor is responsible of matching agents, imposing the system rules, and receiving and transmitting service requests and feedbacks to users through the WAPs. Therefore, we have summarized all of the MAS objectives in a *superviseAgents* goal symbol. The main goal of an agent is to find a union partner so that together they establish a mutually beneficial relationship - *findUnionPartner*.

A MAS actor has a database linked to it wherein all of the system agents are stored and matched before a negotiation sessions are assigned to them. A specific procedure is followed by the MAS in order to register an agent to the system - *registerAgent*. Using this procedure, a MAS exploits the data listed in an SRF to either create a new, or update an existing software agent.

The matching procedure followed by the MAS while putting together agents in negotiation sessions is not showed in the diagram. However, two parallel concepts are considered while the MAS is attempting to match newly arrived agents with existing ones. These concepts are:

(a) **Clustering organization of agents:** A pre-matching step to categorize agents of the same offered service - giving - in different groups so that searching for potential union partners is more directed.

(b) **First-come first-served:** This is to ensure that the oldest agent in each of previously categorized groups is the first to be examined for matching.

In figure 5.10, our negotiation protocol is represented as four different actions the SeekerAgent is free to choose any of them while negotiating a union formation. These actions are *Accept, Reject, Pend, and Deposit*. Each of these actions shows a different standpoint of the seeker agent regarding a specific union formation. The reason we give the negotiating agents these different actions goes to our desire to reflect four different recurring situations in nomadicity-oriented service architectures, these situations are:

1) **Classic Service Acquisition**: In real-world situations, when a buyer finds a seller that offers its services at a value that is within the buyer's range of givings, they both enter into a conversation to discuss the terms of the deal. The seller's and the buyer's acceptance regarding this deal is affected by a set of conditions that is predefined by each of them separately. Among several examples, these conditions may include the *time* available for both to reach their goals, or the quality of the offered service with respect to its given value.

   Similar to the situation above, once a MAS matches the desired service of an agent with the offered service of another agent, and the vice versa. An agent adopts the *accept* action if it is likely to run out of time and, the

current prospective union satisfies - *at least* - one of the adjoined service conditions. Or, the partner of its current negotiation session initially agrees on the optimal combination of the service conditions.

2) **Optimistic Service Acquisition**: There are situations where a seller and a buyer may not come to agreement although both are apparently fulfilling each other's requests. Taking time as an influential factor; if the expectations of a seller are higher than the estimation of the buyer, and *time* is not an issue, a seller may prefer to risk the offered deal and wait for other buyers. On the other hand, while time is still not an issue, a buyer may reject a deal if the object he/she is searching for could be found in different conditions and the current seller is fulfilling their most inferior.

   In an agent-oriented service application, a negotiating agent would decide to go for the *reject* action if the remaining time to work on its most favorable goal is sufficient. Besides, the partner of its current negotiation session *accepts* to satisfy only a suboptimal view of the service's adjoining conditions. Therefore, this rejecting agent is optimistically choosing to search for better alternatives. However, it is not guaranteed by the MAS that another matching agent will be found soon.

3) **Risk-free Service Acquisition**: Situations where the buyer and/or the seller are hesitating about a trade's final decision may occur in a particular context, (e.g., in a flea-market). This is usually happens when the seller and/or the buyer are uncertain about one of the deal terms, (e.g., price, delivery place), and neither time nor a dire need to upgrade the current state is pushing any of them to finalize the deal. One solution for this situation is when the buyer asks the seller - or the opposite - to put the negotiated item on hold for some time until extra incentive for accepting the deal appears. This is usually happens because naturally people tend to maximize their benefits.

   In the context of our research, an attempt for an agent to maximize its benefits will be done through the Pend action. In addition, using the pend action gives less risk for an agent to achieve its delegated goal than the reject action. This is because a potential union formation is on hold while a pender agent is searching for alternatives. However, as mentioned earlier in section 5.3.3, the awaiting union formation is only valid till certain time. Therefore, a pend action is also reflecting the agent's necessity to satisfy its goal on a shorter run than the Reject action.

   The *pend* action is mostly used if the partner of the current negotiation session fulfills a set of adjoining conditions that is not considered as an

optimal solution. The *pend* action is also used if the time remaining for the pender agent to achieve its goal is nearly ending but yet sufficient to look for other potential unions.

4) **Objective-centric service acquisition**: Another real-life situation may occur when a buyer is persuaded by the offered item and, the seller is also accepting the buyer's evaluation of the negotiated item, but both are not willing to finalize the trade instantly. This is usually happens because either the buyer's time to acquire a service is limited but yet permits him to perform a little check for superior opportunities, or the seller is not willing to risk the current offer but yet would like to add to its trading chances (e.g., combine the negotiated item with another one in a bigger deal). However, a buyer may apply a number of different approaches to encourage a seller towards holding the negotiated item for a certain period of time.

   From these approaches, the instance that is of our interest is when the seller is asking the buyer to deposit a part of the total value of the negotiated item until the buyer returns to finalize the deal. Eventually, if the buyer decides to acquire the item being on hold and, the seller have not found a better deal; the trade is then normally completed. If the buyer, in his little check for superior opportunities, found a better offer, and the seller was not successful to find any, then the deposited part of the value is considered as a loss to the buyer and gain for the seller. If the buyer and the seller have both found better deals while the negotiated item was under a deposit condition, then the earlier agreement is canceled and the new ones take place.

   The differences between a negotiating agent the pend and the deposit actions of the negotiation protocol are:

   - In spite of the total time available for an agent to acquire a service, using the *pend* action allows an agent to search for alternative unions only within a limited time, which is managed through the two timelines a system assigns and enforce to each negotiation session. While in situations when a *deposit* action is used, the time an agent spends on finding an alternative union is continuously renewed until the depositor agent communicates an *accept* action.
   - Different from the *pend* action's situations, if an agent that is searching to acquire a service picks the *deposit* action of the protocol as a response to a union acceptance, the service provisioning agent will also be allowed to search for union alternatives.

Among several advantages, (e.g., addressing different agents' commitment levels), the main reason these differences exist goes back to our intentions to tackle two different scenarios. These scenarios are; 1) when an agent is having enough time to search for all possible unions but yet the managing platform encourages fast settlements to reduce system overload, 2) when an agent is having limited time to reach its goal but the system is yet facilitating the possibility to find prospective unions to elevate the overall service application performance.

5.4 The Negotiation Tactics

In [26], autonomous agents' negotiation is tackled with respect to the domain wherein a negotiation process is executed. As a result, three different domains categories were presented in which each provides different negotiation tactics that are suitable for domain-oriented encounters. These categories are: 1) Task-oriented domains, 2) State-oriented domains, and 3) Worth-oriented domains. Task-oriented domains are a subset of state-oriented domains, which are in turn a subset of worth-oriented domains.

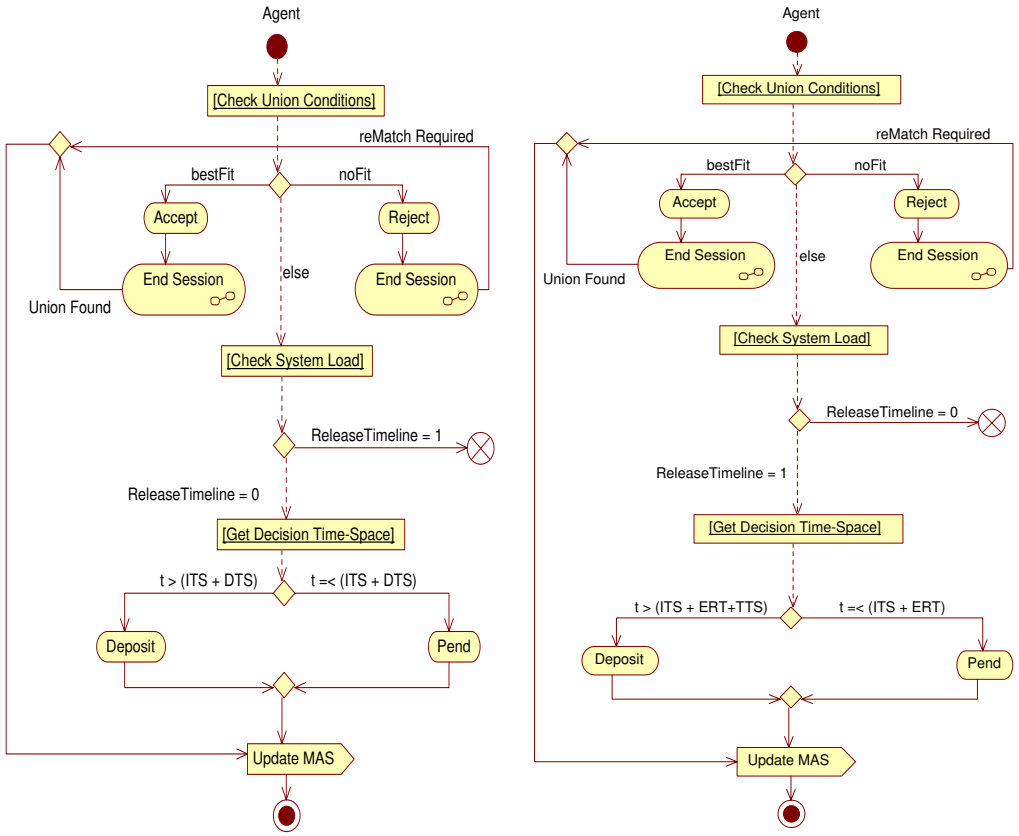
The work presented in [26] is motivated by the abstract notion of machine-machine interactions and, the instance scenarios where two robots encounter in a warehouse and interact to achieve common tasks, or two airplanes are negotiating the landing priorities.

In [27] three families of negotiation tactics are proposed to support agents' decisions while building a counter-offer to an already received offer. These tactics are motivated by the British Telecom (BT) business process management applications authors were addressing at that time. These tactics are; 1) time-dependent, 2) resource-dependent, and 3) imitative tactics. In these tactics new values for each variable in an offer are generated by the decision-making functions proposed in [72].

5.4.1 Time-driven Tactics

In this subsection we explain the time-driven tactics that a negotiating agent is permitted to pick and employ while acquiring a service. As figure 5.11 represents, we show two activity diagrams wherein both are bringing together the notion of system overload and the time available for an agent to establish a union. Within these two diagrams, two different options accompanying each of the abstract tactics are included. In these tactics' options an agent's *deposit* or *pend* decisions are affected by the available decision time-space.

The way a decision time-space is calculated vary depending on whether



(a) Common System State Time-driven Tactic

(b) System Overload Time-driven Tactic

Figure 5.11: The Model's Two Different Time-driven Tactics

the negotiation session ReleaseTimeline is imposed by the system or not. We assume that whenever the ReleaseTimeline - *explained earlier in this section* - is imposed by the system, agents are notified that the number of participating agents has exceeded the average amount; therefore, it is a system overload situation. Otherwise, it is a common system state situation.

Common System State Time-driven Tactic

In figure 5.11(a), once a negotiation session is assigned to an agent by the managing platform, the following step will be taken by each agent to check the conditions a session's partner applies to the union to be made. Three different cases may be encountered, these cases are:

1. **noFit**, wherein none of the conditions the session's partner applies to imminent unions meets any of the checker agent's requirements.

2. **bestFit**, wherein all of the conditions the session's partner is implying to potential unions are precisely fulfilling the checker agent's needs.

3. **else**, wherein the conditions that the session's partner applies to the provisioning of a service are neither fitting the most to the checker agent's requirements nor completely intolerable.

The **noFit** outcome of the [Check Union Condition] process will lead a negotiating agent to a *Reject* activity. Similarly, the **bestFit** outcome of the same process will lead a negotiating agent to an immediate *Accept* activity. Eventually, whether it is an *accept* or *reject*, a sub-activity will follow to terminate the negotiation session and, consequently, log the session's final outcome to the managing platform.

However, the **else** outcome of the [Check Union Condition] will lead a negotiating agent to another process, which is the [Check System Load]. In this process, an agent is basically communicating with the managing MAS to check whether the **ReleaseTimeline** is imposed by the system or not. As explained earlier in section 5.3, the existence of the **ReleaseTimeline** enforces the negotiating agents to set free their current negotiation sessions at certain times, which is a sign of system overload. In figure 5.11(a), we address the case when the ReleaseTimeline does not exist, which will make a negotiating agent moves to the subsequent process wherein the time available to make a decision is calculated, [Get Decision Time-Space].

The later process will have two possible outcomes, these outcomes are:

a) **t > (ITS + DTS):** Here, we use t to refer to the time available for an agent to take a decision about a specific union formation. In addition, from section 5.3, the Initializing Timeslot (ITS) is the time initially granted to a session's negotiating agents in order to examine the conditions of each other. The Decision Timeslot (DTS) is equal to the interval between two subsequent checkpoints located on the DecisionTimeline.

   Recalling the assumption we made earlier in which each negotiating agent has a specific time to achieve its service acquisition goal; that is assigned to it by the delegating end-user. Therefore, if t is equal to more than the sum of the current session's (ITS + DTS), it will be proper for an agent to pick the *Deposit* option of the negotiation protocol. The reason deposit will be the best fit for this situation goes back to the fact that time - *here* - is not a concern for both, the system and the negotiating agent, and the alternative solution - *pend* - will give a meaningless time restriction for the two negotiating agents.

b) **t =< (ITS + DTS):** Another possibly occurring situation while the ReleaseTimeline is not imposed by the system is when a negotiating agent

is actually having limited time to achieve its service acquisition goal. In different words, the decision time-space t available for this agent will be equal to, or less than the sum of current session's ITS + DTS. In this situation, and following this time-driven tactic, it will be best for an agent to pick the *pend* option of the negotiation protocol. Using the *pend* option will restrict the negotiating agent attempts to find alternative unions to a limited time while the current session's partner will not be permitted to do the same; therefore; a less optimal union is guaranteed.

System Overload Time-driven Tactic

Once the managing platform observes a system overload - *the number of participating agent has increased*, an imposition for the ReleaseTimeline is made to give more restriction to the time available for a negotiating agent to reach a union formation. Yet, the use of the *pend* and *deposit* options of the negotiation protocol is applicable.

In figure 5.11(b), we show the sequence of activities a negotiating agent performs whenever the ReleaseTimeline is existent. The differences between this tactic and the one explained in 5.4.1 are related to the way a decision time-space is calculated. The process of calculating the decision time-space here may have two possible outcomes, these outcomes are:

a) **t > (ITS + ERT + TTS):** here, we also use t to refer to the time available for an agent to take a decision about a union formation. Since two different timelines are assigned to negotiation sessions of this scenario, a negotiating agent picks *Deposit* if t is greater than the sum of the Initializing Timeslot (ITS), the Extra Reasoning Time (ERT) and, the Termination Timeslot (TTS).

   By recalling section 5.3, it should be clear that ERT is equal to the time gap between a checkpoint of a DecisionTimeline and its approaching checkpoint of a ReleaseTimeline. Besides, TTS is equal to the interval between two subsequent checkpoints located on the ReleaseTimeline. Therefore, the *Deposit* option here will give to the negotiating agent the best possible time space to search for alternative offers.

b) **t =< (ITS + ERT):** Another situation that may occur here is related to the negotiating agent time to achieve its service acquisition goal. Due to end-users's preferences, a negotiating agent may have a time space that is even less than the time available throughout the two timelines attached to its current negotiation session. Therefore, the *pend* option is picked by the agent if t will be equal to, or less than the sum of the Initializing

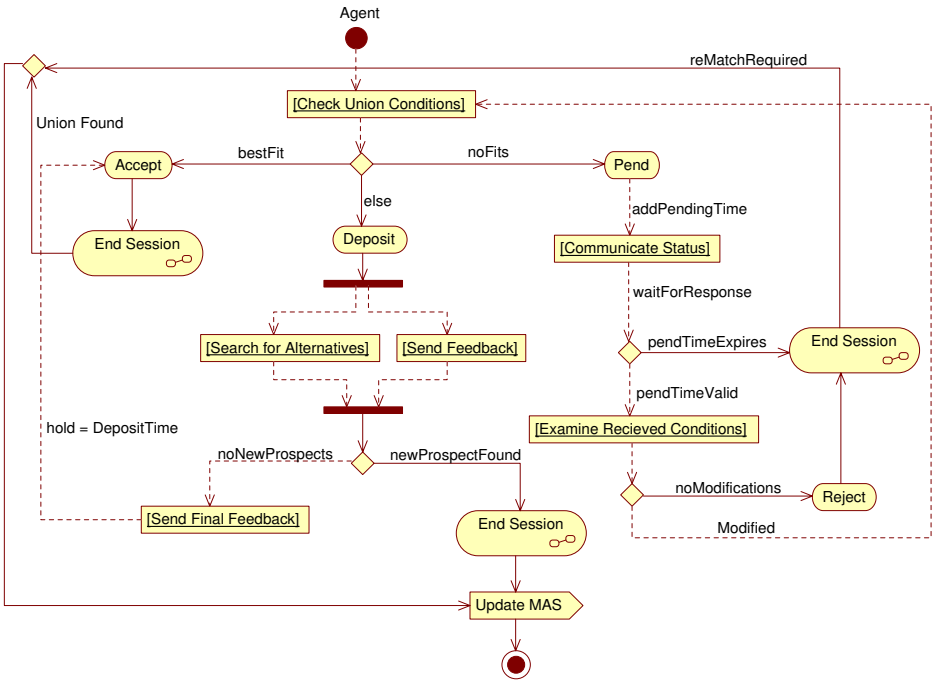


Figure 5.12: Negotiating Agent Actions in Situations of Prompt Communications Channel

Timeslot (ITS) and the Extra Reasoning Time (ERT). Eventually, an agent will take advantage of its total available time - *even though it is not plenty* - to search for better unions while a potential partner is on hold.

5.4.2 Connectivity-driven Tactics

In this subsection, we explain the set of actions a negotiating agent should perform while considering the nature of the communications channel established between the end-user and the service-driven MAS available. We assume the existence of a classification method in which end-users relying on a prompt communications channel are identified and, end-users who are relying on less responsive communications channel are also identified.

By "prompt communications channel" we refer to wireless connectivity that allow end-users to send and receive service requests within a large coverage, and with acceptable data transmission rate, (e.g., Wi-Fi, WiMAX). In addition, by "less responsive communications channel" we refer to wireless connectivity with a number of technological obstacles, such as limited number of concurrent connections, short-range coverage, and low-speed data transmission, (e.g., IrDA, Bluetooth).

Tactic for Prompt Communications Channel

Once a negotiating agent is aware of the communication technology an end-user has used to transmit a service request, it should be uncomplicated for an agent to decide which connectivity-driven tactic to be employed. In figure 5.12, after a negotiating agent is assigned to a specific negotiation session, a process of checking the union conditions a session's partner considers is performed. Traditionally, if these conditions perfectly fulfill the needs of the negotiating agent - **bestFit** - the union is accepted and the session is terminated, and the MAS takes the responsibility of informing the end-users with the session's outcome. Alternatively, other two possible scenarios may occur, these scenarios are:

a) **The *pend* case:** By recalling what figure 5.5 depicts, it should be understandable for us that when two negotiating agents come to a negotiation session it is because the service a session's partner is searching for matches - *abstractly* - the service the other session's partner is providing, and the vice versa.

   Moreover, since the connectivity involved in such scenarios are of encircled area, therefore, the two negotiating agent are actually representing people of the same area. Consequently, if none of the prospective partner's conditions fulfills any of the requirements a negotiating agent carries - **noFits**, a *pend* activity is executed to let the end-user reconsider the service parameters entered in earlier stages.

   Once the negotiation process went on *pend* by an agent, the time a session will remain pending and, the service parameters a session's partner have, are both passed to the MAS in order to be communicated to the end-user, which is shown in [**Communicate Status**]. Eventually, the pender agent activities will be on hold until a response is communicated by the end-user.

   Thus far, two situations may occur here: 1) if the pending time of a pender agent expires without having its current parameters modified by the end-user, then the negotiation session is terminated and the managing MAS attempt to re-match this agent. 2) If the end-user decides to resend the service request, then the received data is examined by the agent. If the service request was modified, then this agent will have to restart the negotiation process. If the received service request was not modified, then this negotiation session is terminated.

b) **The *deposit* case:** Another occurring situation is when the conditions a session's partner is linking to any union formation are neither **bestFit**

nor **noFits**, but they are just partially fitting. We refer to this situation using the *else* term of figure 5.12, and the consequent activity of putting the negotiation session on *Deposit*.

Two parallel actions will be performed following to the *Deposit* activity. These actions are: 1) the [**Search for alternatives**], in which an agent will exclude the current agreed upon session's conditions while searching for other prospective union formations. 2) The [**Send Feedback**], wherein a negotiating agent - *through the MAS* - will communicate the deposited view of the negotiation session with the represented end-user without waiting for responses.

Then, if no new prospective unions are found, and the depositing time is still valid, the negotiating agent holds its final decision until the depositing time expires and then *accepts* the union formation. The reason a negotiating agent holds its final decision until the depositing time expires goes back to the probability that an end-user may communicate any modifications to the system with respect to the feedback communicated to him/her in the earlier phase. However, if a new prospective union is found by the same depositor agent, the current negotiation session is terminated since in our model an agent cannot be involved in two different negotiation sessions at a time.

5.5 Chapter's Summary

In this chapter, we mainly showed how the negotiation model we presented in earlier chapters can be implemented. We introduced a generic implementation description as a step on the way to facilitating the integration of our model in several of the increasingly emerging services for users of pocket computing devices. At certain points, we emphasized our intuitions from approaching specific situations in particular manners, (e.g., the introduction of the Service Request File (SRF)).

In general, we described the overall environment that we foresee our model applied within. Then, we went through users' interactions with this environment. We have also explained our notion of *accepting*, *rejecting*, *pending*, and *depositing* a possible union formation. Then, we employed one of the commonly used agent-oriented methodological tools, figure 5.10, to reflect the proposed negotiation protocol on the addressed environment. We showed our intuition of proposing this negotiation protocol's elements and, connected all that to real-life examples.

To wrap this chapter up, here is a brief connecting description of all what

we demonstrated: we reflected the sequence of service invocation steps that exist prior to the time of agents' negotiation. Using a sequence diagram, figure 5.2, we showed a nomadic user's interactions with its Pocket Computing Device (PCD), this device's interactions with the connecting terminals - *Wireless Access Points (WAPs)*, these terminals' interactions with the agents platform, (e.g., Jade or JACK), and all the way back.

We also showed the expected structure of the Service Request File (SRF) we are expecting a user to manage to create through his PCD, figure 5.3. Then, we gave an abstract intuition of the multi-agent system wherein these earlier SRFs will be represented in software agents' fashion, figure 5.5.

From an implementation perspective, in subsection 5.3, we also showed how software agents can employ our negotiation protocol in four different scenarios, accept, reject, pend, and deposit. Since every negotiation session involves two software agents, then, in this section, we also showed the reaction of the other negotiation party to each of our protocol actions.

In subsection 5.4, using activity diagrams, we showed how our time-based tactics can be implemented in two scenarios: when a community is *Jammed*, and when it is *Tolerable*. Following to that, and using the same approach, we showed how to implement our connectivity-driven tactics, figure 5.12.

Chapter 6

Case Studies

In this chapter, we present two case-studies that we have been working on with our industrial partner, (i.e., ARS LOGICA [98]), for the last 4 years to provide a ridesharing service, (i.e., Andiamo), and a bartering service, (i.e., BarterCell), to users of Pocket Computing Devices in Trento.

Apart from these two case-studies, it is also worth highlighting here that in our Laboratory for Mobile Applications (LaMA) [99] we attempted to address other several needs that the environment surrounding us, (i.e., university), has brought up to our attention. We did that by building a number of agent-oriented software architectures that provide location-based services to students and professors on-the-go; through wireless access points.

6.1 Andiamo

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

In classical agent-based rideshare systems [100], users' desires are represented by delegated agents, and a super agent is available to match the service requests these agents are carrying. However, several contributions were made to the literature of rideshare systems. These contributions has similar notions but different implementation approaches, besides, they are not reachable by holders of lightweight devices.

Rideshare systems allow a substantial number of people to mutually bene-

fit from using less cars in a specific region. This would rationalize energy consumption, save money, and decrease traffic jams and pollution. However, accessibility issues have prevented these architectures from being widely spread. This rideshare system is an agent-based and it is accessible via pocket computing devices. At the time of developing this application - *Andiamo [101]* - we were aiming for accelerating agents' interactions while resolving end-user composite tasks; therefore, *Auction* mechanism was used as a method of negotiation among autonomous and proactive agents.

6.1.1 Application Motivations

Apart from the fact that this PhD's sponsor - *a private sector* - had demanded from us to come up with something that is agent-based and yet of great commercial impact, yet, there are some repetitive and related situations that occur in our daily life. Looking at one of them, we found that using a car to move from a place to another will increase the flexibility to schedule appointments, reliability and comfort. We also noticed that, on the long run, pollution and stress caused by traffic jams will badly affect our life.

If a tool is provided to match peoples' common interests, it would increase cooperation and simplify lots of our daily tasks. It is quite common to see a car owner that has empty seats and commuting daily between two fixed locations (e.g., house and work). It is also common to see commuters of public transportations taking the same route everyday, or people trying to move between different remote places in irregular times.

In fact, average car occupancy in the UK in 2004 is reported to be 1.59 persons/car; 1.2 for commuters and 2.1 for holiday [102], 38% of people traveling in a car in 2004 were unaccompanied drivers, 25% were drivers traveling with one or more passengers and 36% were passengers. And, in Germany is even less [103]

For example, Bob and Alice are two potential system users that are located at the train station, while John is another potential user that is located at the bus stop nearby. Bob has arrived to the train station driving his car, but Alice and John do not have cars. All of them are coming from different locations and again moving to different ones, but it happens that John and Alice destinations are located on Bob's way to work. Moreover, the cost of taking public transportations to reach Alice and John destinations could be of need to Bob in exchange of offering them a ride.

Bob is now able to use his cellular phone to offer a ride to John and Alice that were both using their smartphone or PDA to seek a ride. Intelligent software agents are now delegated by users to seek and offer car rides, then agree on a sharing cost, meeting points and times. According to the pre-

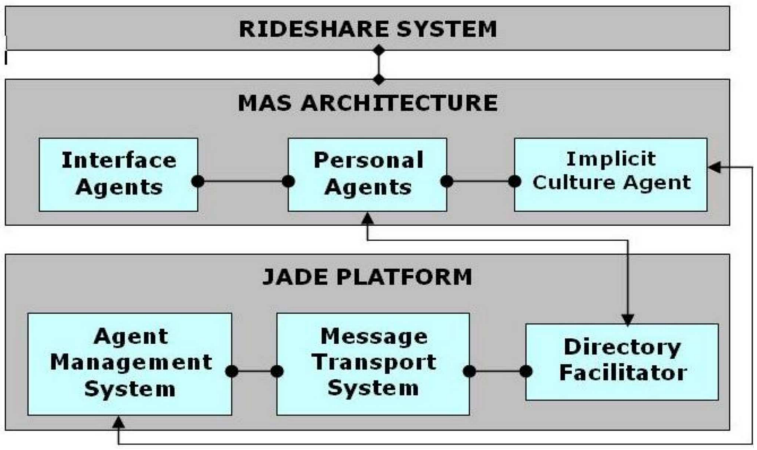


Figure 6.1: The Three-layer Model

configured level of interactivity, these agents would take decisions on behalf of users and communicate the final results.

In a classical agent-based Rideshare system, agents' behavior, level of interactivity and decision making schemes may lead to disagreement and imprecision. On the contrary, an auction mechanism can aid to resolve complex situations; this mechanism ensures ultimate benefits gaining for both service supplier and demander.

6.1.2 The Framework

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

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

As shown in Fig. 6.1, our framework has three different layers. 1) The agent platform, which is based on JADE framework. 2) The Multi-Agent architecture including the Culture (IC) part. 3) The top layer is the handler of user requests and the interface line, which is the Rideshare service management method. Further on, we explain them in details.

The ***MAS Architecture Layer*** is implemented using JADE (Java

Agent Development framework) [9], which is a FIPA-compliant [104] framework for MASs development. JADE provides: (1) an Agent Management System that allow us to create agent containers in distributed hosts, (2) a Directory Facilitator (DF) that provides a yellow-pages service, and (3) a Message Transport System that handles agents communications.

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

Interface Agents

Interface Agents (IA) employ certain techniques to provide assistance to a user dealing with a particular computer application [105].

In Andiamo, each IA has its main features, which are; knowledge acquisition, autonomy and collaboration. At this phase, IAs are managing the creation of new service requests. These agents are recognized by the system as 'AddNewServiceAgent' and they receive from the preceding layer the service request including user details parameters (Bluetooth address, user information, and service request details). Later on, they transmit this data to the corresponding PA and they remain in the system only to achieve this action and then vanish.

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

Personal Agent

These agents represent the system users and the work done on their behalf. The interactions of these agents include two main parts: (1) the elaboration of users' requests, and (2) the negotiation among agents. The PA that elaborates a request for a ride is called **Seeker Agent** (SA while a PA that

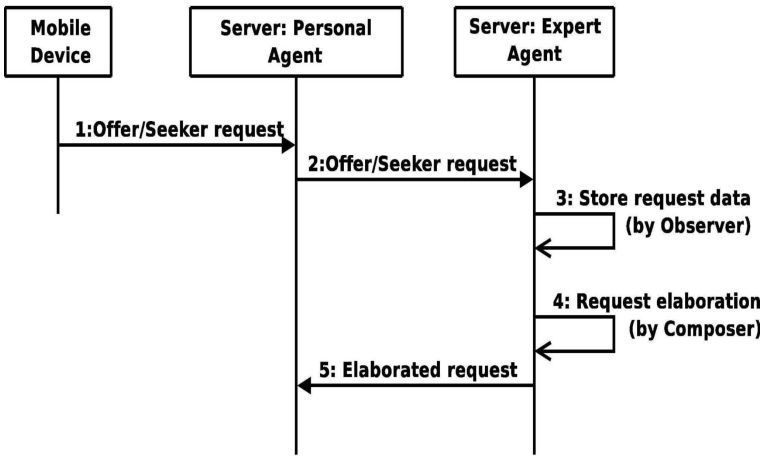


Figure 6.2: User-Request Elaboration Process

elaborates a ride offer is called **Offerer Agent** (OA). The mechanism used for agents' negotiation will be explicitly illustrated in Section 6.1.3.

Agents Interactions

In Fig. 6.2 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) [106, 107]. The SICS consists of three components, 1) the Observer, which uses a database of observations to store information about actions performed by users in different situations, 2) the Inductive Module, which analyzes the stored observations and applies data mining techniques to find a theory about the community culture, 3) the Composer, which exploits the observations and the theory in order to suggest actions in a given situation.

In Andiamo, the use of the IC framework (Implicit Culture Agent, Fig. 6.1) is to let the system suggests the meeting points that are frequently used by other system users and observed by the system. More details about the IC framework are available in [108].

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

SICS needs to gather information about users behavior. To observe user's behavior, EA extracts data from the requests it gets from the PA. Two other additional sources of observation could be added. The first is the database where the results of agent negotiations are stored. This storing takes place every time two PAs agree on sharing a trip and send their proposals to the database.

The EA extracts necessary information (e.g., departure, arrival place and meeting points) from the proposals and stores them in its internal database. The second source is the user's feedback repository. When the ride is finished, the user gives feedback to the other user(s). His evaluation is stored in the mobile phone/PDA and is sent sent to the EA as soon as the user establishes connection with the corresponding server via his mobile device.

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

In Fig. 6.3 a typical Rideshare service transmission protocol is demonstrated. The Multi-Agent platform is located to serve the interactions between an OA and a SA. A sequence of steps is taken between both, offerer and seeker, in order to achieve a successful Rideshare agreement. In addition, the possibility to apply an automatic or a semi-automatic service mode implies that the interaction between two agents can be, either interrupted to prompt an inquiry to the user, or self-decision making. Following to that, we describe the significance of negotiation and we consider its automatic service model.

Service Publication

Within the JADE DF, every PA publishes its carried service requests. If these requests are recognized by the system and that PA is identified, then the service will be registered. The initial interaction starts when a SA tries to find a OA with similar destinations, day and time of departure and with a feedback greater than or equal to the desired value.

Then, the SA will contact every OA found by the system and consequently, communicates with end-user the retrieved data. Notably, if the value of the feedback is less than the requested value, it is possible to contact back the user asking a permission to decrease the requested feedback value so an agreement could be reached. The same thing happens for the time and other similar parameters.

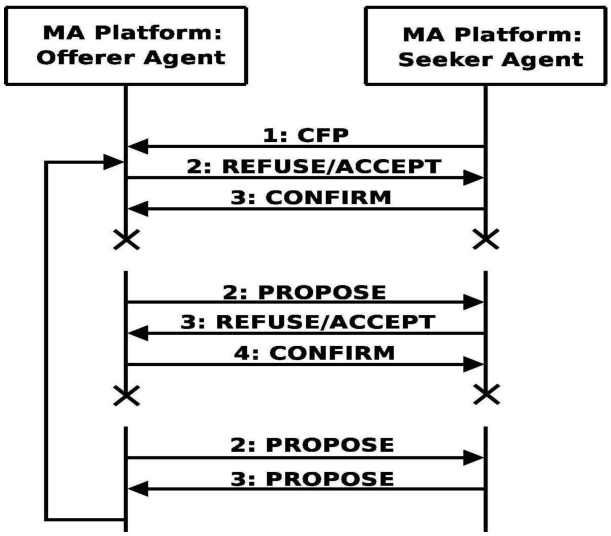


Figure 6.3: A Typical Rideshare Transmission Protocol

6.1.3 The Auction-based Negotiation

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

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

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

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

Algorithm 1 The procedure taken by the Offerer Agents.

to receive new offers from all involved SAs, and a `'val'` is created as a function to calculate the currently obtained best-offer-value.

From `line 8` to `line 10`, if the algorithm had its first round and a `'val'` is gained, the `'bestvalue'` in `line 1` is now updated with the value of `'val'` and the number of loops `'numLoop'` is incremented.

Otherwise, since it is not the first loop, from `line 11` down to `line 19`, the OA checks whether the `'val'` function is increasing in comparison with the previously obtained best value or not.

If `'val'` is greater, the value obtained from the concerned SA is communicated with other SAs and, they are asked to communicate new offer if applicable, then the algorithm is restarted, `line 14` and `line 15`. If the `'val'` is less or equal to the best value previously obtained, the auction is suspended and the best-bid SA wins (the `'bestValue'`), `line 17` to `line 19`. Finally, the algorithm terminates and the auction scenario is ended, `line 20` and `21`. Later to that, we explain the SA behavior in response to OA.

Algorithm 2 is used on the SA side to determine the significance of its role in the impending auction. In `line 1` and `line 2`, a variable `'SABestOffer'`

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

Algorithm 2 The procedures taken by the Seeker Agent.

that carries the SA best offer value is created and set to `'0'`. A variable `'sent'` is initially set to `'false'` and it changes to `'true'` only after a SA has communicated his offer. From `line 3` to `line 6`, SA holds its offer transfer until a communication was received from the OA asking for an auction participation. The SA puts the results from the evaluation function into the `'decision'` variable.

From `line 7` to `line 9`, if the SA accepts the call for auction, a self-revision for the holding parameters is made. This revision refers to the SA insistent to obtain the auctioned item; therefore, it is made with the intention to show extra negotiation flexibility. The part from `line 10` and down to `line 13` refers to the comparison made by the SA to put together the newly obtained value and the existing one. If the new value obtained is greater than the previous one and greater than the `'bestOffer'`, the future offered value `'SABestOffer'` is set to be new one `'newVal'`, and the offer is sent to the concerned OA.

`Line 14` to `line 16`, if the self-revision made by the SA has yielded a disappointing result and the value gained is the same as the previous one, this

specific SA does not send the previous value if `'modificationArePossible'` is `'true'`. The SA continues to review the carried parameters until `'modificationArePossible'` becomes `'false'` or it communicates new best offer. If `'modificationArePossible'` stays on `'false'` and parameters are not sent, SA communicates same offer.

However, from `line 17` to `19`, if SA refuses the auction call, the algorithm terminates. If the user has an inflexible behavior, the algorithm passes the first condition on if (`decision == accept`) but the successive while (`modificationArePossible && !sent`) return `'false'`. The method `'decideIfAcceptOrRefuse'` return `'refuse'` if for instance, a SA has a lot of time before the deadline to achieve the task; therefore, it postpone auction participation. Finally, the algorithm is terminated and the auction scenario is ended, `line 20` and `21`.

Auctioning among agents requires high level of agent-to-user interactivity and increased level of network resources consumption; therefore, agents' intelligence appears when a repetitive scenario occurs. If system user is configuring the mobile-based application to repeat the same service request on daily or weekly basis (e.g., common in mobile news exchange service or carpooling), the created demanding agent would participate in system auctions only if needed.

Once an agreement is settled between a specific supplier and a demander at a certain price, the next time this demander agent will first look-up the very exact supplier agent, which has potential agreement than others in early agreement. This is due to learning agent behavior that maintains an array that saves last successful agreement details.

6.1.4 Rideshare System Layer

In this section we describe the general architecture used for our service delivery. We start from system requirements to the various sub-components and their interaction. The architecture is obtained by extending and customizing ToothAgent [109] [110] 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.

System Components

- **The mobile device** communicates the user's requests with the servers and receives the results.

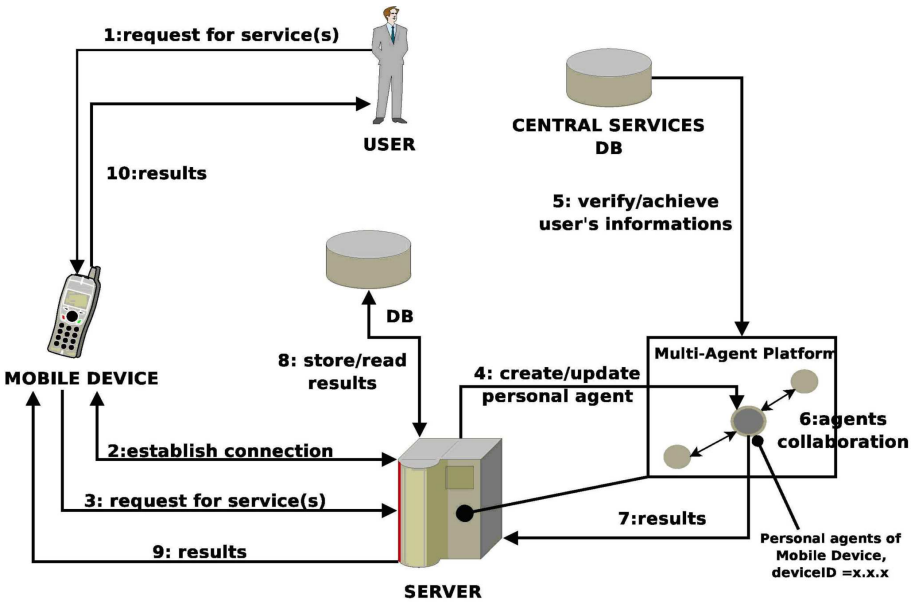


Figure 6.4: Mobile-to-Service Accessability Scenario

- **The distributed servers**. Each of them contains: 1) a Multi-Agent platform with PAs each of which is representing a single user, 2) a database where results are archived, 3) an interface responsible for establishing connections with mobile devices and for redirecting the users' requests to the corresponding personal agents.

- **The central services database**, accessible via web. It contains information about all the servers and their properties, such as name, location, etc. The DB also stores the information about users registered to the system.

Fig. 6.4 illustrates the general architecture of the system and the interaction among its components. The connection between the mobile device and the server is established through Bluetooth but, as we say in Section 6.1.1, it can be established also through SMS or GPRS/UMTS. In particular, a user's cellular phone communicates to the server all the requests, and then receives back the results. The cellular phone may also receive inquiries about a possible modification in the decisions taken by system users and re-communicate the reply with server.

Moreover, cellular phone can be used to send the partner evaluation score, which will be reflected in the future feedback value for whoever will offer/seek a ride. From the server side, a contact is made to the central service DB to check the user's information (age, feedback, etc). Later on, the server updates

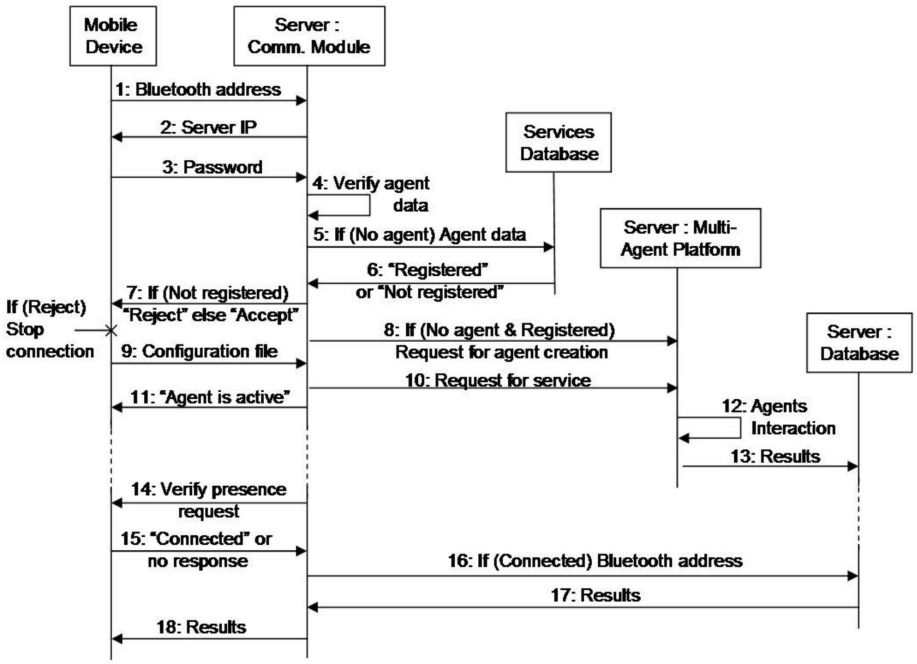


Figure 6.5: Service Accessibility

the offerers and seekers reputation value. The central DB is responsible for storing all the information related to specific service request and the interactions made by its two PAs.

Service Accessibility

Three steps for a user to access the services: (1) to complete a mobile-based identification form; (2) to run the Bluetooth application on the mobile device, and (3) to operate a certain function to activate the required service. The application is written in Java and uses JSR-82 [111] and JSR-120 [112] which are the Bluetooth and the Wireless Messaging API for Java. The application starts a continuous search for Bluetooth-enabled devices in the neighborhood and whenever it finds a server, the software on the device establishes a connection with the server (step 2) and sends the requests related to the Rideshare services (step 3).

The request is then processed by the server and the results are sent back to the user (step 4 -10). The mobile device stores the server's address to keep track of the contacted servers (see Fig. 6.4).

Fig. 6.5 shows the protocol we use for the interaction between different components. A specific communication module on the server is responsible

for managing the interaction with the mobile device. This module receives the Bluetooth address and the password from the mobile device (steps 1 and 3) and checks in the platform running on the server whether a PA is assigned to that mobile device (step 4).

The module employs the Bluetooth address and the password to map the mobile device with a specific PA. If there is no PA previously assigned to this user, the communication module connects to the central services database and verifies whether the user is registered to the system (steps 5–6). In case of a positive response, it creates a new agent and assigns it to the mobile device user (step 8). Then, the mobile device sends the configuration file to the communication module (step 9), which forwards all the user requests to the appropriate PA (step 10).

The PA then starts interacting with other agents on the platform trying to satisfy all the user requests (step 12). In our example a PA receives one or more requests for finding or asking rides. If the agent reaches an agreement with another agent about their users requests it stores the results locally in the server database (step 13). Later the results could be sent back to the user (steps 14–18).

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 Fig. 6.6). The mobile device stores the IP addresses of all the visited servers in an XML file that is used later on to retrieve all pending results. The format of the results produced by the PA may contain the request identifier, contacts (e.g. phone number) of the users interested to share the ride, the departure time, etc.

If the user does not want to use the SMS service, he/she receives the results immediately in his/her mobile device, but only if he/she is still within the Bluetooth server range. In this case, the communication module checks the availability of the mobile device and sends across the results stored in the internal server database by the corresponding PA.

Fig. 6.6 shows the interaction protocol of retrieving pending results via mobile device. Considering our running example, a situation in which a user is close to the server of the train station. After establishing the connection, the mobile device sends the list of IP-addresses of all the previously visited servers (e.g. university servers, city center servers, etc.) to the train station server. The communication module of the server sends the Bluetooth MAC address of the mobile device to all listed servers (step 3).

In turn, the communication module of each server extracts from the in-

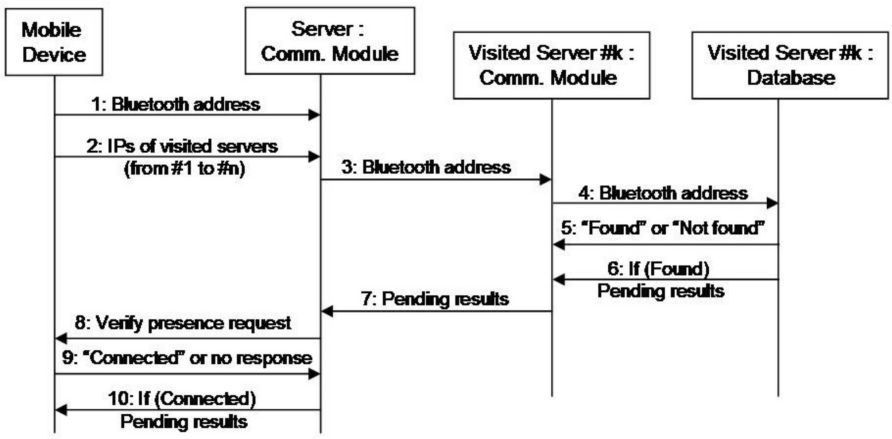


Figure 6.6: Pending Results Retrieval

ternal 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 canceled. Yet these results will still be accessible via the original servers. Therefore, as we already said, a possibility for the server to communicate with the user through SMSs is achievable.

Experiment Facts

We tested the system using Nokia 6630, N73, N70, 6600, Motorola v3 and Sony-Ericsson P910 mobile phones and PC/Server equipped with generic Bluetooth adapter. Bluetooth communications have been implemented using BlueCove [113] which is an open source implementation of the JSR-82 Bluetooth API for Java. We have tested the system on different scenarios, and and in those circumstances we have involved a number of people (students of the university, workers and citizen).

From these tests, we have obtained significant results which were stored as reference. We notice the time to obtain an agreement between two agents is the same for every situation. In the cases in which there are more seekers than the seats offered by the offerers, the agents winning the auction are always the stronger agents (i.e. the agents that offers much money, that have an higher feedback, etc.). A limitation of this model, however, is the lack of a monitoring process of the number of active agents in single MAS.

6.1.5 Andiamo in brief

In this part of the thesis we presented an implemented application of a Mobile-based Rideshare service where Multi-Agent system and Bluetooth and other wireless communication technologies are combined to support co-localized communities of users. We discussed the architecture of the Multi-Agent platform applied for our system, the specific protocols used and the algorithms that have been implemented to realize the Agents interactions and negotiations.

Here, we would like to highlight the fact that this application was lucky enough to get commercial as it was bought by the Province of Trento. Therefore, a realistic verification process of the system scalability through real-life use and a performance test with a considerably high number of users and for a long period of time was made feasible and currently it is a work-in-progress.

6.2 BarterCell

Bartering is a disappearing type of trade where 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.

On behalf of nomadic users and through the use of computing pocket devices, agents of BarterCell can efficiently operate in wireless networks, cooperate to resolve complex tasks and negotiate to reach **Bartering** agreements while attempting to maximize their utilities. In BarterCell we introduced 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 examined 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.

6.2.1 BarterCell Architecture

The architecture of BarterCell, as shown in figure 6.7, relies on the user's capable pocket devices or PC to accomplish a successful bartering task. Via the pre-installed Java application, users start by creating their own profiles

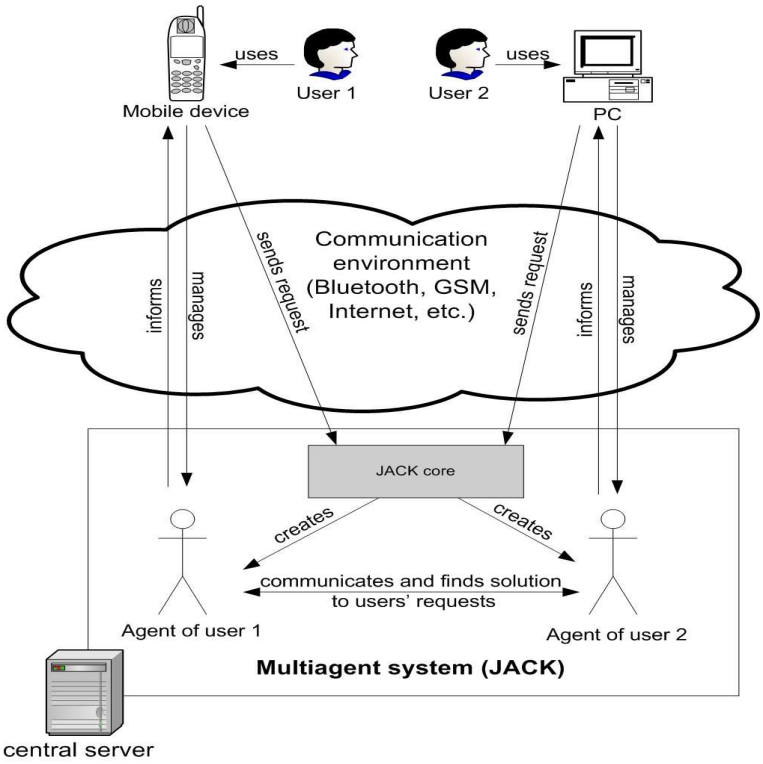


Figure 6.7: The architecture of BarterCell.

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 [114], which is an interactive 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 [115] or FIPA ACL easily accepted by the running architecture.

6.2.2 BarterCell's Adhoc Negotiation Algorithms

Bartering could involve 2 or more users, if there will be some group of users that will mutually satisfy their needs. Such groups of users and order of items to be exchanged will be found by one of agents in the system. Let's call it "main agent". Later on we will describe why we need it, what it does and how other agents will know which agent is the main. Negotiation algorithm considers offered item name, price (estimated by owner), short textual description, time to start offer, period of time to offer and type of the item. The type is defined by user through ontology of item types. Ontology will be predefined during localization of the system. System localization will consider features of working environment, such as physical parameters of a place, social characteristics of system users, etc.

In our application, people using BarterCell should have a capable mobile device with a client application installed in it. This client will serve as interface for a user to access the server-side application where our negotiation algorithm has been implemented. First user will specify what exactly he wants to exchange and what he wants to get from others. This information will be stored in a personal user profile. This profile will be used by software agents. Autonomy of agents, their ability to represent user behavior in real life and their intelligence (that is defined by implemented algorithm) allows having the most efficient bartering between users of the system.

Upon user request, a personal agent will be activated and algorithm 3 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

```
Service_Builder_procedure()
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 ∈ {agentList - 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
```

Algorithm 3 BarterCell's Service Builder Algorithm

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 4"', 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).

There is an additional algorithm - *not showed here* - that is used inbetween 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, ChainMaker 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 queuedChains[] every time ChainMaker gets information of refused chain (Lines 18-22). Accepted bartering chains will simply removed from queuedChains[] (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), ChainMaker will inform all agents from that chain of being involved into it (Line 32). 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

```
1:refusedChains[] = queuedChains[] = newAgentsQueue = optimalChain = NIL;
2:treeRootAgent = currentAgent;
3:while (currentAgent.demands ¡¿ NIL)
4:  chains[] = NIL;
5:  newAgentsQueue = searchNewAgents(agentsList);
6:  if (newAgentsQueue ¡¿ NIL) then
7:    for all a_i ∈{agentsList - currentAgent} do
8:      inform(a_i, "'service finished"');
9:    end for
10:   return NIL
11:  else
12:   for all a_i ∈{agentsList - currentAgent} do
13:     inform(a_i, "'new cycle start"');
14:   end for
15:  end if
16:  for all chain_i ∈ queuedChains[] do
17:   if (hasDecision(chain_i))
18:     if(chain_i.decision == "'No"')
19:       refusedChains[] = refusedChains[] + chain_i;
20:       cDList = restoreCommonDemandsList(chain_i);
21:       cOList = restoreCommonOffersList(chain_i);
22:     end if
23:     queuedChains[] = queuedChains[] - chain_i;
24:   end if
25:  end for
26:  chains[] = findShortestChain(agentsList, treeRootAgent, refusedChains[], "'S"');
27:  chains[] = chains[] + findShortestChain(agentsList, treeRootAgent, refusedChains[], "'SF"');
28:  chains[] = chains[] + findShortestChain(agentsList, treeRootAgent, refusedChains[], "'SFP"');
29:  if (chains[] ¡¿ NIL)
30:    optimalChain = chooseOptimalChain(chains[]);
31:    for all a_i ∈ optimalChain do
32:      informOptChain(a_i, optimalChain);
33:      cDList = removeFromCommonDemandsList(a_i.demands, cDList);
34:      cOList = removeFromCommonOffersList(a_i.offers, cOList);
35:    end for
36:    queuedChains[] = queuedChains[] + optimalChain;
37:    for all a_i ∈{agentsList - optimalChain - currentAgent} do
38:      inform(a_i, "'cycle finished"');
39:    end for
40  else
41:    for all a_i ∈{agentsList - currentAgent} do
42:      inform(a_i, "'no bartering chain"');
43:    end for
44:    return NIL
45:  end if
46:  if (includesFOrPDemands(optimalChain)) then
47:    if (existNextAgent(treeRootAgent, currentMDItem, cDList)) then
48:      treeRootAgent = nextAgent(treeRootAgent, currentMDItem, cDList);
49:    else
50:      if (existNextItem(currentMDItem, cDList)) then
51:        currentMDItem = nextItem(currentMDItem, cDList);
52:      end if
53:    end if
54:  end if
55:  treeRootAgent = ChainMaker(cDList, cOList, currentMDItem, dG, oS);
56:end while
57:for all a_i ∈agentsList - currentAgent do
58:  inform(a_i, "'service finished"');
59:end for
```

Algorithm 4 BarterCell's ChainMaker Operationing Algorithm.

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 was not 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.

6.2.3 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 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 devices 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.

6.2.4 BarterCell's Negotiation Evaluation

In setting up our simulation, we chose to compare our protocol with the Strategic Negotiation Model [18] 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 [71].

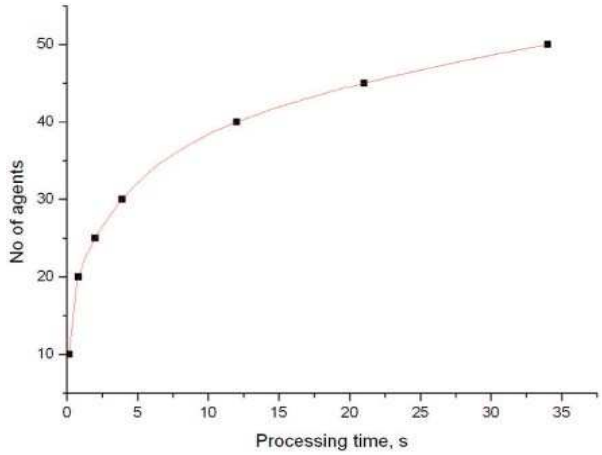


Figure 6.8: Simulating the number of Agents in BarterCell

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 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 6.8 shows how fast the main agent finishes searching for possible

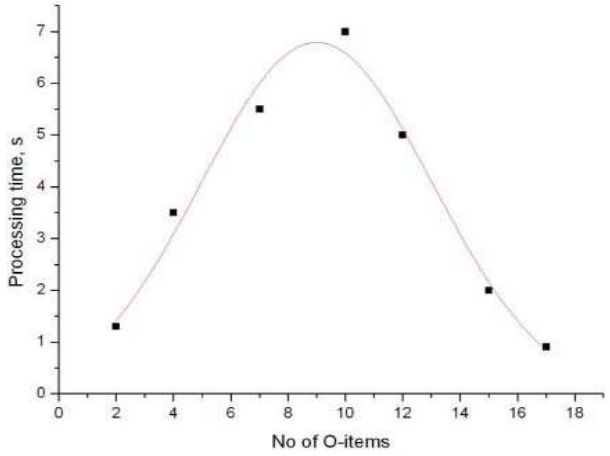


Figure 6.9: System Load Distribution

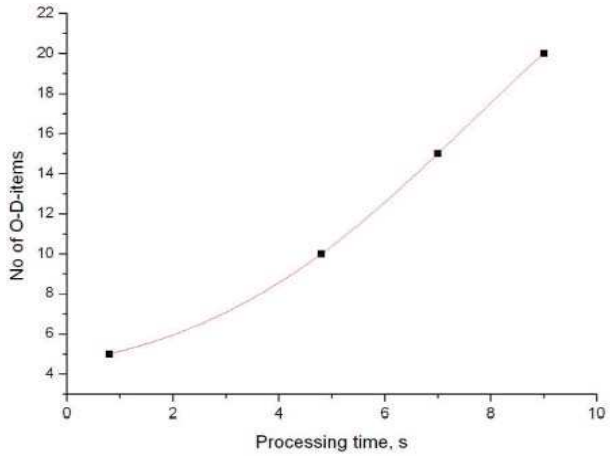


Figure 6.10: Simulating the number of items at each agent level

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 6.9 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 6.10 shows how fast main agent will finish searching for all possible optimal chains depending on number of items at each known agent. In Fig.

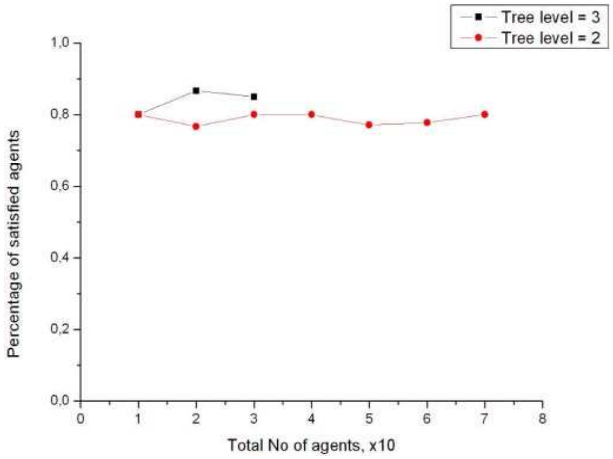


Figure 6.11: Agent Satisfaction Level

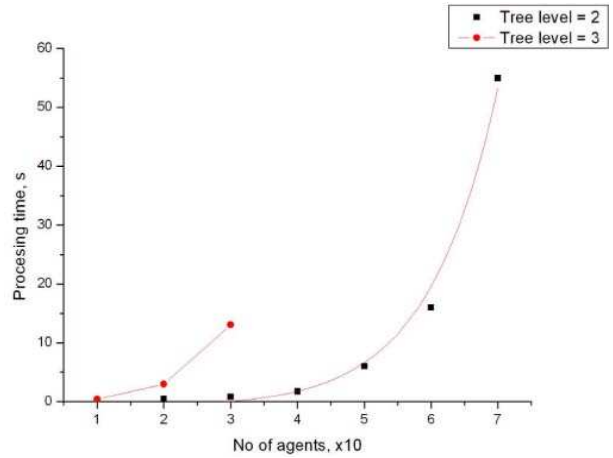


Figure 6.12: Processing time representation of agent satisfaction level

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 and the number of agents is 30.

Figure 6.11 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 6.12, we show the processing time representation of the simulated agent satisfaction level of figure 6.11.

Once more, since we have chosen the Strategic Negotiation protocol [18]

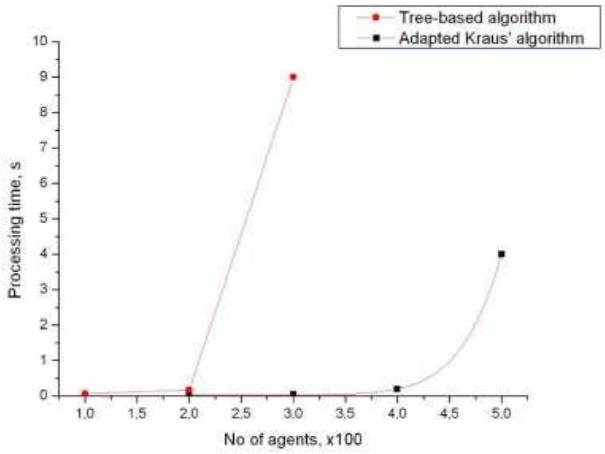


Figure 6.13: Abstract Comparison of the Different Negotiation Protocols

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 6.13 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.

The table here 6.1 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.

Table 6.1: Bartering Chain Length & Creation Time

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	$O(n^9)$	$O(n^5)$

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.

6.2.5 BarterCell in brief

In this part of the thesis we introduced BarterCell that is an software architecture for providing location-based bartering service for users of pocket computing devices. This application was developed to: 1) revive the idea of bartering within members of a specific community and promoting the benefits of location-based services, 2) to test an auctioning negotiation protocol for agents representing nomadic users and interacting very actively on-the-go, 3) motivate 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.

Then we used BarterCell as an agent-based software application to examine the performance of this auctioning protocol. Then we simulated the behavior of agents in strict situations where time and network resources are limited. We then adjusted a negotiation protocol that existed in the literature, which is the *Strategic Negotiation Model* [18] and applied it to BarterCell. At last, we compared the results obtained in both scenarios: Auctioning & Strategic Negotiation.

Chapter 7

Experimental Results

In this chapter we go through the experimental results we have obtained after applying our negotiation model on the two case-studies we have introduced in the previous chapter of this thesis. Earlier to that; we would like to highlight the fact that for each case-study three runs were performed wherein a different negotiation model was implemented in each run.

7.1 Bartering On-the-go

For BarterCell [116]; as explained earlier; we have already implemented this application twice; once using its initially proposed adhoc negotiation algorithms, (i.e., ServiceBuilder & ChainMaker), and again using the Strategic Negotiation Model of Sarit Kraus [18]. The third run of BarterCell was made using the newly proposed negotiation model of this thesis and then; all the three were compared.

7.1.1 BarterCell: First Run

During the first run of BarterCell there were 200 users of pocket computing devices that are covered by a common atmosphere, (i.e., Faculty of Science), and interested in exchanging one or two of their belongings with each other using BarterCell client application. Each user was given the right to create one or two agents in which each agent represents a user's single service request; a desired bartering deal. (all data of BarterCell's first run is summarized in 7.1).

We then fixed a set of 30 items, (e.g., watch, javaBook, cBook, iPod, laptop), which we assumed to be likely exchangeable - *Bartered* - among these 200 users. We then associated two values with each of these items;

Table 7.1: BarterCell's first run: Data Sheet.

Running Time	5 Hours (18000 seconds)
Number of Users	200
Number of Agents	286
Number of Access Points	15
Type of Access Points	Bluetooth (class 2)
AP's concurrent connections	1
Available Bartered Items	30
Negotiation Model Used	Adhoc (ServiceBuilder & ChainMaker)

a minimum score value, (i.e., an item cannot be bartered below it), and a desired score value, (i.e., an item gets instantly bartered at this value or any value greater than it).

We made BarterCell operational for 5 hours, which is equal to 18000 seconds. During this time all of the 200 users were pushed into interacting with the system by communicating their service requests with the central server throughout a set of 15 distributed access points; Bluetooth Terminals.

Here we are considering the Class 2 of the Bluetooth technology - *Version 2.1 + Enhanced Data Rate (EDR)* - that is the most common one, and it has an approximate range of 10 meters. Here, we would like to also highlight the fact that we prevented all access points from establishing more than one connection at a time. Out of 200 users, 86 have decided to communicate two agents instead of one. Therefore, the total number of BarterCell's interacting agents at that moment was 286 agents.

One of the very first issues we would like to highlight here is the fact that, different from Andiamo: "'*our second case study*"', BarterCell uses Jack [114] as the multi-agent platform wherein interacting software agents operate. The second issue that characterizes this run of BarterCell is the fact that during this first run we have applied the adhoc algorithms that we have initially used to developed this application [116], and we explained in the previous chapter, (i.e., the ServiceBuilder & the ChainMaker).

Recalling the fact that each of the 200 users was given the possibility to create one or two service requests - *Software Agent* - then, in this context we can observe from figure 7.1 the following:

1. Out of 286 agents, the number of agents that succeeded to get into a beneficial union is equal to 128 agents in 5 hours.

2. Starting from the 10855 seconds - *approximately 180 minutes after the service's kick-off announcement and 2 hours before it ends* - agents were

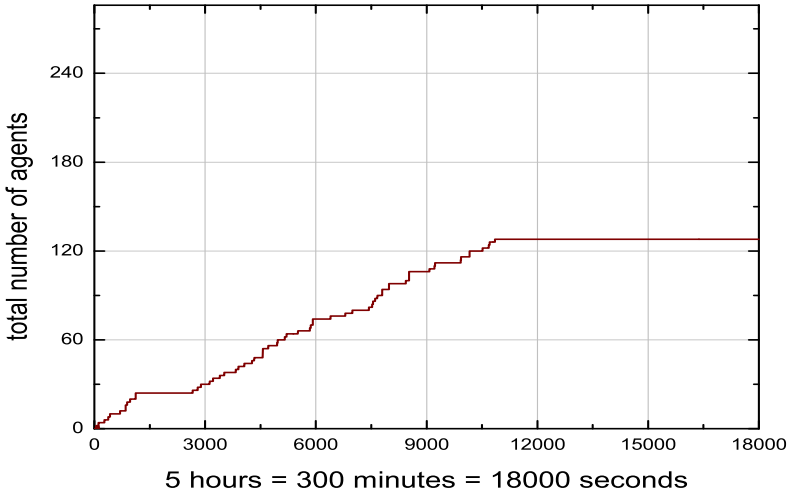


Figure 7.1: BarterCell's first run

unable to reach agreements and remained idle until the simulation time ends.

3. Since the number of users involved is equal to 200, and at least each user has made a service request, and the total number of successful agents is 128, therefore, almost 45% of users' requests were fulfilled.

7.1.2 BarterCell: Second Run

During the second run of BarterCell we removed the adhoc algorithms that were initially introduced with the application - *explained in earlier chapters* - and then adjusted the Strategic Negotiation Model of Sarit Kraus presented in [18] to fit into the specific needs of BarterCell's service acquisition scenario.

Specifically, in Kraus's model a total focus was given to the situation wherein bilateral negotiation between two self-interested agents is occurring. There, it is assumed that the "*attached agent*" is using a resource that the "*waiting agent*" is interested to use, and accordingly the later agent invokes a negotiation process with the earlier one. However, Kraus's approach does not fit into the context of BarterCell since its negotiation processes are assuming that both agents are equal - *none are holding resources* - and the negotiations are made to agree on establishing a mutually beneficial union.

Therefore, the first thing we had to adjust in Kraus's negotiation model before applying it to BarterCell was to program agents to have equal knowledge of each other's belongings, which is equal to Zero; while in Kraus's model it was not. The "'*waiting agent*"' primarily knew that the required resource has been under use by the "'*attached agent*"'.

In addition, since the bartering service addressed by BarterCell is actually exceeding the traditional one-to-one bartering situations wherein a user of pocket computing device is having one or two items to exchange with another user, then an extra adjustment to Kraus's model was required. As explained in 6.2, BarterCell permits the creation of long chains of mutually interested users and therefore; an agent - *in most of the cases* - will have to move from a negotiation session to another while maintaining a *vector* of previously encountered negotiations.

In table 7.2, we summarize the setup of the second run of BarterCell. As shown in this table we have made BarterCell operational for 5 hours, which is the same time of the previous run. We have imposed the same number of users, 200 users, and also assigned to these users the same number of created agents, 286 agents. We have also fixed the number of emulated Bluetooth access points and the technology class they represent.

Table 7.2: BarterCell's second run: Data Sheet.

Running Time	5 Hours (18000 seconds)
Number of Users	200
Number of Agents	286
Number of Access Points	15
Type of Access Points	Bluetooth (class 2)
AP's concurrent connections	2
Available Bartered Items	25
Negotiation Model Used	Strategic Negotiation [18] (tailored)

However, since Kraus's model intuition was not to target the service acquisition scenarios of nomadic users but yet it is a very influential model, therefore, we have given an extra programmatic functionality to these access points in order to permit them to establish more than one concurrent connection at a time. The customization of the Access Points concurrency were directly made to the library of BlueCove JSR-82 Emulator module [113], which is an open source implementation of the JSR-82 Bluetooth API for Java with an additional module to simulate Bluetooth stack.

In addition, since an agent will now have to maintain a *vector* of all encountered negotiations and use this maintained history to decide upon up-

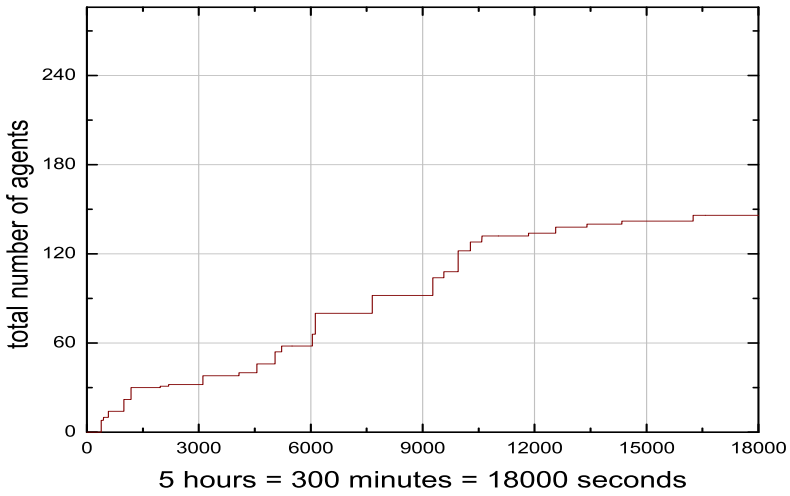


Figure 7.2: BarterCell's second run

coming unioning, then an extra unconsidered computation time is predicted. In order for us to address this extra computation time, in this run, we we have also decreased the number of Bartered items so that the probability of an item being repeated among the interacting agents is increased and, consequently, the possibility of an agent to find a union partner is made easier.

Figure 7.2 shows the number of successfully unioned agents with respect to the total number of interacting ones and simulation time. From this figure we can observe the following:

1. The number of agents that succeeded to get into a beneficial union within the 5 hours of simulation time is equal to 146 agents from a total number of interacting agents that is equal to 286 agents.

2. Interacting software agents in this run were able to get into beneficial unions and carry out negotiations until the 16253 seconds of the simulation time, which is approximately 29 minutes before the simulation ends.

3. In this run, the number of users involved was equal to 200 users and at least each user has made a service request, and the total number of successful agents is 146 out of 286 agents, therefore, almost 51% of the service requests were fulfilled..

7.1.3 BarterCell: Third Run

During the third run of BarterCell we avoided the use of neither the adhoc algorithms initially introduced with the application nor the tailored version of the Strategic Negotiation Model of Sarit Kraus presented in [18]. Instead, we have applied the negotiation model we presented in chapters 3 and 4.

We have deployed our model within BarterCell alike the generic implementation roadmap presented in chapter 5. However, since in BarterCell the outcome might involve more than two agents coming into a single union many parties, many issues, and our model addresses scenarios that tackle the situation wherein two agents are negotiating the possibility to establish a union and then instantly both terminate their activities tow parties, one issue. Then some modifications were also required here.

Table 7.3: BarterCell's third run: Data Sheet.

Running Time	5 Hours (18000 seconds)
Number of Users	200 users
Number of Agents	286 agents
Number of Access Points	15
Type of Access Points	Bluetooth (class 2)
AP's concurrent connections	1
Available Bartered Items	30 items
Negotiation Model Used	Our Nomadicity-driven Negotiation Model

Here, we also added an extra functionality for an agent to store the previously encountered negotiations in an extendable *vector* that combined will define this agent's final chain of bartering partners.

In addition, it is also worth recalling here the distinction our model imposes between the *Pend* and *Deposit* conditions of the protocol. In our model, if two agents agree on pending a union, the *pend* proposer will be granted the right to search for alternatives while the pended agent will be temporarily excluded from the available agents list until the proposer either find a better deal or convert a *pend* to *accept*.

From a simulation perspective, it was expected, and then proven throughout the following results, that the number of successful agents will not always be in an ascending fashion because of this *pend* option. Meaning that indications for the total number of successful agents are expected to go down at some points since it is possible for a pended agent to eventually get rejected by the pender. This will have to make some of the interacting agents counted as successful ones for a while, (i.e., unavailable for further negotiations), and

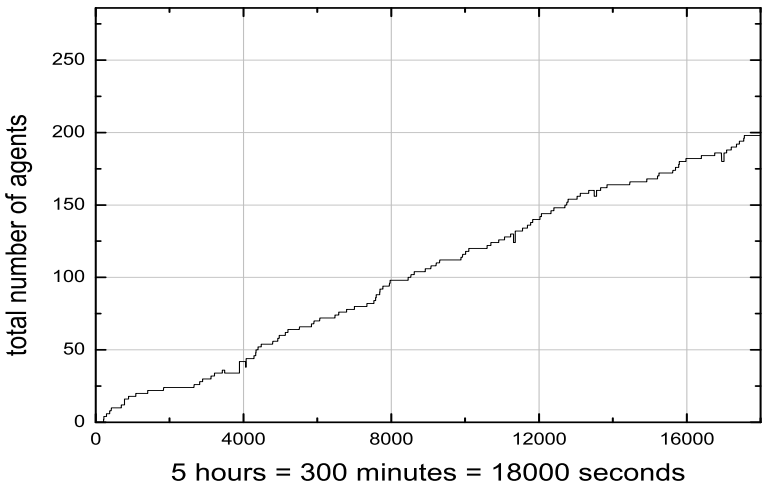


Figure 7.3: BarterCell's third run

then again available if the prospective union at that time was not effectively finalized.

In table 7.3 we summarize this run's facts and assumptions. We have made BarterCell operational for 5 hours as it was made in the previous two runs. We have virtually assumed the existence of 200 users where 86 of these users have decided to communicate two services requests - *delegate two agents to handle different bartering deals*. We again used the java-based BlueCove JSR-82 Emulator module [113] to impose and distribute 15 access points.

We link access points to users in a way that reflects the nomadic nature of this users. We do that by applying a simple communication method that assigns a different access point to a user every time a communication is required to be done. While a single concurrent connection is permitted per access point it is then expected that some difficulties may occur when a user is sending over his agent or receiving back any feedbacks. In addition, in this run we have put back the number of Bartered Items to 30 items. We removed the adhoc algorithms experimented in the first run and Kraus's model of the second run, then we applied our which we introduced in chapters 3 and 4.

In figure 7.3 we show the number of agents that succeeded to reach mutually beneficial agreements - *unions* - while our negotiation model was applied. From this figure we can observe the following:

1. During the 5 hours of simulation 198 agents were able to get into successful unions, which make the total number of agents that didnt suc-

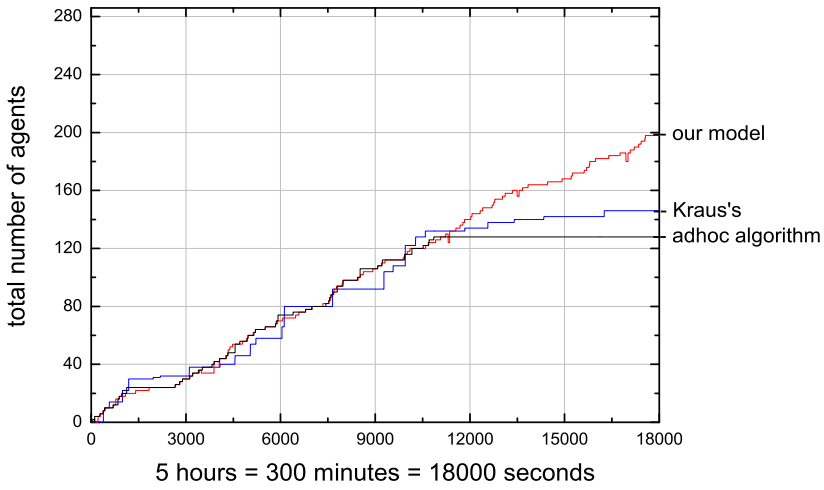


Figure 7.4: BarterCell's three negotiation models evaluation

ceed to union equal to 88 agents.

2. In this run, agents were able to negotiate and get into successful unions until the 17560 seconds, which is approximately the end of the simulation time, (i.e., 18000).

3. The total number of users involved in this run was is equal to 200 users and at least each user has made one service request, (i.e., delegated an agent). The total number of successfully unioning agents is 198 out of 286 agents, therefore, almost a 70% of the users involved have got at least one of their service requests fulfilled.

7.1.4 Our model in Bartering context

In this subsection we put together the results of BarterCell's three different runs and outline the advantages and disadvantages of using our negotiation model in similar contexts. We do that by looking at the collected statistics from three different perspectives, which are: 1) the ability for a negotiation model to achieve early results, 2) the behavior of the applied negotiation model along the simulation time, and 3) the total number of successful agreements reached at the application's process termination.

Early Results

Regardless of the overall outcome of the three negotiation models, we can observe from figure 7.4 that the negotiation model presented in [18] was able to produce the biggest number of mutual agreements earlier than the others. If we take the simulation time from 0 to 1500 seconds we will see that the highest number of unioned agents was achieved through the application of Kraus's model, (i.e., 30 agents). At the same time, the adhoc algorithm of BarterCell was able to drive only 24 agents into successful unions while our model that we presented in chapters 3 and 4 was able to reach to 22 agents.

In some scenarios it is highly desirable by the provider of a certain service to reach the highest number of possible unions as early as possible rather than achieving better results on the long run. For example, when providing a service for users of pocket computing devices in an airport or a train station, it is then already known that these users are likely to have a tighter time frame within this location than others in a university or a factory. Consequently, it might be of better performance to have a short term plans and pick the negotiation model that achieve the highest number of early results rather than looking at the end results. Therefore, here we can possibly say that the advantage of using Kraus's model in similar context is to reach early results.

Behavior

Observing the behavior of a specific negotiation model along the simulation time makes it easier for the service provider to draw a general image of the model prior to its deployment. Here, in this subsection, we attempt to capture a general idea about the behavior of all three models. Following the depictions of figure 7.4 we can observer the following:

- The adhoc algorithm used during the first run of BarterCell have showed a several phases of idleness within the given 5 hours of simulation, (e.g., from 1123 seconds to the 2660 seconds), which then ended up with a complete stoppage at early stages, (i.e., from the 10855 seconds till the end: that led to 2 hours of no actions).

- Kraus's negotiation model had another type of behavior to be observed, which is related to some frequent large jumps accompanied with state of idleness that was occurring all along the simulation time. For example, at the 6045 seconds the total number of unioned agents was equal to 58 agents, while at the moment right after, 6046 seconds, the number of total unioned agents was equal to 66 agents. Also the jump happened at the 6120 seconds from 66 to 80 agents, which then lasted at this number till the 7664 seconds - *a slight idleness*.

- Apart from the sudden drops that figure 7.4 depicts, which we explained the reason behind them earlier in this chapter, our model did not show neither large jumps nor long moments of idleness all through the simulation time.

Overall Outcome

The overall outcome that we can get after putting together the numbers of successfully unioned agents reached at every different run of BarterCell is summarized in these points:

1. The negotiation model we presented in chapters 3 and 4 is the best fitting approach for such a service as long as a stable and a lasting activeness are required.

2. The negotiation model proposed by Sarit Kraus in [18] is the appropriate in this context if early results and long periods where resources need to be on hold are required.

3. The adhoc algorithms applied within BarterCell's first run might be competitive in achieving early results, however, the early complete idleness encountered 2 hours to the end of the simulation may lead this approach to a second round of improvements / modifications before actual deployment.

7.2 Carpooling On-the-go

For Andiamo [101], we have collected the statistics of its existing Auctioning negotiation approach that we have showed earlier in this thesis, then we have made a re-run to Andiamo after implementing the Service-oriented Negotiation Model of Sierra., et al., [27]. The third run of Andiamo has included the implementation of the newly proposed negotiation model of this thesis and then; all the three were compared.

Different from BarterCell, in the three runs of Andiamo we have fixed the simulation parameters so that the same circumstances accompanying the implementation of each negotiation model are alike.

As table 7.4 illustrates, we made Andiamo run for 3 hours (10800 seconds) each time a different negotiation model was applied. During this time 300 users was instructed to communicate their software agents, (i.e., service requests), to a central multi-agent platform, which is JADE [9].

This user-to-server communication was done as if a mediator point is placed in-between, which are the 15 access points. These access points are

implemented through the BlueCove JSR-82 Emulator module [113] that is an open source implementation of the JSR-82 Bluetooth API for Java with an additional module to simulate Bluetooth stack. Then a customization was made to this later module so that a single agent is transferred at a time: *AP's concurrency = 1.*

Since Andiamo is a carpooling service application, then people who are willing to offer a car-ride are expected to participate as well as people that are actually searching for a ride. Therefore, from the total number of 300 users, 100 users were assumed to be car ride givers and accordingly their software agents are the Ride-giver agents mentioned in table 7.4. Out of Andiamo's actual deployment and experience here in Trento (IT) the number of those looking for car rides were always greater than those willing to give one, therefore, we have assumed that the remaining 200 users are actually car ride seekers and, accordingly, their created agents are Ride-seeker agents.

Table 7.4: Andiamo's Simulation Parameters

Running Time	3 Hours (10800 seconds)
Number of Users	300 users
Number of Ride-Giver Agents	205 agents
Number of Ride-Seeker Agents	200 agents
Number of Free Seats per Giver	From 1 to 3 places
Number of Access Points	15 (emulated)
Type of Access Points	Bluetooth (class 2)
AP's concurrent connections	1 connection
Available Destinations	28 subsequent points (English Alphabet)
Destinations' Interval	20 minutes
Rides Timeframe	From 10:00 To 16:00

Since a ride-giver may have more than one available seat to share with ride-seekers then a possibility for each ride-giver to create from one to three agents / service requests was considered in the three runs of Andiamo. Out of the 100 users that are assumed to be responsible of communicating ride-giver agents we have made 30 of them propagate the availability of one seat to share, 35 ride-giver agents to will make available two seats, and the remaining 35 will search to fulfill three seats. Therefore, the total number of ride-giver agents / negotiated seats will be equal to $\{30 + (35 * 2) + (35 * 3)\} = 205$ agents. The destination points, arrival time, and departure time are the same for all of the created agents as long as they are of the same user.

We assumed that there are 28 destination available to all of the 300 users.

These destinations are given names as of the English Alphabet, (i.e., A, B, C, D, ..., Z). Randomly, each user whether he is a ride giver or a seeker was assigned a departure point that is one of these alphabet letters excluding the **Z** letter. In addition, each user was also randomly assigned an arrival point but this point must be located anywhere after the departure letter and not before. For example, if a giver.agent was randomly assigned the departure point **D** then any arrival point can be assigned to the same agent from **E** to **Z** but neither **A** nor **C** or **B** can be a possible option.

The last issue that we address by the fixed simulation parameters mentioned in table 7.4 is the departure and arrival times. Each created agent is given a time that is chosen randomly from the permitted timeframe, which is anytime in-between 10:00 and 16:00. If it is a ride-giver agent then this time will be its ride departure time and, if it is a ride-seeker agent then this time will be its desired departure time.

Then, the arrival time or the desired arrival time for both the ride seeker and the ride giver agents are calculated according to the number of letters located between the departure and the arrival points, meaning that 20 minutes will keep on adding up for each considered destination / letter between both the departure and the arrival points.

7.2.1 Andiamo: First Run

During Andiamo's first run the adhoc Auctioning negotiation algorithms that were presented in 6.1, and initially used while Andiamo was first developed, are made active for the purpose of evaluating the overall application performance. Since the Auctioning approach of agents negotiation was introduced to mainly address the problem where demands are higher than the supply: meaning that number of available seats is less than those willing to share them.

Therefore, only for those agents representing users of one available seat, (i.e., 30 agents), we have imposed an extra function which ensures that at least each of these 30 seats are desired by a minimum of two ride-seeker agents, (i.e., departure point and destination are matched).

Figure 7.5 shows the obtained results from Andiamo while the adhoc Auctioning algorithms were applied. From this figure we can observer the following:

1. Out of 205 available seats, the total number of seats that actually got occupied by ride seekers is 117 seat.

2. Starting from the 4100 seconds there was no any further unions achieved

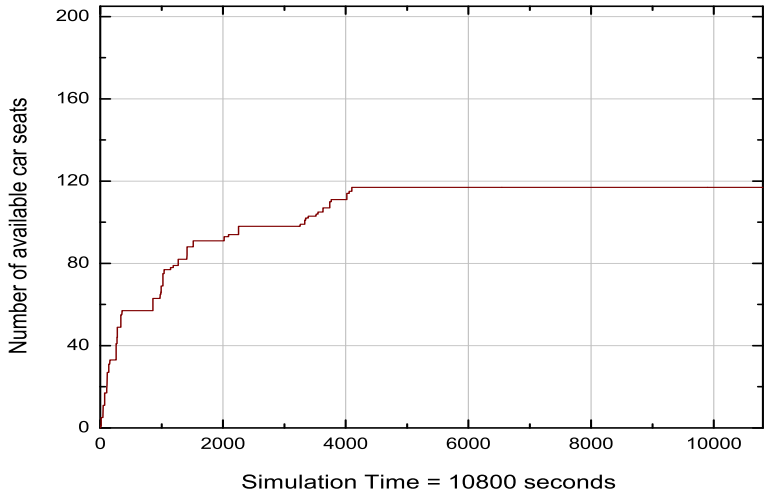


Figure 7.5: Andiamo's first run

between ride seekers and givers. This means that the reached 117 deals were all achieved at the first hour and 8 minutes of the simulation.

3. Since reaching an agreement about each available seat requires the unioning of two users - *the ride giver and the ride seeker*. Therefore, the 117 seats fulfill were the result of the unioning of 234 agents.

4. Since the total number of interacting agents was 405 agents, which is the ride-giver + the ride-seeker agents. Besides, since 234 agents were reached into mutually beneficial unions then more than 50% of the total number of interacting agents was led out of successful negotiation.

7.2.2 Andiamo: Second Run

During Andiamo's second run the Service-oriented Negotiation Model of Sierra., et al., [27] was made active instead of the adhoc Auctioning negotiation algorithms initially used by Andiamo.

In figure 7.6 we demonstrate the results obtained after making Andiamo operational for 3 hours while the earlier mentioned negotiation model was applied. From this figure we came to these conclusions:

1. The total number of unions reached in these 3 hours was equal to 152, which refers to the fact that 76 seats were shared between ride givers and ride seekers.

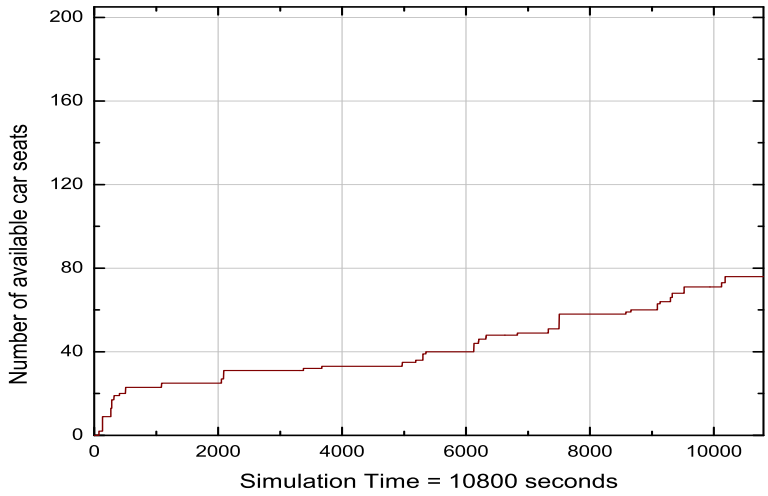


Figure 7.6: Andiamo's second run

2. At the 10183 seconds there no more unions achieved between ride seekers and givers. This means that the reached 76 deals were all achieved almost 10 minutes before the simulation ends.

3. Even though the number of successful negotiations kept on escalating along the 3 hours of simulation, yet repetitive cases of idleness were observed at several times, (e.g., from the 2089 to the 3374 seconds: 31 seats). Therefore, it should be wise here to assume that the state of idleness wherein the simulation ended was another temporary situation and more seats were about to be fulfilled if the simulation would have lasted longer.

4. Since the total number of interacting agents in this run was 405 agents summing up ride givers and seekers and, those went into agreements were 152, therefore; less than 50% of the overall number of agents have made it through by means of Sierra's model [27].

7.2.3 Andiamo: Third Run

During Andiamo's third run we have made active the negotiation model we presented in chapters 3 and 4. Figure 7.7 shows the obtained results from Andiamo while our negotiation model was applied and below are the observation we have on these results:

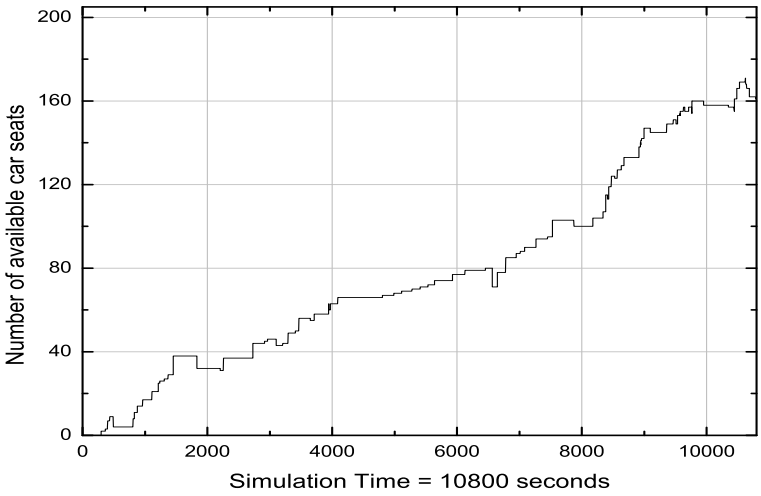


Figure 7.7: Andiamo's third run

1. Out of 205 available seats, the total number of seats that actually got occupied by ride seekers is 161 seat. That refers to the fact that 161 unions were reached wherein each union involve a ride-giver agent and a ride-seeker agent. In this context it is worth highlighting that each user may have more than a free seat to share and consequently more than one ride-giver agent to create and, that makes the same user also eligible for more than one union involvement.

2. In general, the number of successfully shared rides kept on increasing all through the 3 hours of simulation. However, few short periods of idleness can be put into consideration. For example, the period of time from the 1454 seconds to the 1831 seconds wherein the number of achieved unions or shared seats was always equal to 38 seats, and also the period from 4089 seconds to 4803 seconds wherein 66 seats were the so far achieved.

3. The 161 seats fulfilled by means of our negotiation model refers to the fact that 322 agents were reached to mutually beneficial agreements. Out of the 405 agents that were interacting during this run's 3 hours the unioned 322 agents means that more than 75% of the overall number of agents were unioned.

4. As explained through BarterCell's results earlier this chapter, while

applying our model it is recognizable that some drops in the number of successful unions is usually encountered as for the period from 3103 to the subsequent second where the total number of shared seats, (i.e., successful unions), was dropped from 46 to 43 seats. These drops occur because of the *Pend* option available for the negotiating agents throughout our models negotiation protocol. In our model, if two agents agree on pending a union, the *pend* proposer will be granted the right to search for alternatives while the pended agent will be temporarily excluded from the available agents list, (i.e., programmatically counted as unioned), until the proposer either find a better deal or convert a *pend* to *accept*.

7.2.4 Our model in Carpooling context

In this subsection we conclude this chapter by putting together the results of Andiamo's three different runs and outline the advantages and disadvantages of using our negotiation model in similar contexts. Similar to what we have done in BarterCell, we do so by considering three different evaluation perspectives, which are: 1) the ability for a negotiation model to achieve early results, 2) the behavior of the applied negotiation model along the simulation time, and 3) the total number of successful agreements reached at the application's process termination.

Early Results

In BarterCell the results obtained of the three runs - *where each had a different negotiation model applied* - were very close when early results are considered, and our negotiation model was not the best solution to achieve that. However, in Andiamo the situation is quite different and the results obtained with each model in focus are dramatically varying as figure 7.8 depicts.

Taking the period of time from 1 to 2000 seconds as a benchmark for evaluating the acquisition of early result throughout the application of each negotiation model we come with the following conclusions:

1. At the 2000 second, Andiamo while employing its adhoc auctioning algorithms has succeeded to put together 182 agents of ride givers and seekers into 91 mutually beneficial deals or as we call it *unions*.

2. Within the same context and exact timeframe, the service-oriented negotiation model of Sierra., et al., presented in [27] has made Andiamo succeed to union 50 agents and reach a total number of 25 seats shared.

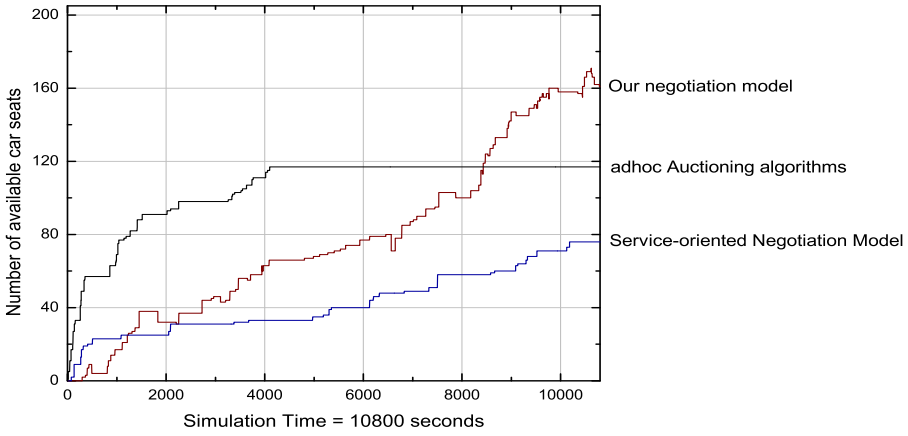


Figure 7.8: Andiamo's three negotiation models evaluation

3. Also at the 2000 second, while the Nomadicity-driven negotiation model we presented in chapter 3 and chapter 4 was employed by Andiamo, 32 unions that involved 62 agents were made possible.

4. Our negotiation model does not do well while obtaining highly effective early results is a must. Since in BarterCell the same case occured, then here we may come to a more general conclusion that is: *While early results is a major requirement of a nomadic service acquisition platform applying our model at its current state is not the best fit.*

Behavior

In this subsection we give general observations on the behavior of each negotiation model along the 3 hours of simulation accompanied with reasonable explanation for hardly understandable behaviors, if existed. Therefore, following the depictions of figure 7.8 we can come up with these conclusions:

1. Even though it was the first to achieve the highest number of early unions, yet the adhoc algorithms used in the first run of Andiamo showed a very early phase of idleness - *2 hours*.

2. The service-oriented negotiation model of Sierra., et al., had another type of behavior to observe. Several but yet short periods of idleness were observed all along the simulation time. We believe that these subsequent phases of short idleness are occurring because:

(a) The whole negotiation model was proposed for client/server software applications wherein *Scoring Functions*, like those presented, are calculated and accordingly a decision is directly communicated to the opponent agent without a bottleneck, (i.e., Access Point).

(b) Only *accept* & *reject* were defined as the only condition of the negotiation protocol, and since we have prohibited an agent for re-negotiating a union formation with previously rejected agent, therefore, a time by the *head-agent* to re-attach new agents into new session was always required and it is yet not reflecting at any part of the simulation results.

(c) The set of *Imitative Tactics* proposed by authors, (i.e., Relative Tit-For-Tat, Random Absolute Tit-For-Tat), together with the narrow flexibility of the negotiation protocol, (i.e., only *accept* & *reject*), have made - *when these tactics are used* - the total number of rejected agents increases faster than usual. Therefore, a longer process for searching for unrejected ones was required, and at that time the simulation remain idle until the *head-agent* succeeds to assigns new negotiations sessions to newly negotiating agents.

3. As a general comment on the behavior of our negotiation model we see it increasingly achieving a higher number of unions very stably among all time segments. However, the sudden drops caused by the use of the *Pend* option of our negotiation model yet remains a significant remark on our model that worth highlighting here.

Overall Outcome

In overall, The negotiation model we presented in chapters 3 and 4 is an appropriate approach for providing services for users of pocket computing devices if no high number of early positive results is required and, also if a long lasting responsiveness with less periods of idleness is a major concern.

The service-oriented negotiation model of Sierra., et al., presented in [27] fits well when averagely short periods of times wherein involved resources are required to be on hold or refreshed without the overall process being disturbed is required. However, the overall number of achieved unions while this model is applied in a nomadic context where bottlenecks are existing, such as Bluetooth Access Points, is not entirely guaranteed to be high.

By far, the adhoc Auctioning algorithms applied within Andiamo's first run might be competitive in achieving early results. However, the early complete idleness encountered 1 hour and 8 minutes after the start while having a large number of agents not unioned yet is yet a drawback.

We believe this later highlighted idleness was caused because, as presented in [101], the auctioning process of a seat might go on until the cost exceeds the logical price predefined by us, and consequently the ride-seeker decides to reject any potential union with this ride-giver. Then, it causes the head-agent to have many available agents that assigning a negotiation session for them is impossible because they all rejected each other earlier.

7.3 Model Limitations

1. Putting in mind that our negotiation model was motivated by the idea of addressing the emerging vision of Nomadicity and the new requirements it imposes; still, our model did not address all of the negotiation issues making a service application capable of supporting users on-the-go. For instance, our tactics lack the understanding of nomads' privacy, nomadic service data accuracy, and security.

2. Considering the fact that nomadic services can be deployed within those rapidly changing circumstances, such as in a train station, or within a considerably less active location-based environment, such as in a university campus. Since our negotiation model is unable to achieve a large number of early agreements. Therefore, our model efficiency is better reached through the location-based approach of nomadic services because it is linked to the availability of nomads within the coverage of the concerned service application for longer periods of time.

3. We tested our negotiation model by emulating the behavior of Bluetooth Access Points, which is a disappearing approach to distributed terminals, (i.e., replaced with Wi-Fi, and in the future WiMAX). Therefore, the usability of our model is directly connected to the evaluation of its behavior from an alternative connecting technology perspective.

4. As the experimental results showed, our model escalates all along the execution period, therefore, all of the involved resources, such as bandwidth, terminal, computing servers, and agents platform, must be dedicated. Since the negotiation processes will never get on hold and, as long as frequent related end-user feedbacks is a characteristic of the model, then, our model is always fully dependent on the linked resources, and this will limit the possibility for the concerned service application architecture to run multiple types of services alongside.

Chapter 8

Conclusions & Future Work

The main contribution of this thesis is related to the introduction of a new negotiation model, (i.e., protocol, set of tactics, strategy), for software agents to employ while attempting to acquire a service for users on-the-go. The purpose of our model is to maximize the benefits of the interacting agents while considering the limitations of the communication technologies involved and, the nomadic nature of the users they represent. We showed how our model can be generically implemented within a service-oriented multi-agent platform. Then, we introduced two case-studies that we have been working on with our industrial partner and, we demonstrate these cases' experimental results before and after applying our negotiation model.

We started this thesis by explaining the related state-of-the-art. Since three different literatures are intersecting in order to construct negotiation models similar to the one we proposed, we gave a literature review about the, 1) advanced Agents Negotiation in Common Settings, 2) Agents' Negotiation for Service Acquisition, and 3) Agents' Negotiation in Wireless Networks. In each literature, we gave a comparison between the research effort of three different groups of scholars that are most related to our research.

Then, we formally introduced our negotiation model wherein we started by formalizing the general setting we expect our model to be applied, then, a clear description of the negotiation issue that any two interacting agents are expecting to resolve. We then formally introduced the negotiation protocol, set of tactics and strategies that we believe to have a great impact on the act of agents' negotiation in a modern service-acquisition environment.

Then, we explained how our negotiation model can be implemented within any agent-based service acquisition software platform. However, within this chapter we did not restrict the implementation of our model to any particular application; instead, we attempted to prove its broad applicability. Then, we presented two case-studies we were working on with our industrial partner to

provide a ride-sharing and bartering services to users of Pocket Computing Devices. At the end; we explicitly went through the experimental results we have obtained and we also emphasize the advantages of employing our model on others.

8.1 Model's Conclusion Statement

In the context of providing services for users of pocket computing devices on-the-go, particularly Carpooling and Bartering services, such as Andiamo [101] and BarterCell [116], our model will perform the best if:

1. No extraordinary number of successfully matching agents is required in early stages of the application's operational time.

2. Resources exploited by the concerned service application while running are dedicated and it will not get on hold or shared at any point of the operating time.

3. So far, when Bluetooth is the only embedded technology in the access points linking users on-the-go with the central server responsible of provisioning the service.

4. An outstanding number of mutually beneficial unions is possibly reached after considerable periods of times while this number is consistently escalating along the total operational time of the service application.

8.2 Future Work

First: We intend to further expand our range of supported Wireless Access Points - *The Bottlenecks* - to cover the technologies of current days. As we mentioned earlier in chapter 7, wherein our Experimental Results are demonstrated, we have been always emulating the Bluetooth terminals assuming that these are the only possible bridges between users and the central multi-agent platform where all the negotiations take place.

However, other technologies such as Wi-Fi and UMTS can also be emulated separately or combined and, we believe that interesting results can then be obtained. In addition, since nowadays Bluetooth access points are hardly found in any environment, and instead Wi-Fi terminals are placed, then integrating Wi-Fi emulators to our simulations will make the foreseen results more realistic.

Second: As a subsequent step, we also intend to integrate our nomadicity-driven negotiation model with the commercial version of Andiamo and evaluate its performance while real data is available. Beside, since the University of Trento, and in specific the Department of Information Engineering and Computer Science, is working very closely with Andiamo's buyer, which is The Autonomous Province of Trento, then we expect our negotiation model to be integrated to Andiamo's architecture without any obstacles.

Third: a real important contribution to the literature of software agents negotiation for service acquisition would be a simulation platform / tool that reflects precisely the nomadic nature and obstacles of nowadays users of pocket computing devices. This point of our future work is an actual work-in-progress these days. A well-designed graphical user interface that allows a user to describe the simulated nomadic environment will be included in this simulator in a Wizard fashion.

Within this simulation tool we also intend to embed a set of recognized negotiation models and the negotiation model we proposed in this thesis so that different behaviors under several negotiation approaches can be observed. Last but not least, since this simulator will be developed in Java, then we will be integrating the JFreeChart [117], which is a Java chart library that makes it easy for developers to display professional quality charts in their applications based on the comma-separated values (CSV) files produced by the simulator.

Fourth:, and currently our last future work intentions, is related directly to the model's proposed tactics and strategies. As we saw in subsections 4.1.1 and 4.1.2 where Time-based Tactics and Connectivity Related Tactics are explained, these tactics are not covering all of the issues facing users of pocket computing devices while acquiring services on-the-go. For example, Profit-based Tactics for nomadic services that are supposed to generate revenues for their users. In these tactics no considerations will be given to Time or Resources but only Money. Therefore, we intend to look into the set of tactics we have proposed so far with the intention to extend it.

Bibliography

[1] Dan Steinbock. *The Mobile Revolution: The Making of Mobile Services Worldwide.* Kogan Page, Philadelphia, USA, January 2007.

[2] Harry Bouwman, Henny De Vos, and Timber Haaker, editors. *Mobile Service Innovation and Business Models.* Springer, NL, July 2008.

[3] Jarkko Vesa. *Mobile Services In The Networked Economy.* IRM Press, April 2005.

[4] Steve Munroe, Tim Miller, Roxana A. Belecheanu, Michal Pěchouček, Peter McBurney, and Michael Luck. Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *The Knowledge Engineering Review*, 21(4):345–392, 2006.

[5] Jez Mckean, Hayden Shorter, Michael Luck, Peter Mcburney, and Steven Willmott. Technology diffusion: analysing the diffusion of agent technologies. *Autonomous Agents and Multi-Agent Systems*, 17(3):372–396, 2008.

[6] Nicholas R. Jennings and Barry Crabtree. The practical application of intelligent agents and multi-agent technology. *Applied Artificial Intelligence*, 11(5):3–4, 1997.

[7] Maurizio Bombara, Davide Calì, and Corrado Santoro. KORE: A multi-agent system to assist museum visitors. In *WOA*, pages 175–178, Villasimius, CA, Italy, September 2003.

[8] Oana Bucur, Olivier Boissier, and Philippe Beaune. A context-based architecture for learning how to make contextualized decisions. In *Proceedings of the First International Workshop on Managing Context Information in Mobile and Pervasive Environments*, Ayia Napa, Cyprus, May 2005.

[9] Fabio Bellifemine and Giovanni Rimassa. Developing multi-agent systems with a fipa-compliant agent framework. *Software - Practice & Experience*, 31(2):103–128, 2001.

[10] Manolis Koubarakis. Multi-agent systems and peer-to-peer computing: Methods, systems, and challenges. In *The 7th International Workshop on Cooperative Information Agents, CIA 2003*, pages 46–61. Springer, August 2003.

[11] Marco Ughetti, Tiziana Trucco, and Danilo Gotta. Development of agent-based, peer-to-peer mobile applications on android with jade. In *UBICOMM'08: Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 287–294, Washington, DC, USA, 2008. IEEE Computer Society.

[12] Open Handset Alliance. Android, 2007. http://www.openhandsetalliance.com/android_overview.html.

[13] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, USA, August 1988.

[14] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[15] Edmund H. Durfee. Distributed problem solving and planning. In *EASSS'01: Selected Tutorial Papers from the 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School on Multi-Agent Systems and Applications*, pages 118–149, London, UK, 2001. Springer-Verlag.

[16] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, USA, March 1999.

[17] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, New York, NY, USA, June 2002.

[18] Sarit Kraus. *Strategic negotiation in multiagent environments*. MIT Press, Cambridge, MA, USA, September 2001.

[19] Howard Raiffa. *The Art and Science of Negotiation*. Belknap Press of Harvard University Press, Cambridge, MA, USA, March 1985.

[20] Marie Pierre Gleizes, Alain Léger, Eleutherios Athanassiou, and Pierre Glize. Abrose: Self-organization and learning in multi-agent based brokerage services. In *IS&N 1999: Intelligence in Services and Networks Paving the Way for an Open Service Market*, pages 41–54, Barcelona, Spain, April 1999. Springer.

[21] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence.*, 77(2):321–357, 1995.

[22] Jim E. Doran, S. Franklin, Nicholas R. Jennings, and Timothy J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[23] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 912–917, San Mateo, CA, 1989. ACM.

[24] Reid 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, 1981.

[25] Leonard Kleinrock. Nomadicity: Anytime, anywhere in a disconnected world. *Journal of Mobile Networks and Applications*, 1:351–357, 1996.

[26] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, July 1994.

[27] Carles Sierra, Peyman Faratin, and Nicholas R. Jennings. A service-oriented negotiation model between autonomous agents. In *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, pages 201–219, London, UK, 1999. Springer-Verlag.

[28] Michael Wooldridge. Agent-based software engineering. *Software Engineering*, 144(1):26–37, February 1997.

[29] Brian Henderson-Sellers and Paolo Giorgini. *Agent-oriented methodologies*. Idea Group Pub., Hershey, PA, USA, June 2005.

[30] Michael Wooldridge. Agent-based computing. *Interoperable Communication Networks*, 1(1):71–97, January 1998.

[31] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[32] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[33] Emile Aarts, Rick Harwig, and Martin Schuurmans. Ambient intelligence. pages 235–250, 2002.

[34] Chen Rui, Hou Yi-bin, Huang Zhang-qin, and He Jian. Modeling the ambient intelligence application system: concept, software, data, and network. *IEEE Transactions on Systems, Man, and Cybernetics.*, 39(3):299–314, 2009.

[35] Michael Wooldridge and Nick Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[36] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[37] Tarek K. Abdel-Hamid. The slippery path to productivity improvement. *IEEE Software*, 13:43–52, 1996.

[38] Victor R. Basili and Robert W. Reiter. A controlled experiment quantitatively comparing software development approaches. *IEEE Transactions on Software Engineering*, 7(3):299–320, 1981.

[39] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[40] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[41] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3:285–312, September 2000.

[42] Andrea Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *AOSE 2000: The First International Workshop on Agent-Oriented Software Engineering*, pages 185–193. Springer, June 2000.

[43] Xiyun Wang and Yves Lesprance. Agent-oriented requirements engineering using congolog and i*. In *In Working Notes of the Agent-Oriented Information Systems (AOIS-2001) Workshop*, pages 59–78, 2001.

[44] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):1–42, 2009.

[45] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM.

[46] John Mylopoulos, Lawrence Chung, and Eric Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

[47] John Mylopoulos. Information modeling in the time of the revolution. *Information Systems*, 23(3-4):127–155, 1998.

[48] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

[49] Michael Wooldridge Rafael H. Bordini, Jomi Fred Hbner. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, October 2007.

[50] Dickson K. W. Chiu, S. C. Cheung, and Ho fung Leung. A multi-agent infrastructure for mobile workforce management in a service oriented enterprise. In *HICSS'05: Proceedings of the 38th Annual International Conference on System Sciences*, pages 85–95, Big Island, Hawaii, USA, January 2005. IEEE Computer Society.

[51] J. Bravo, X. Alamán, and T. Riesgo. Ubiquitous computing and ambient intelligence: New challenges for computing. *Journal of Universal Computer Science*, 12(3):233–235, 2006.

[52] Philips Research. Ambient Intelligence Research in ExperienceLab. http://www.research.philips.com/technologies/syst_softw/ami/.

[53] MIT. Ambient Intelligence Research Group. http://ambient.media.mit.edu/.

[54] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[55] Paolo Remagnino and Gian Luca Foresti. Ambient intelligence: A new multidisciplinary paradigm. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(1):1–6, January 2005.

[56] Norbert Streitz Paddy Nixon. The disappearing computer. *Communications of the ACM*, 48(3):32–35, March 2005.

[57] Albrecht Schmidt. Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2):191–199, June 2000.

[58] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC'99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK, 1999. Springer-Verlag.

[59] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC'99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[60] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[61] John Krogstie, Kalle Lyytinen, Andreas Lothe Opdahl, Barbara Pernici, Keng Siau, and Kari Smolander. Research areas and challenges for mobile information systems. *International Journal of Mobile Communications*, 2(3):220–234, 2004.

[62] Kalle Lyytinen and Youngjin Yoo. Research commentary: The next wave of nomadic computing. *Information Systems Research*, 13(4):377–388, 2002.

[63] Giovanni Acampora and Vincenzo Loia. Dynamic services for open ambient intelligence systems. In Janusz Kacprzyk, editor, *Studies in Fuzziness and Soft Computing*, pages 105–122. Springer Berlin Heidelberg, 2006.

[64] Richard R. Brooks. Distributed sensor networks: A multiagent perspective. *International Journal of Distributed Sensor Networks*, 4(3):285–301, July 2008.

[65] Gregory M.P. OHare, M.J. OGrady, R. Collier, S. Keegan, D. OKane, R. Tynan, and D. Marsh. Ambient intelligence through agile agents citation. In Yang Cai, editor, *Ambient Intelligence for Scientific Discovery: Lecture Notes in Computer Science*, pages 286–310. Springer Berlin Heidelberg, 2005.

[66] Satoshi Kurihara, Kensuke Fukuda, Toshio Hirotsu, Shigemi Aoyagi, Toshihiro Takada, and Toshiharu Sugawara. Multi-agent human-environment interaction framework for the ubiquitous environment. In *MMAS 2004: International workshop on Massively multi-agent systems*. Springer, December 2004.

[67] Eric Yu. Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, 43(3):123–132, April 2001.

[68] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. A knowledge level software engineering methodology for agent oriented programming. In *AGENTS'01: Proceedings of the fifth international conference on Autonomous agents*, pages 648–655, New York, NY, USA, 2001. ACM.

[69] Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, Jim Conallen, and Kelli A. Houston. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[70] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[71] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.

[72] Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24:159–182, 1998.

[73] Erich Bircher and Torsten Braun. An agent-based architecture for service discovery and negotiation in wireless networks. In *Proceedings of*

the 2nd International Conference on Wired/Wireless Internet Communications (WWIC 2004), pages 295–306, February 2004.

[74] FIPA TC Communication. FIPA Contract Net Interaction Protocol Specification. Technical report, Geneva, Switzerland, 2002. The Foundation for Intelligent Physical Agents.

[75] FIPA TC Communication. FIPA English Auction Interaction Protocol Specification. Technical report, Geneva, Switzerland, 2002. The Foundation for Intelligent Physical Agents.

[76] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *The Proceedings of the first international Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 75–90, 1996.

[77] Jian Cao, Jie Wang, Shensheng Zhang, and Minglu Li. A multi-agent negotiation based service composition method for on-demand service. In *SCC'05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 329–332, Washington, DC, USA, 2005. IEEE Computer Society.

[78] Ohbyung Kwon, Jong Min Shin, and Seong Woon Kim. Context-aware multi-agent approach to pervasive negotiation support systems. *Expert Systems with Applications*, 31(2):275–285, 2006.

[79] Matthias Klusch. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, July 1999.

[80] Dionisis Kehagias, Andreas L. Symeonidis, Kyriakos C. Chatzidimitriou, and Pericles A. Mitkas. Information agents cooperating with heterogenous data sources for customer-order management. In *SAC'04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 52–57, New York, NY, USA, 2004. ACM.

[81] Angela Carrillo-Ramos, Jérôme Gensel, Marlène Villanova-Oliver, and Hervé Martin. Pumas: a framework based on ubiquitous agents for accessing web information systems through mobile devices. In *SAC'05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1003–1008, New York, NY, USA, 2005. ACM.

[82] Paul Klemperer. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–86, July 1999.

[83] Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. The MIT Press, Cambridge, Massachusetts, January 2006.

[84] Tom Wanyama and Behrouz H. Far. Negotiation coalitions in group-choice multi-agent systems. In *AAMAS'06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 408–410, New York, NY, USA, 2006. ACM.

[85] Ted Scully, Michael G. Madden, and Gerard Lyons. Coalition calculation in a dynamic agent environment. In *ICML'04: Proceedings of the twenty-first international conference on Machine learning*, page 93, New York, NY, USA, 2004. ACM.

[86] Bastian Blankenburg, Rajdeep K. Dash, Sarvapali D. Ramchurn, Matthias Klusch, and Nicholas R. Jennings. Trusted kernel-based coalition formation. In *AAMAS'05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 989–996, New York, NY, USA, 2005. ACM.

[87] Fernando Menezes Matos and Edmundo R. M. Madeira. An automated negotiation model for m-commerce using mobile agents. In *ICWE'03: Web Engineering*, pages 313–328, Heidelberg, January 2003. Springer.

[88] Jeferson M. Anjos and Linnyer B. Ruiz. Service negotiation over wireless mesh networks : an approach based on economic agents. In *IEEE Wireless Days: WD'08: 1st IFIP*, pages 1–5, November 2008.

[89] John Bigham and Lin Du. Cooperative negotiation in a multi-agent system for real-time load balancing of a mobile cellular network. In *AAMAS'03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 568–575, NY, USA, 2003. ACM.

[90] Shohei Yoshikawa, Takahiko Kamiryo, Yoshiaki Yasumura, and Kuniaki Uehara. Strategy acquisition of agents in multi-issue negotiation. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 933–939, Washington, DC, USA, 2006. IEEE Computer Society.

[91] Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge. *Agent-Oriented Software Engineering for Internet Applications*, pages 326–346. Springer Verlag, 2001.

[92] Goran Trajkovski. *An Imitation-based Approach to Modeling Homogenous Agents Societies (Computational Intelligence and Its Applications Series) (Computational Intelligence and Its Applications Series)*. IGI Publishing, Hershey, PA, USA, 2006.

[93] Jeremy Pitt, Abe Mamdani, and Patricia Charlton. The open agent society and its enemies: a position statement and research programme. *Telematics and Informatics*, 18(1):67–87, 2001.

[94] Nicholas R. Jennings, Peyman Faratin, Alessio R Lomuscio, Simon Parsons, Michael Wooldridge, and Carles Sierra. Automated negotiation: Prospects methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[95] Onn Shehory, Katia Sycara, Prasad Chalasani, and Somesh Jha. Agent cloning: an approach to agent mobility and resource allocation. *IEEE Communications*, 36:58–67, 1998.

[96] John Thangarajah, Lin Padgham, and Michael Winikoff. Prometheus design tool. In *AAMAS'05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 127–128, New York, NY, USA, 2005. ACM.

[97] Lin Padgham and Michael Winikoff. Prometheus: A practical agent-oriented methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, chapter V, pages 107–135. IGI Publishing, 2005.

[98] ARSLOGICA IT Laboratories. Overview, 2003. www.arslogica.it.

[99] UNITN-DISI. Laboratory for mobile applications, 2006. http://lama.disi.unitn.it/index.php.

[100] Amit B. Kothari. *Genghis - A Multiagent Carpooling System.* B.Sc. Dissertation work, submitted to the University of Bath., may 2004.

[101] Sameh Abdel-Naby, Stefano Fante, and Paolo Giorgini. Auctions negotiation for mobile rideshare service. In *Proceedings of the Second International Conference on Pervasive Computing and Applications (ICPCA07)*, Birmingham, UK, July 2007. IEEE.

[102] Fermentas Inc. UK DEPARTMENT OF TRANSPORT., 2006. http://www.dft.gov.uk/pgr/statistics/datatablespublications/personal/mainresults/nts2004/.

[103] eNotions. Dynamische Fahrgemeinschaften., 2006. http://www.mobilitaet21.de/infrastruktur-und-mobilitaet/effiziente-nutzung.html?user_umm21_pi1[detail]=115&cHash=767b343a0a.

[104] Paul O'Brien and Richard Nicol. Fipa — towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[105] Robyn Kozierok and Pattie Maes. A learning interface agent for scheduling meetings. In *IUI'93: Proceedings of the 1st international conference on Intelligent user interfaces*, pages 81–88, New York, NY, USA, 1993. ACM.

[106] Aliaksandr Birukou, Enrico Blanzieri, and Paolo Giorgini. Implicit: an agent-based recommendation system for web search. In *AAMAS'05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 618–624, New York, NY, USA, 2005. ACM.

[107] Enrico Blanzieri, Paolo Giorgini, Sabrina Recla, and Paolo Massa. Information access in implicit culture framework. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 565–567, New York, NY, USA, 2001. ACM.

[108] Aliaksandr Birukou, Enrico Blanzieri, Vincenzo D'Andrea, Paolo Giorgini, Natallia Kokash, and Alessio Modena. IC-service: a service-oriented approach to the development of recommendation systems. In *SAC'07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1683–1688, New York, NY, USA, 2007. ACM.

[109] Volha Bryl, Paolo Giorgini, and Stefano 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-2006)*, Hakodate, Japan, May 2006.

[110] Volha Bryl, Paolo Giorgini, and Stefano Fante. An implemented prototype of bluetooth-based multi-agent system. In *in Proceedings of WOA 2005: 6th AI*IA/TABOO Joint Workshop "'From Objects to Agents"'*, Camerino, MC, Italy, November 2005.

[111] Sun Microsystems. Java APIs for Bluetooth., 2006. http://www.jcp.org/en/jsr/detail?id=82.

[112] Sun Microsystems. Java Wireless Messaging API., 2006. http://www.jcp.org/en/jsr/detail?id=120.

[113] BLUECOVE. Java library for bluetooth, 2007. http://sourceforge.net/projects/bluecove/.

[114] Michael Winikoff. JACK intelligent agents: An industrial strength platform. In Rafael H. Bordini, Mehdi Dastani, Jrgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming*, chapter 7, pages 175–193. Springer US, Soeul, Korea, 2005.

[115] Tim Finin, Rich Fritzson, and Don McKay. A language and protocol to support intelligent agent interoperability. In *The Proceedings of the CE&CALS Conference*, Washington, USA, June 1992. Morgan Kaufmann.

[116] Sameh Abdel-Naby, Oleksiy Chayka, and Paolo Giorgini. Bartercell: an agent-based bartering service for users of pocket computing devices. Technical report, University of Trento, September 2009. #DISI-09-053.

[117] Object Refinery Ltd. JFree.org / JFreeChart, 2005. http://www.jfree.org/jfreechart/.