Implementing Agents Capable of Dynamic Negotiation

Marcin Paprzycki[1], Ajith Abraham[1], Amalia Pîrvănescu[2], and Costin Bădică[3]

[1] Oklahoma State University, Computer Science Department
Tulsa, OK, 74106, USA
marcin@cs.okstate.edu, ajith.abraham@ieee.org
[2] S.C.Softexpert SRL, Craiova, Romania
amalia_pirvanescu@yahoo.com
[3] Software Engineering Department, University of Craiova
Bvd.Decebal 107, Craiova, 200440, Romania
c_badica@hotmail.com

Abstract. Support for negotiation is one of the more important research issues when developing agent systems utilized in e-commerce. While, depending on the type of the transaction, different negotiation procedures need to be utilized, only very few proposed frameworks are generic and flexible enough to handle multiple scenarios. This paper presents negotiating agents, which can change their negotiation protocol and strategy through dynamic loading of reasoning models. We also describe details of the initial implementation of such a system.

1 Introduction

The advent of Internet and the rapid development of e-commerce, gave a boost to the agent technology research. While there exist many definitions of agents ([10]), for the purpose of this paper we will define them as: encapsulated computer programs, situated in an environment, and capable of flexible, autonomous actions focused on meeting their design objectives ([27]). For such agents, e-commerce is considered to be one of the paradigmatic application areas. The total number of e-commerce WWW sites was recently estimated at more than 150,000 ([11]) with revenue projections up to $1.5 trillion in 2004 [1, 9]. In the context of e-commerce, automated trading using agents are expected to reduce transaction costs (thus reducing the prices and increasing the revenue). Our research indicates that most currently existing automated trading systems are not robust enough to become the foundation of the next generation of e-commerce. For example, the Kasbah Trading System ([5]) supports buying and selling but does not include auctions; SILKROAD ([20]), FENAs ([17]) and Inter-Market ([18]) exist as "frameworks" but lack an implementation (which is typical for most agent systems in general ([21, 22]). This paper is a follow up to ([23, 24]) where we proposed a system in which agents can operate according to different business models including auctions, reverse auctions, trading, e-sales etc. This was to be made possible by constructing agents out of independently pluggable,

loaded remotely on-demand modules. In [24] we have described how this system can be implemented using JADE 2.6 agent platform. Here, we will first, outline the rationale and the design of the proposed system and follow with the description of the implementation of the demonstrator system.

2 Negotiations

Negotiation is a method for coordination and conflict resolution, where conflict is understood broadly as any form of goal disparity (for more details see, [23]). As indicated above, it is expected that, on the Internet, autonomous agents will negotiate on behalf of customers. Overall, research on agent-negotiations can be divided into approaches based on game theory and artificial intelligence. *Game-theoretic approaches* are directed towards developing optimization algorithms (e.g. [29, 2]) and take into account both cooperative and non-cooperative agents. In the case of cooperative agents, the problem space is divided between agents and when a conflict arises, the team members negotiate to find a resolution [26]. In non-cooperative situation, theories like Nash equilibrium are applied to the bargaining problem to find the optimum solution. Their main drawback of game-theoretical approaches is that they involve highly abstract models that assume unrealistic properties of the game: e.g. agents that have the entire common knowledge and unbounded rationality, unlimited computation power and indefinite negotiation time. While impossible to implement they helped develop other theories, i.e. techniques for agents participating in auctions e.g. Dutch, English, Vickery, etc. *Artificial Intelligence based approaches* utilize trading heuristics (decision trees, Q-learning and evolutionary algorithms etc.) for different market mechanisms (e.g. [5, 25]). They focus on the negotiation process rather than the outcome of the negotiation. The agents used are presumed to be adaptable, realistic and sociable. When considering the practical aspects of designing multi-agent negotiations, the negotiation protocol, negotiation objects and the reasoning models ([16]) need to be taken into account.

1. *Negotiation protocol* consists of a set of rules that govern the interaction among agents. Some examples of the rules are: permissible types of participants: negotiators, third parties; negotiation states: accepting bids, negotiation closed; valid actions of the participant in particular states, etc.
2. *Negotiation objects* are ranges of issues over which agreement must be reached.
3. *Reasoning model* is the apparatus that participants employ in order to achieve their negotiation objectives. For instance, it is a mechanism by which the next counter-offer is calculated. Some of the strategies developed are argumentation, persuasion and heuristics-based. Obviously, the selection of the reasoning model depends on both the protocol and the negotiation object [25].

3 System Design

It is often suggested that, communicating mobile agent teams can handle simultaneously very large amounts of information whilst disburdening users who can be off-line, while agents continue working for them [28]. One of the typical scenarios used here is the case of mobile agents carrying out automated negotiations. Unfortunately, mobility comes at a cost. Mobile agents can either be lightweight, or loaded with substantial "reasoning power. To avoid this problem we propose to dynamically load appropriate operational modules into agents that are to be involved in negotiations. In this way, only lightweight "agent-skeletons migrate between sites and only modules required for negotiations are loaded (for a description of a somewhat similar system, see [18]).

More precisely, an agent in our system is composed of a static "core" and three pluggable components (for more details see [23]), as depicted in Figure 1).

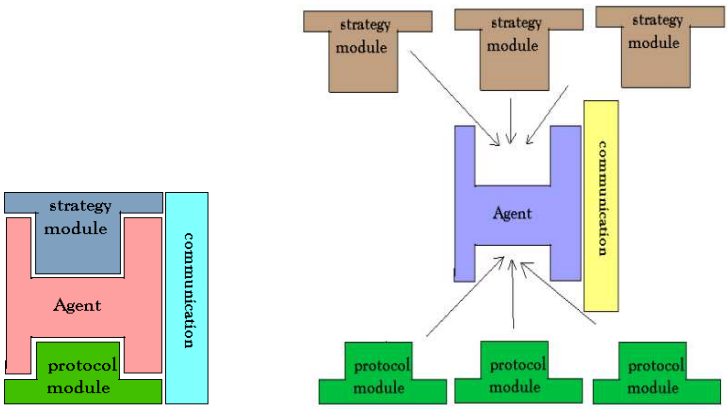

Fig. 1. An agent, its static core, and its plug-in components

1. *Communication module*: responsible for communication between the agents. The Agent Communication Language (ACL) a FIPA standard for agent communication by FIPA is used here [8]. This module is static one (even though this could be an interesting research topic, in this note we are not concerned with dynamically loading communication modules).
2. *Protocol module*: contains general rules of negotiation; when an agent initiates negotiation it finds out which negotiation protocol has to be used and dynamically loads the correct module (from the user's local machine or any agent server, e.g. the nearest one; observe that negotiation protocols can be standardized as it is the strategy that will have to be kept secret).
3. *Strategy module*: contains reasoning policies, which are a set of goals, actions and action rules (triggers). To decide which reasoning model to use, the agent uses the mapping table similar to Table 1 below, containing information about potential sellers (or buyers), resulting from past transactions

and/or additional information sources. Agent carries only information about "sites most often negotiated with. The remaining information as well as individual strategy modules, will have to be stored on a trusted server to prevent problems caused by users machine being off-line.

Table 1. Sample matchmaking table for negotiation initialization

SELLER	PRODUCT	PROTOCOL	STRATEGY	SUCCESS RATE
1	Used Cars	Offer-Counter Offer	Tit-For-Tat	0
2	Used Cars	Offer-Counter Offer	Tit-For-Tat	60
3	Used Appliances	Argumentation	Persuade/Critique	90
4	Used Appliances	Auction	Heuristics	70
5	Travel Package	Offer-Counter Offer	Boulware + Time dependent	40
6	Travel Package	Bidding		
7	Air Tickets	Auction		

4 Current Implementation

Before we proceed, we have to stress that our objective was to implement dynamically loaded protocol and strategy modules working within mobile agents. At this stage our intention *was not* developing and/or implementing sophisticated reasoning algorithms (which are nowadays object of an extensive research[25, 16, 3]). Therefore we have implemented very simple bargaining process for car buying and selling (however, for all practical purposes, this latter choice was inconsequential for our implementation).

Recently, Grifell et al. ([12–14]) have worked on designing plug-in modules and rule-based negotiation. We have taken ideas and build upon them so that new functionality can be added on the fly with need of reconfiguration but no recompilation. An implementation of our system utilizes Java's dynamically loaded classes. In particular, each required functionality is encapsulated in a Java class and an agent loads the required module locally or remotely by using reflection and Java's dynamic class loading.

The initial implementation of the proposed system utilizes the JADE agent platform ([15]), which is an open-source, fully FIPA compliant environment. Furthermore, recent experiments showed, that JADE scales well and therefore, we expect that in the near future we will be able to perform experiments with hundreds agents. Finally, JADE has large and vigorous user community and is being actively developed, with new versions released approximately every 6 month. This latter fact had also some unpleasant consequences. We started implementing our system using JADE 2.61 (see [24] for details) and with the new release we decided to port it to JADE 3.1. This is when we experienced problems caused by changes in the JADE API. We were forced to modify our code at the class level and at the method level.

The most important modifications at the class level were dealing with agent ontologies. Classes of package *jade.onto* have been removed and their functionality has been encapsulated into classes of the *jade.content.onto* package (*SlotDescriptor*, *Schema*, etc.). The addition of roles to ontology items has been replaced with the *ConceptSchema* class that offers similar possibilities (field addition for example). The registration of an ontology is now different, as the method *registerOntology* migrated from the *Ontology* class to the *ContentManager* class. The *DeafultOntology* class has been removed and the *Ontology* class has been transformed from an interface into a concrete class.

At the method level, the method to set a *MessageTemplate* in an agent behavior has been split into three methods to set each behavior element separately: *DataStore*, *MessageTemplate* and *Deadline*. These can be seen in the *DutchAuctionInitiator* and *EnglishAuctionInitiator* classes. Class *Preferences*, that represents agent ontology, must now implement the *Concept* interface from the *jade.content.onto* package. In the communication module the codecs have been changed. A codec is now registered in a different way. In the original implementation we have used the class *SL0Codec*, which doesn't exist anymore. Now there is only the *Codec* class that needs a parameter to determine the type of codec used by the message content. And finally, instead of the already deprecated methods in the original implementation *extractMessageContent* and *fillMessageContent* we now use *extractPreferences* and *fillPreferences*.

4.1 Agents and Agent Interactions; an Overview

In our initial implementation a "personal agent is responsible for creating buyer and seller agents (instances of negotiator agents). In Figure 2 we present the GUI interface to the personal agent. To initiate the process the required fields have to be entered (e.g. price and protocol). The *Start Negotiation* menu item initiates negotiations. Let us present the process from the point of view of a seller agent which has been created and enters a marketplace (we omit all details related to the organization of the marketplace, or a Negotiation Server environment where the negotiations take place). Seller agent tries to find prospective buyers by searching the Directory Facilitator (DF) for all agents of the $Type = buyer$. In the next step, the seller agent determines which negotiation protocol is favored by the majority of buyer agents (at this stage we assume a single seller agent; it is however possible that a number of agents can work together as a team; for instance, the seller agent could clone itself for each encountered protocol). Once the preferred protocol is determined, the seller agent loads the corresponding protocol module as well as an appropriate strategy module and initiates the negotiation process (in the case of multiple protocols, each seller agent clone loads its negotiation protocol and a corresponding strategy protocol). Buyer agents respond and the negotiation process continues until a buyer is found or no buyers are found (the case of multiple agents, the final result of negotiation would have to be further meta-negotiated between the seller agent clones).

Fig. 2. GUI interface of the personal agent

4.2 Negotiation Protocols

We have restricted our attention to auctions and implemented English and Dutch protocols as defined by FIPA. In particular, we have created a *myFipaEnglishAuctionInitiatorBehavior* subclass and a *myFipaDutchAuctionInitiatorBehaviour* subclass of the *EnglishAuctionInitiator* class and the *DutchAuctionInitiator* classes in the *jade.proto package*. Let us now briefly summarize both auction protocols.

A. *FIPA English Auction Interaction Protocol*: the auctioneer seeks to find the market price of a good by initially proposing a price below that of the expected market value and then gradually raising it. Each time the price is announced, the auctioneer waits to see if any buyers will signal their willingness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with a higher price. The auction continues until no buyers are prepared to pay the proposed price, when the auction ends. Commodity is sold only if the last accepted price exceeds the auctioneer's (privately known) reservation price.
B. *FIPA Dutch Auction Interaction Protocol*: the auctioneer attempts to find the market price for a good by starting bidding at a price higher than the expected market value, then progressively reducing the price until one of the buyers accepts the price. The rate of reduction of the price is up to the auctioneer and usually a reserve minimal price is involved.

We have also considered a situation when there are two or more buyers offering the same, winning price. In this case, the actual winner is the first buyer who submitted a winning bid.

4.3 Negotiation strategies

Since the aim of our work was to test-implement an agent system, and *not* to deal with specific negotiation mechanisms, we have implemented two very simple negotiation strategies. The seller increments the price by 10 in, what we named, the *Heuristics Reasoning* module and by 20 in, what we named, the *Argumentation Reasoning* module (obviously, these names only indicate negotiation strategies that could have been used here). Currently we load these two modules randomly as a proof of concept.

5 Proof of Concept Implementation

5.1 Experimental Setup

In order to demonstrate the dynamic loading of modules, we initialized JADE with two containers: the *Main container* and the container attached to it called *Container-1*. The personal agent resides in the *Main container* while the negotiating agents, upon reception of preferences from the personal agent, migrate to *Container-1*. The *Container-1* may (but does not have to) exist on a remote machine and represents the marketplace. As can be seen in Figure 3 the name of host is *ahile* and it is running Windows XP. This setup can be seen from the screen shot presented as Figure 3.

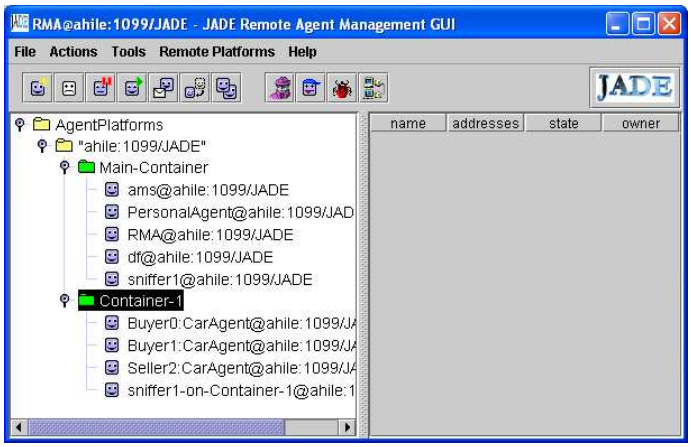

Fig. 3. Screen capture of the Remote Management Agent GUI for JADE set-up with two containers

To run the system the following steps are necessary:

1. Start JADE and create the personal agent.
2. Create a satellite container named *Container-1* that acts as the marketplace.

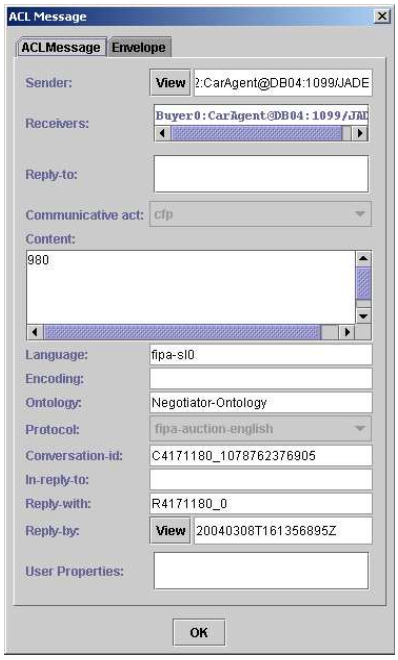

Fig. 4. Screen capture of the content of an ACL message

3. When a screen with default values appears (and the default mode is buyer), create multiple buyer agents.
4. Start a sniffer agent in order to monitor the exchange of transaction messages.
5. Initiate the sniffer for all the agents created thus far.
6. Create a seller agent and enter a reserved price.
7. Initiate the sniffer to include the seller agent.
8. When auction is completed, the messages exchanged by the buyers and the seller are visible in the sniffer agent GUI (Figures 5, 6, 7 and 8). Note that it is possible to check the content of a specific ACL message by right-clicking on an appropriate message-arrow (Figure 4).

5.2 Running the System

Let us now present samples of system operation for both English and Dutch auctions. As indicated above, when two or more buyers have the same reserved price (conflict) the seller will accept the first buyer who submitted the winning bid. For each experiment we present the sequence of messages exchanged between the seller and buyers captured with the help of the sniffer agent.

In the English auction, the seller starts with its reserved price (indicated by the user in the *Personal Agent* GUI) and sends CFP messages to all buyers, no matter their protocol. Only the buyers with the same protocol will reply with

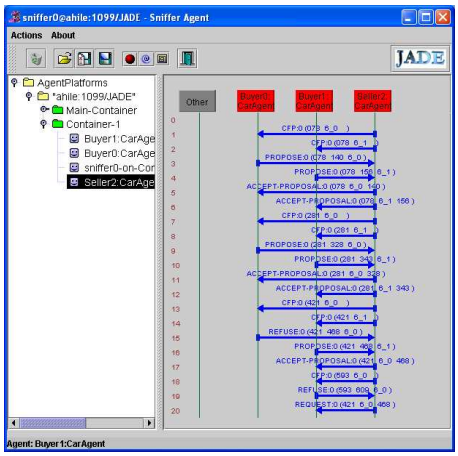

Fig. 5. English auction without conflict

REFUSE or PROPOSE messages. The seller increases the proposed price and responds with CFP messages only to these buyers that have sent a PROPOSE message. The process is repeated, until the seller receives no proposals. In this case, the buyer that wins the auction is the first buyer that submitted a proposal in the previous step. That agent receives an INFORM message from the seller. We performed two experiments with the English auction protocol: without and with conflict. In the first experiment we created two buyers having the reserved prices 1000 and 1020; then we create a seller with the reserved price 980. The buyer that wins the auction is the one with the reserved price 1020. The results are presented in Figure 5.

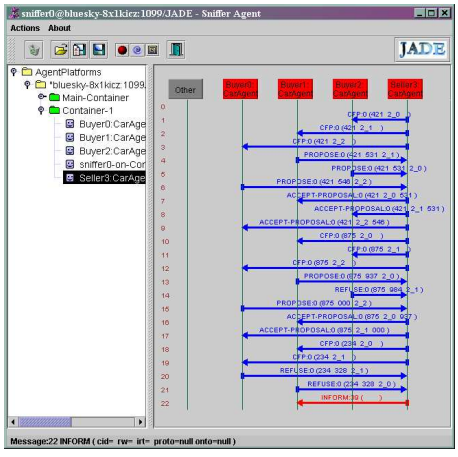

Fig. 6. English auction with conflict

In the second experiment (with conflict) we created three buyers having the reserved prices 1000, 1000 and 980; then we create a seller with the reserved price 960. The buyer that wins the auction is one with the reserved price 1000. The results are presented in Figure 6.

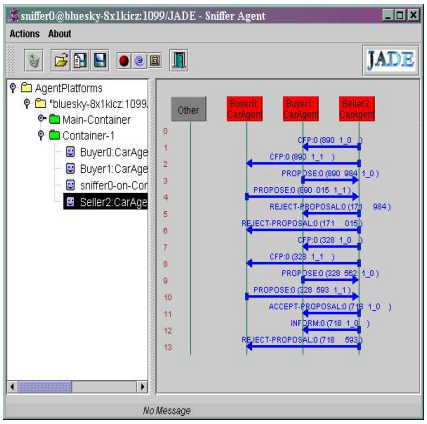

Fig. 7. Dutch auction without conflict

In Dutch auction, the seller begins with a price higher than its reserved price and receives PROPOSE messages from the buyers; if the reserved price of the buyer is smaller that the price proposed by the seller, the content of the propose message is 0; the process is repeated until the first PROPOSE message with a content different from 0 is received. When two or more buyers send a PROPOSE message with the content different from 0 at the same time, the first buyer that sent the message wins the auction. We performed two experiments with the Dutch auction protocol: without and with conflict. In the first experiment we created two buyers having the reserved prices 1000 and 1010; then we create a seller with the reserved price 870. The buyer that wins the auction is one with the reserved price 1010. The results are presented in Figure 7.

In the second experiment we created three buyers having the reserved prices 1000, 1000 and 980; we created a seller with the reserved price 860. The buyer that wins the auction is one of the buyers with the reserved price 1000. The results are presented in Figure 8.

6 Concluding remarks

In this paper we presented an agent framework capable of dynamically loading protocol and strategy modules. negotiations. We have also discussed details of a JADE 3.1 based implementation of an actual demonstrator system. We have shown that it is possible to facilitate adaptive behavior of the system within JADE environment. Obviously, the implemented system is highly simplified, as our goal was to perform an initial assessment of the feasibility of our approach. In

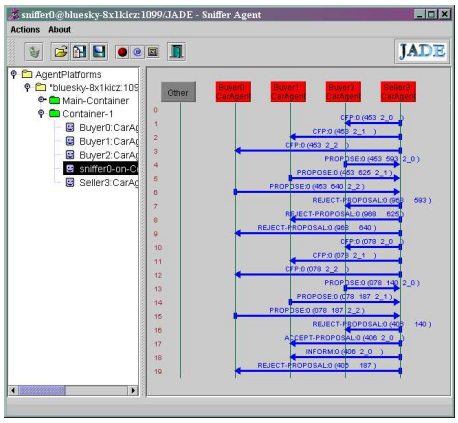

Fig. 8. Dutch auction with conflict

the future we plan to continue developing its capabilities in two main directions. The first one is associated with agent technology itself. Here we plan to, among others, increase the number of agents and computers; test the above-mentioned possibility of utilizing agent cloning; test the assumption that sending modules over the network improves performance of the system. As a step in this direction we have developed the agent-based e-commerce skeleton [6] within which we will now introduce our adaptive negotiating agents. The second is associated with the negotiation process itself. There are two paths to follow here. We would like to add more auction types to our system (as specified by the FIPA standards), like Vickrey auctions and experiment with buyers having various preferences in the product types and their specific features. Also of course, we have to take the existing results and actually implement them in our agents and use such agents in more realistic e-commerce scenarios.

References

1. Andersen Consulting. http://www.andersen.com. (accessed on September 07, 2004)
2. AuctionBot. http://auction.eecs.umich.edu. (accessed on September 07, 2004)
3. Beer, A., d'Inverno, M., Luck, P., Jennings, P.: Negotiation in Multi-Agent Systems, *Knowledge Engineering Review*, 14(3), (1999) 285–289.
4. Benameur, D., Chaib-draa, A., Kropf, H.: Multi-item Auctions for Automatic Negotiation, *Journal of Information and Software Technology*, 44/5, (2002) 291–301.
5. Chavez, V., Maes, P.: Kasbah: An Agent Marketplace for Buying and Selling Goods, Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, 1996.
6. Chmiel, K., Czech, D., Paprzycki, M.: Agent Technology in Modeling E-commerce Processes; Sample Implementation, Proc. of the MISSI-2004 Conference, Szklarska Poreba, Poland, 2004, *to appear*.
7. Chmiel, K., Tomiak, D., Gawinecki, M., Karczmarek, P., Szymczak, M., Paprzycki, M.: Testing the Efficiency of JADE Agent Platform, Proc. of the 3^{rd} Int.Symp.on Parallel and Distributed Computing, Cork, Ireland, 2004, *to appear*

8. http://www.fipa.org. (accessed on September 07, 2004)
9. Forrester. http://www.forrester.com. (accessed on September 07, 2004)
10. Galant, V., Tyburcy, J.: Intelligent Software Agent (in Polish), in: A. Baborski (ed.), Knowledge Acquisition in Databases, Wroclaw Univ. of Economics, (2001).
11. Gartner. http://www.gartner.com. (accessed on September 07, 2004)
12. Tu, M.T., Griffel, F., Lamersdorf, W.: Integration of Intelligentand Mobile Agents for E-commerce. In St. Kirn and M. Petsch (Eds.), Workshop Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarien, TU Ilmenau, FG Wirtschaftsinformatik 2. Arbeitsbericht, (1999).
13. Griffel, F., Lamersdorf, W., Merz, M.: A plug-in architecture for providing dynamic negotiation capabilities for mobile agents, (1998).
14. Griffel, F., Lamersdorf, W., Merz, M.: Interaction-Oriented Rule Management for mobile agent applications, (1999).
15. http://jade.cselt.it. (accessed on September 07, 2004)
16. Jennings, P., Parsons, S.: On Argumentation-Based Negotiation, (1998).
17. Kowalczyk, R.: On Fuzzy e-Negotiation Agents: Autonomous negotiation with incomplete and imprecise information, In: Proc.DEXA'2000, London, UK, (2000) 1034–1038.
18. Kowalczyk, R., Franczyk, B., Speck, A.: Inter-Market, towards intelligent mobile agent E-Market places, Ninth Annual IEEE Int. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS 2002), Lund, Sweden (2002) 268–276.
19. Sandholm, T. and Lesser, V.: Coalitions among Computationally Bounded Agents. *Artificial Intelligence* 94(1), (1997) 99–137.
20. Michael, S.: Design of Roles and Protocols for Electronic Negotiations, *Electronic Commerce Research Journal*, Vol.1 No.3 (2001) 335–353.
21. Nwana, H., Ndumu, D.: A Perspective on Software Agents Research, *The Knowledge Engineering Review*, 14(2), (1999) 1–18.
22. Paprzycki, M., Abraham, A.: Agent Systems Today; Methodological Considerations, *Proc. of 2003 Int.Conf. on Management of e-Commerce and e-Government*, Jangxi Science and Technology Press, Nanchang, China, (2003) 416–421.
23. Parakh, G., Paprzycki, M., Nistor, C.E.: Dynamically Loaded Reasoning Models in Negotiating Agents, Proceedings of the 3^{rd} European E-COMM-LINE 2002 Conference, Bucharest, Romania, (2002). 199–203.
24. Parakh, G., Rani, S., Paprzycki, M., Abraham, A., Thomas, J.: Agents Capable of Dynamic Negotiations, in: M. Paprzycki (ed.), Electronic Commerce; Research and Development, ACTEN Press, Wejherowo, Poland, (2003) 113–120.
25. Sierra C., Faratin P. and Jennings N. R.: A Service-Oriented Negotiation Model between Autonomous Agents, Proc. 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97), Ronneby, Sweden, (1997) 17–35.
26. Qiu, X., Tambe, V: Flexible Negotiation in Teamwork, *Proc. of the third annual conference on Autonomous Agents*, Seattle, United States, (1999) 400–401.
27. Wooldridge, M.: Agent-based software engineering, *IEEE Trans. on Software Engineering*, 144(1) (1997) 26–37.
28. Wooldridge, M.: *An Introduction to MultiAgent Systems*, John Wiley & Sons, (2002).
29. Zlotkin, G., Rosenschein, J.S.: Cooperation and conflict resolution via negotiation among autonomous agents in non-cooperative domains. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), (1991) 1317–1324.