DR-NEGOTIATE – A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies

Thomas Skylogiannis[1] Grigoris Antoniou[1,2] Nick Bassiliades[3] Guido Governatori[4]
Antonis Bikakis[1,2]

1 Department of Computer Science, University of Crete, Greece
2 Institute of Computer Science, FORTH, Greece
3 Department of Informatics, Aristotle University of Thessaloniki, Greece
4 School of ITEE, The University of Queensland, Australia

Abstract. This paper reports on a system for automated agent negotiation, based on a formal and executable approach to capture the behavior of parties involved in a negotiation. It uses the JADE agent framework, and its major distinctive feature is the use of declarative negotiation strategies. The negotiation strategies are expressed in a declarative rules language, defeasible logic, and are applied using the implemented system DR-DEVICE. The key ideas and the overall system architecture are described, and a particular negotiation case is presented in detail.

1. Introduction

As the amount of commercial transactions carried out through the Internet increases at a spectacular rate, the interest for partially or totally automating the negotiation of the terms of these transactions has rapidly become a hot research topic [20]. Consequently, automated negotiation has evolved in a few years from a futuristic vision to a promising technology [27]. In particular, optimal strategies for several forms of automated bargaining and auctioning under simplified assumptions have been identified, and the extensions of these results to more realistic settings have been studied by the game-theory and the distributed artificial intelligence communities ([25], [34], [36]). Furthermore, several tournaments where automated traders compete to maximize their profits in electronic auction houses have been organized (e.g. [22], [40]), and their results are quite encouraging.

Software agents are becoming a choice technology for carrying out automated negotiations [36]. In this approach, each party is represented through an agent who interacts either directly with the other parties, or through a broker. The focus of our work is on the automated negotiation aspect of e-commerce. As stated in [24], *automated negotiation* is the process by which two or more agents communicate and try to come to a mutually acceptable agreement on some matter. The basic dimensions of automated negotiation are negotiation protocols and negotiation strategies. A *Negotiation protocol* is a set of rules which govern the interaction, and a *negotiation strategy* is a decision making model, which participants employ in order to achieve their goal in line with the negotiation protocol. The issue in this context is thus “how to express a negotiation strategy”.

This work reports on an implemented system for automated agent negotiation, which is based on ideas of the abstract negotiation model of [13]. That work presented a simple yet expressive framework for specifying negotiating agents' strategies, in a way that their decisions are predictable and explainable. Specifically, we explore in this paper the suitability of defeasible logic programming ([3], [18], [31]) for expressing the decision-making process of negotiating agents. In agreement with [13], we argue that defeasible logic is suitable for expressing negotiation strategies, since it straightforwardly captures concepts such as preferences, hypotheses, arguments and counter-arguments. Although defeasible logic is certainly not an end-user language but rather a developer's one, it could provide a foundation for designing "user-friendly" interfaces for negotiation strategy specification. And given the low computational complexity of defeasible logic programming [28], these strategies can also be executed in real time.

The viability of these ideas is demonstrated through a prototypical implementation, which constitutes the main contribution of the present work. The implemented system, DR-NEGOTIATE, is based on the JADE multi-agent framework, and uses DR-DEVICE [6] as the deductive engine to apply defeasible logic programs (the negotiation strategies). The overall system design is described, and its functionality is demonstrated through a concrete example.

The paper is organized as follows. Section 2 provides a discussion of the overall approach and the underlying formalisms, with emphasis on the motivation of using defeasible logic programming. Section 3 presents the system architecture, while sections 4-6 illustrate the functionality and use of the system based on a concrete case. Section 7 discusses related work, and section 8 concludes the paper with a summary and description of current and future work.

2. Rationale, Approach and Enabling Formalisms

2.1 An Architecture for Negotiating Agents

In this section we briefly present the architecture proposed by [13]. We view the negotiation process as a set of software agents which interact in order to reach an agreement. Agents participating in a negotiation can interact directly or through a broker. In some situations, the role of a party during the negotiation process is almost entirely carried out by the broker. This is the case for instance in some auction houses, where the auction broker takes the place of the seller.

Following an abstract architecture for agents with memory presented in [44] each of the software agents is composed of four modules: (i) a memory which contains the history of the past decisions and interactions of the agent, including its current intentions, (ii) a communication module responsible of receiving and sending messages to the other agents and interacting with the user, (iii) a reasoning module which encodes the decision-making part of the agent, and (iv) a control module which coordinates the other components. As a refinement to this architecture, we choose to

express the control module as a finite automaton (see section 4), the communication module was implemented using the JADE platform (see section 3), the reasoning module as a defeasible logic program (see next subsection), and the memory as a knowledge base of facts.

Conceptually, each time a negotiating agent is notified of a change in the negotiation status, it updates the base facts stored in the knowledge base accordingly, and it activates the reasoning module. The reasoning module reads these facts from the knowledge base, attempts to deduce new facts and refute existing ones, and updates the derived facts stored in the knowledge base accordingly. Depending on the state of the knowledge base after this revision process, the control module determines whether it should ask the communication module to submit a proposal or counter-proposal immediately, wait for some further event, or retract from the negotiation. It is also through the communication module that the control module communicates with the agent responsible of managing the user interface. This separation between the agent responsible of handling the negotiation process, and the one responsible of interacting with the user (e.g. for collecting the parameters of the negotiation, or for displaying its status), adds considerable flexibility to the architecture. In particular, these two agents can be located in different machines, and the negotiating agent can even be mobile.

2.2 Desired Properties for Negotiation Strategy Representation Formalisms

Before choosing one or several languages for the specification of negotiation strategies, it is important to establish a set of criteria that such languages need to satisfy. The criteria presented below are inspired from those formulated by [21] in the context of techniques for information modeling. They encompass several well-known principles of language design.

Firstly, a language for specifying negotiation strategies needs to be *formal*, in the sense that its syntax and its semantics should be precisely defined. This ensures that the strategy specifications can be interpreted unambiguously (both by machines and humans) and that they are both *predictable* and *explainable*. In addition, a formal foundation is a prerequisite for verification purposes.

Secondly, the language should be *conceptual*. This, following the well-known *Conceptualization Principle* of [17], effectively means that it should allow its users to focus only and exclusively on aspects related to strategies, without having to deal with any aspects related to their realization or implementation. Examples of conceptually irrelevant aspects in the context that we consider are: physical data organization and access, platform heterogeneity (e.g. message-passing formats), and book-keeping (e.g. message queue management).

Thirdly, in order to ease the interpretation of strategies and to facilitate their documentation, the language should be *comprehensible*. Comprehensibility can be achieved by offering a graphical representation, by ensuring that the formal and intuitive meanings are as much in line as possible, and by offering structuring

mechanisms (e.g. decomposition). These structuring mechanisms often lead to *modularity*, which in our setting means that a slight modification to a strategy should concern only a specific part of its specification. Closely related to its comprehensibility, the language that we aim should be *suitable*, that is, it should offer concepts close to those involved in negotiations.

As we are interested in the automation of the negotiation process, the strategy description language should be *executable*, and its execution should exhibit acceptable performances even for complex strategies involving many issues (i.e. the execution performance should be *scalable*).

Finally, the language that we aim should be sufficiently *expressive*, that is, it should be able to precisely capture a wide spectrum of strategies.

2.3 On Defeasible Logic Programming

Defeasible reasoning is a simple rule-based approach to reasoning with incomplete and inconsistent information. It can represent facts, rules, and priorities among rules. This reasoning family comprises defeasible logics ([31], [3]) and Courteous Logic Programs [18], and has the following characteristics:

- They are rule-based, without disjunction
- Classical negation is used in the heads and bodies of rules, but negation-as-failure is not necessarily used in the object language (it can easily be simulated, if necessary [4])
- Rules may support conflicting conclusions
- The logics are skeptical in the sense that conflicting rules do not fire. Thus consistency is preserved
- Priorities on rules may be used to resolve some conflicts among rules
- The logics take a pragmatic view and have low computational complexity [28]

Facts denote simple pieces of information deemed to be true regardless of other knowledge items. A typical fact is that an apartment *a* is air-conditioned: *aircon(a).*

There are two kinds of rules (fuller versions of defeasible logics include also defeaters [3]: *Strict rules* are denoted by $A \rightarrow p$, and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "*Professors are faculty members*". Written formally: $professor(X) \rightarrow faculty(X)$. Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by $A \Rightarrow p$, and can be defeated by contrary evidence. An example of such a rule is $faculty(X) \Rightarrow tenured(X)$ which reads as follows: "*Faculty members are typically tenured*".

A *superiority relation* on R is an acyclic relation > on R (that is, the transitive closure of > is irreflexive). When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

$$r: professor(X) \Rightarrow tenured(X)$$
$$r': visiting(X) \Rightarrow \neg tenured(X)$$

which contradict one another, no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation > with $r' > r$, then we can indeed conclude that he/she cannot be tenured. We assume that the superiority relation is acyclic.

For each literal *p* we define the set of *p-complementary literals C(p)*, that is, the set of literals that cannot hold when *p* does. Let us consider an example: suppose we have the predicates *married* and *bachelor*. Here, we define, for any constant *a*, $C(married(a)) = \{\neg married(a), bachelor(a)\}$. We know that, under the usual interpretation of the predicates, they cannot be true at the same time for one and the same individual. We stipulate that the negation of a literal is always complementary to the literal.

We now give a short informal presentation of how conclusions are drawn in Defeasible Logic. A conclusion P can be derived if there is a rule whose conclusion is P, whose prerequisites (antecedents) are either already been proved or given in the case at hand (i.e. facts), and any stronger rule whose conclusion is in C(P) has prerequisites that fail to be derived. In other words, a conclusion P is (defeasibly) derivable when:

- P is a fact; or
- there is an applicable strict or defeasible rule for P, and either
  - all the rules for P-complementary literals are discarded or
  - every rule for a P-complementary literal is weaker than an applicable strict or defeasible rule for P.

A formal definition of the proof theory is found in [3]. A model theoretic semantics is found in [29], and argumentation semantics is discussed in [16].

In what follows, we rather focus on the application of defeasible logic to automated negotiation. Specifically, we sketch a set of guidelines that can be used to formalize a negotiation strategy in defeasible logic. First of all, the negotiating agent developer needs to conduct an information analysis in order to identify an appropriate set of predicates to encode the negotiation strategy. These predicates must collectively capture the information characterizing a given negotiation situation (e.g. negotiation issues, user parameters, thresholds, limit values, histories of offers, etc.) as well as the conclusions that can be derived from a negotiation situation (e.g. the actions that may need to be undertaken at a given point during the negotiation).

Next, the developer needs to identify the constraints over the negotiation, and the business rules governing the negotiation strategy. These constraints and business rules must then be classified as either hard or soft. Hard constraints (or hard business rules) are those that apply in any situation, regardless of the context. These constraints set the basic boundaries of the negotiating agent behavior, and are used (for example) to model decisions that must be systematically taken when a given condition holds. Hard constraints and hard business rules are formalized directly as strict rules.

Soft constraints (or soft business rules) on the other hand can be violated under particular circumstances (i.e. they have exceptions). They are used to model the guidelines and user preferences that the negotiating agent will consider after making sure that all the hard constraints and rules are met. A soft constraint is formalized as a team of defeasible rules. A soft constraint is formalized as a team of defeasible rules. Specifically the soft constraint itself is first translated into a defeasible rule in this team, and each exception to the constraint is then encoded as a separate defeasible rule, whose conclusion is the negation of the conclusion of the defeasible rule corresponding to the soft constraint. For example, if a soft constraint C is formalized by the defeasible rule *r1: A1,...,An* $\Rightarrow$ *B* (meaning that normally if A1,...,An hold, then so should B), if we know that the state of affairs D constitutes an exception to C, then D must be formalized as *r2: D* $\Rightarrow$ $\neg$*B*. Since D is an exception to C, we have to specify that r2 has precedence over r1, i.e. *r2 > r1*.

In the third step, the developer should identify pairs of incompatible literals. Two literals are said to be incompatible if they cannot both hold at the same time, which essentially means that one of the literals implies the negation of the other. Having identified conflicting literals, and with the aid of an inference tool, the developer can then detect conflicting (defeasible) rules, i.e. rules such that the literals appearing in the conclusions are incompatible. As we have alluded to above, no conclusion can be drawn from conflicting rules in defeasible logic, unless these rules are prioritized. For each pair of conflicting defeasible rules, the developer must analyze what will happen when the conjunction of the antecedents of the rules holds, and must deduce a priority between these rules from this analysis. In some cases, this analysis can be partially supported by an automated tool.

Indeed, several criteria for automatically determining priorities among rules have been put forth, one of the most common being the *specificity* criterion. In a nutshell, a rule is more specific than another one, when it applies in all cases where the other does. In other words, the set of prerequisites of the more general rule is a subset of the set of prerequisites of the more specific rule. In such cases, it can be suggested to the developer to give a higher priority to the more specific rule. However, the developer may decide to do the opposite, since specificity alone is not always an appropriate criterion for ranking rules. In addition, in general, two rules cannot be compared according to the specificity criterion, and thus, the developer must either determine different ranking criteria or perform a manual ranking.

2.4 Why Defeasible Logics for Negotiation Strategies

Applying a negotiation strategy in a particular context is an intensive decision-making process. While most aspects of negotiation strategies could be fully captured in classical logic programming (which has a formal semantics and has proven to be a powerful tool for building decision-making systems), this would put a burden on the developers of strategies, since logic programming is a generic paradigm and offers nothing specific to strategy specification (such as argumentation, defeasibility, hypothetical reasoning, preferences, etc.). Accordingly, we propose to use a logic programming language based on non-monotonic reasoning. Among the many members of the family of non-monotonic logics, we choose defeasible logic [3] for the following reasons.

- A negotiation can be thought of as a dialogue between parties concerning the resolution of a dispute. This suggests that argumentation based reasoning formalisms are suitable to characterized it. In [16], it was shown that defeasible logic can be characterized by an argumentation semantics, thus the formal semantics of defeasible logic is in line with the argumentative nature of negotiations.
- Given the close connection between derivations in Defeasible Logic and arguments, strategies expressed in Defeasible Logic are explainable.
- Defeasible logic is a skeptical formalism, meaning that it does not support contradictory conclusions. Instead it seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding ¬A, the logic does not conclude either of them (thus the name “skeptical”). If the support for A has priority over the support for ¬A then A would be concluded. We believe that non-skeptical reasoning is inappropriate for modeling decision-making processes such as negotiations, since it is quite useless to deduce both that a decision should be taken, and that it should not be taken.
- Defeasible logic integrates the concept of priorities between rules, thereby supporting a direct way of modeling preferences, without having to attach a metric to them, as it is the case of approaches based on utility functions [32].
- Regarding strategy specification, most of the current systems adopt a quantitative approach based on utility functions. Very often, it is not easy to find the right utility functions for a given set of negotiation issues, especially in situations where one needs to express preferences without attaching a metric to them. Moreover, utility functions are mostly used to determine preferences that can otherwise be expressed in a more comprehensible and suitable way through defeasible rules and priorities among these rules. For this reason, we believe that defeasible logic is more suitable than, or at least complementary to, strategy specification approaches purely based on utility functions.

- Defeasible logic has a linear complexity, and existing implementations are able to deal with non trivial theories consisting of over 100,000 rules [30], offering thus an executable and scalable system.

3. Implemented Agent Architecture

The agent framework we used is JADE ([8], [23]), an open-source middleware for the development of distributed multi-agent applications based on the peer-to-peer communication architecture. JADE is Java-based and compliant with the FIPA specification. It provides libraries for agent communication and interaction, based on FIPA standards [14]. It also provides tools for agent lifecycle management, inspection of exchanged messages and debugging. JADE provided us with the agent infrastructure we desired.

For the reasoning module of the agent we used the defeasible reasoning system DR-DEVICE [6]. Its user interface is compatible with RuleML, the main standardization effort for rules on the semantic web and is based on a CLIPS-based implementation of deductive rules [7].

The architecture of the negotiating agent is depicted in Fig. 1. When the agent is notified of an external event, such as an incoming message (step 1), the control module initially retrieves a fact template from the local storage unit (step 2) and consequently, the negotiation parameters from the memory (step 3). The template is an empty placeholder in line with DR-DEVICE system syntax. When the template is filled with the negotiation parameters, is then regarded as “the facts”. The control module updates the knowledge base with the new facts (step 4) and then activates DR-DEVICE (step 5). DR-DEVICE in turn retrieves from the knowledge base the facts, along with the strategy (step 6) and starts the inferencing process. After the inferencing has been completed, the knowledge base is updated with the results (step 7). The control module queries the knowledge base for the result (step 8) and after a short processing an appropriate message is posted to the communication module.

4. The Negotiation Protocol: An Example

As we have already stressed, a basic condition for the automation of the negotiation process among intelligent agents is the existence of a negotiation protocol, which encodes the allowed sequences of actions. Our first thought was to use a well-defined protocol for 1-1 automated negotiation. Although FIPA provides a plethora of standardized protocols, such as FIPA brokering, FIPA English auction, FIPA Contract net etc., we found that there is no standard interaction protocol for 1-1 automated negotiation.

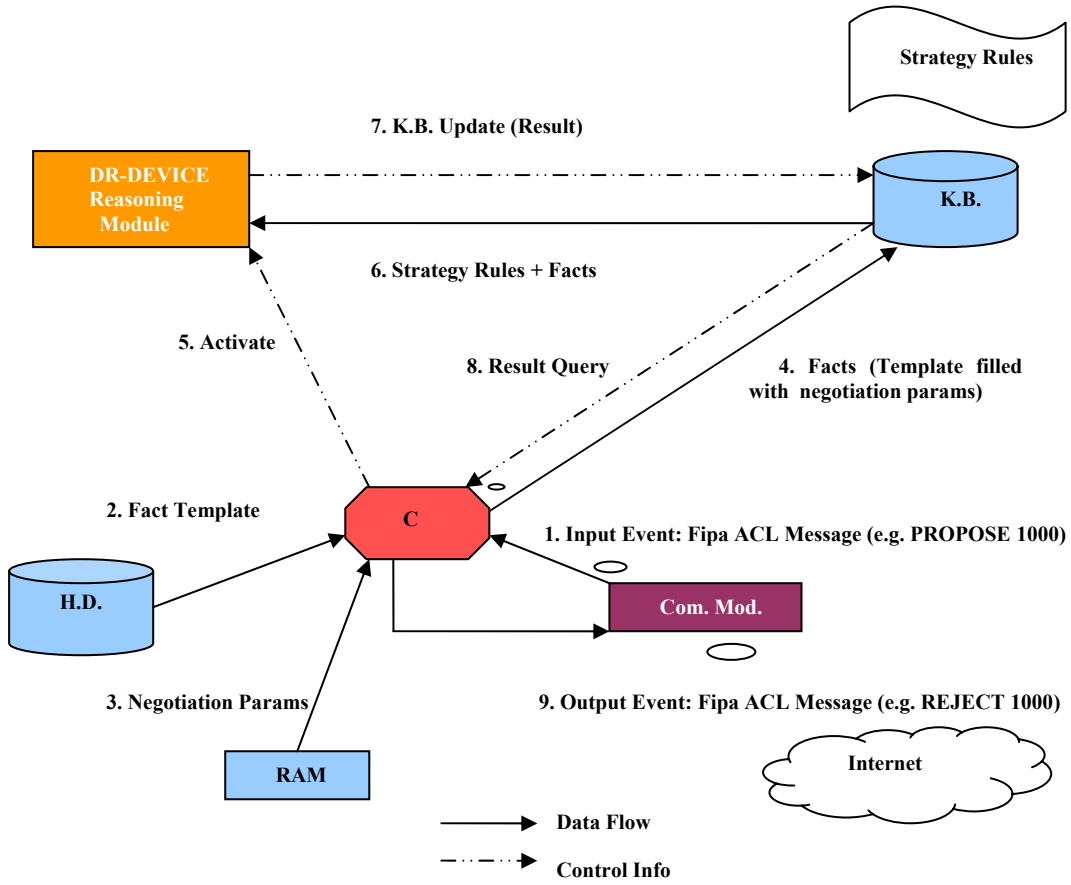

Fig. 1. Architecture of Negotiating Agent.

As a result, we implemented a negotiation protocol proposed in [41]. This protocol is a finite state machine that must be hard-coded to all agents, participating into the negotiation. Bartolini et al. [5] say that most multi-agents systems today use a single negotiation protocol which is usually a finite state machine, hard-coded to all the agents, leading to an inflexible environment, which can accept only agents designed for it. To overcome this inflexibility they propose a generic interaction protocol which can be parameterized with different negotiation rules and give different negotiation mechanisms. The rules can be exchanged among agents that are able to inform their peers, which protocol they wish to use. Governatori et al. [15] show how to use Defeasible Logic to represent both negotiation protocols and strategies. However, the focus of our work is not on protocol design but rather on declarative negotiation strategy specification, therefore we believe the protocol we use is a good solution for our demonstrator.

Our protocol is a finite state machine with discrete states and transition. The protocol is depicted in Fig. 2. INIT is the initial state of the negotiation, PROP_SENT,

PROP_REC, PROP_REJ, PROP_ACC, ACC_PROP, REJ_PROP, represent the states of a negotiation, and TERM is the final state in which there is an agreement, or a failure of agreement between the participants. *Send* and *Recv* predicates represent the interactions which cause state transitions. To clarify the function of the protocol we give an example. If the sequence of transitions is the following: INIT→PROP_SENT →PROP_REC→ACC_PROP→TERM, that means that the agent initially sends a call for proposal message (CFP) to the other negotiating agent (INIT→PROP_SENT), then it receives a propose message (PROP_SENT→PROP_REC) and after the evaluation it decides to send an accept message (PROP_REC→ACC_PROP). Lastly it receives an accept message and the negotiation terminates successfully (ACC_PROP→TERM). We make the convention that the participant that plays the role of the buyer starts the negotiation by posting a CFP message. So, while the protocol can be used as it is by a buyer, it needs a small modification for a seller. Particularly instead of the transition INIT→PROP_SENT there should be a transition INIT→PROP_REC with label "Recv CFP".

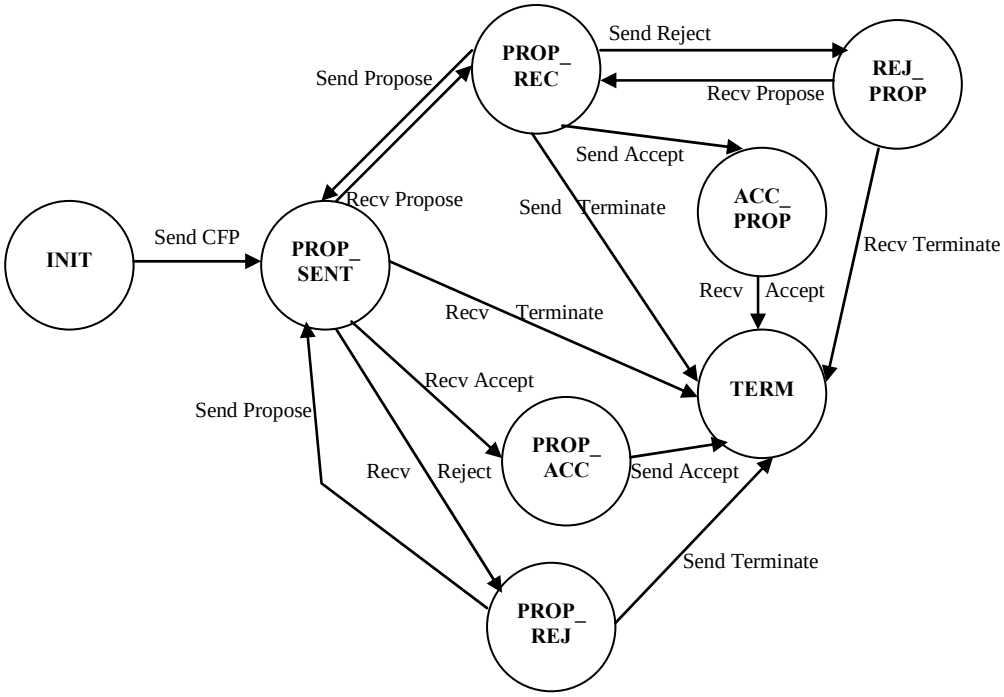

Fig. 2. 1-1 Negotiation Protocol.

5. The Negotiation Strategy: An Example

The strategy of a potential buyer or seller during a negotiation scenario is very critical for the outcome of the encounter. Every strategy is indeed designed in line with a particular protocol. As we have already seen, there is a plethora of strategies classified according to different criteria. We based the strategies we used on the work of Tsang et al. [43]. They define the simple constrained bargaining game between one buyer

and one seller. Some of the most important assumptions are that the seller is constrained by the cost and the number of days within which it has to sell, while the buyer is constrained by its utility and the number of days within which it has to buy. In addition, neither the buyer nor the seller has information about the constraints of the other. The players make alternative bids with the seller to bid first and they can bid only once per day. An agreement is reached when both buyer and seller bid for the same price. Finally, according to the assumptions, if a deal cannot be made before a player runs out of time the negotiation terminates. Tsang et al. propose a number of different strategies both for buyer and sellers. For our buyer we adopted the simple buyer strategy, whose characteristics are found in the following table.

Table 1. Buyer's Strategy Characteristics.

Strategy Name	First Bid Algorithm	Offer-Acceptance Criterion	Last Day BiDDING	General Bid Algorithm
Simple Buyer	Utility/Days to Buy (DTB)	Counteroffer +Minimum Profit (MP)< Utility	Utility –MP	Bid half way between previous bid price and utility

In our work we have modified the strategy of the buyer as follows: Firstly, we relaxed the rule that only one offer per day can be made, which is completely unrealistic in an e-Commerce setting. Instead we allow *one offer per negotiation step*. The negotiation step is handled by the protocol and increases each time a player (buyer or seller) has made an offer and subsequently has received a counteroffer or another message. So, we speak of time to buy (TTB) and time to sell (TTS), measured in negotiation steps. Secondly, except for the offer-acceptance criterion we have added an extra check during the offer submission to avoid non-beneficial results for the player (see below). Thirdly, we incorporated into the strategy parameters relevant to the protocol like the state of the negotiation and the step of the negotiation. Lastly, an agreement is reached when both buyer and seller send an "agree" message.

For the buyer participating into the negotiation, we used the modified strategy of Tsang et al. and we expressed it in defeasible logic and for the seller we used a strategy hard coded in Java to demonstrate that agents with diferrent architecture can interact without any problems – we could have easily expressed the seller's strategy in defeasible logic as well. The seller's strategy is quite similar with that of buyer except for the general bidding strategy. The seller decreases its offer by a fixed amount and not in a linear fashion like the buyer.

First we express the buyer's strategy in defeasible logic (see Fig. 3) and then we give some samples of the strategy expressed in DR-DEVICE's defeasible logic rule language in native CLIPS-like syntax (see Fig. 4). The predicates we use are the following:

- *Step(s)*: The step of the negotiation. When a buyer or seller sends a message and then receives another one the step is increased by one.
- *Counteroffer(c)*: The offer which a buyer or seller receives from the opponent.
- *Min_profit(mp)*: The minimum profit the buyer seeks after buying the product.
- *Utility(u)*: The utility of the buyer if it buys the product.
- *Ttb(ttb)*: The time (negotiation steps) the buyer has at its disposal in order to buy the product.
- *State(st)*: The current state of the negotiation according to the protocol. The possible states are:
  1 (PROP_SENT): The buyer has already sent a CFP or a PROPOSE message.
  2 (PROP_REC): The buyer has already received a PROPOSE message.
  3 (PROP_REJ): The buyer has already received a REJECT message.
  4 (REJ_PROP): The buyer has rejected the received proposal (i.e. has already sent a REJECT message).
  5 (PROP_ACC): The buyer has already received an ACCEPT message.
  6 (ACC_PROP): The buyer has accepted the received proposal (i.e. has already sent an ACCEPT message).
- *Previous_bid(prb)*: The previous bid of the buyer; if it is zero, then the buyer has not yet made a offer.

r_1: State(2), Counteroffer(c), Min_profit(mp), Utility(u), c+mp≤u/2
⇒ ACCEPT_PROPOSAL

r_2: State(2), Counteroffer(c), Min_profit(mp), Utility(u), c+mp>u
⇒ ~ACCEPT_PROPOSAL

r_3: State(5) ⇒ ACCEPT_PROPOSAL

r_4: Step(0), Ttb(ttb), Counteroffer(c), Min_profit(mp), Utility(u), State(2), u/2<c+mp≤u, bid=u/ttb ⇒ PROPOSE(bid)

r_5: Step(s), Ttb(ttb), State(2), Counteroffer(c), Min_profit(mp), Utility(u), Previous_bid(0), 0<s<ttb, u/2<c+mp≤u, bid=u/ttb ⇒ PROPOSE(bid)

r_6: Step(s), Ttb(ttb), State(2), Counteroffer(c), Min_profit(mp), Utility(u), Previous_bid(prb), 0<s<ttb, u/2<c+mp≤u, prb!=0, bid=(u-prb)/2+prb
⇒ PRELIM_PROPOSE(bid)

r_7: Step(s), Ttb(ttb), State(3), Previous_bid(prb), Utility(u), 0<s<ttb, bid=(u-prb)/2+prb ⇒ PRELIM_PROPOSE(bid)

r_8: PRELIM_PROPOSE(bid) ⇒ PROPOSE(bid)

r_9: Min_profit(mp), Utility(u), PRELIM_PROPOSE(bid), bid>u-mp ⇒ ~PROPOSE(bid)

r_{10}: Min_profit(mp), Utility(u), PRELIM_PROPOSE(bid), bid>u-mp, new_bid=u-mp
⇒ PROPOSE(new_bid)

$r_9 > r_8$

Fig. 3. Buyer's Strategy in Defeasible Logic.

Rules R1, R2, and R3 define the conditions for the acceptance or rejection of a proposal. More specifically, rule R1 states that if the current state of the negotiation is 2 (i.e. PROP_REC, when the agent has received a “propose” message) and if opponent’s offer plus the minimum profit is less or equal to half the utility, the counteroffer is accepted in all cases. R2 describes the case in which opponent’s offer plus the minimum profit is greater than its utility and the counteroffer is rejected. There is also an intermediate counteroffer area, between those two limits, whereas the agent cannot immediately accept or reject the proposal, but it should make a counteroffer (see rules R4-R10). Finally, rule R3 defines that if the current state of the negotiation is 5 (i.e. PROP_ACC, when the agent has received an “accept” message) it also sends an “accept” message.

Rules R4 through R10 define different bidding scenarios according to the step of the negotiation. There are three levels for the bidding policy: bidding of first step, bidding of last step and general bidding policy. R4 states that if the negotiation is at state 2 (PROP_REC) and at the first step, the utility divided by the *ttb* (i.e. the starting bid) is offered, when the proposal received is at an intermediate value range (see above). Rules R5, R6, R7, R8, R9, and R10 are the rules for the general bidding policy. According to R5, if the current state of the negotiation is 2 (i.e. PROP_REC, the agent has received a “propose” message) but it has not made one (i.e. previous bid is zero), it offers the utility divided by the *ttb* (i.e. the starting bid). R6 defines that if the current state of the negotiation is 2 (i.e. PROP_REC, the agent has received a “propose” message) and it has made an offer in the previous step, it increases linearly its offer, according to the following type:

$$bid = \frac{utility - previous\_bid}{2} + previous\_bid$$

R7 describes that if the current state of the negotiation is 3 (i.e. PROP_REJ, the agent has received a “reject” message) it also offers the above bid. R8 defines that if R7 or R6 is true then the computed amount for the bid is to be offered. However, R9 checks whether the bid to be offered is lower than the utility minus the minimum profit and if it is not, then R10 is fired. Rules R9 and R10 are additional checks that ensure that the offered amount of money for the product is not against the benefit of the buyer. The control and the termination condition of the negotiation process are handled by the control module and there is no rule for that purpose. However, this is in accordance to the declarative nature of our negotiation specification scheme.

Regarding the DR-DEVICE version of the negotiation strategy (Fig. 4), instead of predicates two classes are used: buyer and protocol. The former models (as object attributes) information needed for the negotiation strategy that concern solely the buyer, namely the minimum profit the buyer seeks, its utility and its *ttb*. The protocol class models information regarding the negotiation process, namely the state of the negotiation, the current step, the previous bid of the buyer and the counter-offer received from the seller. Furthermore, both classes hold additional administrative

information, such as the names of the buyer and seller agents, and a link between each buyer object with the corresponding protocol instance.

```
(defeasiblerule r1
   (buyer (min-profit ?mp) (utility ?u) (protocol ?p))
   (protocol (prot-name ?p) (sel-name ?s) (state 2)
             (counteroffer ?c&:(<= (+ ?c ?mp) (/ ?u 2))))
  =>
   (acceptable (seller ?s) (proposal ?c)))

(defeasiblerule r2
   (buyer (min-profit ?mp) (utility ?u) (protocol ?p))
   (protocol (prot-name ?p) (sel-name ?s) (state 2)
             (counteroffer ?c&:(> (+ ?c ?mp) ?u)))
  =>
   (not (acceptable (seller ?s) (proposal ?c))))

(defeasiblerule r4
   (buyer (min-profit ?mp) (utility ?u) (protocol ?p) (ttb ?ttb))
   (protocol (prot-name ?p) (sel-name ?s) (step 0) (state 2)
      (counteroffer ?c&:(and (> (+ ?c ?mp) (/ ?u 2)) (<= (+ ?c ?mp) ?u))))
  =>
   (calc (bind ?first-bid (/ ?u ?ttb)))
   (propose (seller ?s) (bid ?first-bid)))

(defeasiblerule r7
   (buyer (min-profit ?mp) (utility ?u) (protocol ?p) (ttb ?ttb))
   (protocol (prot-name ?p) (sel-name ?s) (prevbid ?prb)
          (step ?st&:(and (> ?st 0) (< ?st ?ttb))) (state 3) )
  =>
   (calc (bind ?bid (+ ?prb (/ (- ?u ?prb) 2))))
   (preliminary-propose (seller ?s) (bid ?bid)))

(defeasiblerule r9
   (declare (superior r8))
   (buyer (min-profit ?mp) (utility ?u) (protocol ?p))
   (protocol (prot-name ?p) (sel-name ?s) )
   (preliminary-propose (seller ?s) (bid ?bid&:(> ?bid (- ?u ?mp) )))
  =>
   (not (propose (seller ?s) (bid ?bid))))
```

Fig. 4. Fragment of buyer's strategy in DR-DEVICE.

6. Negotiation Trace

In this section we demonstrate the operation of the system and we scrutinize a negotiation trace between a buyer agent and a seller agent. The JADE platform provides a special-purpose agent which is called *sniffer*. Sniffer can monitor the exchanged messages of two or more agents in the agent platform. The specific parameters of the negotiation are given in the next table. We examine the trace from the buyer's viewpoint. The parameters of the negotiation are summarized in Table 2.

As we can see in Fig. 5, the buyer initially issues a "Call for Proposal" message (CFP). At this point, as it is the first time we present a trace, we analyze the structure of exchanged messages. FIPA ACL messages are built up of three layers of languages: (a) elements of the world are defined in an ontology, (b) an agent's

intention to describe or alter the world is expressed by a communicative act or speech-act such as INFORM, and (c) statements about the world are expressed by means of a Content Language. In order for agents to be able to reason about the effects of their communication, ACL messages are inserted into proper Agent Interaction Protocols that describe allowed sequences of actions among agents.

Table 2. Negotiation Parameters for Negotiation Trace.

BUYER	SELLER
Ttb=5	Tts=10
Minimum Profit = 100	Minimum Profit = 100
Utility = 1000	Maximum Profit = 800
	Bid decrement = 40
	Cost = 200

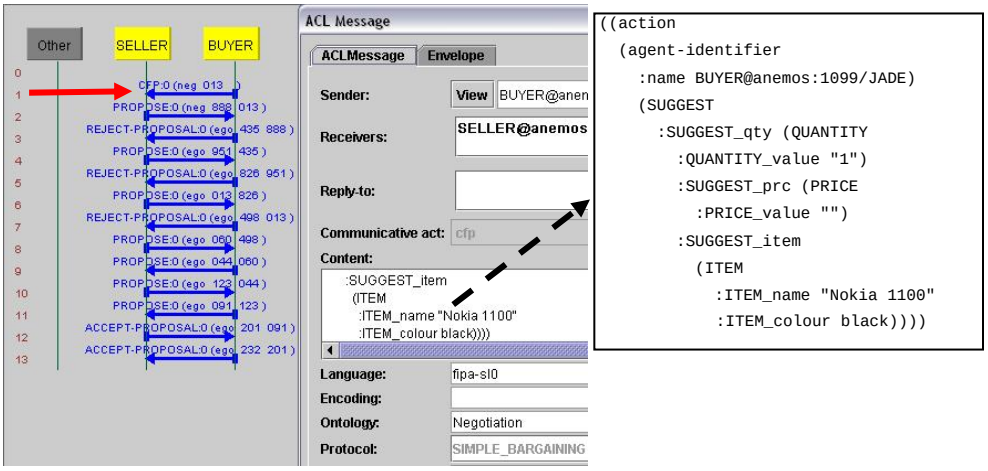

Fig. 5. Initialization of the Negotiation: Buyer issues a "Call for Proposal".

At the left-hand side of Fig. 5 one can see all the interactions between the buyer and the seller agent. We analyze the first interaction. The ACL message that corresponds to interaction 1 is depicted next to the interactions and is indicated by a solid arrow. The Communicative Act (or Speech-Act) of this ACL message is “CFP”. The Ontology, which both buyer and seller share, is called “Negotiation” and the used Interaction protocol (or Negotiation Protocol) is called “Simple-Bargaining”. The message content ontology allows agents to model facts, beliefs, allowed actions, hypotheses and predications about a domain [1]. We have developed an RDF Schema ontology for actions that agents can request and perform (e.g. suggest) during the negotiation procedure, and predicates for error messages. We have also defined a few characteristics for the negotiated products.

The content of the message is indicated by the dashed arrow in Fig. 5. According to the content of the message, the Actor of the action is Buyer who suggests a negotiation template, about a single item of a black NOKIA 1100 mobile phone. For the content language, we have used both FIPA SL0 and FIPA-RDF. All the examples

we show here use FIPA SL0, whereas Fig. 6 shows an interaction example in RDF. Either of those expressions is wrapped in an XML message, before transmission.

The seller responds with a “Propose” and the proposed amount is 1000 (Fig. 7). According to the rule R2 of the buyer’s strategy, as long as the relation c+mp > u is true, the buyer keeps rejecting the offer.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
         xmlns:fipa-rdf="http://www.fipa.org/schemas/FIPA-RDF#"
         xmlns:Negotiation="http://www.csd.uoc.gr/~dogjohn#">
   <fipa-rdf:object>
      <fipa-rdf:CONTENT_ELEMENT>
         <rdf:Description>
            <fipa-rdf:type>action</fipa-rdf:type>
            <Negotiation:actor>
               <rdf:Description>
                  <Negotiation:name>BUYER@anemos:1099/JADE</Negotiation:name>
               </rdf:Description>
            </Negotiation:actor>
            <Negotiation:action>
               <rdf:Description>
                  <fipa-rdf:type>SUGGEST</fipa-rdf:type>
                  <Negotiation:SUGGEST_prc>
                     <rdf:Description><Negotiation:PRICE_value/></rdf:Description>
                  </Negotiation:SUGGEST_prc>
                  <Negotiation:SUGGEST_qty>
                     <rdf:Description>
                        <Negotiation:QUANTITY_value>1</Negotiation:QUANTITY_value>
                     </rdf:Description>
                  </Negotiation:SUGGEST_qty>
                  <Negotiation:SUGGEST_item>
                     <rdf:Description>
                        <Negotiation:ITEM_colour>black</Negotiation:ITEM_colour>
                        <Negotiation:ITEM_name>Nokia 1100</Negotiation:ITEM_name>
                     </rdf:Description>
                  </Negotiation:SUGGEST_item>
               </rdf:Description>
            </Negotiation:action>
         </rdf:Description>
      </fipa-rdf:CONTENT_ELEMENT>
   </fipa-rdf:object>
</rdf:RDF>
```

Fig. 6. Sample message using the RDF Schema message content ontology.

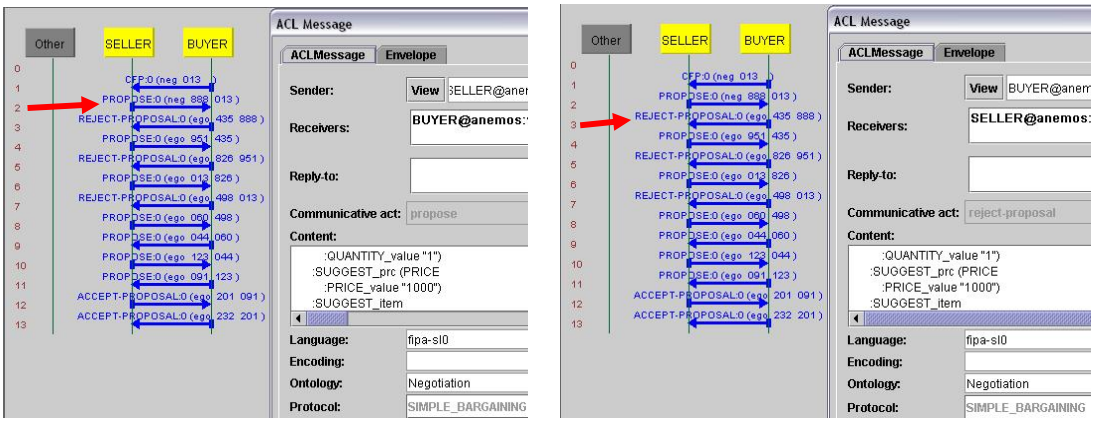

Fig. 7. Step 0 of the Negotiation: Seller proposes 1000 and Buyer rejects it.

As the buyer has rejected the seller's offer, at the next step of the negotiation, the seller decreases its offered amount by 40 (bid decrement) and waits for the response of the buyer (Fig. 8). As the buyer regards (according to its strategy) that the amount of 960 is too high, it continues to reject the offer, without issuing a counteroffer. Although the protocol allows both for a counteroffer or a rejection of a proposal, the decision lies with the agent and is expressed through the strategy.

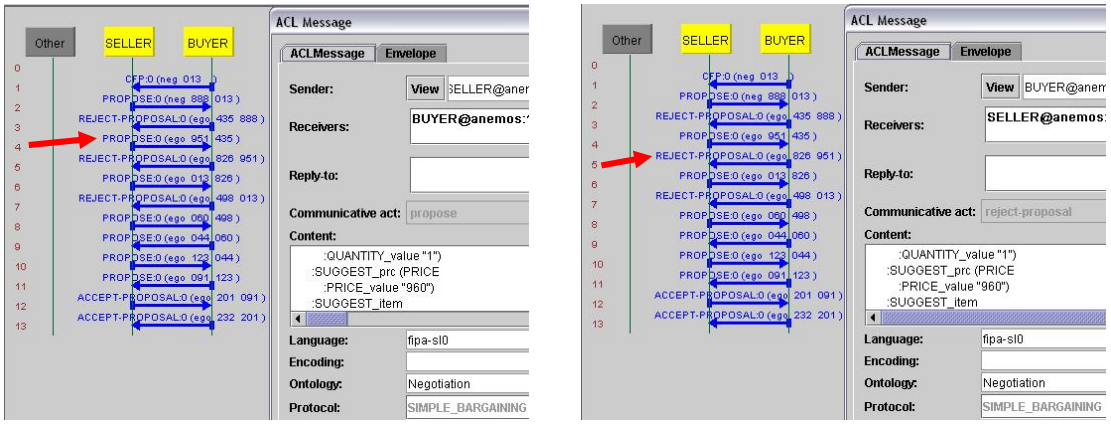

Fig. 8. Step 1 of the Negotiation: Seller proposes 960 and Buyer rejects it.

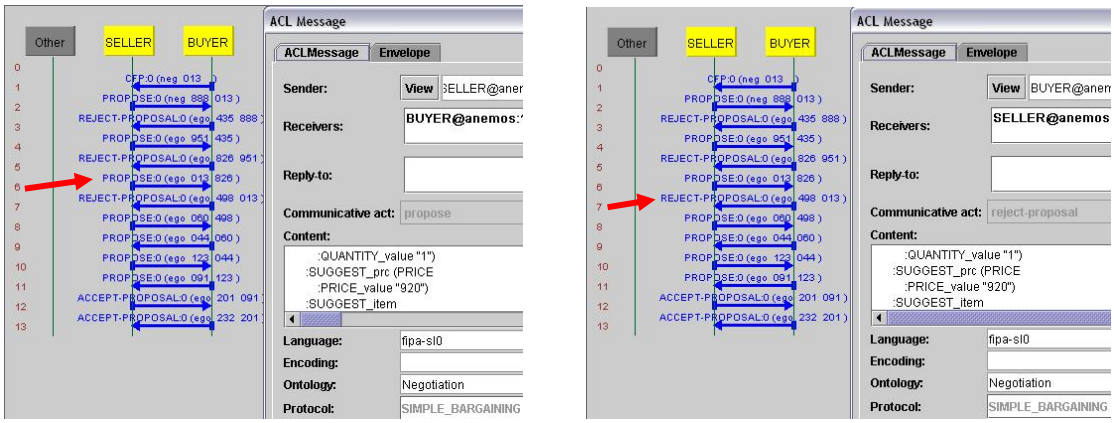

Fig. 9. Step 2 of the Negotiation: Seller proposes 920 and Buyer rejects it.

The seller decreases his offer by another 40, offering 920 (Fig. 9). The buyer still rejects seller's offer and the latter subsequently offers 880 (Fig. 10). As the relation c+mp>u $\Rightarrow$ 880+100>1000 is now false, R5 fires and the buyer offers its initial offer, which is the amount u/ttb = 1000/5.

At the next step, the seller offers 840 and waits the buyer for its response (Fig. 11). Rule R6 now fires and buyer offers the amount 600. The seller in turn issues an "Accept Proposal" (Fig. 12) message and the negotiation terminates. At this point we must notice that if seller's last message were not "Accept Proposal", the buyer's control module would issue a "Cancel" as the *ttb* would exceed 5.

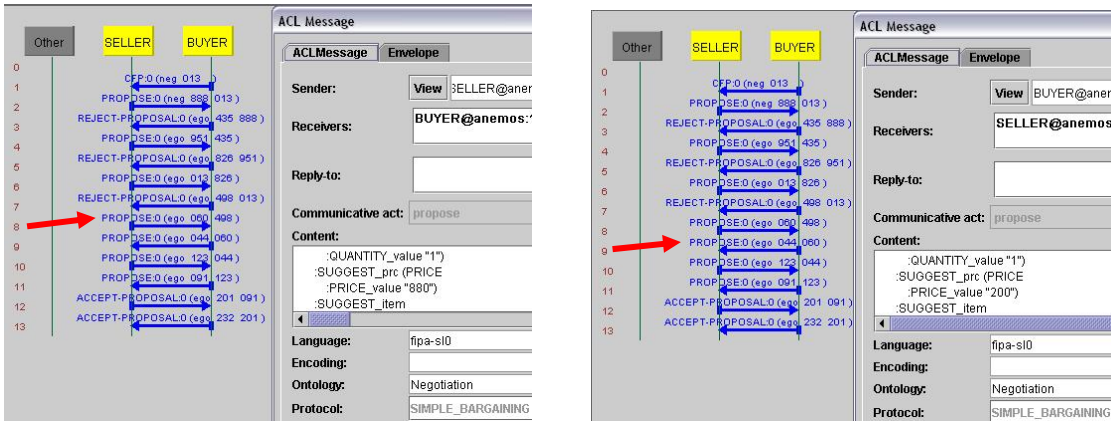

Fig. 10. Step 3 of the Negotiation: Seller proposes 880 and Buyer counter-proposes 200.

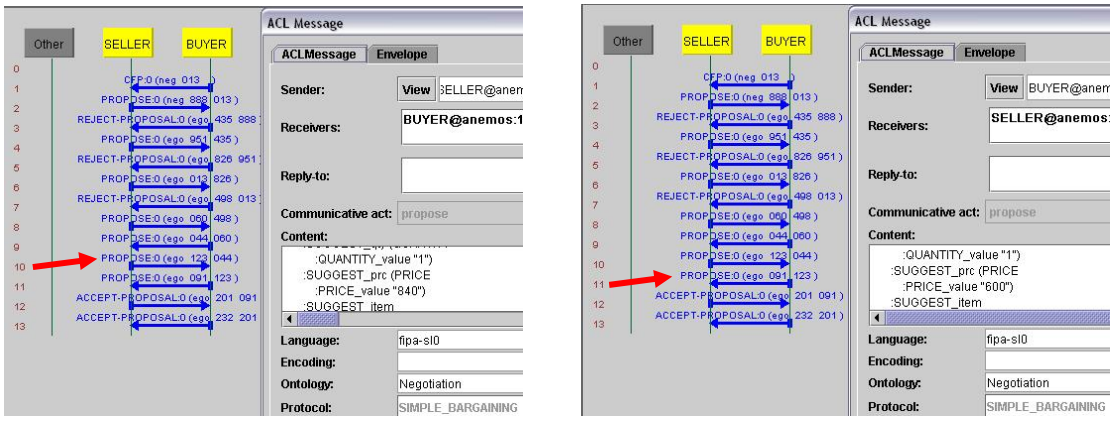

Fig. 11. Step 4 of the Negotiation: Seller proposes 840 and Buyer counter-proposes 600.

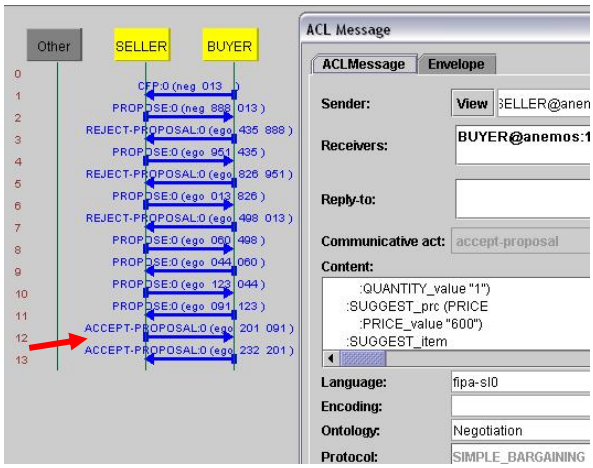

Fig. 12. Last step (5) of the Negotiation: The Seller accepts the buyer's proposal and the Buyer confirms.

7. Related Work

Kasbah [10] is an automated negotiation system. Users can create buyer or seller agents and engage in a negotiation. Each agent is provided with some information through a graphical user interface. Such information includes the date and time within which the agent must buy or sell the item, the desired or reservation price, the lowest (for sellers) or highest (for buyers) price intended to be offered and the strategy that the agent will follow. There are three predefined strategies, which alter the price in linear, quadratic or exponential manner, respectively. Negotiation in Kasbah is bilateral and competitive. Kasbah also provides a reputation mechanism, which allows the buyers or sellers to rate opponent's behavior during the negotiation.

AuctionBot [45] is an auction management system supporting the creation, location and enactment of different kinds of auctions. Users can manually interact with the system through an HTML-based interface, or alternatively, they can develop their own arbitrarily complex bidding agents, and connect them to the auction manager through a TCP/IP-level API. This API is generic enough to deal with several kinds of auctions (e.g. English, Dutch, double, etc.) through a common set of primitives. The task of defining the negotiation strategy is left to the user. Users must develop their own agents form scratch, each time they want to enter a negotiation.

[42] presents a web-based negotiation system for e-commerce, which uses heuristic techniques. It introduces an object-oriented content specification language, which is based on active object model (AOM). The language and its accompanying GUI tools are used both by sellers to advertise their products and by buyers to express their preferences. These are stored in a persistent storage. The same language is also used by clients to define proposals and counter-proposals, which are wrapped in XML. A constraint satisfaction processing component is used for the evaluation of proposals and counter-proposals. Authors adopt a declarative approach and each negotiation strategy is expressed by means of event-trigger rules (ETRs'). The negotiation protocol is an FSM of allowed sequences of actions.

Our work on DR-NEGOTIATE differs from the works above in that it uses a declarative, logical language for expressing the parties' negotiation strategies, offering thus greater expressiveness and flexibility and at the same time allowing the strategy specifications to be executed in real time, keeping low the computational complexity.

[2] presents an approach which is based on heuristic techniques for negotiation. They propose the construction of a software agent, which represents its owner and is able to search online auctions, negotiate with sellers and make purchases in an autonomous fashion. The agent not only decides in which auction to participate but also what bid to offer. Factors which the agents consider to calculate the current maximum bid at any given time are the remaining time, the number of remaining auctions, the level of desperateness, and the level of bargaining desire. These individual constraints are then combined to compose the agent's overall position, i.e. its strategy, using weights. In

addition, Dumas et al. [12] develops a probabilistic approach to an agent architecture for assembling software agents that participate in alternative heterogeneous auctions.

ContractBot [33] is an automatic contract negotiation system, which integrates the three phases of e-contracting, which are discovery, negotiation and execution. Courteous Logic Programs [18] are used to represent contracts and rules in general. A basic concept is that of a contract template. It is a declarative description of all possible outcomes with additional rules, which influence the structure of negotiation. Contract templates have two parts. The proto-contract and the negotiation level rules. The proto-contract refers to conditions of the deal such as the delivery options, payment options, guaranties, etc. It is the part of the contract that remains unchanged. The negotiation level rules, answer questions like what is to be negotiated and how. In other words, they influence the structure of the negotiation. Transactions in the auctions generate additional rules regarding buyers, sellers, prices, quantities etc., which along with the proto-contract form the final contract. The system integrates with the Michigan AuctionBot, which is an auction server. Furthermore, Grosof et al. [19] extended ContractBot to incorporate process knowledge descriptions, i.e. ontologies represented in DAML+OIL. They give a conceptual approach to specifying LP/RuleML rules on top of DL/DAML+OIL language.

e-mediator [37] is an auction management server. It is a system which uses game theoretic and constrains satisfaction techniques for negotiation. It supports combinatorial auctions, in which a bidder may place bids on combination of items and may issue simultaneous bids for many combinations. It consists of three basic components: (a) eAuctionHouse, the configurable auction server, (b) eCommitter, the leveled commitment contract optimizer, and (c) eExchangeHouse, the safe exchange planner.

The focus of the works [2], [33] and [37] is different from our work because they focus on bidding in multiple auctions. In contrast, our work focuses on 2-party negotiations where the negotiation purpose may be other than price.

Tsang and Gosling [43] define the simple constrained bargaining game where one buyer interacts with one seller. They adopt a heuristic approach, and try to find an optimal strategy, by comparing its profit against those by other strategies. The general rules are the following: The seller is constrained by a cost and the number of time units within which it must sell the product (*time to sell - tts*). The buyer is constrained by its utility and the number of time units it has to buy the product (*time to buy - ttb*). None of the participants have information about the other's cost, utility and time thresholds. The players make alternative bids with the seller to bid first. Each player bids exactly once per time unit. When both players bid for the same price, a sale is agreed. If a sale cannot be agreed before a player runs out of time, the negotiation terminates. No one has information about the other player’s past behavior or performance. Our work adopts a different approach in that the negotiation history may be taken into account, and different negotiation parameters may be set.

Rosencheim and Zlotkin [34] present rules of encounter for state-oriented domains. Their approach is based on game theoretic techniques for negotiation. In such domains, the goal of the agent is to move the world from an initial state to one or more desired states. There is interaction with other agents and limited resources. There is the possibility of cooperation, coordination, compromise and conflict. Any goal is described from the set of states that satisfy it. There are primitive operations that an agents alone can do. When these operations are combined into a coherent sequence of actions specifying what both agents are to do, we speak about a joint plan. A joint plan transforms the world in a state that may or may not satisfy both agents. When agents carry out a joint plan, each agent plays some role. Their theory assumes that there is some way of assessing the cost of each role. This measure of cost is essential to how an agent evaluates a joint plan. They use terms such as individual rational and Pareto optimal and tools from game theory to model agents' decision making model. Compared to this work, DR-NEGOTIATE offers more flexibility in defining negotiation strategies. In addition, it adopts a logic-based approach, which carries argumentation semantics [16], while [34] is based on game-theoretic methods.

Sierra et al [38] propose a formal approach for argumentation-based negotiation. They assume that a general and shared social relation is defined between agents. This relation can be modeled as a binary function, over a set of social roles. Authors assume that participants exchange locutions in a common communication language CL defined over a set of illocutionary particles, whose propositional content is expressed in a shared logical language L. CL accounts for the set of illocutionary particles, necessary to model the set of illocutionary acts. The acts can be divided in two sets: Inego corresponds to negotiation particles such as offer, request, accept etc. and Ipers, corresponds to persuasive particles such as appeal, threaten, reward. This work supports more communication acts than DR-NEGOTIATE, but does not offer a general logical language for expressing negotiation strategies.

Travel Agent Game in Agentcities (TAGA) [46] is a framework that extends and enhances the Trading Agent Competition (TAC) scenario to work in Agentcities, an open multi-agent environment, based on FIPA compliant platforms. TAGA uses the semantic web languages and tools (RDF, OWL) to: (a) specify the underlying common ontologies, (b) as a content language within the FIPA ACL messages, (c) as the base for agent knowledge bases via XSB-based reasoning tools, and (d) to describe and reason about services. TAGA extends FIPA protocols to support different types of auctions. The travel market which TAGA simulates includes service registries, service brokerage, wholesalers, peer-to-peer transactions, bilateral negotiation, etc. This provides a rich test bed for experimenting with agents and web services as well as an interesting scenario to test and challenge agent technology. TAGA operates as a continuous open game and anyone can participate and use his/her own agent strategy to compete with the others. This work adopts a logical approach similar to DR-NEGOTIATE, and addresses more aspects than our work (e.g.

brokering). On the other hand, it is tailored to a specific domain, while our system is domain-independent.

8. Conclusions and Future Work

This paper reports on a system for automated agent negotiation, based on a formal and executable approach to capture the behavior of parties involved in a negotiation. It uses the JADE agent framework, and its major distinctive feature is the use of declarative negotiation strategies. The negotiation strategies are expressed in a declarative rules language, defeasible logic, and are applied using the implemented system DR-DEVICE. The key ideas and the overall system architecture are described, and a particular negotiation case is presented in detail.

Defeasible logic seems to be a very promising solution when it comes to negotiation strategies and brokering preferences modeling. It combines all the desired characteristics of a language for data modeling. More specific, it is formal with well defined semantics, conceptual with a high potential of abstraction, comprehensive, modular and executable. It is also highly expressive as it can model a great number of diverse cases as we have seen in section 2.

We plan to extend our work in various ways.

DR-DEVICE [6] uses a specialized Clips-based syntax for the expression of rules, something which makes the rules complicated. There is a need for writing rules in a more abstract syntax. We intend to use the semantic web rule language of DR-DEVICE, which is an extension to RuleML [35].

We will seek to integrate the agent negotiation and brokering functions in one system. In [11] an agent-based architecture for brokering and negotiation is presented. As a foundation we will use the semantic brokering system of [39] which is also based on defeasible logic.

We will implement a graphical user interface for the integrated system. Someone will be able to load, using a file manager, the files which correspond to the rules of negotiation strategy and brokering preferences respectively. The user will also be able to adjust negotiation protocol parameters and monitor the progress of the brokering and negotiation procedure. To this end, we will use the graphical rule authoring tool of DR-DEVICE, which is currently under development.

Finally, we will try to explore the argumentative nature of defeasible logic [16] by building Semantic Web agents that negotiate by exchanging logical arguments [26] within the proof layer of the Semantic Web architecture [9].

Acknowledgements

This work was partially supported by the UQ Early Career grant on “A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies” and the REWERSE Network of Excellence.

References

[1] Aart C. van, Pels R., Caire G. and Bergenti F. (2002). Creating and Using Ontologies in Agent Communication. In *Proc. Workshop on Ontologies and Agent Systems at AAMAS 2002*.

[2] Anthony P., Hall W., Dung Dang V. and Jennings N.R. (2001). Autonomous Bidding Agents for Participating in Multiple On-line Auctions. In *Proc. IJCAI-2001 Workshop on E-Business and the Intelligent Web*, Seattle, WA, 2001.

[3] Antoniou G., Billington D., Governatori G. and Maher M.J. (2001). Representation results for defeasible logic. ACM Transactions on Computational Logic 2, 2 (2001): 255 - 287

[4] Antoniou G., Maher M.J. and Billington D. (2000). Defeasible Logic versus Logic Programming without Negation as Failure. *Journal of Logic Programming* 41, 1.

[5] Bartolini C., Preist C. and Jennings N.R. (2002). A Generic Software Framework for Automated Negotiation. In *Proc. 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna Italy, 2002.

[6] Bassiliades N., Antoniou G. and Vlahavas I. (2004). A Defeasible Logic Reasoner for the Semantic Web. In *Proc. 3rd International Workshop on Rules and Rule Markup Languages for the Semantic Web*, LNCS 3323, Springer 2004, 49-64.

[7] Bassiliades N., Vlahavas I., "R-DEVICE: A Deductive RDF Rule Language", 3rd Int. Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004), Springer-Verlag, LNCS 3323, pp. 65-80, Hiroshima, Japan, 2004.

[8] Bellifemine F., Caire, G., Poggi A. and Rimassa G. (2003). JADE A White Paper. *Telecom Italia EXP magazine,* Vol 3, No 3 September 2003.

[9] Berners-Lee T., Hendler J., and Lassila O. The Semantic Web. (2001). *Scientific American*, 284(5), pp. 34-43.

[10] Chavez A., Dreilinger D., Guttman R. and Maes P. (1997). A Real-life Experiment in Creating an Agent Marketplace. In *Proc. 2nd Int. Conf. on the Practical Applications of Agents and Multi-Agent Technology (PAAM)*, London, UK, 1997.

[11] Delgado J., Gallego I., Garcia R. and Gil R. (2002). An Architecture for Negotiation with Mobile Agents. In *Proc. 4th International Workshop on Mobile Agents for Telecommunication Applications*. pp. 21-32

[12] Dumas M., Aldred L., Governatori G. and Hofstede A.H.M. ter. (2005). Probabilistic Automated Bidding in Multiple Auctions. *Electronic Commerce Research*, 5(1), 23-47.

[13] Dumas M., Governatori G., Hofstede A.H.M ter and Oaks P. (2002). A Formal Approach to Negotiating Agents Development. *Electronic Commerce Research and Applications*, Vol. 1, Issue 2, Summer 2002, 193-207.

[14] FIPA Interaction Protocols Specification. http://www.fipa.org/repository/ips.php3

[15] Governatori G., Dumas M., Hoftede A.H.M. ter and Oaks P. (2001). A formal approach to protocols and strategies for (legal) negotiation. *Proc. 8th Int. Conf. on Artificial Intelligence and Law*. ACM Press. pp 168-177.

[16] Governatori G., Maher M.J., Antoniou G. and Billington D. (2004). Argumentation Semantics for Defeasible Logics. *Journal of Logic and Computation*, 14(5), 675-702.

[17] Griethuysen J.J. van, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036, 1982.

[18] Grosof B.N. (1997). Prioritized conflict handing for logic programs. In *Proc. of the 1997 International Symposium on Logic Programming*, 197-211

[19] Grosof B.N. and Poon T. (2003). SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proc. 12th International Conference on WWW*. Budapest Hungary 2003. pp. 340- 349.

[20] He M., Jennings N.R. and Leung H.-F. (2003). On Agent-Mediated Electronic Commerce. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No 4 July/August 2003.

[21] Hofstede A.H.M. ter. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, The Netherlands, 1993.

[22] IIIA, ISOCO and UPC (organisers). AMECII trading agents' tournament. http://www.iiia.csic.es/Projects/fishmarket/agents2000.

[23] JADE Project. http://jade.cselt.it/

[24] Jennings N.R., Parsons S., Sierra C., Faratin P. (2000). Automated Negotiation. In *Proc. 5th Int. Conf. on the Practical Application of Intelligent Agents and Multi- Agent Systems (PAAM-2000)*, Manchester, UK, pp. 23-30.

[25] Jennnings N.R., Faratin P., Lomuscio A.R., Parsons S., Sierra C. and Wooldridge M. (2001). Automated negotiation: Prospects, methods and challenges. *Journal of Group Decision and Negotiation* 10, 2, March 2001.

[26] Kraus S., Sycara K. and Evenchik A. (1998). Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, Vol. 104, No. 1-2, pp. 1-69.

[27] Maes P., Guttman R.H. and Moukas A.G. (1999). Agents That Buy and Sell. *Communications of the ACM*, Vol. 42, 3, 81-91.

[28] Maher M.J. (2001). Propositional Defeasible Logic has Linear Complexity. (2001). *Theory and Practice of Logic Programming* Vol. 1, No. 6, pp. 691-711.

[29] Maher M.J. (2002). A Model-Theoretic Semantics for Defeasible Logic. In *Proc. Workshop on Paraconsistent Computational Logic*, pp. 67 - 80, 2002.

[30] Maher M.J., Rock A., Antoniou G., Billington D. and Miller T. (2001). Efficient defeasible reasoning systems. *International Journal of Tools with Artificial Intelligence* 10(4), 483-501.

[31] Nute D. (1994). Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming (vol. 3): Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford University Press.

[32] Raiffa H. (1982). *The Art and Science of Negotiation*. Harvard University Press.

[33] Reeves D.M., Wellman M.P., Grosof B.N. and Chan H.Y. (2000). Automated Negotiation from Declarative Contract Descriptions. In *Proc. 17th National Conference on Artificial Intelligence, Workshop on Knowledge-Based Electronic Markets(KBEM)*, Austin, Texas, July 30–31 2000.

[34] Rosenschein J.S. and Zlotkin G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press 1994.

[35] Rule Markup Language initiative. http://www.ruleml.org/

[36] Sandholm T. (1999). Distributed rational decision making. In G. Weiss (ed.), *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*, MIT Press 1999.

[37] Sandholm T. (2002). eMediator: A Next Generation Electronic Commerce Server. *Computational Intelligence* Vol.18, No. 4, 2002.

[38] Sierra C., Jennings N.R., Noriega P. and Parsons S. (1997). A Framework for Argumentation-based Negotiation. In *Proc. 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures and Languages*. pp 177-192.

[39] Skylogiannis T., Antoniou G., Bikakis A. and Bassiliades N. (2005). DR-BROKERING: A Defeasible Logic-Based System for Semantic Brokering. In *Proc. 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05)*, pp. 414-417, Hong Kong.

[40] Strother B. (2000). TAC: A Trading Agent Competition. ACM SIGeCom Exchanges 1(1), August 2000.

[41] Su S.Y.W., Huang C. and Hammer J. (2000). A Replicable Web-based Negotiation Server for E-Commerce. In *Proc. 33rd Hawaii International Conference on System Sciences*.

[42] Su S.Y.W., Huang C., Hammer J., Huang Y., Li H., Wang L., Liu Y., Pluempitiwiriyawej C., Lee M. and Lam H. (2001). An Internet-based Negotiation Server for e-Commerce. *The VLDB Journal* 10, 72-90.

[43] Tsang E. and Gosling T. (2002). Simple Constrained Bargaining Game. In *Proc. Distributed Constrained Satisfaction Workshop*, collocated with the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Bologna Italy, 2002.

[44] Wooldridge M.J. (1999). Intelligent agents. In G. Weiss (ed.), *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*, MIT Press 1999.

[45] Wurman P.R., Wellman M.P. and Walsh W.E. (1998). The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In *Proc. 2nd International Conference on Autonomous Agents*. Minneapolis, Minnesota. pp. 301-308.

[46] Zou Y., Finin T., Ding L., Chen H. and Pan R. (2003). Using Semantic Web Technology in Multi-Agent Systems: A Case Study in the TAGA trading Agent Environment. In *Proc. 5th International Conference on Electronic Commerce*. Pittsburgh, Pennsylvania 2003. pp. 95-101.