A Multi-Agent System for Electronic Commerce including Adaptive Strategic Behaviours

Henrique Lopes Cardoso[1], Max Schaefer, Eugénio Oliveira

Faculdade de Engenharia, Universidade do Porto, NIAD&R-LIACC
Rua dos Bragas
4099 Porto Codex, Portugal
cardoso@oat.ncc.up.pt max@brooks.fe.up.pt eco@fe.up.pt

Abstract. This work is primarily based on the use of software agents for automated negotiation. We present in this paper a test-bed for agents in an electronic marketplace, through which we simulated different scenarios allowing us to evaluate different agents' negotiation behaviours. The system follows a multi-party and multi-issue negotiation approach. We tested the system by comparing the performance of agents that use multiple tactics with ones that include learning capabilities based on a specific kind of Reinforcement Learning technique. First experiments showed that the adaptive agents tend to win deals over their competitors as their experience increases.

1 Introduction

Internet and www popularity has strongly increased the importance and popularity of electronic commerce, which goes far beyond both the searching for services exposed in virtual stores and price comparison. Electronic commerce includes a negotiation process between buyers and sellers, in order to find an agreement over the price and other transaction terms. It also deals with service demands and electronic payment. Common systems that consider mainly the electronic ads, based on the creation of classified ads sites with search capabilities, can be enhanced by semi-intelligent mechanisms relying on agent technology. Such enhanced systems can be applied to the different stages of the *Consumer Buying Behaviour* model, as explained in [3][4]. In particular, when applying to the negotiation process, agents enable new types of transactions, where prices and other transaction issues need no longer to be fixed.

A typical case of negotiation process is the *auction*. Negotiation on the Internet often amounts to one party (typically the seller) presenting a take-it-or-leave-it proposal (e.g., a sale price). Auctions represent a more general approach to look for an appropriate price, admitting a range of negotiation protocols [7]. These auction-based negotiation protocols include the *English* auction, the *Dutch* auction, the *First-price Sealed Bid* and the *Vickrey* auction. These auction-based protocols are called *single-sided* mechanisms, because bidders are all buyers or all sellers, and they also

[1] The work of Henrique Lopes Cardoso has been funded by a scholarship from *Subprograma Ciência e Tecnologia do 2º Quadro Comunitário de Apoio.*

include an auctioneer. *Double-sided* auctions admit multiple buyers and sellers at once, like the *continuous double auction* [7]. There are several auction-based systems that are in use for electronic commerce.

AuctionBot is an auction server where software agents can be created to participate in several types of auctions. *Kasbah* is a web-based system where users create autonomous agents that buy and sell goods on their behalf [1]. It is based on continuous double auction mechanisms. In the *Kasbah* marketplace, buying and selling agents interact and compete simultaneously, making price bids to their counterparts. The agents have a function that changes the price they are willing to propose over time. A different negotiation approach involves negotiating over multiple terms of a transaction, unlike the typical way of auction negotiation, that only considers the price of the good. *Tete-a-Tete* is a system where agents cooperatively negotiate in this way. It intends to provide merchant differentiation through value-added services, like warranties, delivery times, etc. [4].

Following this more general approach, in [2] negotiation is defined as a process by which a joint decision is made by two or more parties. The parties first verbalise contradictory demands and then move towards agreement by a process of concession making or search for new alternatives. In that paper, a many parties, many issues negotiation model is adopted, that is, multilateral negotiations (like a continuous double auction) about a set of issues (the transaction terms). Several negotiation tactics are tested, and a negotiation strategy model is explained.

The work described in the present paper aims to combine those tactics in a dynamic way, in order to endow adaptive market agents with strategies (appropriate ways of selecting combinations of tactics) and to compare their performance with agents that are not adaptive. Since the marketplace is a dynamic environment, adaptive agents are expected to benefit from changing conditions, therefore taking advantage over others. A test-bed has been created to provide the interaction process between market agents with different capabilities.

Section 2 describes our multi-agent platform for electronic commerce (the test-bed), including the negotiation assumptions, model and protocols adopted (which have been based on those introduced in [2]). Section 3 gives details on negotiation tactics (also adopted from [2]) which can be combined to either user-defined strategies or strategies based on Reinforcement Learning techniques. Section 4 focuses on several different scenarios and describes those situations that were chosen for testing purposes. Conclusions drawn from the testing scenarios are presented in section 5. We conclude the paper, in section 6, by presenting some topics of our future work.

2 A System for Electronic Commerce

In this section we describe the basic negotiation approach and the architecture of the SMACE (***S****istema* ***M****ulti-****A****gente para* ***C****omércio* ***E****lectrónico*) system. It is a multi-agent system for electronic commerce, where users can create buyer and seller agents that negotiate autonomously, in order to make deals on services they are requesting or offering. SMACE has been used as a test-bed for different negotiation paradigms, both user-controlled and self-adaptive.

2.1 Negotiation Model

The negotiation model we have adopted is multilateral and based on many issues (that is, multidimensional), as described in [2]. Multilateral refers to the ability that each buying or selling agent has to negotiate simultaneously with many other selling or buying agents. In auction terms, it relates to a sealed-bid continuous double auction, where both buyers and sellers submit bids (proposals) simultaneously and trading does not stop as each auction is concluded (as each deal is made). In practical terms, this multilateral negotiation model works as many bilateral negotiations that can influence each other. The bilateral negotiation model we are using is based on the one defined in [5]. In our work, negotiation is realised by exchanging proposals between agents. The negotiation can be made over a set of issues instead of the single issue price found in most auctions. A proposal consists of a value for each of these issues and is autonomously generated and proposed by one agent.

This negotiation model is also called a service-oriented negotiation model [2], since it involves two roles that are, in principle, in conflict: sellers of services and buyers of services. A service is something that can be provided by an agent and requested by some other agent, and over which they can negotiate. As a general rule, we can say that opponents in the negotiation process have opposing interests over all issues under negotiation. However it could happen that both agents have similar interests on a specific issue, thus proposal evolution may proceed in the same direction.

The sequence of proposals and counter-proposals in a two-party negotiation is referred to as a *negotiation thread*. These proposals and counter-proposals are generated by linear combinations of functions, called *tactics*. Tactics use a certain criteria (time, resources, etc.) in generating a proposal for a given negotiation issue. Different weights can be assigned to each tactic used, representing the importance of each criterion for the decision making. Agents may wish to change their ratings of criteria importance over time. To do so, they use a *strategy*, defined as the way in which an agent changes the relative weights of the different tactics over time.

For each issue $j \in \{1, \ldots, n\}$ under negotiation, each agent has a range of acceptable values $[min^i_j, max^i_j]$, and a scoring function $V^i_j: [min^i_j, max^i_j] \rightarrow [0, 1]$, that gives the score an agent i assigns to a value of issue j in the range of its acceptable values. The higher the score, the better the agent's utility. Agents also assign a weight w^i_j to each negotiation issue that represents its relative importance. Assuming normalised weights ($\sum_j w^i_j = 1$), the agent's scoring function for a given proposal $x = (x_1, \ldots, x_n)$ combines the scores of the different issues in the multidimensional space defined by the issues' value ranges: $V^i(x) = \sum_j w^i_j \, V^i_j(x_j)$. The overall proposal evaluates to a score of zero if any of the issues' values is outside its range.

2.2 Negotiation Protocol

We assume for the negotiation protocol that message delivery is reliable and message delays need not to be considered (because they are presumably short).

At a particular point in time each agent has an objective that specifies its intention to buy or sell a specific service. That objective has to be achieved in a certain amount of time, specified by a deadline. Negotiation stops when this deadline is reached.

A bilateral negotiation starts after the two parties - the buyer and the seller – meet in the marketplace and match their objectives (i.e., they agree that what the buyer wants to buy is what the seller intends to sell). Once this is achieved, a negotiation thread $x_{b \leftrightarrow s}$ becomes active between the two agents. It will stay active while the agents negotiate with each other.

The agents will then exchange a sequence of proposals and counter-proposals. When receiving a *proposal*, an agent will generate a counter-proposal. Both - the received proposal and the generated counter-proposal - will be evaluated using the scoring function described above, and the agent will answer with one of three possible ways:

- *withdraw* from negotiation if the deadline was reached, or if a deal was made with some other agent;
- *accept* the proposal received, if it scores higher than the one generated;
- otherwise, send the generated *proposal*.

When an agent *b* receives an *accept* message from agent *a*, it will respond with a *deal* confirmation or rejection. Since there can be virtually any number of agents in the marketplace, this solves the problem that could rise if an agent receives two simultaneous *accept* messages from two different agents (we are assuming that the agent's objective only admits trading one unit of a service at a time). Therefore, if agent *b* did not commit with any other agent, it will close the deal with agent *a*, the sender of the accept message. This agent will wait until it gets an answer from agent *b*. A deadlock problem could rise if a group of agents was waiting for deal confirmations in a closed loop configuration. We address this problem by making agent *a*, which is expecting a deal confirmation, withdraw any other negotiations and reject any deal with any other agent. This means that eventually it will loose potential deals with other agents, if it receives a rejection from agent *b*. Since we are assuming short message delays, the problem of lost deals is not likely to occur often, and we ignore it.

2.3 SMACE

SMACE allows users to create buyer and seller agents that negotiate under the model and protocol described above. The system was implemented with the *JDK1.1.4 API* [9], and uses the *JATLite* [8] package to easily build agents that exchange *KQML* [10] messages. The agents communicate with each other in the *MarketPlace*, which is an enhanced *JATLite router*, facilitating the message routing between the agents and working as an information centre for the agents to announce themselves and search for contacts.

The SMACE API consists of three layers built on top of the *JATLite* packages. These layers also consist of packages, and using the SMACE system can take place at any of them:

- *Infrastructure* – this layer consists of two fundamental parts:
  - *MarketAgent*: a template for the creation of market agents. It already has implemented the model of negotiation and its associated protocol. The only task left to the user starting in this layer is providing his own negotiation tactics;
  - *MarketPlace*: the application that represents the marketplace, as a space where the agents meet and trade. It includes message routing and agent brokering facilities.
- *Plug&Trade* – this layer includes predefined market agents that can also be seen as examples of how an agent can be built using the *MarketAgent* template:
  - *MultipleTacticAgent (MTA)*: a market agent that is able to use a weighted combination of three tactics, one from each of the tactic families described in the next section, to generate its negotiation proposals.
  - *AdaptiveBehaviourAgent (ABA)*: a market agent that is able to weight the several tactics that it is using in an adaptive way, using Reinforcement Learning techniques.
- *UserInterface* – this layer consists of an application that provides both an *HTML* user interface for the creation and monitoring of *Plug&Trade* market agents operation and their persistence.

While accepting agents from anywhere to enter the marketplace and trade (provided that they use the same negotiation protocol), SMACE allows the user to launch predefined agents (both of *MTA* and *ABA* types) by adjusting its parameters. In order to do so, one can use the SMACE user interface. Through this interface the agents' activities can also be monitored and its parameters setting can be changed as well. Furthermore, the user may create his own agent, with his own tactics, in any programming language or platform he wishes. The SMACE API Infrastructure package assists agent building in Java. This package allows the user not to worry about communication and negotiation protocol details, spending his efforts on building his own negotiation strategy, that is to say, the agent's deliberative knowledge.

2.3.1 Agent Matching

As mentioned before, market agents contact the *MarketPlace* to search for agents that have complementary objectives, i.e., objectives including opposite agents' intentions over the same service.

To facilitate the matching of services, these ones are described using what we call descriptive issues, i.e., descriptor/value pairs that all together define the object of negotiation. Then, values for the same descriptive issues for different objectives are compared, and if the agents agree that they are "talking" about the same service they will negotiate over it. The SMACE system can be easily configured to specify the descriptive issues that market agents will use to describe their own services.

2.3.2 Negotiation Issues

In order to negotiate properly, besides using the same communication and negotiation protocols, market agents should agree in what issues the negotiation will be about.

The SMACE system can be easily configured to consider any number of negotiation issues. All the market agents created in the system will then use the same set of issues. These issues are all considered as uniform, in the sense that, for the market agents, they do not have a semantic attached. Therefore, each market agent will define a weight, a range of acceptable values and a scoring function for each one of the used issues. Opposite intentions (buying and selling) usually imply somehow contrasting scoring functions, e.g., the scoring function for the issue price is decreasing for buyers and increasing for sellers as the price increases.

3 Tactics and Strategies

One of the main aims of our work is to compare the performance of agents using strategies based on dynamic adaptive behaviours with those based on more conventional and less dynamic, or even static, behaviours.

The goal of negotiation is maximising the utility gained in a transaction, and in order to do so the focus is on how to prepare appropriate proposals as well as counter-proposals.

The predefined SMACE market agents use a specific tactic or a combination of several tactics to generate proposals. We focused on tactics that we adopted from [2]:

- Time-dependent tactics: agents vary their proposals as the deadline approaches. These tactics use a function depending on time that can be parameterised.
- Resource-dependent tactics: agents vary their proposals based on the quantity of available resources. These tactics are similar to the time-dependent ones, except that the domain of the function used is the quantity of a resource other than time. This is done either by making the deadline dynamic or by making the function depend on an estimation of the amount of the resource.
- Behaviour-dependent tactics: agents try to imitate the behaviour of their opponents in some degree. Different types of imitation can be performed, based on the opponent's negotiation policy over a sequence of his proposals: proportional imitation, absolute imitation and averaged proportional imitation.

3.1 Time-dependent Tactics

Time-dependent tactics vary the proposals as the deadline approaches. An agent *a* has to find a deal until t^a_{max} . A proposal *x* for issue *j* from agent *a* to agent *b* at time *t*, $0 \le t \le t^a_{max}$, can be calculated as follows:

$$x^t_{a\to b}[j] = \begin{cases} \min^a_j + \alpha^a_j(t)(\max^a_j - \min^a_j), \text{if } V^a_j \text{ is decreasing} \\ \min^a_j + (1-\alpha^a_j(t))(\max^a_j - \min^a_j), \text{if } V^a_j \text{ is increasing} \end{cases} \quad (1)$$

where V^a_j is the scoring function whose gradient reflects the agent's intention (as referred in subsection 2.3.2).

Any $\alpha^a_j(t)$ function defining the time-dependent behaviour must satisfy these constraints: $0 \le \alpha^a_j(t) \le 1$ (offers are inside the range); $\alpha^a_j(0) = \kappa^a_j$ (κ^a_j adjusts the initial value at initial time); $\alpha^a_j(t^a_{max}) = 1$ (the reservation value – the smallest result of the scoring function V^a_j – will be offered at the deadline).

In order to satisfy these constraints two classes of functions are presented:

- Polynomial:

$$\alpha^a_j(t) = \kappa^a_j + (1 - \kappa^a_j)\left(\frac{\min(t, t_{\max})}{t_{\max}}\right)^{\frac{1}{\beta}} \tag{2}$$

- Exponential:

$$\alpha^a_j(t) = e^{\left(1 - \frac{\min(t, t_{\max})}{t_{\max}}\right)^{\beta} \ln \kappa^a_j} \tag{3}$$

The parameter $\beta \in \Re^+$ is used to adjust the convexity degree of the curve, allowing the creation of an infinite number of possible tactics. For values of $\beta < 1$ the behaviour of both classes of functions can be described as *boulware*, i.e., concessions are made close to the deadline, otherwise the proposals are only slightly changed. With $\beta > 1$ the behaviour is called *conceder*. An agent prepared like this urges to make a deal and reaches its reservation value quickly.

3.2 Resource-dependent Tactics

Resource-dependent tactics vary the proposals based on the quantity of a resource available. *Boulware* behaviour, used in the presence of a large amount of resources, should change to *conceder* behaviour when resources run short.

3.2.1 Dynamic-deadline Tactics

This tactic sub-family varies the agent's deadline according to the availability of a particular resource. The resource modelled here is the number of agents that are negotiating and the average length of the active negotiation threads. If a selling agent *a* notices many interested parties for its good then there is no need to urge for an agreement. The set of agents negotiating with agent *a* at time *t* is

$$N^a(t) = \left\{ i \,\middle|\, x^t_{i \leftrightarrow a} \text{ is active} \right\} \tag{4}$$

A dynamic deadline using the resource described above is

$$t^a_{\max} = \mu^a \frac{\left|N^a(t)\right|^2}{\sum_i \left|x^t_{i \leftrightarrow a}\right|} \tag{5}$$

where μ^a is the time agent *a* assumes to be needed to negotiate with an opponent and $|x^t_{i\leftrightarrow a}|$ is the length of the negotiation thread between agent *a* and agent *i*.

3.2.2 Resource-estimation Tactics

Resource estimation tactics measure the quantity of a resource at a time *t*. Function α can be used to model this:

$$\alpha^a_j(t) = \kappa^a_j + (1 - \kappa^a_j)e^{-resource(t)} \tag{6}$$

The function *resource* is here used to evaluate the amount of available resources at time *t*. The following examples model

- interested parties: $resource(t) = \left|N^a(t)\right|$
- interested parties and negotiation threads' length: $resource(t) = \mu^a \dfrac{\left|N^a(t)\right|^2}{\sum_i \left|x^t_{i\leftrightarrow a}\right|}$
- time: $resource(t) = \max(0, t_{\max} - t)$

3.3 Behaviour-dependent Tactics

Behaviour-dependent tactics try to imitate the behaviour of the agent's opponents up to a certain extent. This can be useful once opponents will not be able to exploit the agent's strategy. Tactics of this family make counter-proposals influenced by the opponent's former actions. Following there are three different ways of using imitating behaviours, assuming the negotiation thread $\{\ldots, x^{t_{n-2\delta}}_{b\to a}, x^{t_{n-2\delta+1}}_{a\to b}, x^{t_{n-2\delta+2}}_{b\to a}, \ldots, x^{t_{n-2}}_{b\to a}, x^{t_{n-1}}_{a\to b}, x^{t_n}_{b\to a}\}$ and $\delta \geq 1$.

3.3.1 Relative Tit-For-Tat

These tactics imitate proportionally an opponent's behaviour $\delta \geq 1$ steps ago. The length of the negotiation thread must be $n > 2\delta$. The generated counter-proposal is:

$$x^{t_{n+1}}_{a\to b}[\mathrm{j}] = \min(\max(\frac{x^{t_{n-2\delta}}_{b\to a}[\mathrm{j}]}{x^{t_{n-2\delta+2}}_{b\to a}[\mathrm{j}]} x^{t_{n-1}}_{a\to b}[\mathrm{j}], \min^{\mathrm{a}}_{\mathrm{j}}), \max^a_j) \tag{7}$$

The counter-proposal generated is calculated with the last proposal ($x^{t_{n-1}}_{a\to b}[j]$) and the proportional evolution of two consecutive proposals from the opponent ($x^{t_{n-2\delta}}_{b\to a}[j]$ and $x^{t_{n-2\delta+2}}_{b\to a}[j]$).

3.3.2 Random Absolute Tit-For-Tat

These tactics imitate in absolute terms the opponent's behaviour. They require the existence of a negotiation thread length of $n > 2\delta$. The resulting counter-proposal is:

$$x^{t_{n+1}}_{a\to b}[j] = \min(\max(x^{t_{n-1}}_{a\to b}[j] + (x^{t_{n-2\delta}}_{b\to a}[j] - x^{t_{n-2\delta+2}}_{b\to a}[j]) + (-1)^{s} R(M), \min^{a}_{j}), \max^{a}_{j}) \qquad \textbf{(8)}$$

where $s = \begin{cases} 0, V^{a}_{j} \text{ decreaing} \\ 1, V^{a}_{j} \text{ increaing} \end{cases}$

Parameter *R(M)* provides a way to overcome function's local minima. The function *R(M)* returns a random integer in the space *[0, M]*, where *M* is the threshold of imitative behaviour.

3.3.3 Averaged Tit-For-Tat

These tactics imitate proportionally by calculating the average evolution of a certain number of proposals to the last proposal. The parameter γ refers to the number of past proposals that are considered. The counter-proposal obtained is:

$$x^{t_{n+1}}_{a\to b}[j] = \min(\max(\frac{x^{t_{n-2\gamma}}_{b\to a}[j]}{x^{t_{n}}_{b\to a}[j]} x^{t_{n-1}}_{a\to b}[j], \min^{a}_{j}), \max^{a}_{j}) \qquad \textbf{(9)}$$

for $n > 2\gamma$. The behaviour of averaged Tit-For-Tat when choosing $\gamma = 1$ is similar to relative Tit-For-Tat with $\delta = 1$.

3.4 User-defined Strategies

Once an agent is created and correctly initiated, it can be activated in order to contact the marketplace, thus starting a new episode in its life. Within an episode, the objective can not be changed. The episode ends when the agent is deactivated.

As defined in section 2.1, a strategy can be realised as the way weighted combinations of the former exposed tactics are selected. An agent's strategy determines which combination of tactics should be used at any particular instant within an episode. The simplest strategy is to use the same combination at any time in that episode.

The SMACE *UserInterface* allows the user to create a *MTA*, which combines three different tactics – one from each family described before. The sub-tactics are selected by fixing the parameters in the corresponding tactic family. Along with these parameters, the weight combination remains the same, unless the user explicitly provides different values for it. This means that a fixed weighted combination of tactics is always in use. The user can interact with the *UserInterface* to observe agent actions and change its parameters any time he wants, implementing by this way his own strategy.

In order to make it possible to design and include new agent's learning capabilities or for the user to reuse a successful agent, *MarketAgents* do not terminate after an episode. They can be reactivated again with possibly different objectives and, if desired, with different issues and tactics parameters.

3.5 Adaptive Behaviour Based Strategies

There are several approaches how adaptive behaviour could be achieved in the environment described. First let us precise the expression “adaptive behaviour”. SMACE provides a dynamic environment, i.e., various unknown agents are able to meet and negotiate for an interval of time and achieve some result by following a certain strategy. The strategy of an *MTA* is simple: it uses the weighted combination of tactics that the user supplies. Anyway it should not be necessary to supervise autonomous agents all the time. Once a user has specified the parameters for his agent’s tactics and also specified the combination of tactics by adjusting the respective relative weights, the agent will behave accordingly regardless the situation that it is facing at each particular movement. The tactics provide a means to adapt, in a certain range, to different situations considering certain resources as described in previous paragraphs. The initial proposal also plays an important role in negotiation. How to choose this initial value could also be learned by experience. Using different weighted combinations of tactics along the time in order to match the optimal one in each situation could enable an agent to have an adaptive behaviour and making better deals. However it can not be evaluated which combination of tactics ensures the most success. The space of possible combinations of interrelated variables and situations is indeed too large.

Using the *MarketAgent* template from the SMACE API described in section 2.3, a learning algorithm was included in a new agent – *AdaptiveBehaviourAgent (ABA)* - to provide a way to find out what is the best weighted combination of tactics in any situation for any issue. In the following we are referring to the approach of a matrix with weights per tactic and issue for a proposal [2]. It is obvious that changing the weights that are responsible for the importance of each tactic, depending on situations, leads to an adaptive mechanism that provides an agent with more autonomy to react appropriately depending on both his sensor input and mental state. A matrix of these weights per issue for a proposal from agent *a* to agent *b* at time *t* looks like the following:

$$\Gamma^{t}_{a\to b} = \begin{pmatrix} \omega_{11}\omega_{12}\ldots\omega_{1m} \\ \omega_{21}\omega_{22}\ldots\omega_{2m} \\ . \\ . \\ . \\ \omega_{p1}\omega_{p1}\ldots\omega_{pm} \end{pmatrix} \qquad \textbf{(10)}$$

where ω_{ij} is issue *i*’s weight of tactic *j*.

From the tactics explained before, we may configure many combinations of 10 different tactics per issue per proposal, only by varying the following parameters:

- Time-dependent function: polynomial, exponential
- Resource-dependent tactic: dynamic deadline, resource-estimation
- resource: agents, agents/negotiation threads, time
- Behaviour-dependent tactic: relative Tit-For-Tat, random absolute Tit-For-Tat, averaged Tit-For-Tat

Let there be a vector of 10 weights for these tactics for an issue *j*: $X_j = (x_1, x_2, ..., x_{10})$. Provided that each $x_n \in [0.0,0.1,0.2,\ldots,1.0] \wedge \sum_{n=1}^{10} x_n = 1$ there are 92378 different weighted combinations of the 10 tactics, per issue, considered that the other parameters (First proposal constant, Convexity degree, Negotiation time with single agent, Step and Window Size) are fixed. However, applying the restrictions of an *MTA* that is allowed to combine only 3 tactics, under the same constraints, for the *ABA* it turns out that there are only 66 different weighted combinations left. That seems to be an acceptable amount of tactic combinations to analyse, in order to find out the best one in each situation. The choice to specify the other tactic's parameters is left to the user as well as the selection of the 3 tactics, to be applied separately per issue. Then, it is up to the adaptive mechanism to adjust the weights for the specified tactics.

3.5.1 Applying a Reinforcement Learning Algorithm

In order to search for the best weighted combination of tactics in each situation, Reinforcement Learning seems to be promising as it learns online. In contrast, Supervised Learning is learning offline from examples, provided in a training set. It is not adequate to our domain, since in interacting problems – like negotiating in a dynamic environment – it is often impractical to obtain representative examples of desired behaviours. *RL* enables agents to learn from their own experiences. A kind of *RL*, Q-learning [6], selects an action in a certain state depending on the ranking of that state/action pair. In our case actions are vectors of weights (weighted combinations of tactics). These are referred to as actions in the rest of this paper. The Q-learning is guaranteed to converge to the optimal combinations of state/action pairs after each action has been tried sufficiently often. That seems to be feasible with 66 possible actions per issue. The ranking is due to rewards by matching actions to certain states. Our adaptive agent using *RL* algorithm Q-learning in the SMACE framework executes roughly the following steps per issue:

- *determines the current state per issue*
- *chooses a weighted combination of tactics by selecting one from the available 66*
- *uses this weighted combination of tactics for the next proposal*
- *observes the resulting rewards*

It is obvious that the success of this process mainly depends on how the states are characterised. We therefore select some general settings that are supposed to stay unchanged during an episode and some others to identify states within an episode. We have assumed that what is important to learn is the optimal tactic weights for an episode's static settings. These are the chosen tactics with their parameters, the intention (sell, buy) and the negotiation issue that all states within an episode have in common. This information is referred to as the *FrameConditions*. To distinguish states within an episode we then chose the number of trading partners

(*QuantityOfActiveThreads*); the percentage of time left to the deadline (*PercentageOfTimeLeft*); if negotiating is actually established, the agent is waiting for opponents, it has already reached the deadline or made a deal (*StateFlag*). Thus a state can be represented as follows:

$$State: \begin{pmatrix} StateFlag \\ QuantityOfActiveThreads \\ PercentageOfTimeLeft \\ FrameConditions \end{pmatrix} \quad FrameConditions: \begin{pmatrix} Issue \\ Intention \\ TimeDependentTacticParameters \\ ResourceDependentTacticParameters \\ BehaviourDependentTacticParameters \end{pmatrix}$$

Now, the rewarding mechanism is chosen as follows. While making transitions between states in which the agent is negotiating the reward is always zero for each issue's weighted combination of tactics. That is neutral respectively to the user-specified *conceder* or *boulware* behaviour. Giving a negative reward would force the agent to need fewer proposals to reach a deal. But that would be unlike a predominantly *boulware* behaviour. Changing to a state that is indicated as a deal state, i.e., where the agent got an agreement, is rewarded with the agent's own utility per issue. The circumstances of the deal, i.e., the values on which the agents agreed, influence the rewards. Different rewards honour those actions that increase the utility of an agent for that issue. Communication costs are here not considered. To calculate the utility per issue, the scoring function from each negotiable issue is used. In case of an agreement at the reservation value the utility is zero. The maximum utility is reached for the best possible score of the issue. Loosing a deal because of any reason indicates a bad weighted combination of tactics. Thus making a transition from a negotiating state to one where either the deadline was reached or the opponent decided to stop negotiating is punished, that is, rewarded with a negative value. This allows distinguishing this situation from deals at reservation values. The penalty mechanism also considers that unexplored actions might be better than those that have been punished.

The Q-learning chooses, for each issue, the highest scored weighted combination of tactics. However, as the environment is dynamic, the same action may not lead to a desired result when applied to the same state. As a trade-off between exploitation of already considered good actions and exploration of yet unknown ones (or considered in the past inferior ones), Q-learning selects with a certain probability a non-greedy action. This can be achieved, for instance, with a action selection mechanism that uses either

1. a small probability ε of choosing uniformly a non-greedy action (*Epsilon-Greedy* [6]) or
2. a given degree of exploration T for choosing between non-greedy actions, while considering their ranking (this is called the *Softmax* [6] approach).

4 Experimental Scenarios

In a dynamic environment as the one provided by SMACE there could be an infinite number of scenarios, depending on the existing market agents and their objectives and negotiation settings. In this section we describe some situations that can prevent negotiation from being started or deals from being made. We provide a fixed scenario that avoids those situations in order to successfully test our market agents.

4.1 Problematic Scenarios

As all the agent's negotiation settings are private, agents start negotiations without knowing if a deal is possible [2]. This private information includes their negotiation tactics, as well as their negotiation issues parameters – weights, value ranges and scoring functions.

One problematic scenario involves agents with non-matching value ranges. Since proposals outside the range of acceptable values are always rejected, a deal in this situation is impossible, but as the negotiation process proceeds, communication costs are unnecessarily caused. Even when the value ranges overlap, it is not certain that the result of a negotiation will be a deal. That depends on a number of factors related with the agents' proposal generation process, including the first proposal value. This first value may limit the range of proposed values to a sub-range of the acceptable values. Moreover, since there are no partial agreements, proposal values, that are already acceptable, could run out of its value range while other issues under negotiation converge to an acceptable value. Acceptable values still do not lead to a deal if the agent is looking for better scored values.

4.2 Scenarios Features

In this subsection we describe some assumptions related to the scenario where the first evaluations of our market agents are done.

First, since we are interested on effective negotiation testing, we provide all the agents with the same service offered/requested. This, together with appropriate deadlines, ensures that they will start negotiating. Of course, we provide the marketplace with buyer and seller agents.

Following the considerations discussed in section 2.3.2, all the agents will negotiate, for simplicity, over the single issue *price*. Since the *ABAs* learn independently per issue, this choice does not affect the results.

In order to avoid the problem discussed in the previous subsection, we state that deals are always possible, that is, agents' value ranges match totally or are subsets.

We assume that agents with opposite intentions have concurrent scoring functions (as mentioned in section 2.1).

To evaluate the possible success of the *ABA* model over the *MTA* model, we have chosen two different possible agent configurations. In these, *MTA* agents will use the same tactic configuration in all episodes where they are used.

- *Scenario 1*: one *ABA* trading with one *MTA* (price value range of *[1, 10]* for both).

→ We expected the *ABA* to increase its utility after a number of episodes, so we could confirm the *ABA*'s adaptation process.

- *Scenario 2*: one *ABA* (*[3, 10]*) offering a service to one *MTA* (*[1, 10]*) and competing with another *MTA* (*[1, 5]*). The value ranges were configured like that to make it likely for the *MTAs* to agree faster than the *ABA*, in order to stimulate the *ABA*'s adaptation process.

$\rightarrow$ We expected the *ABA* to win deals with its best possible utility, after a number of episodes.

For all *MTAs*, the tactic configuration was set to a single polynomial time-dependent tactic, with a convexity degree parameter of one ($\beta = 1$) and a first proposal constant of zero ($\kappa = 0$), in order to use the whole value range. This configuration represents the simplest case of a *MTA* for the given value range, because, when generating proposals, it only considers its time left.

The *ABAs* always used a combination of three tactics, all from the time-dependent family. The tactics' settings were similar to that used in the *MTAs*, except that different convexity degree parameters were supplied, in order to enable three different behaviours: *boulware* ($\beta = 0.01$), *linear* ($\beta = 1$) and *conceder* ($\beta = 100$). We chose a combination of three time-dependent tactics to make it easier to provide those three behaviours. These are needed so that the *ABAs* are able to increase their utilities, by trying different weighted combinations inside that behaviour range.

Furthermore, in order to reduce the *ABAs*' learning state/action space, we reduced the number of possible actions, by limiting the available weights.

To consider the trade-off between exploration and exploitation, actions were selected by using the *Softmax* algorithm and a degree of exploration $T = 0.1$. The parameters for the Q-learning algorithm were set to 0.1 for the learning rate and 0.9 for the discount factor.

5 Conclusions

First experimental results suggest that the *ABA*'s learning capabilities were sufficient in order to beat its competitor (scenario 2). We observed on a 150 episodes experiment that the rough tendency was for the *ABA* to significantly increase the number of deals won over its competitor. However, the *ABA* was not successful in improving its utility on the deals it made. We believe this is due to the rewarding mechanism applied, which ranks the fact of getting a deal higher than the distinction between deal utilities. Furthermore, the exploration rate was not sufficient for the agent to try different tactic combinations in the episodes tested. The same difficulty appeared in scenario 1, where the *ABA* agent did not tend to increase the utility of its deals. This was also due to the fact that the use of *conceder* tactic combinations was preferred, since they lead faster to a deal and so their Q-values are increased in early stages of the adaptation process.

Further experiments are needed to reason the observations described above. Also, by limiting to time-dependent tactics, we introduced a limitation to the *ABA*'s behaviour that may have prevented it from a better performance. Learning how to use the convexity degree parameter (β) might prove to be a more efficient way of learning how to use a time-dependent behaviour, since it can make it easier to reduce the state/action space involved.

6 Future Work

In the present paper, we described negotiation behaviours using independent weighted combinations of tactics for each one of the negotiation issues. We intend to further investigate on weighted combinations of tactics that consider correlation between those issues. We believe that agents might benefit from calculating several different issue values, for a proposal, that influence each other.

Another aspect of future work relates to the exploration of other features of the Q-learning, as well as to the application and comparison of different learning algorithms (other kinds of Reinforcement Learning and Genetic Algorithms) that may perform better in a dynamic market environment. We are also interested in testing agents that become somehow specialized in a specific tactic, by optimizing only the parameter values of that tactic to be used in a specific situation.

References

1. A. Chavez and P. Maes. Kasbah: An Agent Marketplace for Buying and Selling Goods. Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96). London, UK, April 1996.
2. P. Faratin, C. Sierra, and N. R. Jennings: Negotiation Decision Functions for Autonomous Agents. Int. Journal of Robotics and Autonomous Systems 24 (3-4) 159-182. 1998.
3. R. Guttman and P. Maes: Agent-mediated Integrative Negotiation for Retail Electronic Commerce. Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET'98). May 1998.
4. R. Guttman, A. Moukas, and P. Maes: Agent-mediated Electronic Commerce: A Survey. Knowledge Engineering Review, June 1998.
5. H. Raiffa: The Art and Science of Negotiation. Harvard University Press, Cambridge, USA, 1982.
6. R. Sutton, and A. Barto: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998.
7. PR Wurman, MP Wellman, and WE Walsh: The Michigan Internet AuctionBot: A configurable auction server for human and software agents. Second International Conference on Autonomous Agents, May 1998.
8. JATLite (Java Agent Template, Lite). http://java.stanford.edu
9. Java(tm) Technology Home Page. http://www.javasoft.com
10. UMBC KQML Web. http://www.csee.umbc.edu/kqml/