Lecture Notes in Artificial Intelligence 2003

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Frank Dignum Ulises Cortés (Eds.)

Agent-Mediated Electronic Commerce III

Current Issues in
Agent-Based Electronic Commerce Systems

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saabrücken, Germany

Volume Editors

Frank Dignum
Utrecht University, Institute of Information and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail: dignum@cs.uu.nl

Ulises Cortés
Technical University of Catalonia
Software Department, Artificial Intelligence Section
Mòdul C5-216, Jordi Girona 1 & 3, 08034 Barcelona, Spain
E-mail: ia@lsi.upc.es

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Agent mediated electronic commerce. - Berlin ; Heidelberg ; New York ;
Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo :
Springer
3. Current issues in agent based electronic commerce systems /
Frank Dignum (ed.). - 2001
(Lecture notes in computer science ; Vol. 2003 : Lecture notes in
artificial intelligence)
ISBN 3-540-41749-4

CR Subject Classification (1998): I.2.11, K.4.4, C.2, H.3.4-5, H.5.3, I.2, J.1

ISBN 3-540-41749-4 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH
http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10782280 06/3142 5 4 3 2 1 0

Preface

In this book we present a collection of papers around the topic of Agent-Mediated Electronic Commerce. Most of the papers originate from the third workshop on Agent–Mediated Electronic Commerce held in conjunction with the Autonomous Agents conference in June 2000. After two previous workshops, one during the Autonomous Agents conference in 1998 in Minneapolis and the second one in conjunction with the International Joint Conference On Artificial Intelligence in 1999, this workshop continued the tradition of the previous ones by setting the scene for the assessment of the challenges that Agent-Mediated Electronic Commerce faces as well as the opportunities it creates. By focusing on agent-mediated interactions, specialists from different disciplines were brought together who contribute theoretical and application perspectives in the narrowly focused topic that nevertheless involves wide ranging concerns such as: agent architectures, institutionalization, economic theory, modeling, legal frameworks and policy guidelines. The main topics for the workshop were:

- Electronic negotiation models for agents
- Formal issues for agents that operate in electronic market places
- Virtual trading institutions and platforms
- Trading strategies for interrelated transactions (respectively auctions)

The workshop received 12 submissions of which 7 were selected for publication in this volume. Although the number of submissions was less then expected for an important area like agent-mediated electronic commerce there is no reason to worry that this area does not get enough attention from the agent community. In fact, we noticed that many papers on agent-mediated electronic commerce found their way to the main conference. We decided therefore to invite a number of authors to revise and extend their papers from this conference and combine them with the workshop papers. Finally, we decided to include a paper that discusses the results of the Fishmarket tournament that was held during the workshop. The result is that this volume contains 12 high quality papers that really can be called representative of the field at this moment.

We have arranged the papers in the book according to the topics indicated above. The first section containing three papers is focused on negotiation models. This section starts with a more theoretical paper on a bilateral negotiation model for agent-mediated electronic commerce from *G. de Paula, F. Ramos and G. Ramalho*. In this paper they describe a model for bilateral negotiation, which offers support for, e.g., alternative product suggestion and ultimatum generation. It is therefore a generalization of the models used in current e-commerce systems such as Kasbah and MAGMA.

The second paper in this section (from *M. Barbuceanu and W. Lo*) discusses simultaneous negotiation over several attributes of a product. It uses constraint optimization as a model to concurrently satisfy several objectives as well as possible.

The same type of techniques are used in the paper of *R. Kowalczyk and V. Bui*, which uses constraint-based reasoning to support the negotiation process of agents in a car-trading system.

The second section of the book contains three papers with a more formal, logical flavor. The first two papers both discuss issues related to the communication between agents in an e-commerce setting. The first paper of *A. Artikis, F. Guerin and J. Pitt* shows how conversations between more than two parties can be formally modeled and given a clear semantics. The theory is used to model some frequently occurring auction protocols.

The second paper (*M. Wooldridge and S. Parsons*) discusses some issues for the design of negotiation protocols for agent communication languages that are based on logic. It indicates that a seemingly simple question as to whether agreement between the parties has been reached is difficult to answer formally (based on the protocol and the messages exchanged). It sets forth to indicate a number of progressively more complex negotiation languages and considers the complexity of these languages.

The last paper of this section, from *M. Pradella and M. Colombetti*, gives a formal description of a practical agent for e-commerce. The formal model makes it possible to prove certain properties of the agent, which would otherwise be hard to discover.

The third section of this book is devoted to platforms and institutions for agent-mediated electronic commerce. *H. Cardoso and E. Oliveira* describe a platform that can be used for e-commerce between agents and which supports adaptive agents, i.e., agents that learn from past experiences. The platform also supports multi-lateral and multi-issue negotiation.

In the paper of *M. Schröder, J. McMann and D. Haynes* it is argued that in trading a universal ontology is often assumed for matching offers and demands. This ontology is often a bottleneck for the scalability of the system. The paper describes a system which circumvents the necessity of such a universal ontology by making the clients a bit more flexible, while the traders specify their products a bit less precisely.

The last paper in this session is from *M. Tsvetovat, K. Sycara, Y. Chen and J. Ying* and discusses the formation of customer coalitions on electronic markets. It discusses the possible formation processes with their advantages and risks and indicates which model is most likely to succeed.

The last section of the book contains four papers related to agent strategies for agents that operate on markets where multiple interrelated auctions are running. The first paper of *C. Preist, C. Bartolini and I. Phillips* discusses the design of an algorithm for an agent that participates in multiple simultaneous auctions. The algorithm is designed to divide the wanted number of products over the auctions in an optimal way.

The other paper of this section is from *J. Béjar and Cortés* and discusses strategies for agents that participate in the Fishmarket games using the Dutch Bidding Protocol. In these games the agent has to participate in a number of successive auctions, trying to buy an optimal amount of fish. The last paper by

J. Béjar and J. Rodríguez-Aguilar describes the exhibition tournament held for this workshop. It also discusses the reasons for the success or failure of some strategies and lessons that can be learned from the tournament.

We want to conclude this preface by extending our thanks to the members of the program committee of the AMEC workshop who were willing to review the papers in a very short time span and also of course to the authors who were willing to submit their papers to our workshop and the authors that revised their papers for this book.

December 2000

Frank Dignum
Ulises Cortés

Workshop Organization

Organizing Commitee

Frank Dignum	Utrecht University, Utrecht, The Netherlands
Ulises Cortés	Technical University of Catalonia, Barcelona, Spain
Juan A. Rodríguez-Aguilar	Massachusetts Institute of Technology, Cambridge, USA

Program Commitee

Javier Béjar	Technical University of Catalonia (Spain)
Boi Faltings	École Polytechnique Fédérale de Lausanne (France)
Peyman Faratin	IIIA-CSIC (Spain)
Fausto Giunchiglia	IRST (Italy)
Robert Guttman	Frictionless Commerce Inc. (USA)
Sverker Janson	Swedish Institute of Computer Science (Sweden)
Nick R. Jennings	Southampton University (UK)
Sarit Kraus	Bar-Ilan University (Israel)
Pattie Maes	MIT (USA)
Joerg Muller	Siemens (Germany)
Julian Padget	University of Bath (UK)
Jeremy Pitt	Imperial College (UK)
Chris Preist	Hewlett-Packard (UK)
Jeff Rosenschein	Hebrew University (Israel)
Katia Sycara	Carnegie Mellon University (USA)
Walter van de Welde	STARLAB (Belgium)
Mike Wooldridge	University of Liverpool (UK)
Frederik Ygge	Enersearch AB (Sweden)

Table of Contents

Section IV: Trading Strategies for Interrelated Transactions

Bilateral Negotiation Model for Agent-Mediated Electronic Commerce

Gustavo E. de Paula, Francisco S. Ramos and Geber L. Ramalho

Universidade Federal de Pernambuco
Centro de Informática
Caixa Postal 7851 50732-970 – Recife – PE – Brazil
{gep,glr}@di.ufpe.br

Abstract. One of the main issues in electronic commerce is the inclusion of the negotiation facilities commonly available in human client-vendor interaction in real world commerce. E-commerce negotiation processes have usually been modeled as self-interested multi-agent systems. In these systems buyers and sellers are represented by agents that have opposite demands and decide what to do on the flight, based on the available information. However, currently e-commerce systems, such as Kasbah and Magma [21], provide a small number of bilateral negotiation facilities. Fortunately, some general negotiation models could be at first applied to e-commerce domain. This is typically the case of Faratin's model, which can be seen as an extension of Kasbah's. In this paper, we propose an original bilateral agent negotiation model, which extends Faratin's one. We introduce various facilities, such as alternative product suggestion, ultimatum generation, local contract agreements, etc. These facilities intend to grant users with a more flexible e-commerce environment. We present our model formalization, including the knowledge base that determines agent behavior. Some empirical validation is also presented to the case of computer purchase.

1 Introduction

As the e-commerce usage has been growing, the involved companies have been trying to offer buy-and-sell facilities to their clients that are as close as possible to the real world ones. These facilities rely essentially on the client-salesman interaction. In fact, clients usually do not know precisely which product to buy or how much to pay for it. The salesman can then (i) help the client to choose the product that best fits his or her needs; (ii) suggest alternative products; and (iii) negotiate shopping prices and conditions. These facilities are desirable because they provide user with a more flexible environment to do their shopping, and this will probably help him to get better purchase conditions.

Agent's metaphor and technologies [9, 22] can be used to model the behavior, the desires and the goals of both salesman and client, providing the typical facilities of client-salesman interaction, which are not yet present in e-commerce. Under this agent mediated e-commerce (AMEC) perspective, one of the main issues is to model and implement negotiation, which involves the specification of the negotiable attributes, agent possible moves, and agent decision making strategies.

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 1-14, 2001.

MIT Kasbah market place [2, 3] is the most significant example of e-commerce negotiation system. This market place simulates an environment where an user can create an autonomous agent to buy or sell a product, negotiating product price on his/her behalf. The agent configuration includes some behavior rules, such as the maximum time to reach a deal, the desired price interval and the price suggestion function. Kasbah provides a small number of facilities to the user's negotiation process.

Fortunately, since negotiation is a central component of many self-interested multi-agent systems [2, 3, 7, 8, 13, 14], some of them could be adapted to e-commerce domain. This is typically the case of Faratin's negotiation model [5], which can be seen as an extension of Kasbah's. In fact, Faratin's model includes new features, such as multiple attributes negotiation, and imitative and resource dependent proposal generation.

In this paper, we detail an agent bilateral negotiation model, which is an extension of Faratin's one applied to e-commerce domain. The main extensions encompass the incorporation of negotiation cost in decision function; the possibility of negotiating over a set of products and multiple attributes; the ultimatum generation; and the suggestion of alternative products.This paper is organized as it follows. In the next section, we present our view of the negotiation problem and processes. In Section 3, we analyze Kasbah and Faratin's model, highlighting their limitations. In Section 4, we present our model, pointing at some solutions to overcome these limitations. We present the negotiable products representation, the agent possible moves (which defines agent complexity), and the agent behavior rules. Some empirical validation to the case of computer purchase are also presented in Section 5. Conclusions and future work are given in the last Section.E-commerce Negotiation.

2 E-Commerce Negotiation

In this Section, we discuss some problems and processes underlying e-commerce negotiation, stressing the difficulties of modeling real world features.

2.1 Real World Negotiation Features

The virtual negotiations, which take place in the Internet, are far away from a real world one, because several features are not considered. Human behavior in real world commerce negotiation involves difficult points, such as:

- *Multiple attributes negotiation*: negotiation usually involves several attributes, such as price, delivery time, taxes, etc.;
- *Similar product suggestion*: clients usually do not know precisely which product to buy. They have only an idea of the desired product class. In this case, the negotiation can regard similar alternative products;
- *Correlated product suggestion*: when a client buys a television, a salesman can offer a discount in the case that the client also buys a video cassette;
- *Ultimatum*: when a negotiator wants to leave the negotiation process, he/she gives an ultimatum to the opponent indicating that this is his/her last offer. This

ultimatum is used to indicate the desire to leave the negotiation process if the last offer is not accepted.

- *Negotiation cost*: a client can buy the product at hand only to avoid the cost (locomotion, parking, etc.) of trying to find a cheaper one somewhere else;
- *Learning*: the experience of previous negotiations is usually taken into account in the future;

As far as we know, only the first of the above features has been incorporated by the e-commerce systems developed so far.

2.2 Principles of Bilateral Agents Negotiation

In automatic bilateral negotiation, two agents (seller and buyer) that have contradictory demands must exchange proposals in order to reach a deal.

Each agent has a private *border proposal*, which is a maximum (or minimum) limit that must be respected in reaching a deal (see Figure 1). The intersection between agents' border proposals defines what we call the *deal range.* If the deal range will be empty, the deal is impossible. A formal definition of these concepts is available in [4].

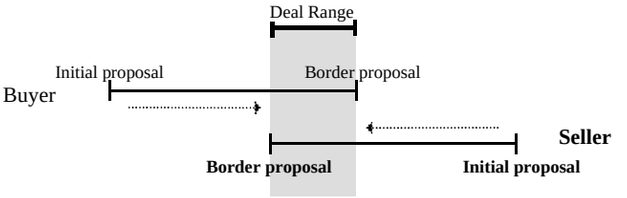


Fig. 1. Bilateral negotiation process. Seller is in bold and buyer is in normal. The dashed area is the deal range.

Each agent's border proposal values are normally kept hidden from its opponents. Some approaches, such as *vickrey* mechanism, multi-unit *vickrey* [6], have been developed to force the opponent agent to reveal its border values. However, it has been shown that this approach has several limitations when applied to bounded rationality applications like the one considered here [19].

2.3 Problems of Bilateral Agent Negotiation

In order to model an environment where agents can negotiate on behalf of human beings taking into account real world features, it is necessary to answer the following questions:

- How to model the proposals?
- How proposals are evaluated?
- Which are the agent possible moves in negotiation?
- How do agents decide which move to take in each of negotiation round?

The first question regards the content and the representation of the proposals exchanged during negotiation. The answers depend on what should be considered in

negotiation. A proposal may contain more than one *deal attribute*, such as price, delivery time, delivery tax, guaranties, etc. *Product attributes* can also be considered in a proposal. For instance, in the purchase of a computer, these attributes could be the computer brand, the processor speed, the memory size, etc.

The answer to the second question depends on the answer to the first one and defines an *evaluation function*. If the proposals consider more than one deal attribute, for instance, it would be necessary to combine theses attributes to obtain a single value. If product attributes are considered, then they should also be included in the evaluation function.

The answer to the third question depends on the desired negotiation complexity. A basic negotiation usually embodies three moves: accept an offer; quit negotiation; and generate a counterproposal. The agent can *accepts* a proposal when it is between its initial and border proposals range and the agent cannot generate a better counterproposal. Normally agent *quits* negotiation when the opponent current proposal is not good enough and a given maximum negotiation time is reached. Otherwise, the agent could generates a counterproposal. A sophisticated negotiation could encompass additional moves, such as alternative or correlated products suggestion; and ultimatum offer (See Section 2.1).

The fourth question addresses the decision making problem. If only the basic moves are adopted, a simple function can be defined to classify the moves importance, based on the proposal evaluation (see second question). The closer is the model to real world negotiation, the greater is its complexity, since further moves are considered and the definition of the classification function becomes harder. In the case of choosing counterproposal generation, it is also necessary to determine its value. This can be made according to some tactics [5], such as: time-dependent tactics, where proposals depend only on the elapsed time in a constant rate; behavior-dependent tactics (tit-for-tat) where proposals depend only on the opponent behavior; resource-dependent tactics where proposals depends on the amount of some resource in the negotiation environment (e.g., number of available sellers).

3 State of the Art

We discuss now how two negotiation systems, Kasbah [2, 3] and Faratin's [5], address the four questions raised in the previous section. As stated in the introduction, Kasbah is the most significant e-commerce system devoted to the simulation of real world negotiation conditions. Although Faratin's model has been applied to a business process management domain, it can be seen as a direct extension of Kasbah's model. We believe that Faratin's model can be easily applied to e-commerce and, besides that, be extended to include new features.

In Kasbah, the proposals consist only of a single item: the product price (the product attributed are not considered). The proposals evaluation in then the simplest as possible consisting of the suggested price. The moves implemented are the basic ones: quit, accept and generate counterproposal (see Section 2.3). All counterproposal generation functions are time-dependent, with variation rates being linear, quadratic or cubic. Negotiation cost is not considered in decision process.

Faratin's model extends Kasbah's in three aspects: proposal representation, proposal evaluation and counterproposal generation tactics. In fact, a Faratin's

proposal, represented by $Proposal_j = \{X_j\}$, may contain multiple deal attributes. Each X_j is then defined as

$$X_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{in}) \quad \textbf{(1)}$$

where x_{ji} represents an deal item value and j the negotiation round number (see section 4.1).

Each x_i is limited to a continuos closed interval $[min_i, max_i]$ and it has a score function $V_i : [\min_i, \max_i] \rightarrow [0,1]$, which serves to normalize it before combining all the attributes. The proposal evaluation function is a weighted combination of the each score function values and it maps the different values composing a proposal into a single point in the negotiation range. It is defined as: $E_x(X_i) = \sum_{j=1}^{n} w_j V_j(x_{ij})$, where $\sum_{j=1}^{n} w_j = 1$.

Faratin's model implements the same moves as in Kasbah's. The negotiation decision making uses an interpretation function, which takes as input the agent current and last proposals, the opponent current proposal and the current negotiation round. This interpretation function is defined below.

$$I^a(t, proposal_{t-1}^b, proposal_t^a) = \begin{cases} reject, If(t > t_{\max}) \\ accept, \\ If(E_a(prop_{t-1}^b) \geq E_a(prop_t^a)) \\ proposal_t^a \end{cases} \quad \textbf{(2)}$$

where $proposal_{t-1}^b$ is the last opponent proposal; $proposal_t^a$ is the agent current proposal; $t_{\max}$ is the maximum time to reach a deal.

Counter-proposal generation can be made according to three tactic families: time-dependent, behavior-dependent or resource-dependent ones. These tactics can also be weighted combined, defining a negotiation strategy.

In our view, there are two limitations in this model. The first one regards the fact that the model does not consider the negotiation cost in the decision making process[5, 17].

The second, and most important limitation regards the decision rule used in decision making (interpretation function) when the proposal is dealing with more than one attribute at the same time. The problem is that, using function E_x, all the deal attributes values are mapped into a single value. The mapping has two bad side effects: local constraint violation and local deal degeneration. The local constraint violation occurs when a proposal is accepted even if negotiation border of one of the deal attributes has not been satisfied. For instance, let us consider a problem where two agents, *B* (buyer) and *S* (seller), are negotiating *product price* and *delivery time*. The agent *B* wants to buy a product for $10 but it is able to give in up to $25. Agent *S* is selling the product for $40 and it is able to give in up to $23. Regarding the delivery time, *B* wants the book in 1 day and it is able to wait until 5. Considering all attribute weights equals to 0.5, the proposal ($26,1 day), whose evaluation is (0.5*0.53) +

(0.5*1) = 0,765, would be accepted, if compared to a previous proposal ($20, 5 days), whose evaluation is (0.33 * 0.5 + 0.5 * 0,5) = 0,415.

The local deal degeneration problem occurs because the evaluation function considers only the whole proposal and does not deliberate about any deal that may occur in one of the negotiable attributes. In other words, when a deal is reached regarding a single attribute, but not all attributes under negotiation, the negotiation process may degenerate. Considering the example above, let us take a look in a possible scenario (Table 1) of exchanged proposals between the agents. The negotiation process degenerates from round 3 since the local deal concerning delivery time is not considered in next rounds. This limitation is not present in Kasbah's, since it does not allow multiple items negotiation.

Table 1. Negotiation proposal exchange

Negotiation round	Buyer		Seller	
	Price($)	Delivery time(day)	Price ($)	Delivery time(day)
1	10	1	40	3
2	12	2	37	2
3	14	3	35	1
4	16	4	33	1
5	18	5	30	1

4 Our Negotiation Model

Our main motivation in developing our e-commerce negotiation model has been to incorporate most of the real-world features listed in Section 2.1. We have succeeded in some of them: multiple product items, multiple products of the same class, ultimatum and negotiation cost. In this section, we present how we have embodied these features, answering to the four questions stated in Section 2.3.

4.1 Negotiation Protocol Overview

Figure 2 shows our model negotiation phases and their connections. A *negotiation cycle* begins when the agent receives a proposal and it ends when it receives a counterproposal. Each cycle is called *negotiation round* and the agent has a maximum number of negotiation rounds t_{max} to reach a deal. It is worth to notice that, as time goes by and the number of rounds increases, a negotiation cost can increase, forcing the agents to reach a deal quickly.

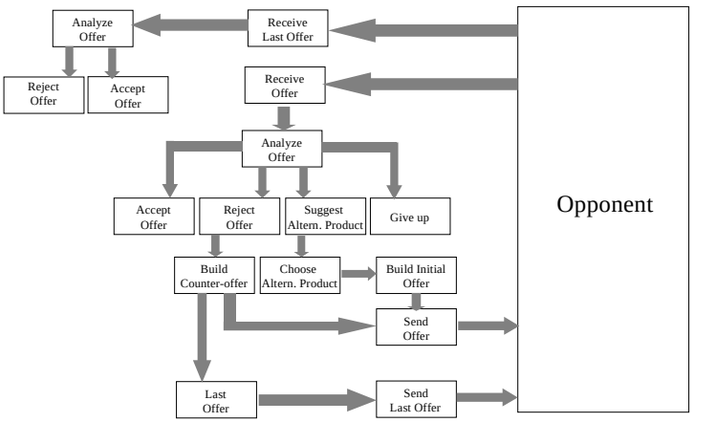


Fig. 2. Negotiation game in a process view. Opponent agent box has the same phases and connections.

4.2 Negotiable Attributes Representation

Since our model includes alternative product suggestion, it is necessary to consider a negotiation not only over a simple product by over a product class $C = \{P_1, P_2, ..., P_m\}$, where P_i represents a product and *m* is the number of products in the class. Each P_i is defined as:

$$P_i = (c_{i1}, c_{i2}, c_{i3}, ..., c_{in}) \ , \tag{3}$$

where c_{ij} represents a characteristic value.

Every c_{ij} is defined in a discrete set of possible values $c_{ij} \in \lfloor v_{1ij}, v_{2ij}, v_{3ij}, ..., v_{hij} \rfloor$ and has a score function $V_{ij}^{c} : \lfloor v_{1ij}, v_{2ij}, v_{3ij}, ..., v_{hij} \rfloor \rightarrow [0,1]$.

A product evaluation funcaion can be defined as $E_p(P_i) = \sum_{j=1}^{n} w_j V_j (c_{ij})$, where $\sum_{j=1}^{n} w_j = 1$.

The proposal contract which agent receives is represented by

$$Proposal_j = \{P_i, X_j\} \tag{4}$$

Each P_i stands for the product attributes being negotiated (Section 2.3) and X_j stands for the *deal items* values. The later have been proposed in Faratin's model (Section 3) with a modification which garanties that only proposals with all attributes values inside the deal range are accepted. To do that, the attribute evaluation function was modified so that values outside the $[\min_i, \max_i]$ were evaluation to $-\infty$. The product representation is an extension made by our model.

4.3 Evaluating Proposals

After the agent receives a proposal, it has to analyze it to decide if it is good or not. To do that, we define an evaluation function $E_{contract}$ which combines the evaluations of the product, E_p, and of the deal items E_x.

$$E_{contract}(\{P, X\}) = W_1 E_p(P) + W_2 E_x(X) \ , \quad (5)$$

where W_1 and W_2 are the weights that we give to the product and the items evaluation and $W_1 + W_2 = 1$.

The reason for considering the product attributes in the proposal evaluation function is that the product quality always influence the negotiator decision. A client do not want to spend much money in a bad quality product.

4.4 Possible Moves

Based on Figure 1, one can identify two different processes in negotiation. In the first one, which we denominate *outside process*, opponent proposals are beyond agent border proposal (blank area in Figure 1). The only possible moves for the agent are either to leave the negotiation process or to generate counterproposals, expecting that the opponent proposals will fall into the deal range. In the second process, denominated *inside process*, opponent proposals are within the deal range (dashed area in Figure 1). In this case, the agent can, besides the previous moves, accept the current opponent proposal and then reach a deal. Figure 3 illustrates these two sets of moves. The triangle notation of counterproposal move is used to represent the continuum of proposal values that can be generated for a product.

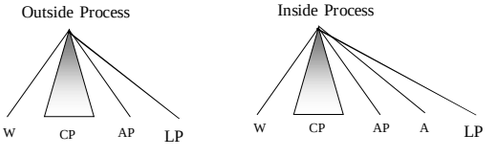


Fig. 3. Outside and Inside Processes moves (where W = Quit; CP = counterproposal generation; AP = Alternative product; LP = Last proposal)

In the following subsections the moves are detailed and the decision rules used to take each of them is presented.

4.4.1 Send a Counterproposals

In our model, when the agent receives a proposal, $Proposal_{rec} = \{P_{rec}, X_{rec}\}$, it immediately generates a counterproposal, $Proposal_{gen} = \{P_{gen}, X_{gen}\}$, to compare it to $Proposal_{rec}$ and determine whether it can get any better proposal in the next rounds. The proposal generation is made according to the tactic families proposed by Faratin [5,14].

We considered two different types of deals, in order to do not discards the local deals:

– *Local Deals*: as the negotiation takes place over multiple items, then a deal can be reached in one item, and the negotiation continues with the other. This is necessary to solve the second limitation of the Faratin's model presented in section 3. A local deal X_{ld} is defined as the following:

$$X_{ld} = \left\{ i \in n \middle| x_i \in X_{gen}, x_i^{'} \in X_{rec}, V_a(x_i) \leq V_a(x_i^{'}) \right\}, \quad \textbf{(6)}$$

where n is the number of items.

The counterproposal X_{prop} is then defined as:

$$X_{prop} = \left\{ \left\{ x_i \in X_{rec} \middle| i \in X_{ld} \right\} \cup \left\{ x_j \in X_{gen} \middle| j \notin X_{ld} \right\} \right\} \quad \textbf{(7)}$$

– *General Deal*: occur when the local deal set contains all the negotiable items.

4.4.2 Suggesting Alternative Products

Alternative product suggestion problem is divided in two sub-problems. The first one is to define which events indicate that an alternative product should be suggested. Some heuristics can be used to this problem. The heuristic we adopted considers the evolution of the opponent proposal over a number of negotiation rounds. Intuitively, this heuristics sates that, if the opponent's proposal is not falling in an appropriated rate for a "long" time, which we called an *increasing window*, so it will probably take too long to reach a deal with this product, and an alternative should be suggested.

Given an increasing window I and considering the set of opponent proposals evaluations $\{oppP_i\}_{i=0}^{numR}$ (where *numR* is the current number of negotiation rounds), the set of proposals outside the minimum increasing tax are:

$$I_{mim} = \left\{ i \in [0, numR - I] \middle| oppP_i - oppP_j > T, i = numR - I,\ j = i + 1 \right\}, \quad \textbf{(8)}$$

where T is the minimum increasing tax

The other problem is which alternative product should be suggested. There are also some heuristics to establish this order, such as product quality decreasing order, product quantity decreasing order, product evaluation decreasing order or random order. We have chosen the product evaluation decreasing order, since it considers all product features. A more complete description of this heuristics is out of this paper's scope and it can be found in [4].

4.4.3 Quitting, Accepting and Sending Last Offer

The agent accepts opponent proposal, when the generated proposal evaluation plus the negotiation cost is smaller than the opponent proposal evaluation. This happens when a general proposal is obtained and $\#X_{dl} = \#X_{prop}$, where # is an operator to give the number of elements of this set.

Before quitting negotiation, the agent always sends a last proposal (ultimatum) indicating that, if opponents, do not accept this proposal, it will leave negotiation in next round. If the opponent accepts the ultimatum, then it sends back an equal proposal. Otherwise, the opponent sends an empty proposal and quits the negotiation.

4.5 Agent Decision Making

To decide what to do, the agent must consider whether its current proposal is far or close to its opponent ones and whether the proposed contract is good or not. In this perspective, we have defined a contract band and a contract classification, which are used in agent's decision rules.

4.5.1 Proposal Bands

According to agent initial proposal and opponent current proposal, we choose to define three possible profit bands: Close to Deal, Middle Way to Deal and Far From Deal. So for a proposal we have that $iniP = E_x(X)$, where X stands for a product contract and $\forall x_i \in X, x_i = \min_i$, where $\min_i$ stands for the minimum possible value of x_i, and with *myP* standing for agent proposed contract evaluation.

$$CloseTo(myP, oppP) = \begin{cases} true, if & oppP < myP \leq closeLim \\ false & otherwise \end{cases} \quad (9)$$

$$MiddleWayTo(myP, oppP) = \begin{cases} true, if & closeLim < myP \leq middleLim \\ false & otherwise \end{cases}$$

$$FarFrom(myP, oppP) = \begin{cases} true, if & middleLimP < myP \leq farLim \\ false & otherwise \end{cases}$$

where $closeLim = oppP + (iniP - oppP)/3$,

$middleLim = oppP + 2/3(iniP - oppP)$ and

$farLim = iniP$

4.5.2 Proposal Classification

Proposal classification defines if a deal is good or not. Based on it, agent can decide if it accepts or not a proposed deal. We define four classifications for the deal: very good, good, bad and very bad. These classifications depend on the values of the maximum and the minimum possible deals for a product. So let $MinP = E_x(X)$, where X stands for a product contract and $\forall x_i \in X, x_i = \min_i$, with $\min_i$ stands for the minimum possible value of x_i, and $MaxP = E_x(X)$, where X stands for a product contract and $\forall x_i \in X, x_i = \max_i$ with $\max_i$ stands for the minimum possible value of x_i:

$$VeryGood(MinP, MaxP, oppP) = \begin{cases} true, if & MaxP < oppP \leq veryLim \\ false & otherwise \end{cases} \quad (10)$$

$$Good(MinP, MaxP, oppP) = \begin{cases} true, if & veryLim < oppP \leq goodLim \\ false & otherwise \end{cases}$$

$$Bad(MinP, MaxP, oppP) = \begin{cases} true, if & goodLim < oppP \le badLim \\ false & otherwise \end{cases}$$

$$VeryBad(MinP, MaxP, oppP) = \begin{cases} true, if & badLim < oppP \le veryBadLim \\ false, & otherwise \end{cases}$$

where $veryLim = MaxP - (MaxP - MinP)/4$ $goodLim = MaxP - (MaxP - MinP)/2$

$veryLim = MaxP - 3/4(MaxP - MinP)$ and $veryBadLim = MinP$

4.5.3 Behavior Rules

The behavior rules define how the agent should act during the negotiation according to payoff value, current profit band and classification (see sections 4.5 and 4.5.2), and should aim at giving the agent a rational behavior (move which should be performed in each negotiation round).

So, let *myP* and *oppP* stand for agent and opponent proposed contract evaluation respectively.

If $CloseTo(myP, oppP) \wedge (VeryGood(oppP) \vee Good(oppP))$ (**11**)

Then $Accept(oppP)$;

If $CloseTo(myP, oppP) \wedge (VeryBad(oppP) \vee Bad(oppP))$

Then $Nextproposal(myP)$;

If $(MiddleWayTo(myP, oppP) \vee FarFrom(myP, oppP))$

Then $Nextproposal(myP)$;

If $t = t_{\max}$

Then $Lastproposal(myP)$

If $\#I_{mim} = 0$

Then $AlternativeSuggestion(P_i)$

If $X_{dl} = X_{prop}$

Then $Accept(oppP)$

If $\#X_{prop} = 0$

Then $Quit(\ )$

where t_{max} is the ,maximum time to reach a deal and # is an operator which gives the number of elements in a set.

5 Implementation and Results

There is no commonly accepted methodology for evaluating negotiation performance of agent-mediated e-commerce systems, because of the intrinsic complexity of the task. Since we have been trying to build a model that fits real world negotiation conditions, we decided to implement an environment where agents and human beings could negotiate with each other, exchanging roles (buyer or seller) without knowing the identity (agent or human being) of the opponent negotiator. The negotiations take place in predefined purchase scenarios, e.g. long time to buy a specific computer spending little money, short time to buy any computer spending much money, selling computers quickly at low prices, etc. The experiments resemble a "Turing test for commerce negotiation".

To perform the experiments, we have implemented a distributed environment (see Figure 4) using java and Stanford JATlite [20] template as the agent communication base.

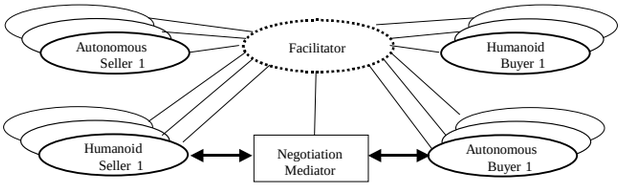


Fig. 4. Environment architecture. Bold line: active agent; regular line: static agent.

There are two kinds of agents in the environment: autonomous and humanoids. The humanoid agents act simply as the system interface to the human negotiator, receiving opponent proposals and forwarding human moves (Figure 5, bottom), and are configured with human initial desired (Figure 5, top left-hand side). The autonomous agents are built according to the model presented in Section 4, and can be configured using the interface presented in (Figure 5, top right-hand side). Negotiation can take place between autonomous or humanoid agents, but the same conditions (moves, configuration, etc.) are given and the same protocol is used despite agent type. Agents do not deliver proposals directly to each other. This is made through a *negotiation mediator agent*, which also saves all the exchanged proposals plus scenario configurations in a log file.

The system is fully implemented and the scenarios are defined. We have not yet run the experiments in all of the scenarios, but the first results are encouraging. The majority of human volunteers mismatch human and machine negotiators in short-term negotiations. For long term scenarios, human beings detect the difference.

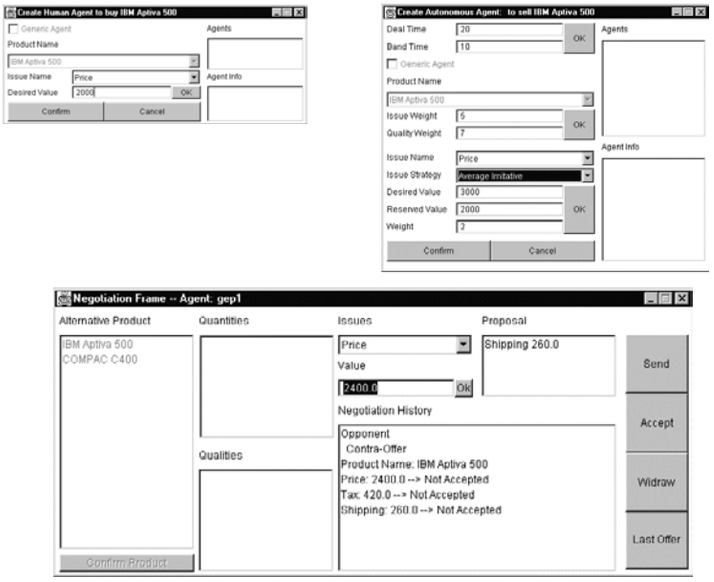


Fig. 5. Human negotiation interface (up-left side), autonomous agent configuration interface (up-right side) and human negotiation interface (down side)

6 Conclusions and Future Work

We have presented an original model for agent bilateral negotiation in e-commerce. In order to fit the best real world negotiation conditions, our model introduces novel features, such as the possibility of suggesting alternative products, the ultimatum generation and allowing local deals. This model extends Faratin's one in the three basic negotiation aspects: proposal representation, negotiation moves and decision making.

As future works, this model will be extended to embody correlated product suggestion and learning capabilities. We intend also to conclude the experimentation covering all purchase scenarios and then propose a formalization of the adopted methodology.

References

1. Alberts, M., Jonker, C. M., Karami, M., Treur, J.: An electronic Market Place: Generic Agent Models, Ontologies and Knowledge. Workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain in 3nd International Conference on Autonomous Agents. Seattle, Washington, May 1, (1999)

2. Chaves, A., Maes, P.: 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. London, UK, (1996).
3. Chaves, A., Dreilinger, D., Gutttman, R., Maes, P.: A Real-Life Experiment in Creating an Agent Marketplace. Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97). London, UK, (1997).
4. de Paula, G. E., Ramos, F. S., Ramalho, G. L.. Electronic Commerce Negotiation Model Formalization. Technical report UFPE-DI 99-20. 1999
5. Faratin, P., Sierra, C., Jennings, N. R.: Negotiation decision Function for Autonomous Agents. Int. Journal of Robotics and Autonomous Systems 24 (3-4) 159-182, (1998).
6. Fudenberg, D., Tirole, J.: Game Theory, Fourth printing, MIT, (1995).
7. Guttman, R., Maes, P.: Cooperative vs. Competitive Multi-agent Negotiations in Retail Electronic Commerce. Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98). Paris, France, (1998).
8. Guttman, R., Maes, P.: Agent-mediated Integrative Negotiation for Retail Electronic Commerce. Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET'98), (1998).
9. Jennings, N. R., Sycara, K. P., Wooldridge, M.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent System, 1, pages 275-306, (1998).
10. Jennings, N. R., Wooldridge, M. J.: Software Agents. IEE Review, January, 17-20, (1996).
11. Kraus, S.: Negotiation and Cooperation in Multi-Agent Environments. Artificial Intelligence journal, Special Issue on Economic Principles of Multi-Agent Systems, 94(1-2):79-98, (1997).
12. Kraus, S., Wilkenfeld, J., Zlotkin, G.: Multi-agent Negotiation Under Time Constraints. Artificial Intelligence journal, 75(2):297—345, (1995).
13. Maes, P., Guttman, R. H. Moukas, A. G.: Agents that Buy and Sell: Transforming Commerce as we Know it. Submitted to the Communication of the ACM, march (1999).
14. Matos, N., Sierra, C., Jennings, N. R.. Determining successful negotiation strategies: an evolutionary approach Proc. 3rd Int. Conf. on Multi-Agent Systems (ICMAS-98), Paris, France 182-189. (1998).
15. Moukas, A. G., Guttman, R. H., Maes, P.: Agent-mediated Electronic Commerce: An MIT Media Laboratory Perspective. Proceedings of the First International Conference on Electronic Commerce (ICEC'98), Seoul, Korea, (1998).
16. Parsons, S., Sierra, C., Jennings, N. R.: Agents that reason and negotiate by arguing. Journal of Logic and Computation 8 (3) 261-292, (1998).
17. Rubinstein, A.: Perfect Equilibrium in a Bargain Model. Econometrica, 50(1):97-109. (1982)
18. Sierra, C., Faratin, P., Jennings, N. R.: A Service-Oriented Negotiation Model between Autonomous Agents. Proceedings of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97), Ronneby, Sweden, pages 17-35, (1997).
19. Sycara, K. P.: Multi-agent systems. AI Magazine, Vol. 10, No. 2, pp. 79-93, (1998).
20. Stanford JATLite Package java.stanford.edu
21. Tsvetovatyy, M., Gini, M., Mobasher, B., Wieckowski, Z.: MAGMA: An Agent-Based virtual Market for Electronic-Commerce. Workshop of AI for Electronic Commerce (AI-EC / AAAI99), July 18, (1999).
22. Woodridge, M., Jennings, N. R.: Intelligent Agent: Theory and practice. The Knowledge Engendering Review. 10(2):115-152, (1995).

Multi-attribute Utility Theoretic Negotiation for Electronic Commerce

Mihai Barbuceanu and Wai-Kau Lo

Enterprise Integration Laboratory
University of Toronto
4 Taddle Creek Road, Rosebrugh Building
Toronto, Ontario, Canada, M5S 3G9
{mihai,wklo}@eil.utoronto.ca

Abstract. We present a generic negotiation architecture that uses Multi-attribute Utility Theory (MAUT) principles to reach agreements that satisfy multiple interdependent objectives. The architecture is built by giving a constraint optimization formulation to the MAUT principles and by using a constraint optimization solver to find the best 'deals' from an agent's local perspective. These are then proposed to other agents via a second component that supports conversational interactions among agents. When received proposals are disjoint from what an agent can currently accept, we provide a systematic constraint relaxation protocol that allows agents to generate the next acceptable 'deal'. This protocol ensures that in the end the Pareto optimal deal will be found, if one exists. The approach is built on top of our Negotiation Engine, a generic architecture for coordination and negotiation that integrates local reasoning, in the form of propositional constraint optimization, with interaction, in the form of conversational exchanges. The system is fully operational, being currently used to automate negotiations in the electronic components domain.

1 Introduction

The distribution of ownership over resources and activities among the members of a group creates dependencies between members, requiring them to negotiate in order to achieve their goals. Negotiation allows agents to agree on which goals to adopt in order to satisfy their individual needs and to plan and execute the agreed on goals through joint work.

In most real situations what is acceptable to an agent can not be described in terms of a single parameter. A buyer of a PC will consider the price, the warranty, the speed of the processor, the size of the memory, etc. A buyer of a service, like Internet access, will look at the speed and reliability of the connection, the disk space offered, the quality of customer service, the pricing scheme, etc. Agreements in such cases are regions in a multi-dimensional space that satisfy the sets of constraints of both sides. Recent experiences with shopping agents like [4] have shown that limiting the comparison of offerings to price only causes the dissatisfaction of merchants who see their offer unfaithfully projected on a single dimension with their other value-added services ignored. Gutman [7] made the case that the increasingly popular online auctions, by pitting buyers

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 15–30, 2001.

against sellers in price only competitions, make an equally great disservice to buyers. The success of ventures like NetGrocer, who charges more than what one would pay in a supermarket, but saves time and guarantees quality, further confirms this. And in the business to business relation, it has long been the case that offers are judged based on a wide array of criteria - the Just-in-Time manufacturing model for example requires sellers to respect given delivery time windows even if this incurs higher prices.

Given the importance of negotiations that can lead to multi-dimensional agreements, what tools and technologies can be used to produce a generic negotiation system of this kind? Multi-Attribute Utility Theory (MAUT) [9] is a tool for making decisions involving multiple interdependent objectives based on uncertainty and preference (utility) analysis. The PERSUADER system [12] is an early system that used MAUT to resolve conflicts through negotiation in the domain of labour disputes. It has been suggested [7] that a combination of MAUT and distributed constraint satisfaction [13] could lead to agents that could automate multi-objective negotiations in e-commerce applications. However, we have not yet seen any concrete technical solution as to how this integration can be achieved.

To purpose of this paper is to propose such a solution. Briefly, our approach is first to integrate MAUT principles into a generic, constraint optimization agent reasoner and second to integrate the reasoner with a conversational agent interaction system. The first step is achieved by giving a constraint optimization formulation to the MAUT problem, and then using the constraint optimization solver to find the best 'deals' from an agents local perspective. These are then proposed to other agents via conversational interaction. When received proposals are disjoint from what an agent can currently accept, we provide a systematic constraint relaxation protocol that allows agents to generate the next acceptable 'deal'. This protocol ensures that in the end the Pareto optimal deal will be found, if one exists.

The approach is built on top of our Negotiation Engine, a generic architecture for coordination and negotiation that integrates local reasoning, in the form of propositional constraint optimization, with interaction, in the form of conversational exchanges. The system is fully operational, being currently used to automate negotiations in the electronic components domain.

The paper is organized as follows. First, we review the Negotiation Engine, discussing its constraint language, the resolution methods and the interaction mechanisms. Then we show how the constraint reasoner can be used to encode MAUT problems and we present the relaxation protocol that ensures Pareto optimality. Then we present our application in electronic component negotiation and address more system architecture issues. We end with conclusions and future work remarks.

2 The Negotiation Engine

The Negotiation Engine is a system that integrates a generic reasoning component with a generic interaction component. Agents use the reasoning component to make decisions about the course of actions they will adopt. They use the interaction component to communicate and coordinate with other agents. The two components operate in an intertwined manner which allows agents to use reasoning to decide when, what and how

to communicate and to integrate the information received through communication in their deliberation. The reasoning component combines a propositional representation with utility-theoretic elements in a constraint optimization framework. The interaction component uses a conversational interaction model based on our previous work [2]. In the following we describe the representation used by the reasoning component, the constraint optimization model implemented by the reasoner, the basic conversational interaction approach and the integration between the reasoner and the conversational component.

2.1 Describing Behavior

Syntax and Semantics. Agents act and thus we first need a language to describe and reason about agent behavior. We define behavior as a course of action aimed at achieving certain goals. The language represents the goals that agents have. There are two kinds of goals, *composed* and *atomic*. Composed goals consist of other (sub)goals, while atomic goals do not (thus can be immediately executed). Both types of goals have temporal execution semantics and their description may include elements like start times, end times and durations. As temporal execution is not used in this paper, we will not detail these aspects here (see [1] for details). For the purposes of this paper we are interested in three kinds of compositions.

- *Parallel* (conjunctive) compositions: $a = par(a_1, a_2, ...a_m)$ defines a as a goal that is achieved iff all components $a_i, m \geq i \geq 1$, are achieved.
- *Choice* (disjunctive) compositions: $a = choice(a_1, a_2, ...a_p)$ defines a as a goal that is achieved iff a non-empty subset of sub-goals $a_i, p \geq i \geq 1$ is achieved.
- *Xchoice* compositions are choices which are achieved iff exactly one sub-goal is achieved.

The complete language also includes *sequential* compositions, $a = seq(a_1, a_2, ...a_n)$ which require that all component goals be achieved in the given order, but these are not used in this paper.

From the execution viewpoint, choices have *or* and xchoices *xor* semantics. A choice g is 'on' - meaning is achieved and written $On(g)$ - iff at least one component is on and 'off' - meaning is not achieved and written $Off(g)$ - iff all components are off. An xchoice is 'on' iff exactly one component is 'on' and 'off' otherwise. Parallels have *and* semantics - 'on' iff all components are on, and 'off' otherwise. Sub-goals can occur negated within composed goals. Thus, $c = choice(a, -b)$ denotes a choice between achieving a and not achieving b.

Controllable and non-controllable goals. A goal is under an agent's control (controllable) if the agent can decide on its own if the goal will be achieved or not (owns the goal). Often, non-controllable goals are part of an agent's plans. In such cases, the agent has to obtain the commitment of the agents controlling these goals about their achievement. Obtaining such commitments is the purpose of negotiation.

Utilities. We quantify an agent's preferences toward achieving goals by attaching two sorts of utilities. $u_{on}(a)$ denotes the utility obtained if a is achieved. $u_{off}(a)$ denotes the utility obtained if a is not achieved. We normally assume that utilities are real numbers

in the $[0, 1]$ interval. Utilities can be used both to describe an agent's own preferences and to quantify influences between agents. The utility of a goal that is controlled by another agent describes in some way the power that the other agent has on the agent needing the goal. An agent that controls a goal (thus can offer it to others) may have a very different utility for that goal from the agents needing it. In this paper we also make the standard assumption that agents will plan for or choose those behaviors that maximize their utility.

Roles. The representation of roles specifies, for each role an agent can play, the goals it controls and the goals it needs. This is used to perform strategic coalition formation - choosing who to involve when needing to achieve certain goals. The details of coalition formation will be described elsewhere.

2.2 Finding Behavior by Constraint Optimization

Terminology. Let $G = \{g_1, ...g_n\}$ be a set of goals, or a goal network. An on-off labeling of the network is a mapping $L : G \rightarrow \{on, off\}$ associating either 'on' or 'off' labels to each action in G. A labeling is *consistent* iff the labels of each composed node and of its subgoals are consistent with the node's execution semantics, e.g. if a parallel goal is 'on', then all its subgoals are also 'on' or if a choice goal is 'off', then all its subgoals are 'off', etc. Consistent labelings thus define *executable* behaviors.

Let $C = \{c_1, ...c_m\}$ be a set of constraints, where each c_i is a constraint of the form $On(g_j)$, $Off(g_k)$ or an implication on both sides of which there are conjunctions of on-off constraints. $On(OnSiteService) \supset On(PayOnSiteService)$ is an example of implication.

Let $U = \{(g_1, u_{on}(g_1), u_{off}(g_1))...(g_l, u_{on}(g_l), u_{off}(g_l))\}$ be a utility list, that is a set of goals with their associated on and off utilities. Given a utility list U and a consistent labeling L of G we can compute the total utility of the labeling, $Util(L, U)$, as the sum of on-utilities for the 'on' labeled goals plus the sum of off-utilities for the 'off' labeled goals.

Finally, a problem is a tuple $P =< G, C, U, criterion >$ where $criterion \in \{max, min\}$. A problem specifies a goal network, a set of constraints, a utility list and

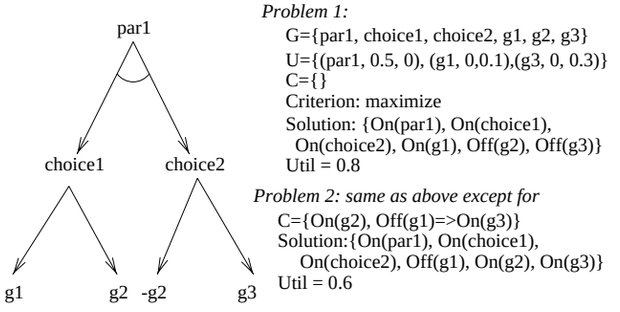


Fig. 1. Goal Network with two problem descriptions.

an optimization criterion, either max or min. A solution to a problem P is a labeling L such that $Util(L, U)$ is either maximal or minimal, according to the $criterion$.

To clarify the vocabulary and illustrate the graphical notation we employ for goal networks, figure 1 shows two simple problems based on the same goal network ($par1$ is a parallel, $choice1$ and $choice2$ are choices, $g1$, $g2$, $g3$ are atomic, g_2 occurs negated in $choice2$), the same utility list and two sets of constraints, together with their solutions. In the first problem, the total utility comes from $On(par1)$ (0.5) and $Off(g3)$ (0.3). In the second one, from $On(par1)$ (0.5) and $Off(g1)$ (0.1).

The Solver. The problem of finding a consistent labeling is equivalent to satisfiability (SAT). The problem of finding a labeling that maximizes (minimizes) utility is a form of MAXSAT. The Solver provided by the Negotiation Engine takes advantage from the recent wave of stochastic search based solutions for SAT [11] (and to a lesser extent MAXSAT [8]) by providing and integrating an incomplete random search procedure with a systematic (complete) branch and bound procedure, both operating over the same representation of goal networks. In both cases implication constraints are translated into disjunctions and then solved with the entire network. The random search method is not guaranteed to find a solution, but performs very well on large scale problems (both in terms of time and of its ability to actually find solutions).

```
Modified-WSAT(goalNetwork){
 for i = 1 to maxTries{
  A = a randomly generated on-off assignment;
  for j = 1 to maxFlips{
   if A is a solution return it;
   else {G = randomly chosen inconsistent goal;
    With probability P,
     Flip G or one of its subgoals that results
     in the greatest utility increase;
    Otherwise (with probability 1-P)
     Flip G or one of its subgoals that results
     in the greatest decrease in the number of
     inconsistent goals;}}}
 return failure;}
```

Fig. 2. Modified WSAT for random search.

The actual random search method we use is a modification of WSAT [11] (figure 2). The algorithm always keeps a complete on-off assignment to goals. Goals are 'flipped' (their value is changed) with probability P (around 0.2 in our experiments) to increase the utility of the assignment (unlike the standard WSAT which flips randomly), and with probability $1 - P$ to increase the number of consistently labeled goals. In each case ties are broken in favor of the other criterion. Based on the current experience, this method has consistently produced high utility solutions, in many cases finding the optimal solution (similar results are reported in [8] for another variant of WSAT used for optimization).

The branch and bound method is a systematic backtracking search procedure that is guaranteed to find the optimal solution. The procedure maintains the utility of the current best solution. If the partial solution currently explored can not be extended to one with better utility than the current best (done by overestimating the utility of the current solution), then it is dropped and a new one is explored. The procedure uses the variable selection heuristic of selecting the most constrained goal first (the one with most subgoals assigned) and forward checks any proposed assignment to ensure it is not inconsistent with past assignments (figure 3).

The solver allows for the integration of the two methods, for example by using random search first for a number of runs and then taking the utility of the best solution produced as the bound constraining branch and bound to find a better solution.

```
BranchAndBound(maxUtil){
 top:
  while(true){
   if Complete solution found whose utility is util{
      if util > maxUtil{maxUtil = util;
                        Save current solution;}
      Backtrack;
      if Backtracking not possible exit;}
   else{
    goal=select most constrained goal;
    Push goal onto stack;
    while(goal inconsistent with past assignments or
          current solution is worse than maxUtil){
     if Another assignment for goal and its
        subgoals exists
          {Continue with top;}
     else {Backtrack;
           if Backtracking not possible exit;}
    }}}}
```

Fig. 3. Branch and bound method.

2.3 Conversational Interactions

To support the interaction dimension of negotiation we use our previous conversational technology [2] of which we only review here a few elements needed for the understanding of this work. The major elements of our conversational technology are *conversation plans*, *conversation rules*, *actual conversations* and *situation rules*. Briefly, a conversation plan is a description of both how an agent *acts locally* and *interacts* with other agents by means of communicative actions. The specification of conversation plans is largely independent from the particular language used for communication, for which we currently use a liberal form of KQML [6]. A conversation plan consists of states

(with distinguished initial and final states) and rule governed transitions together with a control mechanism and a local data base that maintains the state of the conversation. The execution state of a conversation plan is maintained in *actual conversations*. Active conversations can be suspended until other active conversations reach given states or other agent conditions hold, then they can be resumed. To decide which conversations to instantiate and to update the agent's data base when events take place, we provide *situation rules*. The top level control loop of an agent activates all applicable situation rules (suggesting conversations to initiate) and then executes new or existing conversations as appropriate. This technology has been recently reimplemented in Java and extended in several ways. One extension allows an agent to create a Java servlet that will manage a graphical interface between a human user and the agent. Using a Web browser, human users can in this way have conversations with their agents. From the point of view of the agent, the interaction with the user is just a regular conversation. The servlet creates screen forms for data input and translates the information entered by the user into the agent communication language.

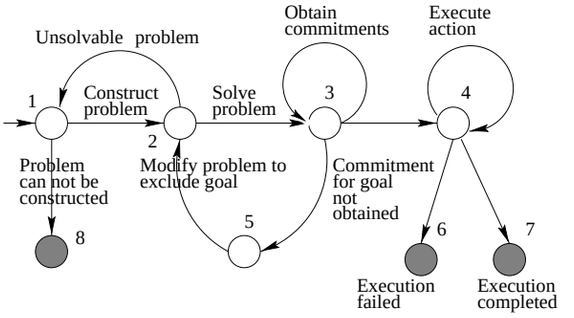


Fig. 4. Conversation plan integrating local reasoning with interaction.

We use the conversational technology for two purposes. First, to represent and execute structured patterns of agent interaction. These patterns correspond to interaction conventions followed by agents in given situations, for example when bidding in an auction or negotiating a contract. Second, as a high level 'scripting language' that allows the integration of local solvers, access to resources, etc. into the final agent behavior. For the latter purpose, the Negotiation Engine provides API-s to its representation language and solvers that allow agents to construct models of situations, use the solvers to reason about these and then use the results of reasoning in interaction with other agents. For example, API-s allow an agent to create and modify goal networks, define problems by associating constraints and utilities to goal networks, solve problems by using the available solvers in various combinations.

As an illustration of how local reasoning can be integrated with interaction in a conversation plan, consider figure 4. It shows a conversation plan used by an agent to:

1. Construct a Negotiation Engine Problem for a situation. This can be guided interactively by the user for example using the Web interface provided by our implementation (state 1).

2. Use the Solvers to solve the problem (state 2). If the problem has no solution, reconstruction is attempted. If no reconstruction possible, terminate in a fail state (8).

3. If the problem can be solved and the solution (goals to be achieved and goals not to be achieved) includes non-controllable goals to be achieved, then for each such goal, try to get the commitment of the owner of the goal for achieving it on behalf of the agent (state 3). Obtaining this commitment triggers a new conversation with the owner. The current conversation is suspended until all the commitment obtaining conversation terminate.

4. If some commitments can not be obtained, try to modify the problem by excluding the problematic goal (e.g. by posting a constraint that sets it to 'off') (state 5). Then solve again.

5. If all commitments have been obtained, execute the actions required for achieving the goals. For non-controllable goals, request execution form owner (state 4).

6. If all action execution were successful, terminate in a success state (7). Else, terminate in a fail state (6).

3 Encoding MAUT

We can encode multi-attribute problem specifications in the language of the Negotiation Engine and use the engine to negotiate about domains described in the multi-attribute utility theoretic language.

Encoding attributes. Let $A = a_1, a_2, ...a_n$ be a set of attributes shared by a number of agents. The attributes describe relevant aspects of the situation. In a situation where a seller agent negotiates with a buyer agent about a product, interesting attributes can be the $price$, the $deliveryTime$, the $quality$ etc. of the product. The domain of an attribute a_i, D_{a_i} is an interval $[l, h]$ where l and h are integers or reals. The domain describes the range of values that the attribute can take. Each value in the domain is assumed to be measured in a unit specific to the attribute. Thus, if the attribute is $price$, a domain $D_{price} = [5, 100]$ measured in dollars specifies that the $price$ can be anywhere between \$5 and \$100. A domain $D_{deliveryTime} = [1, 5]$ measured in weeks specifies that the $deliveryTime$ can be anywhere between 1 and 5 weeks. We assume agents interact by exchanging multi-attribute specifications formed by means of a shared set of attributes that have shared domain specifications. Each agent that has an attribute a_i also has a utility function $U_{a_i} : D_{a_i} \rightarrow [0, 1]$. The utility function expresses the agent's valuation of each value in the attribute's domain. A seller agent for example may assign high utilities to high prices, where a buyer agent may assign high utilities to lower prices.

We assume that an agent's utility function has the form shown in figure 5. That is, the domain of the attribute can be decomposed into a set of disjoint sub-intervals that cover the entire domain, such that on each sub-interval the utility is constant. Any utility function can be approximated in this way. (We will see that the fewer subintervals, the easier it is for the negotiation method to converge). Let $D_{a_i} = [i_0, i_1) \cup [i_1, i_2) \cup ...[i_{n-1}, i_n]$ be a decomposition of D_{a_i} int n subintervals such that $i_0 = l$, $i_n = h$ and

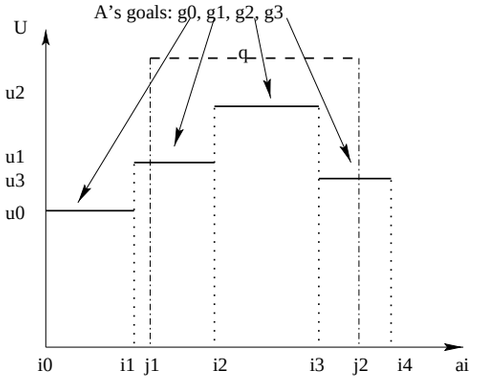


Fig. 5. Representing and encoding attributes.

for any $x \in [i_l, i_{l+1})$ we have $U_{a_i}(x) = u_l$ (figure 5). For each domain sub-interval $[i_l, i_{l+1}]$ we create an atomic goal $g^l_{a_i}$ which is 'on' iff the value of a_i is in the subinterval $[i_l, i_{l+1})$. As the subintervals cover the domain and are disjoint, in any situation only one of these goals can and should be 'on'. This is enforced by posting, for each attribute a_i, the constraint $On(xa_i)$ where $xa_i = xchoice(g^0_{a_i}, g^1_{a_i}, ...g^{n-1}_{a_i})$ (we also call these attribute encoding constraints). The utility function of a_i is translated into a Negotiation Engine utility list where $u_{on}(g^l_{a_i}) = u_l$ and $u_{off}(g^l_{a_i}) = 0$.

Acceptability constraints. An agent may have its own constraints about what attribute values or combinations of values are acceptable. For example, an agent may only accept $deliveryTime \in [1, 3]$ which may be represented as the constraint $On(dT)$, where (assuming a 2 subinterval decomposition) $dT = xchoice(g^1_{deliveryTime}, g^2_{deliveryTime})$. Or an agent may accept to pay more than \$50 only if the quality is greater than some given limit. A proposal from another agent will not be accepted unless all acceptability constraints are satisfied.

MAUT Problem. A MAUT problem is a Negotiation Engine problem whose goals are all the goals generated for all attributes of interest, whose constraints are all the attribute encoding constraints plus all the acceptability constraints and whose utility list is obtained by merging the utility lists of each encoded attribute. A solution of a MAUT problem is an on-off assignment to the goals of the problem that satisfies all constraints. The optimal solution is the solution that has maximum utility. We also call a solution to the MAUT problem a *deal*.

Let s be a solution to agent A's MAUT problem and a_i an attribute of the problem. Because of the attribute encoding constraint, one and only one of the goals associated with the subintervals of the attribute will be 'on' in s. The subinterval associated with this goal defines the acceptable set of values for the attribute in the given solution in the sense that any value in the set is equally acceptable to the agent. Let now s_A and s_B be solutions to A's, respectively B's MAUT problem. The two solutions intersect iff for each attribute a_i the acceptable sets of values for the two agents have a non-empty intersection. This is described graphically in figure 5. Assuming $[i_3, i_4]$ is the set of acceptable values

for A (coming from A's goal g^3 being 'on') and the set $[j_1, j_2]$ is the set of acceptable values for B (coming from one of B's goals that we do not represent), then $[i_3, j_2]$ is the non-empty intersection for the represented attribute. If such an intersection exists for each attribute, then the two solutions intersect. The existence of intersecting solutions represents a possible agreement between the agents, because each solution contains ranges of values acceptable to each agent in part.

4 The Negotiation Process

Assume we have two negotiating agents A and B. Each agent represents its problem as a MAUT problem, with its attributes, goals, constraints and utilities. For each agent, an acceptable solution specifies for each attribute in part an interval of acceptable values that the attribute can take. The branch and bound solver of the Negotiation Engine allows an agent to generate its solutions in decreasing order of utility. The first (best) solution is obtained by running the solver on the original constraints. The next best solution is obtained by logically negating the previous best solution, adding it as a constraint and solving the resulting more constrained problem. Because a solution s is logically the conjunction of all subinterval goals that are 'on', the negation of a solution is a disjunction (choice) of the negations of each 'on' subinterval goal in the solution. In other words, the next solution must differ from the current one in at least one goal (subinterval). For example, if in a 2 attribute problem $s_k = g^i, g^j$ is the k^{th} solution, then $ns_k = choice(-g^i, -g^j)$ is its negation and the posted constraint is $On(ns_k)$.

This use of the solver, in which each solution is negated and added as a constraint for the next round allows each agent to generate all its acceptable solutions in decreasing order of utility. For each agent, the negotiation process takes place as shown by the conversation plan in figure 6.

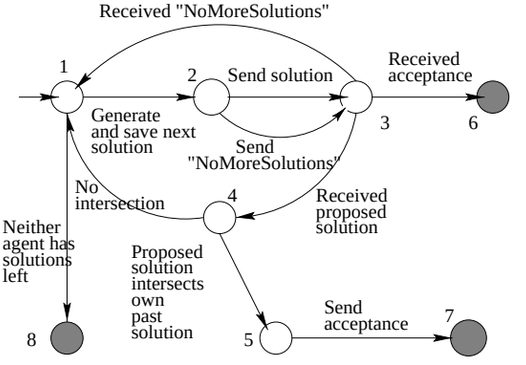
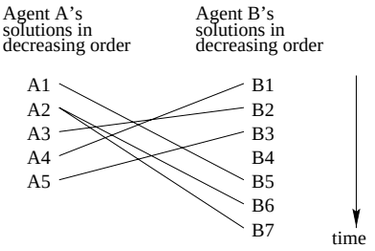


Fig. 6. Negotiation protocol as a conversation plan.

Each agent generates its current best solution for as long as it has solutions to generate (state 1). Each solution is saved locally. The solution is also sent to the other agent (state

2). Then the process waits for a message from the interlocutor (state 3). If the message is an acceptance (state 6), it signals that the sent solution is consistent (has a non-empty intersection) with one of the interlocutor's past solutions. In this case the intersection is a mutually acceptable deal, which terminates the negotiation successfully. If the received message is "NoMoreSolutions", the other agent has run out of solutions to generate. If the same is true for this agent as well, then the negotiation ends unsuccessfully. Otherwise the agent that still has solutions will continue to generate them. Finally, if the message contains a proposed solution from the interlocutor, this solution is checked for compatibility (intersection) with any of the past solutions generated by this agent (state 4). If an intersection is found (state 5), it represents a mutually acceptable deal, which terminates the negotiation successfully. Otherwise the top loop is resumed. The process ends when either a deal acceptable to both parties is found or when both agents have run out of solutions. Figure 7 shows the interaction between two agents using this protocol.



Connections define intersecting pairs of solutions of A and B.
(A3, B2) is the Pareto optimum.

A		B
Generate and send A1	⟷	Generate and send B1
B1 not intersecting A1		B1 not intersecting A1
Generate and send A2	⟷	Generate and send B2
B2 not intersecting {A1, A2}		A2 not intersecting {B1, B2}
Generate and send A3	⟷	Generate and send B3
B3 not intersecting {A1, A2, A3}		A3 intersects B2 - deal!

Fig. 7. The negotiation process.

The most important feature of the algorithm is that it guarantees the discovery of the Pareto optimum. To prove that, assume that the discovered deal is (A_k, B_l) and is not Pareto optimal. Then another deal must be Pareto optimal. The Pareto optimal deal must involve a pair of solutions (A_i, B_j) such that $i \leq k$ and $j \leq l$, because solutions are generated in decreasing utility order. But then any pair (A_i, B_j) with $i \leq k$ and $j \leq l$ has already been generated and verified. Hence it is only possible that $i = k$ and $j = l$.

5 Negotiating about Electronic Components

To demonstrate the applicability of the generic negotiation architecture we now describe a negotiation process for the electronic component suppliers and their OEM customers in the Electronic Design Automation (EDA) industry [15]. The process conforms to the QuickData Evaluation and QuickData Protocol specifications developed by the Electronic Component Information Exchange (ECIX) project [14], designed to facilitate the creation, exchange and evaluation of electronic components information between supply chain partners. These two specifications, however, do not specify any negotiation process that could be used by customers and suppliers to agree on an acceptable "deal". In the remainder of this section, we show how we have extended the specifications to provide a multi-attribute negotiation process by describing a negotiation scenario between a customer and a supplier agent.

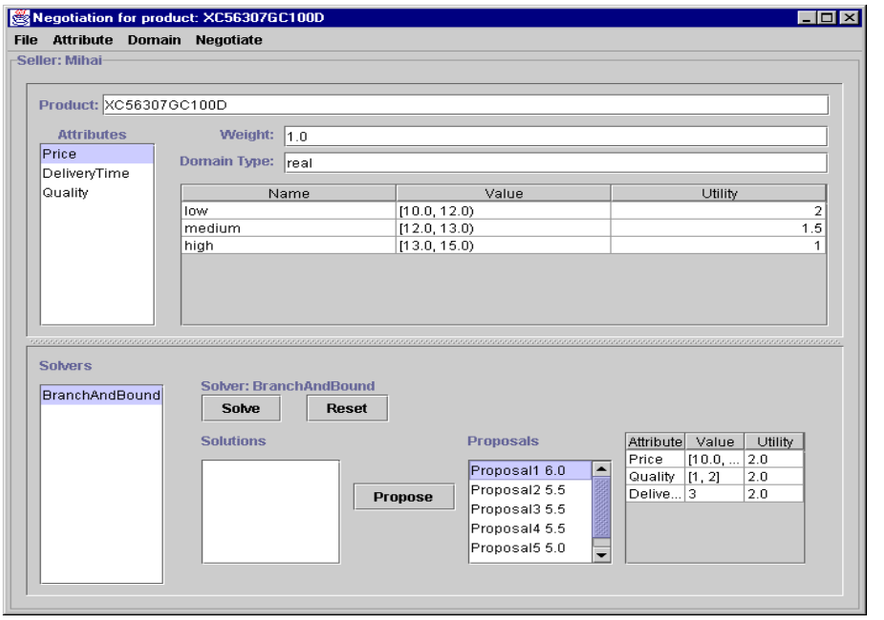


Fig. 8. Negotiation panel for Wai-Kau.

The negotiation scenario involves a customer agent, named "Wai-Kau", and a supplier agent, named "Mihai". Using the QuickData Evaluation protocol, agent Wai-Kau has identified the electronic component with part number XC56307GC100D that it wants to buy from agent Mihai. Agent Wai-Kau uses the GUI shown in figure 8 to specify three attributes that he wants to negotiate with agent Mihai regarding the purchase of the component. The three attributes are *Price* (dollar), *Delivery Time* (week), and *Quality* (number of defects) and their corresponding values and utilities are as follows:

Wai-Kau's proposals

⟨ [10.0, 12.0), 3, [1,2] ⟩, 6.0

⟨ [10.0, 12.0), [1,2], [1,2] ⟩, 5.5

⟨ [10.0, 12.0), 3, [3,4] ⟩, 5.5

⟨ [12.0, 13.0), 3, [1,2] ⟩, 5.5

⟨ [10.0,12.0), [1,2], [3,4] ⟩, 5.0

⟨ [10.0,12.0), 3, [5,6] ⟩, 5.0

Mihai's proposals

⟨ [13.5, 15.0), [2,3], [5,6] ⟩, 6.0

⟨ [12.5, 13.5), [2,3], [5,6] ⟩, 5.6

⟨ [13.5, 15.0), [2,3], 4 ⟩, 5.5

⟨ [13.5, 15.0), [4,5], [5,6] ⟩, 5.5

⟨ [11.5, 12.5), [2,3], [5,6] ⟩, 5.3

⟨ [11.5, 12.0), 3, [5,6] ⟩, 5.3

Fig. 9. Negotiation trace

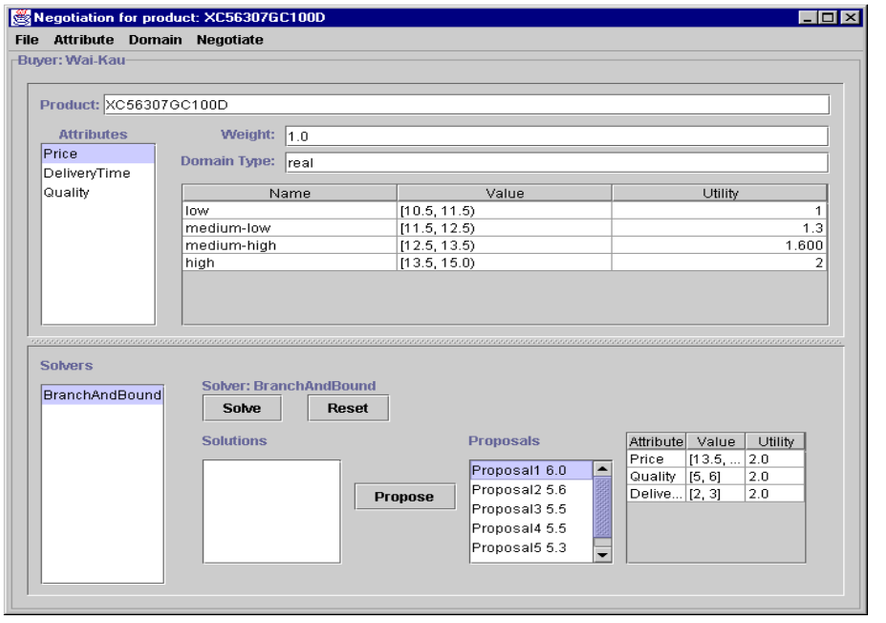


Fig. 10. Negotiation panel for Mihai.

- Price: Low [10, 12.0), Utility 2.0; Medium [12.0, 13.0), Utility 1.5; High [13.0, 15.0), Utility 1.
- Delivery Time: Earliest [1,2], Utility 1.5; Optimal 3, Utility 2; Latest [4,5], Utility 1.
- Quality: High [1,2], Utility 2; Medium [3,4], Utility 1.5; Low [5,6], Utility 1.

Having specified the desirable values and utilities for the attributes, agent Wai-Kau then invokes the negotation engine to start a negotiation conversation with agent Mihai.

Upon receiving the message sent by agent Wai-Kau, agent Mihai must first specify its own values and utilities for the three negotiable attributes. In the example, agent Mihai uses the following values and utilities (figure 10 shows his GUI).

- Price: Low [10.5, 11.5), Utility 1.0; Medium-Low [11.5, 12.5), Utility 1.3; Medium-High [12.5, 13.5), Utility 1.6; High [13.5, 15.0), Utility 2.0.
- Delivery Time: Earliest 1, Utility 1; Optimal [2,3], Utility 2.0; Latest [4,5], Utility 1.5.
- Quality: High [1,2], Utility 1; Medium-High 3, Utility 1.3; Medium-Low 4, Utility 1.5; Low [5,6], Utility 2.

With these specifications in place, agent Mihai then invokes the negotiation engine to join the negotiation conversation initiated by agent Wai-Kau.

Figure 9 shows the sequence of proposals exchanged between agents Wai-Kau and Mihai. For each of them in part, these are shown in their GUI-s (figures 8 and 10). Note that none is aware of the valuations (utility function) of the other. The negotiation conversation terminates with the following acceptable deal: Price: [11.5, 12.0), Delivery Time: 3, Quality: [5,6]. With respect to this proposal, agent Wai-Kau has utility 5.0, while agent Mihai has utility 5.3.

6 Conclusions

One obvious limitation of the presented solution is the fact that each agent generates proposals in decreasing order of their own utility alone. This means that combinations like ‘lowest price and highest quality’ may top the list of a buyer, while ‘lowest quality and highest price’ may top the list of the seller. As none of this is likely to be accepted, this introduces empty zones in the message exchange that only waste both agents time. The solution we are investigating is to have agents also consider the probability that a proposal would be accepted by the other agent. Once this probability known, we can either eliminate all proposals with probability less than a threshold, or order proposals according to their expected utility. The approach is to model attributes as discrete random variables whose values are the subintervals in which their domain is decomposed and to use Bayesian Network inference [10] to calculate the probability of any combination of attribute values. One problem with this approach is that both prunning and reordering risk loosing the Pareto optimality, as this is guaranteed only if all solutions are communicated in the agent’s decreasing order. It would be interesting to see if we can generate probability thresholds for prunning that in most cases would not endanger

Pareto optimality. Another problem has to do with how we can use probability information inside the branch and bound solver either for skiping low probability solutions or for early detecting of low expected utility solutions.

Even with probabilistic reasoning included, the method may still generate a great number of exchanged message (at worst, each agent may produce an exponential number of proposals). To avoid transmitting all of these over the network, we are considering mediator agents that would accept the utility functions of all agents, construct and run the negotiation internally and communicate to parties the final result. This requires agents to trust that the mediator would not divulge their utility functions (these are not disclosed in the current solution).

Another variant of the idea of having an agent generate proposals that are more likely to be accepted by the other party is explored in [5]. Their idea is that when an agent has several equally good offers (from its own perspective), the one to be proposed should be the one that is most preferable to the negotiation partner. Since the partner's utility function is not known, they propose to use fuzzy similarity measures to select an offer that is similar to a previous offer made by the partner. Our framework can be extended to use this idea as follows. First, we need to modify the search procedure to produce all solutions that have the same utility (perhaps by allowing all solutions whose utilities are within a δ from each other). From these solutions, the agent would then select one that is more similar to any of the previous offers made by the interlocutor. As the agent knows the order in which the interlocutor has made its offers, it also knows that the more similar the chosen solution to an earlier offer, the better it will be for the other side - thus the agent can fine tune how benevolent it wants to be.

The fundamental issue with this approach, and with ours as well, is ensuring that the negotiation partner will be equally benevolent. In other words, we must find ways to discourage partners from deviating from the prescribed behavior. As far as our framework is concerned, we will show in future papers how this problem can be addressed.

From the implementation point of view, the Negotiation Engine is part of our Java-based JCOOL multi-agent framework. This is a ground-up re-implementation of the Lisp-based Agent Building Shell [3]. The new implementation adds a host of new extensions that facilitate the construction and execution of complex multi-agent applications. One significant extension is the integration of various web technologies, e.g., XML/XSL, HHTP, and Servlet, into the new framework. This extension allows us to build agents that carry out conversations with other agents or human users over the Web, with dynamically generated web interfaces adapted to the context of the interaction. Finally, being written in Java, the framework and applications built on top of it can be deployed in any system that supports a Java Virtual Machine. This significantly expands the applicability of the JCOOL multi-agent framework.

Acknowledgements. This research is supported, in part, by Materials and Manufacturing Ontario, Mitel Corp., Metex Systems, Communications and Information Technology Ontario and the Natural Science and Engineering Research Council of Canada. The authors wish to thank an anonymous reviewer for the useful comments made about the paper.

References

1. Barbuceanu, M. 1998. Agents that work in harmony by knowing and fulfilling their obligations. *Proc. of AAAI-98*, Madison, WI, 1998.
2. Barbuceanu, M. and Fox, M. S. 1997. Integrating Communicative Action, Conversations and Decision Theory to Coordinate Agents. *Proceedings of First International Conference on Automomous Agents (Agents'97)*, 47-58, Marina Del Rey, February 1997.
3. Barbuceanu, M. and Fox, M.S. The Architecture of an Agent Building Shell, In M.Woolridge, J.P. Mueller and M. Tambe (eds), *Intelligent Agents II: Agent Theories, Architectures and Languages*, Springer Verlag Lecture Notes in Artificial Intelligence Vol. 1037, 1996.
4. Doorembos, R., Etzioni, O., Weld, D. A Scalable Comparison-Shopping Agent for the World Wide Web. *Proceedings of First International Conference on Autonomous Agents (Agents'97)*, Marinal del Rey, CA, 1997.
5. Faratin, P., Sierra, C. and Jennings, N. 2000. Using Similarity Criteria to Make Negotiation Trade-Offs. *Proceedings of the Fourth Intl. Conference on MultiAgent Systems*, Boston, MA, July 2000, 119-126.
6. Finin, T. et al. 1992. Specification of the KQML Agent Communication Language. The DARPA Knowledge Sharing Initiative, External Interfaces Working Group.
7. Gutman, R., Moukas,A., and Maes, P. Agent Mediated Electronic Commerce: A Survey. *Knowledge Engineering Review*, June 1998.
8. Jiang, Y., Kautz, H. and Selman, B. Solving Problems with Hard and Soft Constraints Using a Stochastic Algorithm for MAXSAT. *First International Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, Oregon, 1995.
9. Keeny,R. and Raiffa, H. Decisions with Multiple Objectives: Preferences and Value Tradeoffs. *John Willey & Sons*, 1976.
10. Pearl, J. Probabilistic Reasoning in Intelligent Systems, *Morgan Kaufmann*, 1988.
11. Selman, B., H.J. Levesque and D. Mitchell. 1992. A new method for solving hard satisfiability problems. *Proceedings of AAAI-92* San Jose, CA, pp. 440-446.
12. Sycara, K. The PERSUADER. In *The Encyclopedia of Artificial Intelligence*, D. Shapiro (ed), JohnWilley & Sons, January 1992.
13. Yokoo, M. Ishida, T., Durfee, E. and Kuwabara, K. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. *Proceedings of* 12^{th} *IEEE Conference on Distributed Computing Systems*, 1992, pp 614-621.
14. www.si2.org/ecix
15. www.edat.com

On Constraint-Based Reasoning in e-Negotiation Agents

Ryszard Kowalczyk and Van Bui

CSIRO Mathematical and Information Sciences,
723 Swanston Street, Carlton 3053, Australia,
Ph: +61-3-8341 8216, Fax: +61-3-8341 8222
ryszard.kowalczyk@cmis.csiro.au

Abstract. Negotiation typically involves a number of parties with different criteria, constraints and preferences that determine the individual areas of interest, i.e. the range and order of the preferred solutions of each party. The parties usually have a limited common knowledge of each other's areas of interest. Therefore a range of possible agreements, i.e. the common area of interest is typically not known to the parties a priori. In order to find a mutual agreement the parties explore possible agreements by the process of exchanging information in the form of offers. During the negotiation process the range of possible offers of each party changes according to the current information available. As negotiation progresses and more information become available the ranges reduce until an agreement can be found or the parties withdraw from negotiation. This interpretation allows one to consider the negotiation problem as a constraint satisfaction problem and the negotiation process as constraint-based reasoning. This paper presents some aspects of that interpretation. In particular it outlines the constraint-based representation and constraint propagation mechanisms used in an experimental system of e-Negotiation Agents (eNAs). The eNAs can autonomously negotiate the multi-issue terms of transactions in an e-commerce environment tested with the used car trading problem.

1 Introduction

Negotiation is a form of decision making where two or more parties jointly explore possible solutions in order to reach a consensus [15]. Negotiation has traditionally been a subject of study in the game-theoretic [19], economics [3, 6] and management science [10] research. It has also been an active area of research in Artificial Intelligence (AI) and in particular in distributed AI (DAI) and multi-agent systems (MAS) (e.g. [4, 9, 14, 16, 18, 21]). The increased potential of AI technology in supporting and automating negotiation has been recognized in a wide range of real-world problems including group conflict resolution [13, 18], business negotiations [22], resource allocation and scheduling [17] and e-commerce [1, 4, 24, 25].

Negotiation typically involves a number of parties with different criteria, constraints and preferences that determine the individual areas of interest, i.e. the range and order of the preferred solutions of each party. The parties usually have a limited common knowledge of the criteria, constraints and preferences of each other.

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 31-46, 2001.

Therefore the range of possible agreements, i.e. common area of interest (also called a negotiation set [19]), which is an intersection of the individual areas of interests, is typically not known to the parties a priori. In order to find a mutual agreement that satisfies all the parties, i.e. a solution from the common area of interest, the parties move towards and explore possible agreements by the process of exchanging information. They exchange information in the form of offers[1], i.e. currently preferred solutions from the individual areas of interest. During the negotiation process the range of possible offers of each party changes according to the current information available. Typically as negotiation progresses and more information becomes available the ranges reduce until an agreement can be found or the parties withdraw from negotiation. In other words if the common area of interest reduce to one solution then this is an agreement and negotiation is successful. Otherwise it becomes empty and the agreement is not possible.

It should be noted that the information available to the parties during negotiation act as constraints that progressively reduce the individual areas of interest towards an agreement within the common area of interest. In this context the objective of negotiation can be to find any solution from a common area of interest or to find a solution within a common area of interest that optimizes some objectives of the parties (e.g. maximizes the gain). This interpretation allows one to consider the negotiation problem as a constraint satisfaction problem (CSP) and the negotiation process as constraint-based reasoning [8, 12]. CSPs form a class of general problems defined by a set of variables with the associated domains and a set of constraints acting on the variables with the objective of finding an instantiation of the constrained variables such that all constraints are satisfied at the same time. Constraint-based reasoning is often used in solving CSPs with the principles of constraint satisfaction, constraint consistency and constraint propagation [8, 12].

This paper presents some aspects of the constraint-based reasoning in the negotiation problem considered as a distributed CSP. In particular it outlines the constraint-based representation and constraint propagation mechanisms used in an experimental system of e-Negotiation Agents (eNAs) [5, 7]. The agents can autonomously negotiate the multi-issue terms of transactions in an e-commerce environment tested with the used car trading problem. The eNAs are rational and self-interested in the sense that they are concerned with achieving the best outcomes for themselves. They are not interested in social welfare or outcomes of other agents (as long as they can agree on a solution). The rationality of the agents is bounded by the availability of the information and computational resources. It means that the agents try to achieve as good outcome as possible. In other words they do not have always the information and computational resources to obtain the theoretically optimal outcome (e.g according to the game theoretical results).

It should be noted that some principles of constraint-based reasoning have also been proposed in the Tete-a-Tete agent system [4, 24] to support cooperative negotiation across multiple terms of transactions. However the Tete-a-Tete agents perform a semi-automated argumentative negotiation in the sense that they aim at

[1] In general the parties may exchange any information including the evaluation criteria, constraints and preferences. If available they may also acquire any additional information from the environment (e.g. third parties) that may assist in negotiation (e.g. utility distribution, reservations). In this paper without loosing the generality we limit the information exchanged (and the common knowledge) to the offers only.

automating the negotiation process for merchants and assisting shoppers during negotiations with multiple sellers. The Tete-aTete shopping agent evaluates and orders the offers received from the sellers, and presents them to the user for consideration. Then it broadcast the user's critiques to the selling agents in order to receive better offers [4, 24]. The eNAs described in this paper are self-interested agents that autonomously evaluate, generate and exchange multi-issue offers on behalves of both the sellers and the buyers.

The paper is organized as follows. Section 2 defines a constraint-based representation of negotiation as a distributed CSP. Section 3 presents some aspects of constraint-based mechanisms of negotiation used by the eNAs agents. An overview of the eNAs system demonstrator is given in section 4. Section 5 presents some implementation aspects and the results of some comparative experiments for selected trading situations with a combination of different negotiation strategies. Finally the concluding remarks and an outline of future work are presented in section 6.

2 Constraint-Based Representation of Negotiation

Negotiation involves a number of parties with different criteria, constraints and preferences that need to be satisfied in order to get an agreement between the parties. This implies that negotiation may be modeled as a constraint satisfaction problem (CSP) [8, 12] and in particular as a distributed CSP (DCSP) [11, 17, 20]. In general, CSPs are defined by a set of variables with the associated domains and a set of constraints acting on the variables with the objective of finding an instantiation of the constrained variables such that all constraints are satisfied at the same time. In DCSPs the variables and/or constraints are distributed among the agents that exchange the coordination information in order to solve a given problem. In the context of negotiation the individual constraints are partitioned between the parties and information is exchanged in the form of offers, i.e. the preferred instantiations of the variables corresponding to the issues of negotiation. It should be noted that the offers already exchanged between the parties influence (i.e. constrain) the negotiation process and the future decisions of the parties. For example a rational negotiator would not propose an offer of the lower value than a value of the offers received already from another party.

More formally negotiation can be represented in the terms of a DCSP as follows:

- a set of variables $\mathbf{X}$ consisting of all decision variables distributed among p negotiating parties, i.e. $\mathbf{X} = \bigcup_{j=1,..,p} x^j$, where each $x^j = \{x_i^j\}, \quad i = 1,\ldots,n_j$ is a set of the decision variables of the j^{th} party. It includes a non-empty set of the issues of negotiation $x \subseteq \mathbf{X}$ that are the variables shared between the parties[2], i.e. $x = \bigcap_{j=1,..,p} x^j$.

[2] The parties may have private decision variables that are not the issues of negotiation and/or may become the negotiation issues. Without loosing the generality in the subsequent sections of the paper we will assume that all parties have the same decision variables that are the issues of negotiation, i.e. $x = \mathbf{X}$.

- a set of domains $\mathbf{D} = \{D_i^j\}$ for the variables of all the parties $\mathbf{X} = \{x_i^j\}$, $i = 1,\ldots,n_j$, $j = 1,\ldots,p$, where each domain D_i^j consists of a set of values that a variable x_i^j may assume, i.e. $x_i^j \in D_i^j$. Each party can have private (and different) domains for the variables including the issues of negotiation.
- a set of constraints $\mathbf{C}$ consisting of all constraints distributed among p negotiating parties, i.e. $\mathbf{C} = \bigcup_{j=1,..,p} C^j$, where each $C^j = \{C_k^j\}$, $k = 1,\ldots,m_j$ is a set of constraints between the variables of the j^{th} party. A constraint $C_k^j, k = 1,\ldots,m_j$, $j = 1,\ldots,p$ relating a subset of the variables $\{x_{i_1}^j,\ldots,x_{i_l}^j\} \subseteq x^j$ is defined as a relation $C_k^j(x_{i_1}^j,\ldots,x_{i_l}^j)$ on the Cartesian product space $D_{i_1}^j \times \ldots \times D_{i_l}^j$ representing a set of assignments to the variables that satisfy the constraints. Similarly the sets of constraints C^j and $\mathbf{C}$ are defined as relations $C^j(x^j) = \bigwedge_{k=1,..,m_j} C_k^j(x^j)$ and $\mathbf{C}(\mathbf{X}) = \bigwedge_{j=1,..,p} C^j(x^j)$ on the respective Cartesian product spaces.

A solution of a DCSP is an instantiation of all the decision variables such that all the constraints of the parties are satisfied. In general the objective of a DCSP can be to check if a solution exists, to find a solution, to find all solutions or to find the solution that satisfies some other criteria (e.g. maximizes an objective function). In negotiation the sets of constraints C^j and $\mathbf{C}$ prescribe the preferred solutions of each party (individual areas of interest) and the possible joint solutions of negotiation (common area of interest), respectively. In this context the objective of negotiation can be to find any solution from a common area of interest or to find a solution within a common area of interest that optimizes some objectives of the parties (e.g. maximizes the gain).

Figure 1 illustrates an example of a constraint based representation of the negotiation problem involving two parties *a* and *b*. $C^a(x)$ and $C^b(x)$ define individual areas of interest (multi-dimensional in general) of the parties *a* and *b*, respectively. *x** is a solution from an intersection of the individual areas of interest, i.e. the common area of interest, defined by a conjunctive combination $\mathbf{C}(x) = C^a(x) \wedge C^b(x)$.

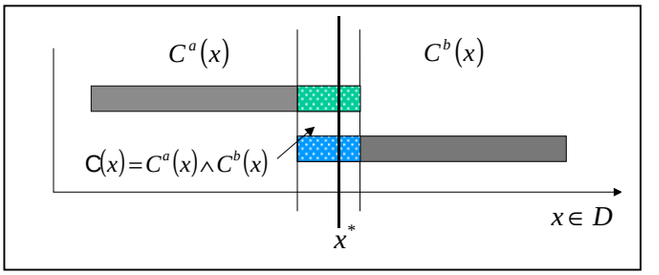


Fig. 1. A constraint-based model of negotiation involving two parties *a* and *b*

It should be recalled that the common area of interest is not known to the negotiating parties a priori. Therefore the main challenges in negotiation are to move towards and to explore potential agreements within the common area of interest. The mechanisms of such a process used by the eNAs are outlined in the following sections.

3 Constraint-Based Mechanisms of Negotiation

Negotiation is a process by which a joint decision is made by two or more parties which first verbalize contradictory demands and then move towards agreements by a process of concession making or search for new alternatives [2, 15]. Typically each party starts negotiation by offering the most preferred solution from the individual area of interest. If an offer is not acceptable by other parties they make counter-offers in order to move them closer to an agreement. During that process the ranges of available options for each party reduce until an agreement can be reached. It can occur when the individual areas of interest of each party become the common area of interest, i.e. they are the same for each party. At this stage the parties can accept any solution or may choose to continue negotiation in order to find better solution/agreement from the available range. Figure 2 illustrates such a process in a simple buyer-seller negotiation setting.

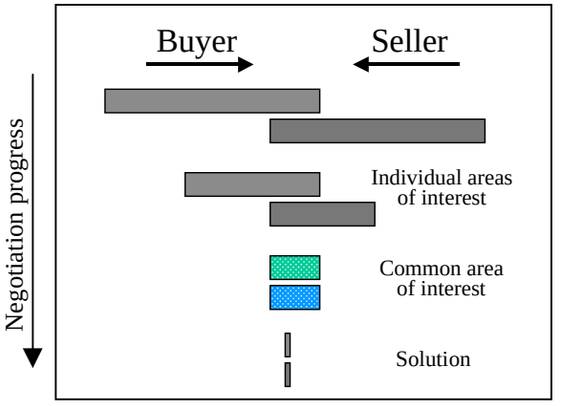


Fig. 2. Buyer-Seller negotiation

In the context of constraint-based reasoning the reduction of the ranges of possible solutions results from the constraint propagation in order to ensure the constraint consistency and satisfaction. Many constraint propagation methods have been developed to provide constraint satisfaction at different levels of constraint consistency such as well known AC-1, AC-2, AC-3 and others [8, 12]. In general they eliminate unfeasible solutions from a search space, i.e. the values that cannot satisfy the constraints are removed from the domains of the variables. This prunes the search space to one that consists only of the possible solutions satisfying the constraints at a required level of constraint consistency. These principles are used by the eNAs at the level of arc-consistency to maintain the ranges of options for each

issue/variable considered by each agent during the negotiation process. In particular they support the mechanisms of evaluation and assessment of the acceptability of solutions, and generation of an initial offer and the consecutive counter offers if the received offers are unacceptable. Some aspects of these mechanisms used by the eNAs are briefly described in the following sections.

3.1 Solution Evaluation

One of the fundamental tasks during negotiation is evaluation of solutions considered by the parties. It can involve assessment of a value of an offer received by a party in order to evaluate its acceptability. It is also required in ordering available solutions within the individual area of interest in order to identify and propose the currently preferred counter offer.

There are many methods of evaluating solutions based on utility theory, decision theory, game theory and others (eg. [6]). In particular a value v of an offer (potential solution) consisting of a number of issues $x = \{x_1,\ldots,x_n\}$ can be defined as a function of the values of the considered issues. For example using the utility theory framework [6, 18] the value function can be represented as an additive utility function as follows: where x_i is the i^{th} issue of negotiation, v_i is a utility (value) function of the i^{th} issue and

$$v(x_1,\ldots,x_n) = \sum_{i=1,..,n} w_i v_i(x_i), \quad \sum_{i=1,\ldots,n} w_i = 1$$

w_i is a weighting factor (e.g. priority) of the value of the i^{th} issue. The utility theory provides a general framework for assessing, comparing and ordering alternative solutions based on their utility value (e.g. see [6, 18])

It should be noted that even though the parties may use the same form of the utility function they do not have common knowledge about it. For example the eNAs consider the value of an offer expressed in dollars where the value functions for the corresponding three issues in the used-car test problem can be defined for the seller and buyer as in Table 1.

It should also be noted that in the context of constraint-based mechanisms the utility function relates (i.e. constrains) the issues of negotiations in respect to the value of a solution. For example the eNAs use it as a constraint on the possible trade-offs between the issues during the offer generation (see section 3.3).

Table 1. Value functions for seller and buyer

Negotiation issue	Buyer	Seller
$x_1 \equiv$ price	$v_1^B(x_1) = -\text{price}$	$v_1^S(x_1) = \text{price}$
$x_2 \equiv$ warranty	$v_2^B(x_2) = \alpha_w^B \text{warranty}$	$v_2^S(x_2) = -\alpha_w^S \text{warranty}$
$x_3 \equiv$ trade-in	$v_3^B(x_3) = \text{trade-in}$	$v_3^S(x_3) = -\text{trade-in}$
$x = \{x_1, x_2, x_3\}$	$v^B(x_1, x_2, x_3) = -w_p^B \text{price} + w_w^B \alpha_w^B \text{warranty} + w_t^B \text{trade-in}$	$v^S(x_1, x_2, x_3) = w_p^S \text{price} - w_w^S \alpha_w^S \text{warranty} - w_t^S \text{trade-in}$

3.2 Acceptability Criteria

In general a solution can be accepted if it satisfies the acceptability criteria of each party. The criteria can involve an acceptable solution value and other constraints such as reservation prices. In our approach each party specifies its preferred values for the issues of negotiation and its desirability to make a deal. An acceptable solution value is then calculated as follows:

$$v^* = d \cdot v(x^*)$$

where v^* is an acceptable solution value, $v(x^*)$ is a value of the preferred solution $x^* = \{x_1^*, \ldots, x_n^*\}$, and d is a coefficient expressing desirability of a party to make a deal. In general the higher desire to make a deal the lower solution value that can be accepted[3]. In general the desire level can also influence the negotiation strategy and behaviour of the parties. For example the higher desire the higher level of concession a party is willing to consider for new offers (see section 3.3).

A party can accept an offer if its value is equal or larger than an acceptable value, i.e. $v(x) \geq v^*$. In addition a party may consider other criteria that can influence acceptability of an offer such as the constraints on the issues of negotiation (e.g. reservation prices). For example the acceptability criteria used by the eNAs agents are defined in Table 2.

Table 2. Acceptability criteria for the seller and buyer agents

buyer B	seller S
$v^B(price^S, warranty^S, tradein^S) \geq v^{B*}$ $price^S \leq price^{B*}$ where: $v^{B*} = d^B v^B(price^{B*}, warranty^{B*}, tradein^{B*})$	$v^S(price^B, warranty^B, tradein^B) \geq v^{S*}$ $price^B \geq price^{S*}$ where: $v^{S*} = d^S v^S(price^{S*}, warranty^{S*}, tradein^{S*})$

It should be noted that the acceptability criteria also play an important role as constraints on the ranges of available solutions during generation of offers. These constraints for example ensure that any offer proposed by a party is also acceptable by the party itself (see section 3.3).

3.3 Offer Generation

Generation of offers is the main decision making process that directs the progress of negotiation and its outcomes. It involves search for prospective solutions from the individual area of interest that move the parties towards an agreement from the common area of interest. The search is typically guided by the negotiation strategies of each party, i.e. the rules for generation of offers taking into account the information available to the party including the individual criteria, constraints and preferences as well as the previous offers and counter-offers.

---

[3] In [18] it has been indicated that an agreement on 70% of the preferred solution value or higher is typically acceptable in the real-world negotiations. In our approach the eNAs agents calculate the desirability coefficient as a function of the desire level, i.e. $d = 1 - 0.3desire$, where $desire \in [0, 1]$.

Considering all information available to a party as constraints on the individual area of interest the mechanisms of constraint-based reasoning (i.e. constraint propagation) can assist the search process by reducing the range of offers available to the party. In other words it can prune the search space to one that consists of feasible solutions satisfying the constraints of the party at any stage of the search process. In this context generation of offers involves both constraint consistency maintenance and searching for the values of issues to be offered.

Constraint Consistency Maintenance. Constraint consistency maintenance involves posting constraints corresponding to the information available to a party so they can be propagated during the search process in order to ensure the individual area of interest is consistent with the constraints of the party. It can include the constraints available to the party before and during the negotiation process. For example some constraints corresponding to the initial and exchanged information used by the eNAs during negotiation are listed in Table 3.

Table 3. Constraints during negotiation

Initial information	Constraints
Negotiation issues $x=(x_1, x_2, x_3)$ e.g. x_1 ≡ price, x_2 ≡ warranty, x_3 ≡ trade-in	$C(x): x_i^{\min} \le x_i \le x_i^{\max}$
Information regarding the first offer price e.g. x_4 ≡ depreciation cost, x_5 ≡ usage cost	$C(x): x_1 \le x_1^{new} - x_4 - x_5$
Values of offer and issues	$C(x): x_{123} = v(x) = \sum_{i=1,\ldots,3} w_i v_i(x_i), \quad \sum_{i=1,\ldots,3} w_i = 1$
Acceptable value	$C(x): x_{123} \le v(x_1^{\max}, x_2^{\min}, x_3^{\min})$
Information exchanged	Constraints
Previous offers generated (x_1', x_2', x_3')	$C(x): x_{123} \ge v(x_1', x_2', x_3')$
Previous counter-offers received (x_1'', x_2'', x_3'')	$C(x): x_{123} \le v(x_1'', x_2'', x_3'')$

Constraint propagation ensures that the individual area of interest is consistent with the information (i.e. constraints) available to the party. Figure 3 illustrates simple examples of consistent domains of the variables related by the constraints on the first offer price and the values of offer and issues.

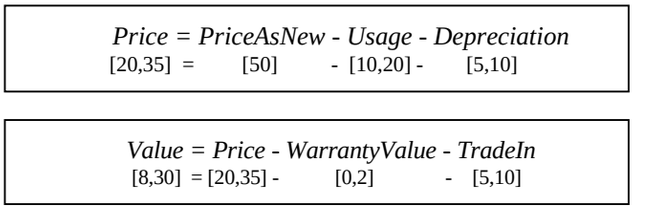


Fig. 3. Examples of constraint consistent domains

It should be noted that after the initial propagation the constraints are also dynamically propagated during the search for new offers to support finding instantiations of variables corresponding to the issues of negotiation.

Searching for New Offers. Search for new offers involves selection of a value of the offer to be generated and finding values of the issues of the offer according to the negotiation strategies used by a party. In general the negotiation strategies prescribe the rules of progressing negotiation towards an agreement. In particular they characterize the level of concession the party is willing to make during a given episode of negotiation. For example the eNAs can use a number of predefined strategies[4] to determine the concession level on the value of an offer to be generated as presented in Table 4.

Table 4. Concession-based negotiation strategies of eNAs

Negotiation strategy	Concession level Δ
Take-it-or-leave-it (one iteration only)	$\Delta = 0$
No-concession	$\Delta = 0$
Fixed concession	$\Delta = f\left(v\left(x_1^{\max}, x_2^{\min}, x_3^{\min}\right), desire\right) = const$
Simple concession	$\Delta = f\left(v\left(x_1, x_2, x_3\right), desire\right)$
Fixed concession (better deal)*	$\Delta = f\left(v\left(x_1^{\max}, x_2^{\min}, x_3^{\min}\right), desire\right) = const$
Simple concession (better deal)*	$\Delta = f\left(v\left(x_1, x_2, x_3\right), desire\right)$

* The better deal strategies are used if a party wants to continue negotiation after receiving an acceptable offer with a hope to get a better deal.

After selecting a value for a new offer, i.e. $x_{123}^t = x_{123}^{t-1} - \Delta$ the issues of negotiation need to be instantiated so their values form the offer value. In other words given x_{123}^t the values for $\left(x_1, x_2, x_3\right)$ need to be found such that $x_{123}^t = \sum_{i=1,..,3} w_i v_i\left(x_i\right)$. The eNAs use the Branch and Bound search with the support of constraint propagation to find instantiations that satisfy the constraints of the party. If there is a number of possible

[4] It should be noted that in general other strategies may be defined by the user (or possibly learned from the past experience).

instantiations then the agents choose a solution according to the preference order, i.e. an instantiation with the higher preferences is selected. It is also possible that there is no instantiation that exactly forms the value of the offer. In such a case the agents choose an instantiation that is closest to the value of the generated offer, i.e. the values for (x_1, x_2, x_3) that minimize $| x^t_{123} - \sum_{i=1,..,3} w_i v_i(x_i) |$.

4 System Overview

The eNAs environment can consist of many autonomous trading agents representing buyers and sellers. They are started from the main interface after finding and selecting a car (or a number of cars) for negotiation. For example the main user interface of a buying agent[5] is shown in Figure 4.

Each pair of the selling and buying agents can negotiate with each other at a time. An agent consists of a Web-based user interface, a communication interface and a negotiation engine. The user interfaces of the buying and selling agents are shown in Figure 5. They display public information about the subject of negotiation, allow the users to define and select the negotiation issues, preferences, constraints and negotiation strategy for an agent, and display the progress and results of each negotiation session.

The negotiation engine provides the main decision making functionality of an agent during negotiation, i.e. evaluation of the received offers and generation of the counter-offers according to the constraints, preferences and negotiation strategies. The negotiation engine uses the principles of constraint based-reasoning involving constraint propagation as presented in the previous sections. This allows the agents to keep tracks of changing options during negotiation including the ranges of issue values in possible offers (i.e feasible solutions at each stage of the negotiation process).

5 Implementation and Experiments

The eNAs system has been implemented in Java and deployed on the WWW. Two versions of the negotiation engines have initially been implemented with the use of two constraint programming libraries based on Java (JSolver [29]) and C++ (Ilog Solver [28]), respectively. It has been found that both implementations provide sufficient performance of the agents and finally Java-based JSolver has been used because of easier integration and deployment of the eNAs on the WWW. The system has been tested with several trading scenarios and is publicly available at http://www.cmis.csiro.au/aai/ITA.htm.

---

[5] It should be noted that in general the search, selection and negotiation initialization may be performed automatically. For the demonstration purposes the buyer invokes all the negotiation agents for both the buyer and the seller.

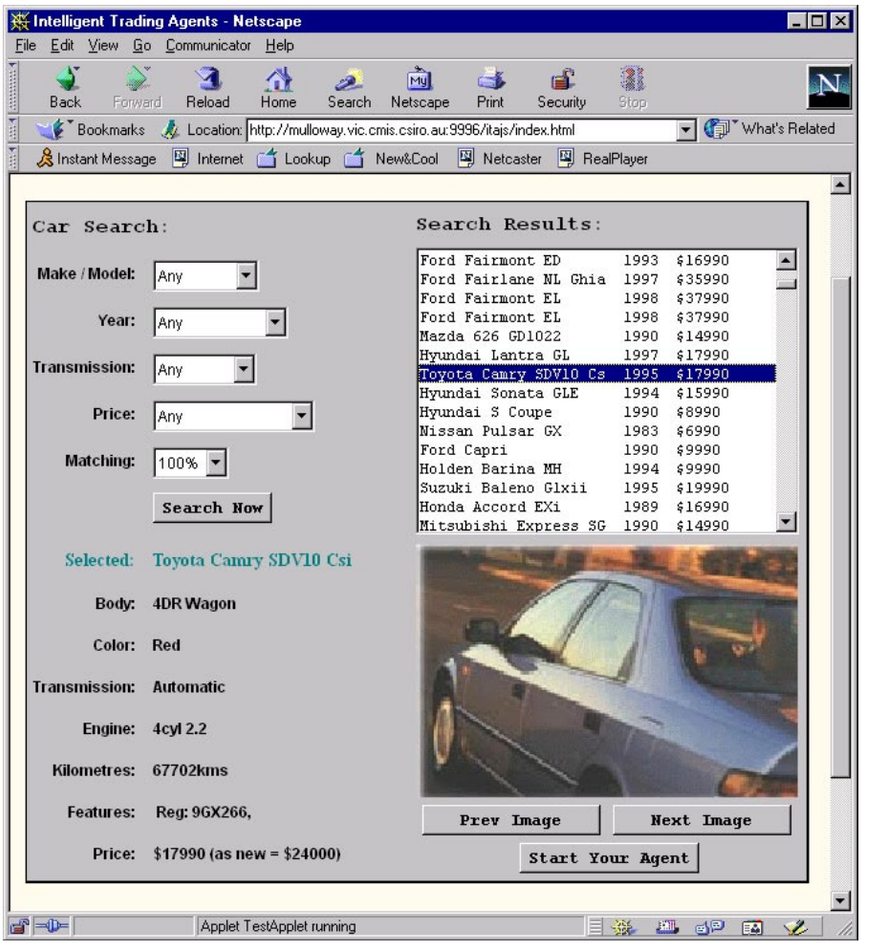


Fig. 4. The user interface to the eNAs demonstration system

A number of experiments have been performed to test the feasibility of the presented approach and to demonstrate how the agents can negotiate with different negotiation strategies. Below are two examples of the results of the single-issue and multi-issue negotiation scenarios, respectively.

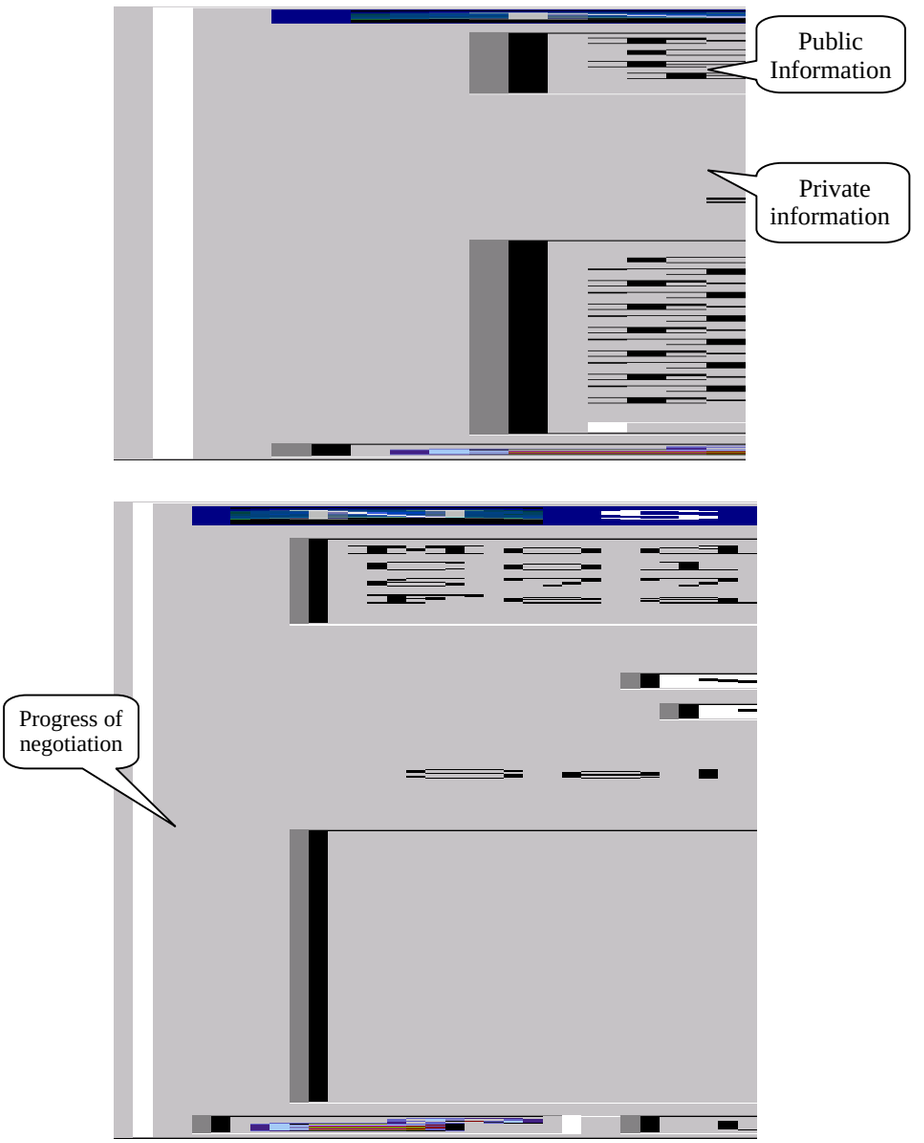


Fig. 5. Interfaces of the buying and selling agents with the negotiation results

5.1 Single-issue Negotiation

A simple scenario for single-issue negotiation based on price only is presented in Table 5.

Table 5. A scenario for single-issue negotitiation

Public (shared) information	Model: 1991 Toyota Celica ST184R SX, Kilometres: 132568, Price as New: $37000, Advertised Price: $18990.	
Private information (constraints and preferences)	Buyer	Seller
	Budget = $17100 Desire to buy = 70 %	Minimum price = $14200 desire to sell = 70 %

Table 6 summarizes the final negotiation results agreed by both the buyer and the seller using different negotiation strategies.

Table 6. Single-issue negotiation results

Seller / Buyer	Take-it-or-leave-it	Fixed concession	Fixed concession better deal	Simple concession	Simple concession better deal	No concession	**Average**
Take-it-or-leave-it	No deal	No deal	No deal	No deal	No deal	No deal	No deal
Fixed concession	No deal	15488	15488	15488	15488	14310	**15252**
Fixed concession better deal	No deal	14288	14288	14310	14310	14310	**14301**
Simple concession	No deal	14468	15551	14468	15522	14468	**14895**
Simple concession better deal	No deal	14468	15551	14468	15522	14468	**14895**
No concession	No deal	15488	15488	15488	15488	No deal	**15488**
Average	No deal	**14840**	**15273**	**14844**	**15266**	**14389**	

From the above results it can be observed that the seller agent works best with both of the better-deal strategies while buyer agent can benefit the most with the fixed concession better-deal strategy in this specific trading scenario. It should also be noted that the take-it-or-leave-it strategy resulted in unsuccessful negotiation, i.e. no deal was possible because the seller's asking offers (advertised price) were unacceptable to the buyer (they exceeded the budgetary constraints of the buyer agent).

5.2 Multi-issue Negotiation

A simple scenario for multi-issue negotiation based on price, warranty and trade-in is presented in Table 7.

The preferences for the issues of negotiation (i.e. price, warranty, trade-in) were set as (0.8, 0.0, 0.5), and (0.1, 1.0, 0.3) for the buyer and the seller, respectively. These preferences could be interpreted as follows: the buyer does not mind having a shorter warranty as long as he/she can get a good price for the car and trade-in. On the other hand the seller prefers lower warranty. Table 8 summarizes the final negotiation results agreed by both the buyer and the seller using different negotiation strategies.

The relative performance of different negotiation strategies is similar to the previous example. The results also demonstrate that given different preferences of different issues the agents could achieve more win-win deals.

Table 7. A scenario for multi-issue negotitiation

Public (shared) information	Model: Toyota Landcruiser Wagon 1995, Kilometers: 61240, Price as New: $49400, Advertised price: $36990.	
Private information (constraints and preferences)	Buyer	Seller
	Budget = $13500 Warranty = - Trade-in = $20000 Desire to buy = 70 %	Minimum price = $30000 Warranty = 1 Trade-in (valuation) = $19000 Desire to sell = 90 %

Table 8. Multi-issue negotiation results

Seller Buyer	Fixed concession	Fixed concession better deal	Simple concession	Simple concession better deal	No concession	**Average value**
Fixed concession	30821,1,19000 (10281)	30821,1,19000 (10281)	30352,1,19000 (9836)	30352,1,19000 (9836)	30320,0,19977 (10343)	**10115**
Fixed concession better deal	30000,1,19000 (9500)	30000,1,19000 (9500)	30000,1,19000 (9500)	30000,1,19000 (9500)	30320,0,19977 (10343)	**9669**
Simple concession	30320,0,19959 (10361)	31231,0,19989 (10670)	30320,0,19959 (10361)	31225,1,19000, (10668)	30320,0,19959 (10361)	**10484**
Simple concession better deal	30320,0,19959 (10361)	31231,1,19000 (10670)	30320,0,19959 (10361)	31229,1,19000 (10068)	30320,0,19959 (10361)	**10484**
No concession	30821,1,19900 (10281)	30821,1,19900 (10281)	30352,1,19000 (9836)	30352,1,19000 (9836)	NO DEAL	**10058**
Average	**10156**	**10280**	**9979**	**9984**	**10352**	

* Each cell consists of the agreed price, warranty and trade-in . Inside the brackets are the utility values of the final deals.

6 Conclusion

This paper presents some aspects of the constraint-based reasoning in the negotiation problem considered as a distributed CSP. In particular the paper outlines the constraint-based representation and constraint propagation mechanisms used in an experimental system of e-Negotiation Agents (eNAs) [5, 7]. The agents of the eNAs system can autonomously negotiate the multi-issue terms of transactions in the presence of limited common knowledge. The demonstration version of the system for the used car trading is publicly available at http://www.cmis.csiro.au/aai/ITA.htm.

Based on the results of the initial experiments it seems that the approach based on constraint-based reasoning is promising in providing automation support for multi-issue negotiations in the presence of limited common knowledge. In particular the experiments confirm that even a simple negotiation strategy can be superior to the take-it-or-leave-it strategy based on a fixed price. Although the overall results are encouraging a number of research issues need further investigation. Our current work includes extending the presented approach to more flexible negotiation with fuzzy constraints [5, 23], multiple party negotiation including dynamic coalition formation

[26] and improving negotiation strategies by learning from experience, modeling opponents and adaptation [27]. In particular it has led to the development of a fuzzy version of the eNAs system that uses the principles of fuzzy constraint-based reasoning in order to deal with imprecise information during negotiation (see [5, 23]).

Acknowledgments. The authors would like to thank Dr Leila Alem, Dr Maria Lee, Mr Wai Yat Wong and Dr Dong Mei Zhang of CSIRO Mathematical and Information Sciences for their contribution to the Intelligent Trading Agency project.

References

1. C. Beam and A. Segev. Automated Negotiations: A Survey of the State of the Art. CMIT Working Paper 97-WP-1022. May, 1997, http://haas.berkeley.edu/~citm/wp-1022.pdf
2. P. Faratin, C. Sierra, N. Jennings and P. Buckle. Designing Flexible Automated Negotiators: Concessions, Trade-Offs and Issue Changes, 1999, Institut d'Investigacio en Intel.ligencia Artificial Technical Report, RR-99-03
3. R. H. Frank. Microeconomics and Behaviour. 3rd ed. McGraw-Hill, 1996.
4. R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated Electronic Commerce: A Survey. Knowledge Engineering Review, June 1998
5. http://www.cmis.csiro.au/aai/ITA.htm
6. R. Keeney and H. Raiffa. Decisions with Multiple Objectives: Preferences and Value Trade-offs. John Willey and Sons, 1976.
7. Kowalczyk R. and Bui V. (1999) Towards Intelligent Trading Agents. The International Conference on Intelligent Systems and Active DSS in Turku/Åbo, Finland, 1999
8. V. Kumar. "Algorithms for Constraint-Satisfaction Problems: A Survey". AI Magazine, Spring 1992, 32-44.
9. S. Lander and V. Lesser. Understanding the Role of Negotiation in Distributed Search Among Heterogenous Agents. Proc. Of the 13th International Joint Conference on Artificial Intelligence. Chambery, France, August 1993. 438-444
10. R. Lewicki, D. Saunders, and J. Minton. Essentials of Negotiation. Irwin, 1997.
11. Liu, J., Sycara K.: Emergent constraint satisfaction through multi-agent coordinated interaction. Decentralized Artificial Intelligence, IV, Spronger Verlag, 1995.
12. A.K. Mackworth. "Constraint satisfaction". S.C. Shapiro (Ed.) Encyclopedia of Artificial Intelligence, John Wiley & Sons, pp. 205-211, 1990.
13. J. F. Nunamaker Jr., A. R. Dennis, J. S. Valacich, and D. R. Vogel. Information Technology for Negotiating Groups: Generating Options for Mutual Gain. Management Science, October 1991.
14. S. Parsons and N. R. Jennings. Negotiation through argumentation – a preliminary report. Proc. 2nd Int. Conf. On Multi-Agent Systems, ICMAS'96, Japan, 1996, 267-274.
15. D. G. Pruitt. Negotiation Behaviour. Academic Press, 1981.
16. T. Sandholm and V. Lesser. Issues of Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. Proc. 1st International Conference on Multiagent Systems (ICMAS'95), San Francisco, 1995.
17. Sycara, K., Roth, S., Sadeh, N., Fox, M.: Distributed constraint heuristic search. IEEE Trans.on System, Man, and Cybernetics. 21 (1991) 1446-1461.
18. Sycara, K. The PERSUADER. In D. Shapiro (ed.). The Encyclopedia of Artificial Intelligence., John Wiley Sons, 1992.
19. J. Rosenschein and G. Zlotkin. Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. MIT Press, 1994.
20. Yokoo, M., Ishida, T., Kuwabara, K.: Distributed constraint satisfaction for DAI problems. Proc. of the 10th International Workshop on Distributed AI. (1990).

21.M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed Constraint Satisfaction for Formalising Distributed Problem Solving. Proc. Of the 12th IEEE International Conference on Distributed Computing Systems, 1992.
22.A. Foroughi. “A Survey of the Use of Computer Support for Negotiation”. Journal of Applied Business Research, Spring 1995, 121-134.
23.R. Kowalczyk & V. Bui. FeNAs: A Fuzzy e-Negotiation Agents System. Conference on Computational Intelligence for Financial, New York City, March 26-28, 2000.
24. http://ecommerce.media.mit.edu/Tete-a-Tete/
25. http://kasbah.media.mit.edu/
26. Alem L, Kowalczyk R., Lee M. Recent Advances in Negotiation Agents for Electronic Trading. The International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, SSGRR 2000, l‘Aquila, Italy (in press)
27. Zhang, D.M., Wong, W. and Kowalczyk, R. Reusing previous negotiation experiences in Multi-agent Negotiation. Agents in Electronic Commerce, WAEC'99, p.79-87
28. www.ilog.com
29. Hon Wai Chun. Constraint Programming in Java with JSolver 2.0. (http://www.aotl.com)

Integrating Interaction Protocols and Internet Protocols for Agent-Mediated E-Commerce

Alexander Artikis, Frank Guerin, and Jeremy Pitt

Intelligent & Interactive Systems Group,
Department of Electrical & Electronic Engineering
Imperial College of Science, Technology & Medicine,
Exhibition Road, London, SW7 2BT, UK
{a.artikis,f.guerin,j.pitt}@ic.ac.uk
URL: http://www.iis.ee.ic.ac.uk/

Abstract. Conversations involving three or more agents often occur in multi-agent systems, for example in brokering and auction protocols typically used in e-commerce. For developing agents in open systems, it is important that the interactions in such conversations have a precise and unambiguous meaning. We address this issue by generalising a protocol-based semantic framework for expressing the semantics of Agent Communication Languages. The generalisations involve exploiting mechanistic aspects of the interaction (conversation identifiers), greater flexibility in the space of possible replies, and a richer representation of protocol states. We define intentional specifications for some brokerage and auction protocols, including event-based clocks to determine the ordering of events. We then discuss how these agent interaction protocols can be integrated with internet protocols, using the Agent Communication Transfer Protocol (ACTP), an application layer protocol designed to generalize communication between heterogeneous agents. We conclude that this approach to specifying multi-party protocols and the implementation platform of ACTP leads to clearer interfaces for open systems and easier re-use, with a potentially significant impact on e-commerce systems deployment and standardisation efforts.

1 Introduction

Conversation policies and interaction protocols have proved useful and indeed almost essential for high-level interoperability between heterogeneous agents in multi-agent systems (MAS). However, they have primarily been based on one-to-one conversations, i.e. dialogues between only two agents. It is a feature of many MAS, though, that there is some 'well known' agent (cf. *well known ports* in TCP/IP networks) that provides generic facilities to all other agents, for example directory and management services.

For example, the KQML language specification includes a Facilitator agent [5]. Facilitators support multi-agent and third-party conversations in some commonly recurring patterns of interaction between three agents. In some cases,

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 47–69, 2001.

there is even indirection, as one agent may not be aware of who one of the others is in the conversation.

For developing agents in open systems, it is important that the interactions in such conversations have a precise and unambiguous meaning. This paper addresses the issue by generalising the protocol-based, semantic framework of Pitt and Mamdani [12,11], developed for describing the semantics of Agent Communication Languages (ACLs) at different levels of abstraction. These generalisations enable us to provide formal specifications of the interaction patterns described in [5], and also for auction protocols.

Section 2 reviews the KQML brokerage protocols and the nature of the semantic problem, reviews the general semantic framework which will be used to frame a solution, and introduces the Agent Communication Transfer Protcol (ACTP). ACTP is an application layer internet protocol designed to generalize communication between heterogeneous agents. Section 3 demonstrates the solution, which is based on exploiting conversation identifiers and a richer space of possible replies. Section 4 gives a further generalisation which caters for auction protocols. In both cases we give an intentional specification of expected agent behaviour. Section 5 describes a proposed implementation of the conversation identifier mechanisms and its integration within ACTP, which can then be used to support the protocol specifications for brokerage and auctions described in the previous two sections. Section 6 summarises the work, and argues that interpreting speech acts in context is preferable to using complex speech acts. We conclude that the emphasis on design of multi-party protocols and implementation over common platforms can lead to 'public' interfaces for open systems, easier re-use, and can expose unexpected problems. This has a potentially significant impact on standardisation efforts, and the development and deployment of 'plug-and-play' agents for electronic commerce.

2 Background and Motivation

2.1 Multi-party Conversations and Protocols

There is a definite requirement for multi-party conversations in MAS applications where brokerage and/or auctions are required. In KQML a special class of agent called facilitators was introduced to perform various useful communication services, in particular mediation or brokerage services. In [5], four interaction patterns based on these services were described, for recruitment, brokerage, recommendation and subscription (clockwise from top right in Figure 1).

As with the well-known contract-net protocol, the value of these protocols was that, given the frequency with which they occurred in different applications, the specifications could be simply re-used. However, the problem with understanding and applying these diagrams is that in [5], the semantics of KQML was an open issue. This meant some difficulty in interpretation. For example, in the recommend protocol, agent A was supposed to ask F if there was an agent willing to provide a certain service (e.g. $\mathsf{ask}(X)$), and F would reply "once it

learns that B is willing to accept $\mathsf{ask}(X)$ performatives" [5]. But this means that agent A may have to block until B advertises, whereas in fact it would be more useful for A to know directly that no agent has advertised the service it requires.

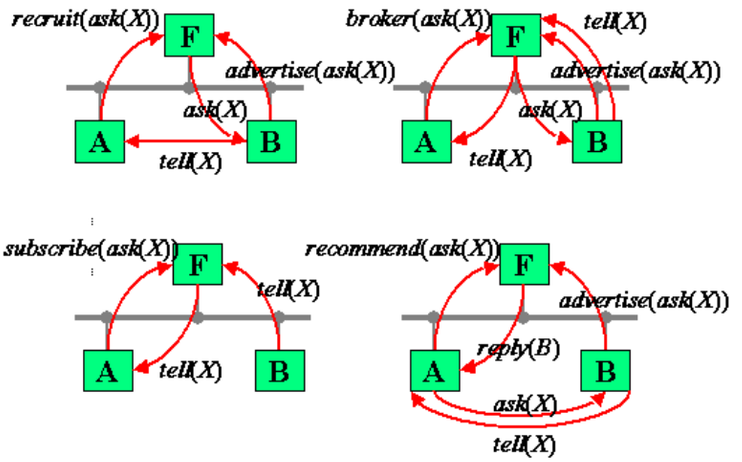


Fig. 1. KQML communication facilitation services

This problem has largely been addressed in KQML by the use of Coloured Petri Nets [3], but it remains a problem for the FIPA ACL semantics [6]. In a series of papers submitted to FIPA, an attempt has been made to express the semantics of the broker communicative act in the SL logic of the FIPA semantics [16,15]. The experience has been that it is very hard to do, understand, verify and apply.

Nevertheless, these protocols have been implemented and widely used in a variety of applications. Probably, a number of specific solutions have been developed but it is unlikely that such systems could then interoperate. What we are trying to achieve with this paper is to show how such protocols can be specified in a general semantic framework. This framework is briefly reviewed in the next section.

2.2 The General Semantic Framework

The protocol-based semantic framework has been introduced in [12]. The main idea in the original work was to separate out the 'external', action-level semantics (i.e. observed speech acts) from the intentional semantics (agent-internal reasons for and acting and replying) and the content-level semantics (i.e. the meaning of the content actually communicated). This core idea remains but we here we will extend the framework in a consistent manner in order to handle multi-party protocols.

An ACL is a 3-tuple $< Perf, Prot, reply >$ where $Perf$ is a set of perfomative names, $Prot$ is a set of protocol names, and $reply$ is a partial function given by:

$$reply : Perf \times Prot \times \mathrm{N}^{+} \nrightarrow P(Perf \times Prot)$$

where N^{+} is the domain of positive integers. The $reply$ function is then defined for each distinct state of each protocol, identified by a unique (for each protocol) integer. This gives for each speech act, 'performed' in the context of a conversation being conducted according to a specific protocol, what performatives in which protocols are acceptable replies. The $reply$ function therefore specifies a finite state diagram for each protocol named in Prot. (Note also that in $Perf$ we include the null performative which is a 'do nothing' (no reply) performative (cf. 'silence' in [13])).

To fully characterise the intended semantics, three further functions are required, which are specified relative to each agent a, and state what that agent does with a message, not how it does it. The three functions in [12] were (1) a procedure for computing the change in an agent's information state from the content of an incoming message; (2) a procedure for selecting a performative from a set of performatives (valid replies), and (3) a function $conv$ which mapped a conversation identifier onto the current state of the protocol. For these functions, we specify intentional (logical) descriptions of the reasons for and reactions to a speech act. These serve as reference implementation models that agent developers could use to implement the appropriate internal and external behaviours for their agents. Furthermore, where the import of the the content level meaning was required, further specifications could be supplied, and this is dependent upon the application.

An agent s then communicates with (and communicates information to) an agent r via a speech act. This (possibly infinite) set is denoted by $speech\_acts$, a single member sa of which is represented by:

$$sa = \ll s, \mathsf{perf}(r, C, L, O, cp, ci, t_s) \gg$$

This is saying that s does (communicates with) performative perf with content C in language L using ontology O in the context of protocol (conversation policy) cp as part of a conversation identified by ci at time of sending t_s. The notation $sa.perf$ denotes the performative of a speech act, and so on. (Note that the issues of time (explicitly represented), language and ontology are not addressed in this paper, so that later representations of speech acts in section 3 and 4 will omit these parameters).

The meaning of such a speech act sa from agent s to agent r is then given by:

$$[\![\ll s, \mathsf{perf}(r, (C, L, O, cp, ci, t_s)) \gg]\!] = \mathcal{I}_r \ll r, rep\_sa \gg \text{ s.t.}$$
$$(rep\_sa.perf, rep\_sa.prot) \in reply(\mathsf{perf}, cp, conv_r(ci))$$

This means that, in this framework, at the observable action level, the meaning of a speech act is the intention to give a reply. Note that this defines only

the meaning of a speech act at the action level. This is therefore an external semantics, i.e. from the perspective of an observer that cannot 'see' the agent internals. It may well be that two speech acts get the same apparent meaning (for example, whenever the intended reply is null). However, we contend that from the observer's perspective that is exactly right: all the observer sees is the actions. If two different actions don't entail any response then the actions mean the same thing (and in the absence of any observable change of state, are ostensibly meaningless (sic)) – at the action level.

At other levels of meaning, we are concerned with the change of state (information and motivational) that occurs in both the sender and a receiver as a result of a speech from one to the other. It is for this reason that we are concerned with intentional specifications to characterise the decision making for sending and replying to a speech act. Some of this will also be influenced by the content, its interpretation and context. Note that there are also certain mechanistic aspects in our framework that we presuppose: that both both sender and receiver believe that the speech act was 'done', that the both change state, and both update (where necessary) the conversation identifiers.

Therefore, the protocol-based semantic framework supports a general methodology for designing an ACL for a particular application [11]:

- Generalization at the action level: additional performatives and protocols can be introduced to create a new ACL and new patterns of interaction;
- Specialization at the intentional level: a reference implementation model, possibly referring to agents' beliefs, desires and intentions, could be specified to give intended behavioural meanings for performatives in the context of the protocols;
- Instantiation at the content level: the specific decision-making functionality for deciding which reply from a set of allowed replies can also be specified. For example, the same protocol may be used in quite different application domains and the decision making may be dependent on a number of different factors.

Section 3 will show how the brokerage protocols of [5] can be specified at the action level, and give logical specifications to complete the meaning at the intentional level. Section 4 analyses an auction protocol. First though, we complete our background review with a brief description of the Agent Communication Transport Protocol, which is the basis for implementation discussed in section 5.

2.3 Agent Communication Transfer Protocol

Inter-agent communication requires knowledge of an interaction protocol, an agent communication language (ACL), and a transfer protocol [5]. In the previous sections we presented a general framework for designing ACLs and interaction protocols. In this section, we briefly overview our solution to the third requirement, which is the transfer protocol. For full details, see [1].

Humans use multiple channels for communication, and if one fails, we try another (e.g. with no reply to a phone call, we may send e-mail). If we seek to

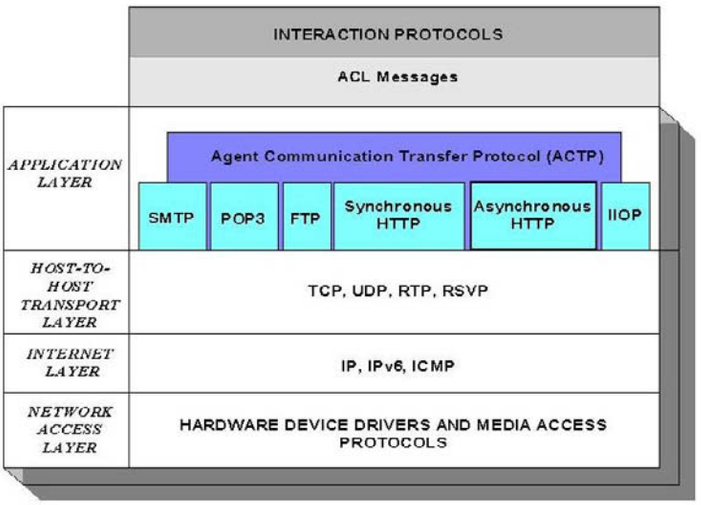


Fig. 2. KQML communication facilitation services

support higher-level communication between agents rather than processes, some of the same flexibility should be seen, without forgetting that we are essentially dealing with a mechanistic interaction. The Agent Communication Transfer Protocol (ACTP) is an application layer internet protocol that uses various communication protocols for information exchange, thus supporting various methods of communication and ensuring reliable agent-interactions.

The ACTP is an 'umbrella' protocol in the sense that it hides the low-level networking details from the agents (see Figure 2). The communicating agents provide the ACTP with the ACL message via a simple API and the ACTP is responsible for the delivery of that message. The application protocols that the ACTP uses as transport mechanisms use other protocols in order to carry out their tasks. In other words, many of the application protocols can be viewed as a wrapper, in the sense that they hide the implementation and use of other protocols from the ACTP. For example, the FTP uses the synch signal of Telnet to abort an in-progress transfer. Nevertheless, the ACTP does not interact with Telnet; it just uses the FTP and it does not deal with the way the FTP is implemented. In the architecture of the ACTP there are two levels of encapsulation; in the higher one, the networking details of the communication processes are hidden from the agents. In the lower one, the implementation of the underlying application protocols is concealed from the ACTP.

Normally, an agent will specify the exact protocol that it wants to be used for the data transfer. In this case the ACTP will use the specified protocol for the communication. In other cases, an agent might not specify a particular transport mechanism and let the ACTP decide about that. The ACTP has the flexibility

to make those kinds of decisions. In particular, the ACTP offers a decision-making algorithm that is primarily based on the properties of the message being exchanged, like the size and the time-criticality of the message. The output of this decision-making algorithm is the choice of transport medium that will be used for the message transfer.

This agent-oriented middleware system supports the use of a name. The agent that sends a message denotes the peer agent not by its physical address, but by its name. It is up to the ACTP to convert this name to a physical address. This function is performed with the use of the Name Server of the ACTP. The Name Server is a process that provides the ACTP with low-level information about the agents, such as their logical name and domain, their physical addresses in relation to each underlying protocol and any authentication data that may be needed in order to contact each agent.

The ACTP Library can be extended for handling conversation identifiers as discussed in the next section, and a possible implementation is reviewed in Section 5. This makes the ACTP eminently suitable for carrying messages generated using the brokerage and auction protocols specified in the next two sections.

3 A Semantics for Brokerage

3.1 Conversation Identifiers

Conversation identifiers in ACLs are used to identify a dialogue between two agents, so that a message in one conversation is not confused with message in another, simultaneous, conversation. In the FIPA specifications, the conversation identifier is a parameter of the ACL message, but it is not part of the semantics. It is noted that there are parts of the specification that fall outside the scope of the semantics, and it is arguably this omission that makes the logical specification of the broker performative harder than it needs to be, because there is no 'handle' in the semantics for referring to a conversation as a 'first-class object'.

Our approach brings the conversation identifiers into the semantics, but these need to be unique. One way of guaranteeing uniqueness is to use the FIPA naming system for Globally Unique Identifiers for agent names and a timestamp generated from the system clock. Modulo any operating system vagaries, a unique identifier is then generated for each conversation in which an agent participates [4].

An alternative mechanism for generating unique conversation identifiers is to use a 2-tuple, and for each participant to supply one component: we use an integer.

Each agent maintains a count of the number of conversations it has engaged in, and increments this by one for each new conversation. Then, it assigns the current count for its first speech act in the conversation (see Figure 3).

If both agents in a conversation do this, then the combination of protocol state and the conversation identifier uniquely identifies the conversation and which message has been sent in reply to which other, i.e. there is no need to

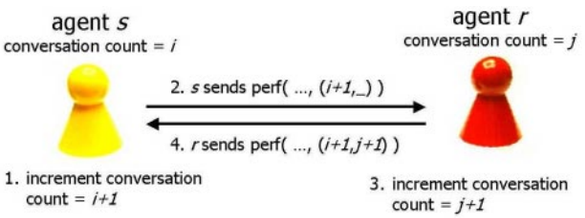


Fig. 3. Communication with conversation identifiers

have separate `:reply-with` and `:in-reply-to` message parameters in the ACL. Where no reply is expected or required, the receiving agent need take no action about replying, but still labels the conversation for later reference.

The advantage of having one component of the 2-tuple conversation identifier provided by each party is that each one can then label its own component with additional information, and return the other party's information intact. The conversation identifier can be treated as a term (in the Prolog sense) and can contain structured information. The functor of the term is the uniquely-assigned integer, and the arguments can be the extra information.

This then means that the conversation identifiers can be treated like "Our Reference" and "Your Reference" labels in human information transactions (e.g. memo passing). By convention, one uses one's own system for "our reference", and respects whatever system is used by the other party. The advantage for agent communication is the knowledge of how to interpret the reference in context: for example, in the KQML-style communication [5] of Figure 4, the interpretation of the reference $j(A, i)$ is for agent B to send the reply to the content of the broker (speech act) received from the Facilitator F to agent A quoting its reference i. (Note the convention in Figure 4 is the sender's reference first and the receiver's second.) Then, the conversation identifier is constant but the additional information can be variable throughout the lifetime of a single conversation.

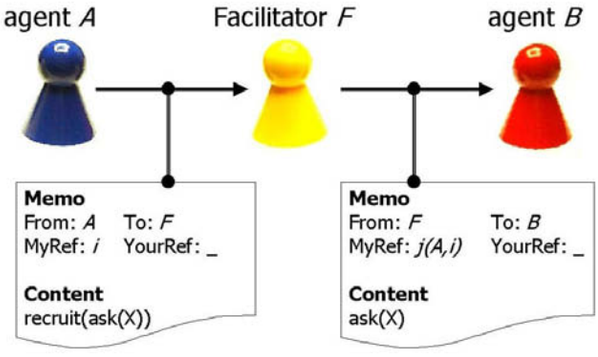


Fig. 4. Parameterised conversation identifiers

We define the following function cid_a for each agent a:

$$cid_a : N^+ \times AgentIdentifier \longrightarrow N^+$$

This gives, for each agent a, for a conversation identifier and the name used by a to identify some other agent, the identifier used by that agent to identify the same conversation.

3.2 Protocols and Intentional Specifications

The KQML brokerage protocols can each be analysed as a composite of two or three separate one-to-one conversations, which can be intrinsically linked by the *reply* function. In this section, these conversations will first be specified at the action level as finite state diagrams, with the underlying meaning given by intentional specifications. The formal language for the specification is a multi-modal dynamic action logic using the 'triggers and tropisms' style of [11], i.e. specifying the reason for doing and responding to speech acts. Formulas like $[a, A]p$ intuitively state that after agent a does action A, then an agent which 'observed' A and has this formula in its belief state should endeavour to make p true. An operational semantics for this language is considered in [10].

Note that after the first speech act in a protocol, we assume that the receiver assigns its component of the conversation identifier, and that after that DONE($\ll a, \mathsf{perf}(\ldots, (i, j)) \gg$) is true, where (i, j) is the conversation identifier.

The finite state diagrams for the various parts of the recruit and broker performative/protocols are illustrated in Figure 5. The finite state diagrams for subscribe and recommend are simple variations on a simple question/answer pair. Intentional specifications of the meaning of the speech acts in the context of these protocols, together with an English paraphrasal (of the recruit and broker protocols; the subscribe and recommend specifications can be expressed similarly) are given in Table 1.

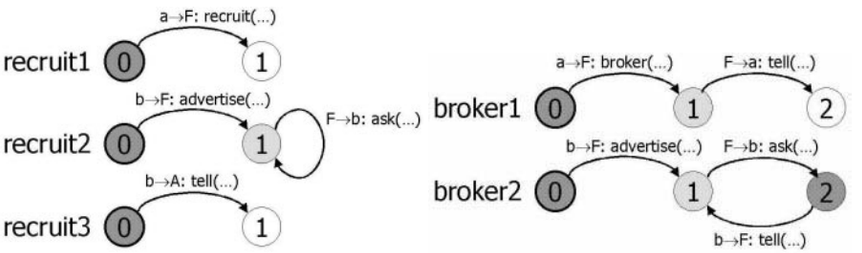


Fig. 5. Finite State Diagrams for KQML Brokerage

Table 1. Intentional Specifications: KQML Brokerage Protocols

1	$[A, \mathsf{recruit}(F, \mathsf{ask}(X), \mathit{recruit1}, (i, \_))]$
2	$\exists b, j, k.\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{recruit2}, (k, j)) \gg)$
3	$\rightarrow \mathcal{I}_F \ll F, \mathsf{ask}(b, X, \mathit{recruit2}, (j(A, i), k)) \gg$
$1'$	after A performs a recruit speech act using the *recruit1* protocol with conversation identifier $(i, \_)$,
$2'$	if an agent b has done an advertise using *recruit2* with identifier (k, j)
$3'$	then form the intention to ask b about X with identifier $j(A, i)$, meaning reply to A quoting identifier i
4	$[F, \mathsf{ask}(B, X, \mathsf{recruit2}, (j(A, i), k))]$
5	$\mathcal{I}_B \ll B, \mathsf{tell}(A, X, \mathsf{recruit3}, (l, i)) \gg$
$4'$	after F performs an ask using *recruit2* with identifier $(j(A, i), k)$,
$5'$	form the intention to tell A about X using *recruit3* with identifier (l, i)
6	$[A, \mathsf{broker}(F, \mathsf{ask}(X), \mathit{broker1}, (i, \_))]$
7	$\exists b, j, k.\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{broker2}, (k, j)) \gg)$
8	$\rightarrow \mathcal{I}_F \ll F, \mathsf{ask}(b, X, \mathit{broker2}, (j(i), k)) \gg$
$6'$	after A performs a broker using *broker1* with identifier $(i, \_)$,
$7'$	if an agent b has done an advertise using *broker2* with identifier (k, j)
$8'$	then form the intention to ask B about X with identifier $(j(i), k)$
9	$[B, \mathsf{tell}(F, X, \mathit{broker2}, (k, j(i)))]$
10	$\exists a, l.\mathrm{DONE}(\ll a, \mathsf{broker}(F, ask(X), \mathit{broker1}, (i, l)) \gg)$
11	$\rightarrow \mathcal{I}_F \ll F, \mathsf{tell}(a, X, \mathit{broker1}, (l, i)) \gg$
$9'$	after B replies with a tell using *broker2* with identifier $(k, j(i))$,
$10'$	then there is an agent a who did a broker using *broker1* with identifier (i, l)
$11'$	and form the intention to tell agent a about X using *broker1* with identifier (l, i)
12	$[A, \mathsf{subscribe}(F, \mathsf{ask}(X), \mathit{subscribe}, (i, j))]$
13	$\exists b.\mathrm{DONE}(\ll b, \mathsf{tell}(F, X, \_, \_) \gg \rightarrow \mathcal{I}_F \ll F, \mathsf{tell}(A, X, \mathit{subscribe}, (j, i)) \gg$
14	$[A, \mathsf{recommend}(F, \mathsf{ask}(X), \mathit{recommend}, (i, j))]$
15	$\exists b.\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{recommend}, \_) \gg)$
16	$\rightarrow \mathcal{I}_F \ll F, \mathsf{reply}(A, b, \mathit{recommend}, (j, i)) \gg$

3.3 Extending the Framework

The first observation to make is that a conversation identifier is no longer a single unique identifier, as specified in [12], but is a pair, one element of which is supplied by each party to the conversation. The function of the identifiers is the same as before, but the fact that they can be parameterised allows us to provide additional information. This information can be interpreted (in the context of a protocol) to forward to third parties and so on, giving a precise formal specification of the KQML brokerage protocols.

The second observation is that under the previous semantics, an agent either replied with a speech act in the same protocol, or, if the conversation was over, performed the null speech act. The above logical specifications are correct with respect to that semantics; however, it is not quite capturing the notion that although a speech act in this conversation is not expected (or allowed), another

speech act in a new conversation is intended. Therefore the range of the *reply* function is a set of pairs of speech act and protocol.

The third observation to make is that using these generalisations in conjunction with the intentional specifications, we can be more precise about the order of events and what actions the Facilitator should take. The value of such specifications is that it is now clear what the order of actions is expected to be, and under what circumstances certain actions are expected. Note that the protocols remain normative in that they specify what actions can (must) be done, but the intentional specifications give declarative statements of what decisions should be taken, but do not specify *how* this decision making should be implemented. In this sense the intentional specification is informative, but any agent hoping to work in a system based on this specification of the protocols is expected to comply with these intentions, and so the specification is providing a reference implementation model for an agent implementer.

However, even now there are certain elements which remain underspecified: for example, actions to recover from errors (for example, in lines 1–3, if there was not an agent who had performed an $\mathsf{advertise}$). Therefore, we might specify the following:

$$
\begin{aligned}
&[A, \mathsf{broker}(F, \mathsf{ask}(X), \mathit{broker1}, (i, \_))] \\
&\qquad (\exists b, j, k.\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{broker2}, (k, j)) \gg) \\
&\qquad\qquad \rightarrow \mathcal{I}_F \ll F, \mathsf{ask}(b, X, \mathit{broker2}, (j(i), k)) \gg) \\
&\vee \\
&\qquad (\neg\exists b.\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{broker2}, \_) \gg) \\
&\qquad\qquad \rightarrow \mathcal{I}_F \ll F, \mathsf{tell}(A, \mathsf{not\_advertised}, \mathit{broker1}, (j, i)) \gg)
\end{aligned}
$$

Recall that this is a receiver's side specification interpreted locally (not a system-wide (global) specification), so this formula states that after the broker speech act, if the facilitator knows of no agent that has advertised the service, then it should tell the sender that this is the case. Of course, we also need to update the *broker1* state diagrams to reflect this improvement to the specification.

In addition, from inspection of Table 1, it might be considered odd that agent B needs to advertise both in the *recruit2*, *broker2* and *recommend* protocols. However, these are different protocols, with different state diagrams; furthermore, advertising with the protocol name offers an implicit contract that the agent will understand the different conversation identifiers that will be used in each case. In addition, an agent may want to preserve anonymity, and while be willing to participate in brokerage, it may not want to participate in recruitment (note the difference in Figure 1: in the broker protocol everything gores through the Facilitator).

By specifying the decision-making and error recovery, we also see that the protocols remain incomplete. What action is taken, for example, for an agent B that wishes to withdraw an advertise, or an agent A that wishes to cancel a subscription? To handle this, we might introduce new performatives ($\mathsf{withdraw}$ and cancel, say), update the protocols, and introduce extra conditions into the intentional specifications, for example:

$$[A, \mathsf{recruit}(F, \mathsf{ask}(X), \mathit{recruit1}, (i, \_))]$$
$$\exists b, j, k.(\mathrm{DONE}(\ll b, \mathsf{advertise}(F, \mathsf{ask}(X), \mathit{recruit2}, (k, j))\gg)$$
$$\wedge\neg\mathrm{DONE}(\ll b, \mathsf{withdraw}(F, \mathsf{advertise}(\ldots), \mathit{recruit2}, (k, j))\gg)$$
$$\rightarrow \mathcal{I}_F \ll F, \mathsf{ask}(b, X, \mathit{recruit2}, (j(A, i), k))\gg$$

This is not the first time such problems have been identified with the KQML brokerage protocols [13]. Indeed we are endorsing the work of [13], in that we argue for a formal semantics, a development method, and (ideally) automated tools, in order to support rigorous design allowing development of solutions to such problems. We can see also that this method of protocol specification is becoming unwieldy as the protocols grow more complex. The entire history of the conversation is appearing in the intentional specification of the meaning of the recruit act. Ideally the withdraw and advertise should make a change to the state of the conversation, and a subsequent recruit would be received in the context of this new state. The state effectively summarises relevant information from the history of the conversation. The semantics of recruit would then include a check on the current state before giving permission to perform the ask. This idea of storing the relevant information as the conversation state and using it to guide the conversation is the subject of ongoing research [7].

4 Auction Protocols

In this section, we consider auctions, another type of multi-party agent conversation. We concentrate in particular on what are typically called English Auctions, in which an auctioneer announces a proposed sale price (the asking price) to a group of potential bidders. If one of them accepts the price, the auctioneer increases it by a set amount, and announces a new asking price. If no-one accepts, the goods on auction are sold to the last bidder.

Auction protocols can be distinguished from the brokerage protocols studies in the previous section, in that instead of several one-to-one conversations which are conducted, sequentially or nested, according to different protocols, auction protocols comprise, in effect, several one-to-one conversations being conducted concurrently according to the same protocol.

The techniques used to specify the semantics of an English Auction are a richer representation of state, and parameterised conversation identifiers, as in the previous section. Conversation identifiers in auctions will now be parameterised by vector timestamps [17].

4.1 English Auction: State Diagram

The finite state diagram for an English Auction is illustrated in Figure 6(a). We assume a previous stage of registration to participate and declaration of lots, etc., covered by some alternate protocols. We consider here only the process of a single auction.

The English Auction Protocol is used between an Auctioneer agent A and a set of buyer agents B. However, it is effectively a single conversation between A

and each agent $b \in B$. After the first announce, which is multi-cast to all agents, if no agent bids (accepts), the auctioneer can end the auction. Otherwise, if an agent decides to try to buy the lot at this price, it sends an accept to the auctioneer. The auctioneer accordingly increases the price, and announces this to all agents. This process repeats, until no agent makes a bid (accept). Then the auction is won by the agent that made the last bid. This agent is told of its win by a sold message and all other agents are informed that the auction is over.

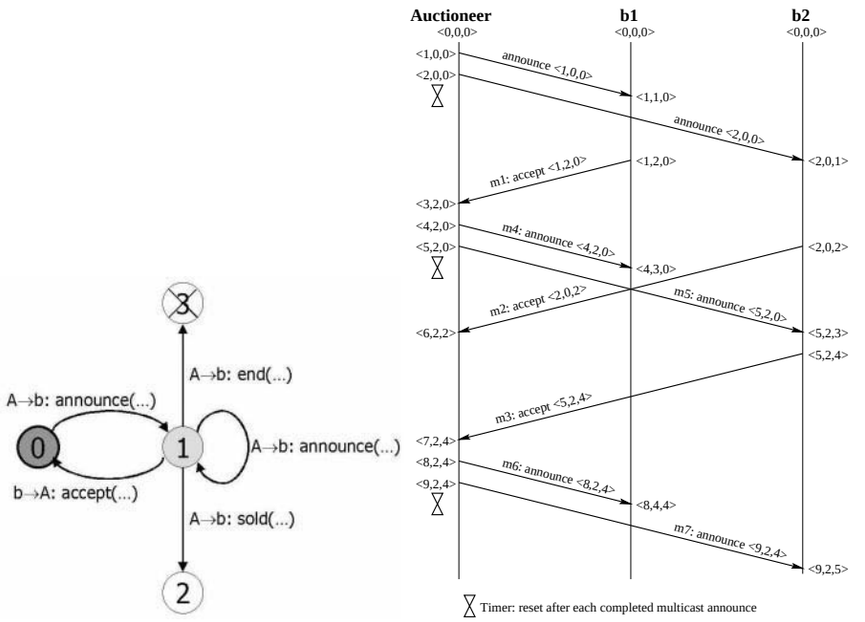


Fig. 6. English Auction Protocol: (a) Finite State Diagram, and (b) Message Sequence Chart

4.2 Vector Timestamps

The problem with the protocol described in the previous sub- section is that a transition from state 2 is context-sensitive, i.e. for an agent making a bid, it goes into state 1 first; for all other agents, all they 'see' is the new announce. However, these other agents may also have sent accept messages – except accepting the old price and have therefore arrived too late. Therefore, it is necessary to resolve:

1 For the auctioneer, if an accept is in response to the most recent multi-cast announce;
2 For the bidding agents, whether or not it is their bid which has been accepted.

Our solution to this is firstly to use a unique conversation identifier between the auctioneer and each agent, in the style of the previous section. However,

in the English Auction Protocol, the conversation identifier is parameterised by the sending agent with a vector timestamp [17] which will enable the agents to determine potential causality.

The idea of potential causality in distributed systems without global clocks is that the ordering of event can only be determined locally, i.e. with respect to any single observer. If an event e precedes an event f, from one observer's point of view, then e potentially caused f. To determine if one event preceded another, each agent uses its own *vector clock*.

Following [17], a vector clock is defined over a group of agents with cardinality n as an (n)-ary vector of natural numbers $v =< a_1, a_2, \ldots, a_n >$. The starting clock of every agent is the vector $< 0, 0, \ldots, 0 >$. Each agent increments its entry in the vector when it performs a local event, i.e. sending or receiving a message. It attaches its entire vector clock as a timestamp to every message it sends out. When it receives a message, it takes the element-wise max of its vector clock and the timestamp on the incoming message, increments its entry in the vector by 1 (for the local event), and sets this value to be its new vector clock.

One vector is defined as later than another if the value of at least one entry in the first vector is greater than the corresponding entry in the second, and no value in the second is greater than the first, i.e. v is later than u if and only if firstly $\forall i.1 \leq i \leq n : u_i \leq v_i$ and secondly $\exists i.1 \leq i \leq n : u_i < v_i$. The notation $u \prec v$ indicates that v is later than u.

Figure 6(b) shows a message sequence chart for a possible exchange of a number of announce and accept message between an auctioneer A and two bidding agents $b1$ and $b2$ during an English Auction. (The labelling 'mx' on some messages is incidental and used to explain cause and effect between messages below.) Note that after the first announce, both $b1$ and $b2$ accept, but $b1$'s bid arrives first. Therefore this is the accepted bid which causes the second (multi-cast) announcement. Note also that we stipulate that the Auctioneer must 'block' (not read) any incoming messages after an accept, until it has finished the multi-cast announce, i.e. a separate announce message has been sent to each participating agent in B, and so has been assigned a timestamp.

The agents can now use the vector timestamps to determine which announce (potentially) caused which accept, and vice versa, using the two following rules:

- R1 For the auctioneer, if the timestamp on an incoming accept message is later than the auctioneer's vector clock after the last accepted bid (the last accept to cause an announce), then it was caused by the most recent multi-cast announce;
- R2 For a bidding agent, if the timestamp on an incoming announce is later than their vector clock, then the announce was caused by its accept.

Applying these rules to the message sequence chart of Figure 6(b), the causality relations of messages m1-m7 are summarised below. Rule R1 is applied by the auctioneer to messages m1-m3, and Rule R2 is applied by bidding agents to messages m4-m7.

m1	120 $\prec$ 000 accept is potentially caused by most recent (multi-cast) announce
m2	202 $\not\prec$320 accept not caused by most recent (multi-cast) announce
m3	524 $\prec$ 320 accept is potentially caused by most recent (broadcast) announce
m4	420 $\prec$ 120 (multi-cast) announce potentially caused by b1's accept (m1)
m5	520 $\not\prec$202 (multi-cast) announce not caused by b2's accept (m2)
m6	824 $\not\prec$430 (multi-cast) announce not caused by b1's accept
m7	924 $\prec$ 524 (multi-cast) announce potentially caused by b2's accept (m3)

4.3 Intentional Specifications

Finally in this section, we specify the reference implementation model for the English Auction. The problem now is that the change of state is more complex than in ordinary one-to-one protocols, and in the brokerage protocols of KQML. Therefore the simple state representation as an integer is no longer sufficient. The state of an auction is conditional on the announced price, the current winner (performer of the last accept), the set of bidding agents, and the auctioneer's vector clock of the last accepted bid, used in R1.

In fact, there are four parts to the specification that need to be formalised:

1 The change of state of both speaker and hearer after a speech act (message is sent/received);
2 The intentions of hearer in response to a speech act;
3 The (real) time aspects, i.e. if no accept is received within a certain time, then the auctioneer terminates the auction;
4 Rules R1 and R2 upon which 1–3 above are conditional.

For the first item, we require a richer representation of the protocol states, so need the following state variables:

buyer the agent currently winning the auction (i.e. the agent who made the last accepted bid)
price current asking price of the item for auction B set of agents bidding in this auction ($b \in B$ denotes a single bidder in B)
la timestamp of last accepted bid *time* real time remaining before auction is ended

Access and change to a state variable *svar* is indicated by the following notation:

$conv_a(i).svar = val$ to test the value for equality
$conv_a(i).svar \leftarrow newval$ to overwrite the current value

The following identifiers are also required:

vc, ts, la are all vector clocks, where
vc is the vector clock of an agent
ts is the timestamp on a message
la is the vector clock of last accepted bid

$(i(ts), j)$ i is the unique number identifying this conversation for the sender, ts is its vector clock at the time of sending, and j identifies the conversation for the receiver

$conv_a(i)$ returns the protocol state of the conversation identified by i for agent a (as in section 3.1)

$cid_a(i, b)$ returns the integer used by agent b in the conversation identified (for agent a) by i (as in section 3.2)

Finally, we need a notation to multi-cast a speech act (where ; is a conventional sequence operator for actions):

$$\ll s, multicast(\mathsf{perf}, R, C, P, i) \gg = ;_{r \in R} \ll s, \mathsf{perf}, C, P, (i(vc), cid_s(i, r)) \gg$$

This describes the multi-cast of a performative perf, to a set of receivers R, with content C, in protocol P, performed by an agent s as the composition of a sequence of individuals acts to each $r \in R$. Since each agent can only do one act at a time, the specific conversation identifier for each speech act in the multi-cast is constructed from vc is the vector clock at the time of occurrence of the local event (and is incremented after performing each individual announce speech act), and so is different for each $r \in R$.

The English Auction multi-party protocol is formally specified (and paraphrased) by the intentional specifications contained in Table 2 for the communicative acts involved. Note that there are different axioms for both sender (auctioneer) and receiver (bidding agents) for the announce act. Note that we are assuming that vector clocks will be correctly updated to account for the occurrence of local events. Table 3 shows the auctioneer's actions for dealing with the timeout of the timer represented in Figure 6(b).

5 Experimental Implementation

In this section we describe a proposed implementation of the conversation identifiers (CIDs) mechanism as it was described in section 3.1, and of the integration of this mechanism with the Agent Communication Transfer Protocol (ACTP) described in section 2.3. In previous work [14] we produced an experimental implementation of this mechanism. In this paper, we reconsider this implementation and propose a modified specification of it.

There are two different ways in which a conversation could be established:

- The handshaking mode: the initiating agent sends only its CID to the receiving agent, and waits for that agent's identifier to construct the final CID tuple (cf. handshaking in TCP before establishing a connection). This method requires two messages to be passed before the commencement of the actual conversation.

Table 2. Intentional Specifications: English Auction Protocol

Auctioneer's trigger for multi-cast announce
17 $[A, multicast(\mathsf{announce}, B, price, EA, (i, \_))]$
18 $conv(i).time \leftarrow 50$
17′ after A does a multi-cast announce using the EA protocol with identifier $(i, \_)$
18′ reset the *time* state variable to some set number of time units (e.g. 50)
Bidding agent's reaction on receiving its (individual) announce
19 $[A, \mathsf{announce}(b, price, EA, (i(ts), j))]$
20 $(vc =< 0, 0, \ldots, 0 >) \vee (ts \prec vc)$
21 $\rightarrow compute\_accept(price)$
22 $\rightarrow \mathcal{I}_b \ll b, \mathsf{accept}(A, \_, EA, (j(vc'), i)) \gg$
19′ after A does an announce in the EA protocol with conversation identifier containing timestamp ts
20′ if the vector clock is all zeros (the start of the auction) or rule R2 does not apply (otherwise b is winning)
21′ then if the agent decides to bid at this price ($compute\_accept(price)$ is true)
22′ then form the intention to send an accept to the auctioneer, vc' being the vector clock of the send local event
Auctioneer's reaction after receiving an accept
23 $[b, \mathsf{accept}(A, \_, EA, (j(ts), i))]$
24 $conv(i).time > 0$
25 $\rightarrow ts \prec conv(i).la$
26 $\rightarrow conv(i).buyer \leftarrow b$
$\wedge\, conv(i).price \leftarrow conv(i).price + increment$
$\wedge\, conv(i).la \leftarrow vc$
27 $\wedge \mathcal{I}_A \ll A, multicast(\mathsf{announce}, conv(i).B, conv(i).price, EA, i) \gg$
23′ after b does an accept in the EA protocol with conversation identifier $(j(ts), i)$
24′ if the auction hasn't timed out and been terminated
25′ then if rule R1 applies
26′ update all the relevant state variables
27′ and form the intention to multi-cast the updated price to all the bidding agents

– The blank identifier mode: the initiator sends its part of the CID with the first ACL message, leaving the receiver's part of the tuple blank. If the peer agent wishes to reply, it then sends its ACL reply message as well as filling in its part of the CID tuple.

We follow the latter approach, thus reducing the number of messages exchanged.

In figure 7, we illustrate an example of a multi-party conversation (these are the recruit and ask messages appearing in Table 1) with the use of the ACTP. Each agent interacts only with its ACTP module. This module is responsible for sending the ACL message as well as the CID. Figure 8 shows a decomposition of the ACTP module illustrating the conceptual model of the ACTP and the way the CIDs can be integrated within it.

In the intentional specifications appearing in sections 3 and 4, we saw the CID appearing as a parameter within the ACL message (that is the syntax of our custom ACL), this must now be separated since the ACTP is intended to be

Table 3. Intentional Specifications: Timeout Events

Auctioneer's triggers for dealing with timeouts	
28	$\exists i.conv(i).time < 0 \rightarrow \mathcal{I}_A \ll A, timeout(i) \gg$
28′	if there is any conversation identifier i for which the value of the *time* state variable drops below zero, then form the intention to perform a *timeout* action
29	$[A, timeout(i)]$
30	$conv(i).B \leftarrow conv(i).B \ \{conv(i).buyer\}$
31	$\wedge \mathcal{I}_A \ll A, \mathsf{sold}(buyer, \_, EA, (i(vc), cid(i, buyer)) \gg$
32	$\wedge \mathcal{I}_A \ll A, multicast(\mathsf{end}, conv(i).B, \_, EA, i) \gg$
29′	after a timeout in a particular auction
30′	remove the winner (*buyer*) from set of bidding agents
31′	confirm the sale to the buyer (strictly we should write $conv(i).buyer$ but this is obvious from context)
32′	end the auction with a multicast to the rest

sufficiently general to handle a message from any ACL (e.g. FIPA or KQML). Each agent that seeks to communicate will provide its ACTP module with the ACL message as well as the CID (a tuple). In this way we give explicit semantics to the CIDs. The ACTP cannot be expected to be capable of parsing a message from any unknown ACL, so its Conversation Identifiers Module simply creates a wrapper for the ACL message as a string and the CIDs (see Figure 9 for the decomposition of the Conversation Identifiers Module). The CID tuple will either contain only one integer (if this is the first message of the interaction) or two integers (if this a message of a conversation under way).

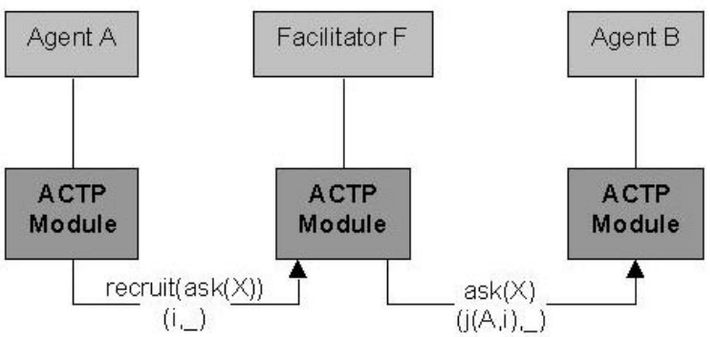


Fig. 7. ACTP Test Environment

The ACTP stores in its memory the full history of every conversation and it uses the CIDs of each conversation to sort them. In the event of an agent crashing during a conversation, and subsequently restarting with no recollection

of it, the memory enables the agent to resume the conversation from the point where it left it. This extra robustness is achieved by imitating the mechanistic interactions of the internet protocols (for example the way TCP exchanges SYN and ACK numbers, although our system can do this concurrently as well as in a handshaking phase). We can provide the agents, at a higher level of abstraction, with a more robust and faster way of dealing with conversations and errors in the communications infrastructure. These additional properties are particularly relevant to e-commerce applications, where vendors and consumers (represented by selling and buying agents) are especially sensitive to delays and downtime.

6 Summary and Conclusions

This paper has addressed two primary concerns:

- specifying multi-party protocols in formal, precise and unambiguous terms; and
- implementing such specifications over an open, robust and efficient platform.

Both aspects are essential for agent-mediated electronic commerce. We argue that it is necessary for open multi-agent systems that employ Facilitators and Auctioneers to make the language and protocols publically available, so that heterogeneously designed, developed and deployed agents can interoperate seamlessly. Furthermore, the communications infrastructure has to be quick and reliable to ensure take-up and continued usage. The combined developments described in this paper are, we believe, useful steps in addressing these concerns.

In this paper, we have taken a general semantic framework for specifying the semantics of an Agent Communication Language, which concentrated on one-to-one conversations, and extended it to cover multi-party conversations as found in brokerage and auction protocols. The specific extensions introduced were:

- Increased range of the possible intended replies, so that the 'meaning' of a speech act between a speaker and hearer in one protocol could, at the 'action' level, be the intention to perform another speech act in a new protocol with a third party;
- Conversation identifiers that were structures (2-tuples), one element of which was supplied by each party to the speech act, and could be parameterised with additional information. The identifiers were then brought into the semantics at the intentional level rather than being or considered as pragmatics as in the FIPA ACL semantics;
- A richer representation of protocol states, so that states were no longer single integers but could include a number of variables. These variables could change value after either the speaker sent the message, and reflect the fact that both the speaker and hearer are changing state when a speech act is performed.

We then used these extensions to give precise, detailed formal specifications (reference implementation models) of the meaning of particular speech acts in the

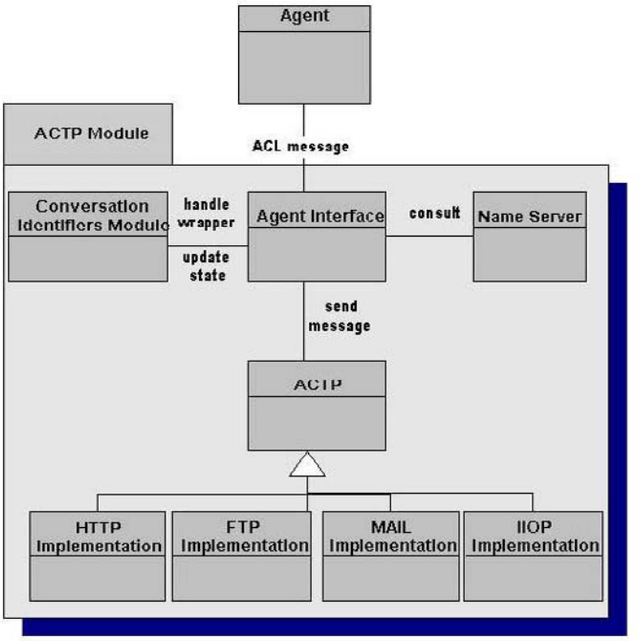


Fig. 8. ACTP Module

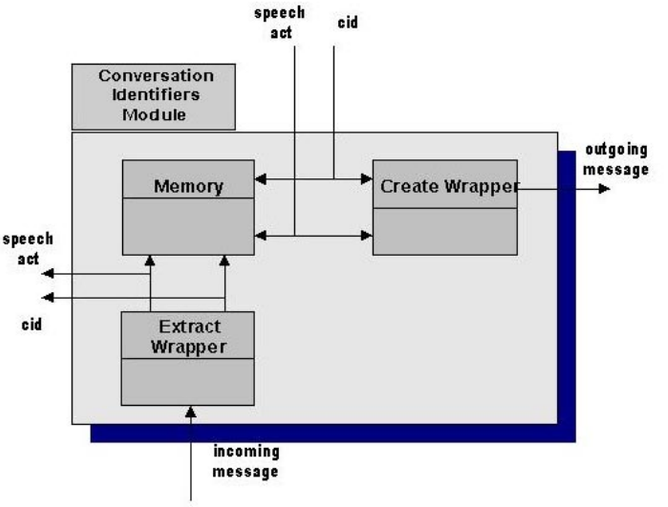


Fig. 9. CIM (Conversation Identifier's Module)

context of a specific protocol. The intention is then, of course, for developers to be able to implement such specifications in agents which could then inter-operate in any open multi-agent system that was using this communication language (performative, protocols and specifications) for uni-cast and multi-cast messages. Broadcast messages are currently out of the scope of this framework (see [8] for an alternative method).

We used an agent-oriented middleware that offers a number of novel features that enhance communication in multi-agent systems. The development of the ACTP was accomplished on the basis of an extensible and well-structured design. Furthermore, the ACTP provides various levels of encapsulation and handles conversations in an efficient manner, thus shielding the agents from the complexity of the communication process. Due to the fact that a significant effort was spent on the implementation of the error handling mechanism, this communication infrastructure has proved to be very reliable and robust.

Having developed the first version of the Agent Communication Transfer Protocol we suggest that it contains a significant degree of generality, reliability and flexibility to support agent conversations in heterogeneous environments. Therefore, the ACTP can be standardised as a communication platform for multi-agent systems. Ultimately, the ACTP can be integrated with other middleware systems (i.e. FIPA-OS [9]) in order to produce a more complete communication transport that will offer high-level interoperability in agent interactions.

We believe that analysing multi-party conversations in terms of their individual components makes the requirements on each interaction clearer, and so assists in determining where the dependencies occur (e.g. sequence, cross-reference, and so on). Having done this analytic design, it is of course feasible to re-constitute the original presentation, if it helps the designer. It is also our contention that the performative semantics is clearer from being defined in context of use (pace Wittgenstein) rather than being defined, syntactically or semantically, as a composite or complex speech act [2,6].

We would further argue that if conversation identifiers are going to be used in an ACL, then it is worth using them, mechanistically as here, for guaranteeing unique conversations and for passing additional information, to be interpreted in context. However, in [2] it was shown how the 'commitments' in composite acts could be derived from the conditions on the primitive ones, and one area for further research is how such commitments are preserved across linked protocols.

We also observed that the auction and brokerage protocols have been used widely in MAS, although formal specifications appeared to be in short supply. The FIPA experience, though, and the work in this paper, suggest that achieving unambiguous specifications for multi-party protocols requires a wide range of considerations to be met. Indeed, the logical specification language required to capture all the different aspects (time, state, events, intentions, axioms, etc.) needs to be extremely powerful and expressive, and correspondingly complex. It is therefore clear that when it comes to providing standard specifications, we are treading a fine line between being too dense and arcane, and being useful and understandable. It remains the case, too, that the formal specification language

here has an intuitive but somewhat loose semantics, and this is being addressed in current research [10].

We maintain though that a systematic approach to design at different levels of 'meaning' offers definite development advantages, in terms of clearer interfaces, open standards, modularity and re-use. Furthermore, the design process can expose new problems. For example, the Facilitator tells in lines 12-13 of Table 1 have F telling X, without necessarily believing X itself. In open systems it might be that F is liable for what it 'says', and that therefore quoting is required, or some other safeguard.

The way the English Auction protocol has been specified and implemented within our framework may point the way forward. We have had to make use of variables which control how the agents follow the protocol. There is also scope for identifying the various roles that the participants play at certain points of the protocol according to the state. It therefore appears that there is an underlying richer state description which cannot be represented in the finite state diagram. Ideally we would like to have an explicit representation of roles and control variables within the general framework, where the action level diagrams and semantics use these roles, rather than specifying them solely in the logical implementation stage. This will require another extension of the semantic framework and is the subject of ongoing research [7], but the result will be a clearer separation of the constituent 'meanings' of speech acts and protocols, plus their dependencies and relations, which combined together provide a more accessible specification.

Acknowledgements. This work has been undertaken in the context of the UK-EPSRC/Nortel Networks funded Project CASBAh (Common Agent Service Brokering Architecture, GR/L34440) and the support from these funding bodies is gratefully acknowledged. We are also very appreciative of the detailed and constructive comments of the anonymous reviewers of the original articles, and thank the editors for their support.

References

1. A. Artikis, J. Pitt, and C. Stergiou. Agent communication transfer protocol. In C. Sierra and M. G. an dJ. Rosenschein, editors, *Proceedings Autonomous Agents AA2000*, pages 491–498. ACM Press, 2000.
2. P. Cohen and H. Levesque. Communicative actions for artificial agents. In V. Lesser, editor, *Proceedings ICMAS95*. AAAI Press, 1995.
3. R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Using colored petri nets for conversation modeliing. In F. Dignum and B. Chaib-draa, editors, *IJCAI'99 Workshop on Agent Communication Languages,*. Stockholm, Sweden, 1999.
4. FACTS. EU ACTS Project AC317: FIPA agent communication technologies and services. http://www.labs.bt.com/profsoc/facts, 1999.
5. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1995.

6. FIPA. FIPA'97 specification part 2: Agent communication language. FIPA (Foundation for Intelligent Physical Agents), http://drogo.cselt.stet.it/fipa/, 1997.
7. F. Guerin and J. Pitt. A semantic framework for specifying agent communication languages. In *Proceedings ICMAS-2000*, pages 395–396. IEEE Computer Society Press, 2000.
8. S. Kumar, M. Huber, D. McGee, P. Cohen, and H. Levesque. Semantics of agent communication languages for group interaction. In *Proceedings 17th National Conference on Artificial Intelligence*. Austin, 2000.
9. NortelNetworks. FIPA-OS (FIPA Open Source). http://www.nortelnetworks.com/fipa-os, 2000.
10. J. Pitt. An operational semantics for intentional specifications of agent behaviour in communication protocols. In M. Greaves, editor, *Proceedings AA2000 Workshop on Agent Languages and Conversation Policies*, pages pp31–40. Barcelona, 1999.
11. J. Pitt and A. Mamdani. Designing agent communication languages for multi-agent systems. In F. Garijo and M. Boman, editors, *Multi-Agent System Engineering MAAMAW'99*, volume LNAI1647, pages 102–114. Springer-Verlag, 1999.
12. J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *Proceedings 16th International Joint Conference on Artificial Intelligence IJCAI'99*, pages 485–491. Morgan-Kaufmann, 1999.
13. I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory. In Y. Demazeau, editor, *Proceedings ICMAS98*. IEEE Press, 1998.
14. C. Stergiou, J. Pitt, F. Guerin, and A. Artikis. Implementing multi-party conversations. In R. Loganantharaj, G. Palm, and M. Ali, editors, *Proceedings IEA/AIE 2000*, pages pp4–13. Springer-Verlag, 2000.
15. Y. Takada, H. Iciki, M. Okada, and T. Mohri. Agent brokerage proposal to cfp6. Fujitsu Laboratories Ltd., Fukuoka, Japan. FIPA CFP6_003,, 1999.
16. Y. Takada, H. Iciki, M. Okada, and T. Mohri. A proposal on agent brokerage. Fujitsu Laboratories Ltd., Fukuoka, Japan. FIPA CFP5_008, http://www.fipa.org/Common_docs/cfp5_008.html, 1999.
17. M. Venkatraman and M. Singh. Verifying compliance with commitment protocols: Enabling open web-based multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):pp217–236, 1999.

Issues in the Design of Negotiation Protocols for Logic-Based Agent Communication Languages*

Michael Wooldridge and Simon Parsons

Department of Computer Science
University of Liverpool, Liverpool L69 7ZF
United Kingdom
{m.j.wooldridge,s.d.parsons}@csc.liv.ac.uk

Abstract. This paper considers the design of negotiation protocols for logic-based agent communication languages. We begin by motivating the use of such languages, and introducing a formal model of logic-based negotiation. Using this model, we define two important computational problems: the success problem (given a particular negotiation history, has agreement been reached?) and the guaranteed success problem (does a particular negotiation protocol guarantee that agreement will be reached?) We then consider a series of progressively more complex negotiation languages, and consider the complexity of using these languages. We conclude with a discussion on related work and issues for the future.

1 Introduction

Negotiation has long been recognised as a central topic in multi-agent systems [10,6]. Much of this interest has arisen through the possibility of automated trading settings, in which software agents bargain for goods and services on behalf of some end-user [8].

One obstacle currently preventing the vision of agents for electronic commerce from being realised is the lack of standardised agent communication languages and protocols to support negotiation. To this end, several initiatives have begun, with the goal of developing such languages and protocols. Most activity in this area is currently focused on the FIPA initiative [3]. The FIPA community is developing a range of agent-related standards, of which the centrepiece is an agent communication language known as "ACL". This language includes a number of performatives explicitly intended to support negotiation [3, pp17–18].

Our aim in this paper is to consider the use of languages like FIPA's ACL for negotiation. In particular, we focus on the use of *logical* languages for negotiation. The use of logic for negotiation is not an arbitrary choice. For example, logic has proved to be powerful tool with which to study the expressive power and computational complexity of database query languages. We believe it will have similar benefits for the analysis of negotiation languages.

We consider two distinct aspects of logical negotiation languages:

* This research was supported by the EPSRC under grant GR/M07076 and the EC under grant IST–1999-10998 (SLIE).

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 70–83, 2001.

– *Adequacy*: Which negotiation languages are appropriate for which application domains? Is a given language sufficiently expressive for a particular domain?
– *Simplicity*: Is a chosen negotiation language likely to be usable in practice?

Of course, there is a tradeoff between adequacy and simplicity: the more powerful a language is, the less tractable it becomes.

In the following section, we introduce a general formal framework for logic-based negotiation. In particular, we define the concept of a negotiation history, and consider various possible definitions of what it means for negotiation to succeed on such a history: we refer to this as the *success* problem. In section 4, we define *protocols* for negotiation, and consider the problem of when a particular protocol guarantees that agreement between negotiation participants will be reached: we refer to this as the *guaranteed success* problem. In section 5, we consider three progressively more complex languages for negotiation. We begin with propositional logic, and show that, for this language, the guaranteed success problem is in the second tier of the polynomial hierarchy (it is Π_2^p-complete, and hence unlikely to be tractable even if we were given an oracle for NP-complete problems). We then investigate how the complexity of the problem varies, depending on the choice of negotiation language and the properties of the protocol.

We then present two further negotiation languages, which are more suited to electronic commerce applications; the second of these is in fact closely based on the negotiation primitives provided in the FIPA agent communication standard [3]. We show that the success problem for these languages is provably intractable (they have double exponential time lower bounds). We conclude by briefly discussing related work and issues for future work.

1.1 Background

We are interested in designing negotiation protocols and strategies that may be used by semi-autonomous software agents in electronic commerce scenarios [8]. Furthermore, we require that the agents engaged in such scenarios will make use of communication languages such as KQML [7], or the increasingly well-known FIPA agent communication language [4].

There is by now a significant literature on automated negotiation; Rosenschein and Zlotkin's pioneering work on the use of game theoretic techniques in negotiation is possibly the best known example [10]. Kraus provides a summary of relevant work in [6], and Sandholm provides a detailed survey of the game theoretic foundations in [11]. By and large, this work has focussed on two issues. The first is the design of negotiation protocols, which provide the "rules of encounter" for interacting agents. Typically, the designer of a negotiation protocols (the "principal" in game theory terminology [1, p526]) has some criteria in mind when designing the protocol. A classic example is that of someone who wishes to sell some particular good to one of a group of agents; the obvious criteria that such a person will have is to maximise revenue. In game theory, the process of designing a protocol with the aim of satisfying such criteria is known as *mechanism design* [1, p523]. There are many possible properties that one might wish to prove of a negotiation protocol. Possible properties include, for example [11, p204]:

- *Guaranteed success*.
  A protocol guarantees success if it ensures that, eventually, agreement is certain to be reached.
- *Maximising social welfare*.
  Intuitively, a protocol maximises social welfare if it ensures that any outcome maximises the sum of the utilities of negotiation participants. If the utility of an outcome for an agent was simply defined in terms of the amount of money that agent received in the outcome, then a protocol that maximised social welfare would maximise the *total* amount of money "paid out".
- *Pareto efficiency*.
  A negotiation outcome is said to be Pareto efficient if there is no other outcome that will make at least one agent better off without making at least one other agent worse off. Intuitively, if a negotiation outcome is not Pareto efficient, then there is another outcome that will make at least one agent happier while keeping everyone else at least as happy.
- *Individual rationality*.
  A protocol is said to be individually rational if following the protocol — "playing by the rules" — is in the best interests of negotiation participants. Individually rational protocols are essential because without them, there is no incentive for agents to engage in negotiations.
- *Stability*.
  A protocol is *stable* if it provides all agents with an incentive to behave in a particular way. The best-known kind of stability is *Nash equilibrium*: two strategies s and s' are said to be in Nash equilibrium if under the assumption that one agent is using s, the other can do no better than use s', and vice versa.
- *Simplicity*.
  A "simple" protocol is one that makes the appropriate strategy for a negotiation participant "obvious". That is, a protocol is simple if using it, a participant can easily (tractably) determine the optimal strategy.
- *Distribution*.
  A protocol should ideally be designed to ensure that there is no "single point of failure" (such as a single arbitrator), and ideally, so as to minimise communication between agents.

The fact that even quite simple negotiation protocols can be proven to have such desirable properties as these accounts in no small part for the success of game theoretic techniques for negotiation.

In this paper, we combine ideas from mechanism design with the use of agent communication languages such as KQML and FIPA [7,4]. These languages are being developed in an effort to enable agents built by different organisations using different hardware and software platforms to inter-operate with one-another. The two languages are similar, and we will here review KQML. This language is essentially an "outer" language for messages: it defines a simple LISP-like format for messages, and 41 *performatives*, or message types, that define the intended meaning of a message. Example KQML performatives include `ask-if` and `tell`. The *content* of messages was not considered part of the KQML standard, but another language called KIF was also defined, to express such content. KIF is essentially classical first-order predicate logic, recast in a LISP-like syntax.

To better understand the KQML language, consider the following example of a KQML message [7, p354]:

```
(ask-one
   :content  (PRICE IBM ?price)
   :receiver stock-server
   :language LPROLOG
   :ontology NYSE-TICKS
)
```

The intuitive interpretation of this message is that the sender is asking about the price of IBM stock. The performative is `ask-one`, which an agent will use to ask a question of another agent where exactly one reply is needed. The various other components of this message represent its attributes. The most important of these is the `:content` field, which specifies the message content. In this case, the content simply asks for the price of IBM shares. The `:receiver` attribute specifies the intended recipient of the message, the `:language` attribute specifies that the language in which the content is expressed (the *content* language) is called LPROLOG (the recipient is assumed to "understand" LPROLOG), and the final `:ontology` attribute defines the terminology used in the message.

In this paper, we will investigate some issues surrounding the design of agent interaction protocols — specifically, *negotiation* protocols — for agents that communicate using such languages. Negotiation is recognised as an important application of such communication languages, and the FIPA language explicitly provides performatives that are intended to support negotiation. For example, the FIPA performative `cfp` (call for proposals) allows an agent to advertise the fact that it has something (e.g., a task that it desires to be solved), and invites bids from other agents with respect to this task.

Although there have been many implementations of both the KQML and FIPA languages, to date there is little theory that either supports or disproves the view that such languages can be used for applications such as negotiation. In this paper, we begin to rectify this omission. We present a simple formal model of negotiation, in which agents communicate by exchanging formulae of some logic-based agent communication language. We then consider a number of progressively richer logics for negotiation, and investigate the design of protocols for the use of such languages. In particular, we investigate the computational complexity of proving that protocols have certain properties. Before proceeding, we survey the basic concepts from complexity theory that are required for the article.

1.2 Complexity Concepts and Classes

Throughout the paper, we make use of the terminology and tools of complexity theory. Although we provide a summary of main concepts from complexity theory that we use, we emphasise that a detailed presentation is beyond the scope of this paper. We refer the reader to [5,9] for details. We start from the complexity classes P (of languages/problems that may be recognised/solved in deterministic polynomial time), and NP (of languages/problems that may be recognised/solved in non-deterministic polynomial time). If $\mathcal{C}_1$ and $\mathcal{C}_2$ are complexity classes, then we denote by $\mathcal{C}_1^{\mathcal{C}_2}$ the class of languages/problems

that are in $\mathcal{C}_1$ assuming the availability of an oracle for languages/problems in $\mathcal{C}_2$ [9, pp415–417]. Thus, for example, NP$^{\text{NP}}$ denotes the class of languages/problems that may be recognised/solved in non-deterministic polynomial time, assuming the presence of an oracle for languages/problems in NP. A language that is complete for NP$^{\text{NP}}$ would thus be NP-complete even if we had "free" answers to NP-complete problems (such as propositional logic satisfiability). We define the *polynomial hierarchy* with reference to these concepts [9, pp423–429]. First, define

$$\Sigma_0^p = \Pi_0^p = \text{P}$$

Thus both Σ_0^p and Π_0^p denote the classes of languages/problems that may be recognised/solved in deterministic polynomial time. We then inductively define the remaining tiers of the hierarchy, as follows:

$$\Sigma_{u+1}^p = \text{NP}^{\Sigma_u^p} \qquad \Pi_{u+1}^p = \text{co-}\Sigma_{u+1}^p$$

Thus Σ_1^p is simply the class NP, and Π_1^p is the class co-NP, while $\Sigma_2^p = \text{NP}^{\text{NP}}$ and $\Pi_2^p = \text{co-NP}^{\text{NP}}$.

2 Formal Preliminaries

We begin by assuming a non-empty set $Ag = \{1, \ldots, n\}$ of *agents*. These agents are the negotiation participants, and it is assumed they are negotiating over a finite set $\Omega = \{\omega, \omega', \ldots\}$ of *outcomes*. For now, we will not be concerned with the question of exactly what outcomes are, or whether they have any internal structure — just think of outcomes as possible states of affairs.

Each agent $i \in Ag$ is assumed to have preferences with respect to outcomes, given by a partial pre-order $\succeq_i \subseteq \Omega \times \Omega$. Following convention, we write $\omega \succeq_i \omega'$ to mean $(\omega, \omega') \in \succeq_i$. If $\omega \succeq_i \omega'$, then agent i likes outcome ω at least as much as ω'.

Negotiation proceeds in a series of rounds, where at each round, every agent puts forward a proposal. A proposal is a *set of outcomes*, that is, a subset of Ω. The intuition is that in putting forward such a proposal, an agent is asserting that any of these outcomes is acceptable.

In practice, the number of possible outcomes will be prohibitively large. To see this, consider that in a domain where agents are negotiating over n attributes, each of which may take one of m values, there will be m^n possible outcomes. This means it will be impractical for agents to negotiate by explicitly enumerating outcomes in the proposals they make. Instead, we assume that agents make proposals by putting forward a formula of a *logical negotiation language* — a language for describing deals. In much of this paper, we will be examining the implications of choosing different negotiation languages, and in order to compare them, we must make certain general assumptions. The first is that a negotiation language $\mathcal{L}$ is associated with a set $wff(\mathcal{L})$ of *well-formed formulae* — syntactically acceptable constructions of $\mathcal{L}$. Next, we assume that $\mathcal{L}$ really *is* a logical language, containing the usual connectives of classical logic: "$\wedge$" (and), "$\vee$" (or), "$\neg$" (not), "$\Rightarrow$" (implies), and"$\Leftrightarrow$" (iff) [2, p32]. In addition, $\mathcal{L}$ is assumed to have a Tarskian satisfaction relation "$\models_{\mathcal{L}}$", which holds between outcomes Ω and members of $wff(\mathcal{L})$.

We write $\omega \models_{\mathcal{L}} \varphi$ to indicate that outcome $\omega \in \Omega$ satisfies formula $\varphi \in \mathit{wff}(\mathcal{L})$. The classical connectives of $\mathcal{L}$ are assumed to have standard semantics, so that, for example, $\omega \models_{\mathcal{L}} \varphi \wedge \psi$ iff both $\omega \models_{\mathcal{L}} \varphi$ and $\omega \models_{\mathcal{L}} \psi$. If $\varphi \in \mathit{wff}(\mathcal{L})$, then we denote by $[\![\varphi]\!]_{\mathcal{L}}$ the set of outcomes that satisfy φ, that is, $[\![\varphi]\!]_{\mathcal{L}} = \{\omega \mid \omega \models_{\mathcal{L}} \varphi\}$.

As we noted above, negotiation proceeds in a series of rounds, where at each round, every agent puts forward a formula of $\mathcal{L}$ representing the proposal it is making. A single round is thus characterised by a tuple $\langle \varphi_1, \ldots, \varphi_n \rangle$, where for each $i \in \mathit{Ag}$, the formula $\varphi_i \in \mathit{wff}(\mathcal{L})$ is agent i's proposal. Let R be the set of all possible rounds:

$$R = \underbrace{\mathit{wff}(\mathcal{L}) \times \cdots \times \mathit{wff}(\mathcal{L})}_{|\mathit{Ag}| \text{ times}}$$

We use $r, r', \ldots$ to stand for members of R, and denote agent i's proposal in round r by $r(i)$.

A *negotiation history* is a finite sequence of rounds $(r_0, r_1, \ldots, r_k)$. Let $H = R^*$ be the set of all possible negotiation histories. We use $h, h', \ldots$ to stand for members of H. If $u \in \mathbb{N}$, then we denote the u'th round in history h by $h(u)$. Thus $h(0)$ is the first round in h, $h(1)$ is the second, and so on.

3 Types of Agreement

Given a particular negotiation history, an important question to ask is whether or not agreement has been reached with respect to this history. For many negotiation scenarios, this problem is far from trivial: it may well not be obvious to the negotiation participants that they have in fact made mutually acceptable proposals.

In fact, we can identify several different types of agreement condition, which may be used in different negotiation scenarios. It is assumed that the negotiation participants will settle on the agreement condition to be used before the actual negotiation process proper begins. The selection of an agreement condition is thus a *meta-negotiation* issue, which falls outside the scope of our work.

To understand what agreement means in our framework, it is helpful to view a negotiation history as a matrix of $\mathcal{L}$-formulae, as follows.

$$\begin{array}{cccc} \varphi_1^0 & \varphi_1^1 & \cdots & \varphi_1^k \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_n^0 & \varphi_n^1 & \cdots & \varphi_n^k \end{array}$$

In this matrix, φ_i^u is the proposal made by agent i in round $u \in \mathbb{N}$. The simplest type of agreement is where "all deals are still valid" — once an agent has made a proposal, then this proposal remains valid throughout negotiation. One important implication of such agreement is that since all previous offers are still valid, it makes no sense for agents to make more restrictive proposals later in negotiation: we emphasise that our formal approach does not depend on this assumption — other types of agreement are possible, as we demonstrate below.

In this case, determining whether agreement has been reached means finding at least one outcome $\omega \in \Omega$ such that every agent i has made a proposal $\varphi_i^{u_i}$ where $\omega \models_{\mathcal{L}} \varphi_i^{u_i}$. In other words, agreement will have been reached if every agent i has made a proposal $\varphi_i^{u_i}$ such that $[\![\varphi_1^{u_1}]\!]_{\mathcal{L}} \cap \cdots \cap [\![\varphi_n^{u_n}]\!]_{\mathcal{L}} \neq \emptyset$. This will be the case if the formula $\varphi_1^{u_1} \wedge \cdots \wedge \varphi_n^{u_n}$ is satisfiable. Given a history h, expressed as a matrix as above, agreement has been reached iff the following formula is satisfiable:

$$\bigwedge_{i \in Ag} \left(\bigvee_{u_i \in \{0,\ldots,k\}} \varphi_i^{u_i} \right) \tag{1}$$

Given a history $h \in H$, we denote the formula (1) for h by φ_h. We refer to the problem of determining whether agreement has been reached in some history h as the *success problem*.

Note that the success problem can clearly be reduced to the satisfiability problem for the negotiation language using only polynomial time. The types of agreement we consider in this paper are all variants of satisfiability. However, it is important to understand that (1) is *by no means* the only type of agreement that we can define.

A different type of agreement is where prior negotiation history is disregarded: the only proposals that matter are the most recent. Agreement will be reached in such a history iff the conjunction of proposals made on the final round of negotiation is satisfiable. The success condition is thus:

$$\bigwedge_{i \in Ag} \varphi_i^{|h|-1} \tag{2}$$

A third possible definition of agreement is that agents must converge on "equivalent" proposals: where the proposals made agree on all particulars. Such agreement is captured by the following condition.

$$\varphi_1^{|h|-1} \Leftrightarrow \cdots \Leftrightarrow \varphi_n^{|h|-1} \tag{3}$$

4 Protocols

Multi-agent interactions do not generally take place in a vacuum: they are governed by *protocols* that define the "rules of encounter" [10]. Put simply, a protocol specifies the proposals that each agent is allowed to make, as a function of prior negotiation history. Formally, a protocol π is a function $\pi : H \rightarrow \wp(R)$ from histories to sets of possible rounds. One important requirement of protocols is that the number of rounds they allow on any given history should be at most polynomial in the size of the negotiation scenario. The intuition behind this requirement is that otherwise, a protocol could allow an exponential number of rounds — since an exponential number of rounds could not be enumerated in practice, such protocols could never be implemented in any realistic domain.

We will say a history is *compatible* with a protocol if the rounds at each step in the history are permitted by the protocol. Formally, history h is compatible with π if the following conditions hold:

1. $h(0) \in \pi(\epsilon)$ (where ϵ is the empty history); and
2. $h(u) \in \pi((h(0), \ldots, h(u-1)))$ for $1 \leq u < |h|$.

Now, what happens if $\pi(h) = \emptyset$? In this case, protocol π says that there are no allowable rounds, and we say that negotiation has *ended*. The end of negotiation does not imply that the process has succeeded, but rather simply that the protocol will not permit it to continue further.

Notice that negotiation histories can in principle be unrealistically long. To see this, suppose that the set Ω of outcomes is finite. Then every agent has $2^{|\Omega|}$ possible proposals, meaning that even if an agent never makes the same proposal twice, negotiation histories can be exponentially long. We say protocol π is *efficient* if it guarantees that negotiation will end with a history whose length is polynomial in the size of Ω and Ag. Efficiency seems a reasonable requirement for protocols, as exponentially long negotiation histories could never be practical.

When we create an agent interaction protocol, we attempt to *engineer* the protocol so that it has certain desirable properties [10, pp20–22]. In this paper, we will be concerned with just one property of protocols: whether or not they *guarantee success*. We will say a protocol π guarantees success if agreement is reached on every negotiation history compatible with π. Protocols that guarantee success are desirable, for obvious reasons. But consider the nature of this problem. In general, a protocol allows *branching* during the negotiation process. This branching arises because negotiation participants are allowed to make a number of proposals at each round. It is easy to see that the number of negotiation histories of length l compatible with a negotiation protocol with branching factor b will be b^l, that is, exponential in the length of the protocol. Thus determining whether or not a protocol guarantees success will intuitively involve solving an exponential number of individual success problems, which are themselves logical satisfiability problems. This suggests that the guaranteed success problem is likely to be computationally hard; in the next section, when we consider some concrete negotiation languages, we will see just how hard it actually is.

Before proceeding, however, we need to say something about how protocols are *represented* or *encoded*. (This is a technical matter that is important when we come to consider some decision problems later in the paper.) We will assume that (efficient) protocols are represented as a two-tape Turing machine: the machine takes as input a representation of prior negotiation history on its first tape, and writes as output the set of possible subsequent rounds on the second tape. We will further assume that the Turing machine requires time polynomial in the size of $|Ag \times \Omega|$ in order to carry out this computation.

5 Example Negotiation Languages

In this section, we present a series of progressively more complex negotiation languages and protocols. We begin with propositional logic. Although this logic is not appropriate for many negotiation domains, it is a useful starting point for our analysis, and the results we establish for propositional logic can be regarded as "lower bounds" for other, more expressive negotiation languages. It is difficult to imagine a simpler negotiation language

being of any practical use. (Proof for the theorems given in this section may be found in [13].)

Example 1: Classical Propositional Logic.

For the first example, we will assume that agents are negotiating over a domain that may be characterised in terms of a finite set of attributes, each of which may be either true ($\top$) or false ($\bot$). An outcome is thus an assignment of true or false to every attribute. The proposals possible in this kind of language are exactly the kind of outcomes typically considered in decision theory. For example, in the classic "oil wildcatter" problem, agents might be involved in a negotiation about which of two oil fields to drill in, and proposals might be of the form:

- $drillField_A \wedge \neg drillField_B$
- $\neg drillField_A \wedge drillField_B$

The obvious language with which to express the properties of such domains is classical propositional logic, which we will call $\mathcal{L}_0$. The set $wff(\mathcal{L}_0)$ contains formulae constructed from a finite set of proposition symbols $\Phi = \{p, q, r, \ldots\}$ combined into formulae using the classical connectives "$\neg$" (not), "$\wedge$" (and), "$\vee$" (or), and so on. It is easy to see that the success problem for $\mathcal{L}_0$ histories will be NP-complete. More interesting is the fact that we can establish the complexity of the guaranteed success problem for $\mathcal{L}_0$.

Theorem 1 *The guaranteed success problem for efficient $\mathcal{L}_0$ protocols is Π_2^p-complete.*

Note that Π_2^p-complete problems are generally reckoned to be worse than, say, co-NP-complete or NP-complete problems, although the precise status of such problems in the relation to these classes is not currently known for sure [5]. Theorem 1 should therefore be regarded as an extremely negative result.

An obvious question to ask is whether the complexity of the guaranteed success problem can be reduced in some way. There are two main factors that lead to the overall complexity of the problem: the complexity of the underlying negotiation language, and the "branching factor" of the protocol. It is possible to prove that if we chose a negotiation language whose satisfiability problem was in P, then the complexity of the corresponding guaranteed success problem would be reduced one level in the polynomial hierarchy. To make this concrete, let us consider the subset $\mathcal{L}_0^{HC}$ of $\mathcal{L}_0$ in which formulae are restricted to be conjunctions of *Horn clauses*. A clause is said to be Horn if it contains at most one positive (unnegated) literal. It is well known that there is a deterministic polynomial time algorithm for determining the satisfiability of $\mathcal{L}_0^{HC}$ [9, pp78–79]. We can prove the following.

Theorem 2 *The guaranteed success problem for efficient $\mathcal{L}_0^{HC}$ protocols is Π_1^p-complete.*

With respect to the branching factor of the protocol, suppose we have a *deterministic* $\mathcal{L}_0$ protocol π — one in which $|\pi(h)| \leq 1$ for all $h \in H$. Such a protocol is guaranteed to produce a single negotiation history. This allows us to easily establish the following result for deterministic $\mathcal{L}_0$ protocols.

Table 1. The complexity of the guaranteed success problem for deterministic and non-deterministic protocols, using variants of propositional logic.

	Protocol	
Language	Non-deterministic	Deterministic
$\mathcal{L}_0$	Π_2^p-complete	NP-complete
$\mathcal{L}_0^{HC}$	Π_1^p-complete	P

Theorem 3 *The guaranteed success problem for efficient deterministic $\mathcal{L}_0$ protocols is* NP*-complete.*

Similarly, we can establish the following for $\mathcal{L}_0^{HC}$.

Theorem 4 *The guaranteed success problem for efficient deterministic $\mathcal{L}_0^{HC}$ protocols is in* P.

Table 1 summarises our complexity results for protocols based on propositional logic. We remark that determinism is a far too restrictive property to require of realistic protocols.

Before leaving this section, we present a simple example of a protocol that guarantees success for agents negotiating using a subset of propositional logic. We refer to the subset of propositional logic in question as $\mathcal{L}_0^*$. Formulae of $\mathcal{L}_0^*$ are constrained to take the form

$$l_0 \wedge \cdots \wedge l_m$$

where each l_m is a *literal*, that is, either an element of Φ or the negation of an element of Φ. If $\varphi \in \mathit{wff}(\mathcal{L}_0^*)$, then we denote the set of literals in φ by $\mathit{lit}(\varphi)$.

Given two formulae φ and ψ of $\mathcal{L}_0^*$, we say that φ is a *concession* with respect to ψ if φ is a subformula of ψ and $\mathit{lit}(\varphi) \subset \mathit{lit}(\psi)$. So, for example, $p \wedge \neg q$ represents a concession with respect to $p \wedge \neg q \wedge \neg r$, which in turn represents a concession with respect to $p \wedge \neg q \wedge \neg r \wedge s$. If φ represents a concession with respect to ψ, then φ is "less constraining" than ψ, in that $[\![\psi]\!] \subset [\![\varphi]\!]$.

We now turn to the protocol in question. The protocol is simply the monotonic concession protocol of Rosenschein and Zlotkin [10, pp40–41], recast into our logical framework. This protocol is defined by the following two rules:

1. on the first round, every agent is allowed to propose any formula of $\mathcal{L}_0^*$;
2. on round u ($u > 0$), one agent $i \in \mathit{Ag}$ must make a proposal φ_i^u such that φ_i^u represents a concession with respect to φ_i^{u-1}; every other agent puts forward the same proposal that it put forward on round $u - 1$.

It is not difficult to prove the following.

Theorem 5 *The monotonic concession protocol for $\mathcal{L}_0^*$ guarantees success. Moreover, it guarantees that agreement will be reached in $O(|\mathit{Ag} \times \Phi|)$ rounds.*

Note that when using this protocol, the obvious strategy for an agent to play in order to ensure that negotiation concludes as quickly as possible involves conceding literals that *clash* with those of other negotiation participants. For example, in a two agent negotiation scenario, suppose that agent 1 proposed $p \wedge \neg q$ and agent 2 proposed $p \wedge q$. Then the obvious concession for agent 2 to make would involve proposing p subsequently. Otherwise, successful termination would require a further negotiation round.

Example 2: A Language for Electronic Commerce.

Propositional logic is a simple and convenient language to analyse, but is unlikely to be useful for many realistic negotiation domains. In this example, we focus on somewhat more realistic *e-commerce* scenarios, in which agents negotiate to reach agreement with respect to some financial transaction [8]. We present a negotiation language $\mathcal{L}_1$ for use in such scenarios.

We begin by defining the outcomes that agents are negotiating over. The idea is that agents are trying to reach agreement on the values of a finite set $V = \{v_1, \ldots, v_m\}$ of *negotiation issues* [12, pp181–182], where each issue has a natural number value. An outcome $\omega \in \Omega$ for such a scenario is thus a function $\omega : V \rightarrow \mathbb{N}$, which assigns a natural number to each issue.

In order to represent the proposals that agents make in such a scenario, we use a subset of first-order logic. We begin by giving some examples of formulae in this subset.

- $(\mathit{price} = 20) \wedge (\mathit{warranty} = 12)$
  "the price is \$20 and the warranty is 12 months"
- $(15 \leq \mathit{price} \leq 20) \wedge (\mathit{warranty} = 12)$
  "the price is between \$15 and \$20 and the warranty is 12 months"
- $\exists n \cdot \mathit{warranty} \geq 12$
  "the warranty is longer than 12 months"

Formally, $\mathcal{L}_1$ is the subset of first-order logic containing: a finite set V of variables, (with at least one variable for each negotiation issue); a set C of constants, one for each natural number; the binary addition function "$+$"; the equality relation "$=$"; and the less-than relation "$<$".

There is both good news and bad news about $\mathcal{L}_1$: the good news (in comparison with full first-order logic) is that it is decidable; the bad news (in comparison with propositional logic) is that it is *provably* intractable. In fact, we can prove that $\mathcal{L}_1$ has a double exponential time lower bound. In what follows, $TA[t(n), a(n)]$ is used to denote the class of problems that may be solved by an alternating Turing machine using at most $t(n)$ time and $a(n)$ alternations on inputs of length n [5, p104].

Theorem 6 *The success problem for $\mathcal{L}_1$ is complete for $\bigcup_{k>0} TA[2^{2^{n^k}}, n]$.*

The details of the class $TA[t(n), a(n)]$ are perhaps not very important for the purposes of this example. The crucial point is that any algorithm we care to write that will solve the general $\mathcal{L}_1$ success problem will have *at least* double exponential time complexity. It follows that such an algorithm is highly unlikely to be of any practical value. With respect to the guaranteed success problem for $\mathcal{L}_1$, we note that since the success problem gives a lower bound to the corresponding guaranteed success problem, the $\mathcal{L}_1$ guaranteed success problem will be at least $\bigcup_{k>0} TA[2^{2^{n^k}}, n]$ hard.

Table 2. Illocutions for the negotiation language $\mathcal{L}_2$.

Illocution	Meaning
request(i, j, φ)	a request from *i* to *j* for a proposal based on φ
offer(i, j, φ)	a proposal of φ from *i* to *j*
accept(i, j, φ)	*i* accepts proposal φ made by agent *j*
reject(i, j, φ)	*i* rejects proposal of φ made by agent *j*
withdraw(i, j)	*i* withdraws from negotiation with *j*

Example 3: A Negotiation Meta-language.

The language used in the previous example is suitable for stating deals, and is thus sufficient for use in scenarios in which agents negotiate by just trading such deals. However, as discussed in [12], the negotiation process is more complex for many scenarios, and agents must engage in persuasion to get the best deal. Persuasion requires more sophisticated dialogues, and, as a result, richer negotiation languages. One such language, based on the negotiation primitives provided by the FIPA ACL [3], and related to that described in [12], includes the illocutions shown in Table 2[1]. In this table, φ is a formula of a language such as $\mathcal{L}_0$ or $\mathcal{L}_1$. In this sense, the language which includes the illocutions is a *meta-language* for negotiation — a language for talking about proposals. For the rest of this example, we will consider a language $\mathcal{L}_2$ which consists of exactly those illocutions in Table 2, where φ is a formula in $\mathcal{L}_1$.

These illocutions work as follows. There are two ways in which a negotiation can begin, either when one agent makes an *offer* to another, or when one makes a *request* to another. A request is a semi-instantiated offer. For example, the following illocution

$$\textit{request}(i, j, (\textit{price} = ?) \wedge (\textit{warranty} = 12))$$

is interpreted as "If I want a 12 month warranty, what is the price?".

Proposals are then traded in the usual way, with the difference that an agent can reply to a proposal with a *reject*, explicitly saying that a given proposal is unacceptable, rather than with a new proposal. Negotiation ceases when one agent *accept*s an offer or *withdraw*s from negotiation. Note that this protocol assumes two agents are engaged in the negotiation. (Many-many negotiations are handled in [12] by many simultaneous two-way negotiations.)

To further illustrate the use of $\mathcal{L}_2$, consider the following short negotiation history between two agents negotiating over the purchase of a used car:

1. $\textit{request}(a, b, (\textit{price} \leq 4000) \wedge (\textit{model} = ?) \wedge (\textit{age} = ?))$
2. $\textit{offer}(b, a, (\textit{price} = 3500) \wedge (\textit{model} = \textit{Escort}) \wedge (\textit{age} = 8))$
3. $\textit{reject}(a, b, (\textit{price} = 3500) \wedge (\textit{model} = \textit{Escort}) \wedge (\textit{age} = 8))$
4. $\textit{offer}(b, a, (\textit{price} = 3900) \wedge (\textit{model} = \textit{Golf}) \wedge (\textit{age} = 6))$
5. $\textit{offer}(a, b, (\textit{price} = 3200) \wedge (\textit{model} = \textit{Golf}) \wedge (\textit{age} = 6))$

[1] Note that the language proposed in [12] also includes illocutions which include the reason for an offer. We omit discussion of this facility here. We also omit the timestamp from the illocutions.

Table 3. The protocol $\pi_{\mathcal{L}_2}$ for $\mathcal{L}_2$ at the uth step of the negotiation.

Agent i says	Agent j replies
$\mathit{request}(i, j, \varphi_i^u)$	$\mathit{offer}(j, i, \varphi_j^u)$
$\mathit{offer}(i, j, \varphi_i^u)$	$\mathit{offer}(j, i, \varphi_j^u)$, or $\mathit{accept}(j, i, \varphi_j^u)$, or $\mathit{reject}(j, i, \varphi_i^u)$, or $\mathit{withdraw}(j, i)$
$\mathit{reject}(i, j, \varphi)$	$\mathit{offer}(j, i, \varphi_j^u)$ or $\mathit{withdraw}(j, i)$
$\mathit{accept}(i, j, \varphi_j^{u-1})$	end of negotiation
$\mathit{withdraw}(i, j)$	end of negotiation

6. $\mathit{offer}(b, a, (\mathit{price} = 3400) \wedge (\mathit{model} = \mathit{Golf}) \wedge (\mathit{age} = 6))$
7. $\mathit{accept}(a, b, (\mathit{price} = 3400) \wedge (\mathit{model} = \mathit{Golf}) \wedge (\mathit{age} = 6))$

Broadly speaking, the illocutions in $\mathcal{L}_2$ are syntactic sugar for the kinds of proposal that we have discussed above: we can map them into $\mathcal{L}_1$ and hence into the framework introduced in section 2. To do this we first need to extend the condition for agreement. In the case where we have two agents, *a* and *b* negotiating, the agreement condition we use is a combination of (2) and (3):

$$(\varphi_a^{|h|-1} \wedge \varphi_b^{|h|-1}) \wedge (\varphi_a^{|h|-1} \Leftrightarrow \varphi_b^{|h|-1}) \tag{4}$$

Thus the agents must not only make mutually satisfiable proposals on the final round, they must make equivalent proposals. Given this, we can prove the following result.

Theorem 7 *The augmented success problem for $\mathcal{L}_2$ is complete for $\bigcup_{k>0} TA[2^{2^{n^k}}, n]$.*

There is also the question of whether success can be guaranteed when negotiating in $\mathcal{L}_2$, and this, of course, depends upon the protocol used. Table 3 gives the protocol used in [12]. We will call this $\pi_{\mathcal{L}_2}$.

Clearly this protocol can lead to negotiations which never terminate (since it is possible for agents to trade the same pair of unacceptable offers for ever). However, it is not unreasonable to insist that conditions are placed upon the protocol in order to ensure that this does not happen and that negotiations eventually terminate. One such condition is that agents make concessions at each stage, that is, that each offer made by an agent is less preferable to that agent than any of its predecessors. The protocol $\pi_{\mathcal{L}_2}$ with this additional condition is an $\mathcal{L}_2$ version of the monotonic concession protocol discussed easier. With the concession condition, and assuming that agents withdraw once φ drops below some threshold, we have:

Theorem 8 *Protocol $\pi_{\mathcal{L}_2}$ guarantees termination. If agents are not allowed to withdraw, then $\pi_{\mathcal{L}_2}$ guarantees success.*

One simple scenario which is captured by $\pi_{\mathcal{L}_2}$ with monotonic concessions is that in which one agent, *i* say, rejects every offer made by the other, *j*, until suitable concessions have been gained. Of course, provided that the end-point is acceptable for *j*, there is nothing wrong with this — and if the concession *i* is looking for are too severe, then *j* will withdraw before making an acceptable offer.

6 Discussion

This paper has identified two important computational problems in the use of logic-based languages for negotiation — the problem of determining if agreement has been reached in a negotiation, and the problem of determining if a particular negotiation protocol will lead to an agreement. Both these problems are computationally hard, and the main contribution of this paper was to show quite how hard they are. In particular the paper showed the extent of the problems for some languages that could realistically be used for negotiations in electronic commerce. This effort is thus complementary to work on defining such languages. Obvious future lines of work are to consider the impact of these results on the design of negotiation languages and protocols, and to extend the work to cover more complex languages. In particular, we are interested in extending the analysis to consider the use of argumentation in negotiation [12].

References

1. K. Binmore. *Fun and Games: A Text on Game Theory*. D. C. Heath and Company: Lexington, MA, 1992.
2. H. B. Enderton. *A Mathematical Introduction to Logic*. The Academic Press: London, England, 1972.
3. FIPA. Specification part 2 — Agent communication language, 1999. The text refers to the specification dated 16 April 1999.
4. The Foundation for Intelligent Physical Agents. See `http://www.fipa.org/`.
5. D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume A: Algorithms and Complexity*, pages 67–161. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
6. S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1-2):79–98, July 1997.
7. J. Mayfield, Y. Labrou, and T. Finin. Evaluating KQML as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 347–360. Springer-Verlag: Berlin, Germany, 1996.
8. P. Noriega and C. Sierra, editors. *Agent Mediated Electronic Commerce (LNAI Volume 1571)*. Springer-Verlag: Berlin, Germany, 1999.
9. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley: Reading, MA, 1994.
10. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press: Cambridge, MA, 1994.
11. T. Sandholm. Distributed rational decision making. In G. Weiß, editor, *Multiagent Systems*, pages 201–258. The MIT Press: Cambridge, MA, 1999.
12. Carles Sierra, Nick R. Jennings, Pablo Noriega, and Simon Parsons. A framework for argumentation-based negotiation. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 177–192. Springer-Verlag: Berlin, Germany, 1998.
13. M. Wooldridge and S. Parsons. Languages for negotiation. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, Germany, 2000.

A Formal Description of a Practical Agent for E-Commerce

Matteo Pradella and Marco Colombetti

Dipartimento di Elettronica e Informazione,
Politecnico di Milano
P.za Leonardo da Vinci, 32,
20133 Milano, Italia
tel. +39-02-2399-{3666, 3686}
fax. +39-02-2399-3411
{pradella,colombet}@elet.polimi.it

Abstract. Software agents are starting to play a significant role in the field of electronic commerce. Particularly, as mediators they must be completely trusted by the user. This paper tackles this subject by proposing a practically tailored formal model, based on BDI. This model is tested on the ground, firstly, by describing an existing agent for electronic commerce; then, by proving some useful trading properties fulfilled by the specification.

1 Introduction

Software agents are mainly programs to which one can delegate a task. Their quality of (semi)-autonomy is especially useful for the increasingly complex environment of electronic commerce. There is little doubt that software agents will play an increasing variety of roles as mediators in this sensitive field. This roles could be risky: think about an agent delegated to buy, e.g., a car. Somehow a delegating person needs to be confident about the automated agent's correctness. This should be an effective stimulus towards the creation and spreading of a truly automated electronic market. It's a strong motivation for the use of clearly written formal specifications for software agents: agents with great powers should be carefully designed and analyzed.

A formal language is a fundamental tool to describe and analyze the behavior of a software agent. So far some well-founded and interesting formal languages have been proposed, suited to describe and to reason about fairly complex agent environments.

This work consists of a practical application of one of these languages and formal tools in the field of agent-based e-commerce. We used one of the most promising of these languages to specify a practical, implemented software agent, namely a variant of a MIT's Kasbah-based selling agent. We then simplified the chosen language and the chosen set of axioms, just to cover what is needed by our agent.

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 84–95, 2001.

Kasbah turned to be a quite interesting testing ground: in fact it is one of the more interesting implemented software agent environment for agent-mediated e-commerce (see [7,9,6,5] for recent surveys of the field).

2 Kasbah

Kasbah [2] is a web-based classified ads system, created at MIT. It implements a virtual marketplace with two classes of autonomous agents: *sellers* and *buyers*.

A selling agent is very like to a classified ad. When users create a new selling agent, they give it a description of the item they want to sell. Unlike traditional classified ads, Kasbah selling agents are active. Basically, they try to sell themselves, contacting interested parties (namely, buying agents) and negotiating with them to find the best deal.

Users have some control over the agent's negotiation strategy. They can specify the decay function the agent uses to lower the asking price over its given time frame. The three decay functions offered by Kasbah are linear, quadratic, and cubic with respect to time.

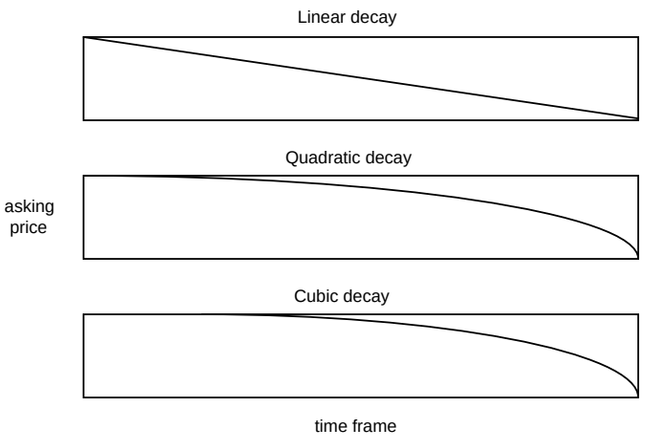


Fig. 1. The three decay functions

To summarize, a Kasbah agent, along with its being a seller or a buyer, is denoted by the following characteristics:

- the good to sell/buy;
- latest desired date to sell/buy the good;
- desired (starting) price;
- lowest/highest acceptable price;
- negotiation strategy (linear/quadratic/cubic with respect to time);

– minimum acceptable reputation of the buyer/seller.

In this work, we will consider only a buyer, without any loss of generality: in fact, a seller is a kind of mirror image of a buyer.

At the time of writing, an improved version of Kasbah is available online, renamed *MarketMaker*, (see `http://maker.media.mit.edu`), which actually introduces the last item of the above dotted list, i.e. the "reputation" management.

3 A Stripped-Down BDI Model

BDI logics is based on a philosophical theory of intentional action originally stated by Bratman (see [1,11,12,10]). It models an autonomous agent, structuring its mental state using three main components: *Belief*, *Desire* (or *Goal*), and *Intention*.

For our specification, we used a modified version of the BDI logic presented in [11,12]. In fact the specified Kasbah seller turned out to be quite simple; therefore we discarded the D and I components.

Moreover, we never used the branching time Computation Tree Logic (or CTL* - see [3]), as Rao and Georgeff did, and we decided to use a somewhat simpler linear time logic to cover the main timing aspects (see, e.g., the thorough description of Temporal Logic of Concurrency in [4]).

Informally, our semantics works as follows. A *world* is modeled using a linear temporal structure, with single past and single future. A particular time point in a particular world is called a *situation*.

Events transform one time point into another. Of course, the agent may attempt to execute some event, but fail to do so. Thus we distinguish between the successful execution of events and their failure.

As for the timing aspects, as we stated before, we use a version of the Temporal Logic of Concurrency. The standard temporal operator of this logic are: $\bigcirc$, meaning "in the next time instant"; $\Diamond$, meaning "eventually"; $\Box$, meaning "always in the future"; $\mathcal{U}$, meaning "until".

Belief is modeled in a quite conventional way. That is, to each situation, we associate a set of *belief-accessible* worlds: these are, intuitively, these are the worlds that the agent *believes* to be possible (relation $\mathcal{B}$).

The *Intention-accessible* worlds and the *Goal-accessible* worlds of the full BDI model are not considered in this paper.

Let us now enter into some formal details. The following definitions are from [11], slightly modified and tailored to our case.

Definition 1. *An* interpretation *M is a tuple $M = \langle W, E, T, \prec, U, \mathcal{B}, \Phi \rangle$. W is a set of worlds, E is a set of primitive event types, T is a set of time points, $\prec$ is a total order relation on T (linear time), U is the universe of discourse, and Φ is a mapping of first-order entities to elements in U for any given world and time point. A* situation *is a world, e.g. w, at a particular time point, e.g. t, and is denoted by w_t. The relation $\mathcal{B}$ map the agent's current situation to its belief-accessible worlds: $\mathcal{B} \subseteq W \times T \times W$. We will use $\mathcal{B}_t^w$ to denote the set of worlds $\mathcal{B}$-accessible from world w at time t.*

Definition 2. *Each world w of W is a tuple $\langle T_w, \mathcal{S}_w, \mathcal{F}_w \rangle$, where $T_w \subseteq T$. A* fullpath *is an infinite sequence of time points $(t_0, t_1, \ldots)$ such that $t_i \prec t_{i+1}$. We will use the notation $(w_{t_0}, \ldots)$ to make the world of a particular fullpath explicit. $\mathcal{S}_w : T_w \times T_w \to E$ and similarly of $\mathcal{F}_w$, moreover both are partially functional and have disjoint domains.*

Consider an interpretation M, with a variable assignment v. We take v_d^i to be that function that yields d for the variable i and is the same as v everywhere else. The semantics of first-order formulas can be given as follows.

- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models q(y_1, \ldots, y_n)$ iff $\langle v(y_1), \ldots, v(y_n) \rangle \in \Phi(q, w, t_0)$ where $q(y_1, \ldots, y_n)$ is a predicate formula.
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \neg\phi$ iff $M, v, (w_{t_0}, w_{t_1}, \ldots) \not\models \phi$
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \phi_1 \vee \phi_2$ iff $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \phi_1$ or $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \phi_2$
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \exists i \phi$ iff $M, v_d^i, (w_{t_0}, w_{t_1}, \ldots) \models \phi$, for some d in U
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \bigcirc\phi$ iff $M, v, (w_{t_1}, \ldots) \models \phi$
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \phi_1 \mathcal{U} \phi_2$ iff $\exists k, k \geq 0$, such that $M, v, (w_{t_k}, \ldots) \models \phi_2$ and $\forall j, 0 \leq j < k, M, v, (w_{t_j}, \ldots) \models \phi_1$
- $M, v, (w_{t_0}, w_{t_1}, \ldots) \models \mathrm{BEL}(\phi)$ iff $\forall w' \in \mathcal{B}_{t_0}^w, M, v, w'_{t_0} \models \phi$
- $M, v, (w_{t_1}, \ldots) \models succeeded(e)$ iff $\exists t_0$ such that $\mathcal{S}_w(t_0, t_1) = e$
- $M, v, (w_{t_1}, \ldots) \models failed(e)$ iff $\exists t_0$ such that $\mathcal{F}_w(t_0, t_1) = e$

Other useful operators are derived as usual:

- $\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2)$;
- $\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$;
- $\phi_1 \leftrightarrow \phi_2 = (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$;
- $true = (\phi \vee \neg\phi)$;
- $false = \neg true$;
- $\Diamond\phi = true\ \mathcal{U}\ \phi$;
- $\Box\phi = \neg\Diamond\neg\phi$;
- $\exists! i\ p(i) = \exists i(p(i) \wedge \forall j(j \neq i \rightarrow \neg p(j)))$.

Here are some of the main definitions about events:

- $succeeds(e) \leftrightarrow \bigcirc succeeded(e)$;
- $fails(e) \leftrightarrow \bigcirc failed(e)$;
- $done(e) \leftrightarrow succeeded \vee failed(e)$;
- $does(e) \leftrightarrow \bigcirc done(e)$.

4 A Kasbah Selling Agent

In this section, we introduce the formal description of our Kasbah-like selling agent. This informal description comes quite directly from [2] and defines the behavior of a Kasbah selling agent.

An agent consists of the following components: control parameters, negotiation history, and internal state. The control parameters are the six user-specified parameters described earlier in the paper. The negotiation history records each conversation that the agent has had with other agents. An example conversation is "I offered agent B a price 100. B rejected the offer" or "Agent C asked my selling price. I replied 91".

Time is partitioned by Kasbah in discrete slices: only one agent is active in a given time slice. The internal state of an agent contains information that the agent uses to decide what will do during its time slice. The internal state stores a list of "potential contacts", which are those agents interested in buying what the agent is selling. With each potential contact, the last known offering price (i.e. what the agent is willing to buy for), and whether it has been asked this round are recorded. The internal state also stores the agent's own current asking price. The strategy an agent uses to decide what to do in each time slice is described below.

- *Current asking price*: the agent lowers its asking price according to the specified price decay function. When the agent is created, its asking price is set to the desired price. By the time to sell by, the asking price is the lowest price. At any moment in between, the current asking price can be interpolated according to the decay function.
- *Decide which agent to talk to*: the agent's strategy is to talk to each potential contact exactly once per round. In other words, an agent will never talk again to a given potential contact until it has first talked to all other potential contacts. The algorithm for deciding which potential contact to talk to during a slice works as follows: consider the potential contacts that have not yet been spoken to in the current round. If all have been spoken to, then begin a new round and consider all the potential contacts. From this set of agents (*suitable*), pick one that has never been contacted, or, if all agents under consideration have been contacted, then pick the one whose last known offering price is the highest. The idea is to first talk to those agents which seem the most promising, first those who have never been spoken to, and then the agents who have indicated a willingness to pay a higher price.
- *Talk to the potential contact* (candidate): the agent offers to sell the item at its current asking price. If the contacted agent accepts, then the agent's job is done. If the contacted agent rejects the offer, then it is asked what its offering price is. This price is recorded for that potential contact.

4.1 The Price Function

The axioms in this section describe the temporal behavior of the offering price function: *price* is a time-dependent function, local to the agent. The specifier needs to include one of the following propositions, depending on the chosen behavior of the price functions: *linearSeller, quadraticSeller, cubicSeller*. These are the only curves available in Kasbah, but other behaviors can be easily introduced by adding a price definition with the chosen function shape.

$dp = max_p - min_p$ and $dt = max_t - min_t$ are agent-dependent constants: i.e. the maximum price variation, and the maximum expected life of the agent, respectively.

$$linearSeller \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = max_p - \frac{dp}{dt} k) \right)$$

$$quadraticSeller \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = max_p - \frac{dp}{dt^2} k^2) \right)$$

$$cubicSeller \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = max_p - \frac{dp}{dt^3} k^3) \right)$$

The following are the corresponding price functions for the "mirror image": a buying agent.

$$linearBuyer \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = min_p + \frac{dp}{dt} k) \right)$$

$$quadraticBuyer \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = min_p + \frac{dp}{dt^2} k^2) \right)$$

$$cubicBuyer \leftrightarrow \bigwedge_{0 \le k \le dt} \left(start \leftrightarrow \bigcirc^k (price = min_p + \frac{dp}{dt^3} k^3) \right)$$

These axioms permit a very easy way of coding the price function in the specification. Ideally, using the same structure it is possible to describe whatever price function in a pointwise fashion, thus extending the fixed Kasbah behaviors.

4.2 Actions and Predicates

Let Ag and Pr be two distinct subsets of U, let $a \in Ag$ be an agent, and $p \in Pr$ a price. The following are the main possible actions:

- wants(a,p): a offers the price p for the good;
- info(a,p): a asks the agent its current price, the agent replies p;
- offer(a,p): the agent offers a to buy at price p;
- ask(a,p): the agent asks to a its price, a answers p.

Note that a successful transaction is modeled using *succeeded*. For example, $succeeded\, offer(a, p)$ means that during the previous time instant a offered a price p, and the agent accepted.

The following are needed predicates about the other agents.

- suitable(a): a is a suitable agent (i.e. it is looking for what we are trying to sell and it has an acceptable reputation);

– dead(a): the agent a is dead (in fact, it ended its activity).

These predicates are "local" to the agent. They are needed to record every negotiation phase and to denote the staring (and ending) of activity.

– candidate(a): a is chosen for negotiation;
– db(a,p): database with agent a and associated price p;
– nc(a): agent a has never been contacted;
– start: the agent is starting its activity;
– end: the agent finished its dt time slot, or successfully sold its good: end of activity.

4.3 Main Axioms

The following axioms define the start and the end of the agent's trade activity (ld1,2,4,5); while (ld3) states about other agents' ending of activity (in fact $dead(a)$ means only that a is no more active for trading). Axiom (ld6) tells about actions and time slice: the agent can execute only one action per round.

(ld1) $start \rightarrow \bigcirc\Box\neg start$
(ld2) $start \rightarrow \bigcirc^{dt}\Box end \wedge \neg end \mathcal{U}\, end$
(ld3) $dead(a) \rightarrow \Box dead(a)$
(ld4) $end \rightarrow \Box end$
(ld5) $end \rightarrow \neg\exists a \exists p(does\, offer(a,p) \vee does\, ask(a,p))$
(ld6) $does\ s \rightarrow \neg\exists t(t \neq s \wedge does\ t)$

The next groups of axioms deal with the mental state of the agent.

A first, necessary axiom is based on the previously introduced price function axioms, e.g. if we want to define a linear seller, then we must introduce the axiom BEL($linearSeller$).

Negotiation Management. These axioms deal about asking and offering prices for the good. The agent must contact every suitable - and not contacted-before - candidate; while it must offer to candidate and contacted-before agents a price depending to their negotiation history.

(nm1) BEL$(candidate(a) \wedge nc(a) \wedge \neg failed\, offer(a,p)) \rightarrow$
$does\, offer(a, price)$
(nm2) BEL$(candidate(a) \wedge nc(a) \wedge failed\, offer(a,p)) \rightarrow$
$does\, ask(a,q)$
(nm3) BEL$(candidate(a) \wedge db(a,p) \wedge p \leq price) \rightarrow\ does\, offer(a, price)$
(nm4) BEL$(candidate(a) \wedge db(a,p) \wedge p > price) \rightarrow\ does\, offer(a,p)$
(nm5) BEL$(done\, wants(a,p) \wedge p \geq price) \rightarrow succeeded\, wants(a,p)$
(nm6) BEL$(\exists a \exists p(succeeded\, offer(a,p) \vee succeeded\, wants(a,p))) \rightarrow end$

Negotiation recording. These axioms define how to insert new entries in the negotiation management database. The agent must remember every highest offer coming from buying agents; moreover it must hold every useful db entry (i.e. concerning only "alive" agents).

(nr1) $(\mathrm{BEL}(done\, wants(a,p)) \vee \mathrm{BEL}(done\, ask(a,p))) \rightarrow \mathrm{BEL}(db(a,p))$
(nr2) $\mathrm{BEL}(db(a,p) \wedge \neg\exists q(done\, wants(a,q)) \wedge \neg\exists q(done\, ask(a,q))) \rightarrow$
$\mathrm{BEL}(\bigcirc(db(a,p) \vee dead(a)))$

DB Consistency. The following axioms are basically needed to maintain the negotiation database consistency.

(db1) $\mathrm{BEL}(start \wedge suitable(a) \rightarrow nc(a))$
(db2) $\mathrm{BEL}(db(a,p) \rightarrow \neg nc(a))$
(db3) $\mathrm{BEL}(nc(a) \rightarrow \neg\exists p(db(a,p)))$
(db4) $\mathrm{BEL}(dead(a) \rightarrow \neg nc(a) \wedge \neg\exists p(db(a,p)))$
(db5) $\mathrm{BEL}(suitable(a) \rightarrow (nc(a) \vee \exists! p(db(a,p))))$
(db6) $\mathrm{BEL}((nc(a) \vee \exists p(db(a,p))) \rightarrow suitable(a))$
(db7) $\mathrm{BEL}(\neg dead(a) \rightarrow (nc(a) \rightarrow suitable(a)))$
(db8) $\mathrm{BEL}(\neg dead(a) \rightarrow (\exists p(db(a,p)) \rightarrow suitable(a)))$
(db9) $\mathrm{BEL}(suitable(a) \rightarrow \bigcirc(suitable(a) \vee dead(a)))$

Candidate Management. These axioms describes how a candidate is chosen by the agent, and the relation between a candidate and a suitable agent.

(cm1) $\mathrm{BEL}(candidate(a) \wedge db(a,p) \rightarrow$
$\neg\exists b(b \neq a \wedge suitable(b) \wedge db(b,q) \wedge q > p))$
(cm2) $\mathrm{BEL}(candidate(a) \wedge db(a,p) \rightarrow \neg\exists b(b \neq a \wedge nc(b)))$
(cm3) $\mathrm{BEL}(candidate(a) \wedge nc(a) \rightarrow$
$\neg\exists b(b \neq a \wedge nc(b) \wedge failed\, offer(b,p)))$
(cm4) $\mathrm{BEL}(\exists! a(suitable(a) \wedge candidate(a)))$
(cm5) $\mathrm{BEL}(candidate(a) \rightarrow suitable(a))$
(cm6) $\mathrm{BEL}(succeeded\ wants(a,p) \rightarrow \neg\exists b(candidate(b)))$
(cm7) $\mathrm{BEL}(candidate(a) \rightarrow \neg\exists b(b \neq a \wedge does\ offer(b,p)))$

4.4 Cognitive Architecture Axioms

In this section we introduce the cognitive architecture of our agent. Indeed, the architecture turns out to be quite simple: the agent is purely reactive.

Time. The agent must hold its belief in the immediate future.

(tm) $\mathrm{BEL}(\bigcirc t) \rightarrow \bigcirc(\mathrm{BEL}(t))$

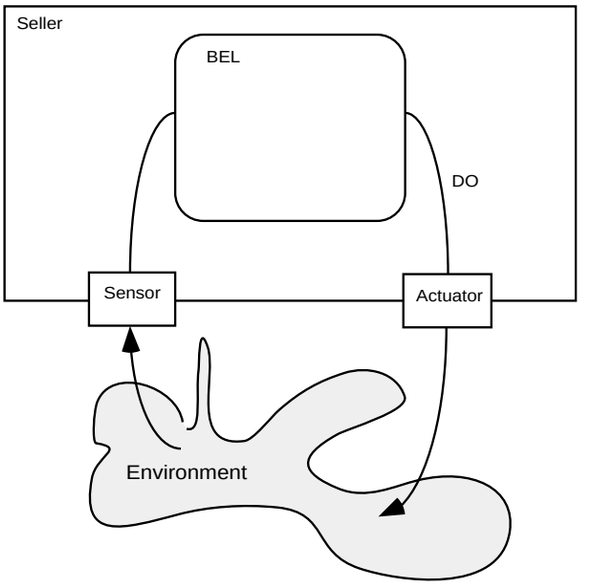


Fig. 2. The agent's cognitive architecture

Sensors. Let e be an observable action. The following axioms show how environment affects the agent's "inner belief".

Clearly we use the term "sensors" speaking of a software: in this case the environment is the Kasbah virtual marketplace.

(se1) $done(e) \rightarrow \text{BEL}(done(e))$
(se2) $suitable(a) \rightarrow \text{BEL}(suitable(a))$
(se3) $dead(a) \rightarrow \text{BEL}(dead(a))$
(se4) $start \rightarrow \text{BEL}(start)$
(se5) $end \rightarrow \text{BEL}(end)$

Actuators. Originally we used the following axiom to describe the "virtual-actuator":

(ac) $\text{INTEND}(does(e)) \rightarrow does(e)$

Now, this uses the INTEND operator, but this modal operator was previously present only in this simple form: $\text{INTEND}(does(e))$.

Actually, it is convenient to replace this subformula with $does(e)$, because the INTEND part is totally absent from this agent's architecture (as well as the GOAL part). Therefore (ac) can be deleted. Clearly, as was stated before, this is a purely reactive agent, and therefore the concept of intention is redundant.

About Axiom Systems. Rao and Georgeff preferred axiom system for the BEL component is KD45 (see [12]). In our example, the BEL component is in fact functioning like a simple database for the agent. We need basic logic deduction within this BEL component (K). We want belief to be consistent (D), but we do not need introspection and there are not nested applications of the BEL operator: we can forget axioms like (4) or (5). Therefore, we use just the following axioms:

(K) $\mathrm{BEL}(\phi) \wedge \mathrm{BEL}(\phi \rightarrow \psi) \rightarrow \mathrm{BEL}(\psi)$
(D) $\mathrm{BEL}(\phi) \rightarrow \neg\mathrm{BEL}(\neg\phi)$

Another relevant aspect is *time*: considering the CTL component, we do not need a branching time logic, because all time operators have the form *inevitable OP*. In our opinion, branching time logic is effective only with more sophisticated agents, able to deal with quite complex situations in a deliberative way. Therefore linear temporal logic suffices, allowing us to drop the *inevitable* operator.

5 Some Useful Trading Properties

As we stated in the introduction, the user may want to be assured about the correctness of his agent. Now, having a formal definition of it, we can assert -and prove- useful properties.

A first interesting property is the following: the agent will never offer a price below the price function. This is stated by the next theorem.

Theorem 1. $BEL(candidate(a)) \wedge does(offer(a,p)) \rightarrow p \geq price$

Proof. We will partition the scenario in mutual exclusive cases, depending on the presence/absence of the entry referred to agent a in the agent's database (see axioms (db$n$)).

Let us suppose that $db(a,p')$, with $p' \leq price$. Then, by (mn3), it implies $does(offer(a,price))$. But, by (ld6), we cannot ask/receive other offers. Therefore $p = price$: the statement holds.

Now suppose $db(a,p')$, with $p' > price$.

Then, by (mn4), this implies $does(offer(a,p'))$. Like in the previous case, by (ld6), we cannot ask or receive other offers. Therefore $p = p' > price$: the statement holds.

Consider now the case $nc(a)$. We can suppose $done(offer(a,p'))$ (case a) and specifically succeeded $offer(a,p')$ (case a1). This implies, by (nm6), *end*, therefore (by axiom (ld5)): $\neg\exists a \exists p(does(offer(a,p)))$. The premise of the statement is false: the statement holds.

Otherwise, we can suppose $failed(offer(a,p'))$ (case a2). This implies (nm2) $does(ask(a,p'))$, therefore we have (ld7) $\neg\exists a \exists p(does(offer(a,p)))$. Like in the previous case, the premise of the statement is false: the statement holds.

Now consider (case b): $\neg done(offer(a,p'))$. This implies (definition of *done*): $\neg failed(offer(a,p'))$.

Therefore, by (nm1), we obtain $does(offer(a,price))$, but, by (ld6), we cannot ask/receive other offers. Therefore $p = price$: the statement holds.

The following other simple statement assures that the agent, after receiving an offer higher or equal to the current *price*, will buy immediately. Moreover, it will not offer/ask anymore (*end of activity*).

Theorem 2. $done(wants(a,p)) \wedge p \geq price \rightarrow$
$succeeded(wants(a,p)) \wedge \neg\exists b(does(offer(b,q)))$

Proof. By (se1), the premise of the statement implies the premise of axiom (nm5). Therefore succeeded $wants(a,p)$ holds. But this, by (nm6), implies the end of activity (*end*).

This means, using (ld5), that $\neg\exists b(does(offer(b,q)))$ holds, which is the other consequence to be proved.

Actually, the previous two properties are fairly trivial but nonetheless useful. They are thought just as an example, to show how this can be an effective framework for asserting trading properties in the delicate field of e-commerce.

As stated in [8], a practical formal method should be supported by analysis techniques that can be invoked with the mere push of a button (e.g. model checking). A really *simple* formal framework, like the one presented in this paper, is naturally suited to "pushbutton" techniques.

6 Conclusions

First generation systems for agent-mediated electronic commerce are already creating new markets and beginning to reduce transaction costs in a variety of business tasks. However, many business aspects are now only partially (if any) covered, and sometimes in an ineffective or unreliable way. A more extensive use of formal methods should change and improve the current situation.

For this point, this experience turned to be quite useful in some respects.

We effectively use some well-known formal tools to *reverse-engineer* a real implemented e-market autonomous software agent. This naturally permits to proof useful properties about the correctness of the agent's behavior.

We simplified the BDI formalism in a somewhat weaker, but easier and more practical-suited logic. Quite naturally, an "easy" logic usually means easier deductions and the possibility of efficient (semi-)automatic analysis tools.

We did a simple and straightforward reasoning about the cognitive architecture of our agent. It seems quite natural that a practical agent for e-commerce must be simple and straight-minded: network bandwidth and host processing power are at present too weak to adequately support the refined and complex-minded agents we often see in some theoretical/philosophical papers. As a matter of fact, sometimes the expressive power of the chosen logic results misdirected, compared to the requirements of present (and probably near-future) automated electronic market.

Acknowledgments. We wish to thank Gianpaolo Cugola and Stefano Gaburri for the fruitful discussions. Last but not least, many thanks even to Mattia Monga, who solved some nontrivial LaTeX formatting problems.

References

1. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
2. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 75–90, 1996.
3. E. Allen Emerson and Jai Srinivasan. Branching time temporal logic. In Jaco W. de Bakker, Willem-Paul de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models of Concurrency*, number vol. 354 in Lecture Notes in Computer Science, pages 123–172. Springer, Heidelberg, Germany, 1989.
4. R. Goldblatt. *Logics of Time and Computation, Second Edition, Revised and Expanded*, volume 7 of *CSLI Lecture Notes*. CSLI, Stanford, 1992 (first edition 1987). Distributed by University of Chicago Press.
5. R. H. Guttman and P. Maes. Cooperative vs. competitive multi-agent negotiations in retail electronic commerce. In Matthias Klusch and Gerhard Weiß, editors, *Proceedings of the 2nd International Workshop on Cooperative Information Agents II: Learning, Mobility and Electronic Commerce for Information Discovery on the Internet*, volume 1435 of *LNAI*, pages 135–147, Berlin, July 4–7 1998. Springer.
6. R. H. Guttman and P. Maes. Agent-mediated integrative negotiation for retail electronic commerce. In Pablo Noriega and Charles Sierra, editors, *Proceedings of the 1st International Workshop on Agent Mediated Electronic Commerce (AMET-98)*, volume 1571 of *LNAI*, pages 70–90, Berlin, May 10–10 1999. Springer.
7. R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. In *Knowledge Engineering Review*, 1998.
8. C. Heitmeyer. On the need for practical formal methods. *Lecture Notes in Computer Science*, 1486, 1998.
9. P. Maes, R. H. Guttman, and A. G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, March 1999.
10. D. Morley. Semantics of BDI agents and their environment. Technical Report 74, Australian AI Institute, Carlton, Australia, 1996.
11. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. Technical Report 14, Australian AI Institute, Carlton, Australia, 1991.
12. A. S. Rao and M. P. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian Artificial Intelligence Institute, Melbourne, Australia, June 1995.

A Platform for Electronic Commerce with Adaptive Agents

Henrique Lopes Cardoso and Eugénio Oliveira

Faculdade de Engenharia,
Universidade do Porto, NIAD&R-LIACC
Rua dos Bragas
4050-123 Porto Codex, Portugal
Phone: +351-22-2041849 Fax: +351-22-2074225
hlc@ipb.pt
eco@fe.up.pt

Abstract. Market research suggests that organisations, in general, have a differentiation strategy when approaching Electronic Commerce. Thus, in order to be useful, agent technology must take into account this market characteristic. When extending its application to the negotiation stage of the shopping experience, one should consider a multi-issue approach, from which both buyers and sellers can profit. We here present SMACE, a layered platform for agent-mediated Electronic Commerce, supporting multilateral and multi-issue negotiations. In this system, the negotiation infrastructure through which the software agents interact is independent from their negotiation strategies. Taking advantage of this concept, the system assists agent construction, allowing users to focus in the development of their negotiation strategies. In particular, and according to experiments here reported, we have implemented a type of agent that is capable of increasing the performance with its own experience, by adapting to the market conditions, using a specific kind of Reinforcement Learning technique.

1 Introduction

Agent technology has been applied to the Electronic Commerce domain, giving birth to what is known as agent-mediated Electronic Commerce. Many of these online implementations refer only to the first stages of the *Consumer Buying Behaviour* model [9], those of discovering what particular product a shopper should buy (*product brokering*) and finding it through the online merchants (*merchant brokering*). These automated search engines help the user on finding the best merchant offer, classifying those offers according to the price that they state. Examples of such systems include BargainFinder [2] and Jango [10]. In some extent, this is the only presently viable implementation of agent technology to online shopping, since many of the merchant sites consist of catalogues, including product and service descriptions and a so called *take-it-or-leave-it* way of doing business, which consists of presenting the demanded price of the good.

Most of the commercial online sites where it is possible to negotiate over the terms of a transaction consist of auctions, mostly based on the traditional *English auction*

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 96-107, 2001.

protocol. In this kind of interaction, it is common to have the shoppers bidding on the price they are willing to pay for a given good, with all the remaining product and transaction characteristics being fixed. This is what makes these auctions *single-sided* mechanisms, in which shoppers compete for limited resources. On the other hand, *double-sided* auctions, like the *continuous double auction*, admit multiple buyers and sellers at the same time, both parts being able to bid. This possibility enables a way to implement a more realistic agent-based approach to today's online shopping experiences. In fact, the Internet has changed radically the "rules of the game". Shoppers have today a virtually world wide marketplace, allowing them to compare merchant's offers in a global scale. Thus, the competition among merchants has increased, and they are obliged to find out new ways of capturing shoppers' attention.

Some examples of agent technology applied to auctions are the academic projects AuctionBot [1, 19], which is an auction server where software agents can be created to participate in several types of auctions, and Fishmarket [8, 14], an electronic auction house for the *Dutch auction*.

According to market research [15], from the three well-known generic business strategies with which organisations may align Electronic Commerce – cost leadership, differentiation, and focus –, differentiation is the preferred one. This fact poses questions about the advantages of using the currently available information seeking agent-based systems applied to the Electronic Commerce domain. In order to be helpful, this technology should take into account that merchants want to differentiate themselves, and that shoppers can and want to benefit from that.

In order to build agent-based tools that respond to these requirements, merchants must provide in their sites multi-issue product and service descriptions that can be treated in an automatic way. This would enable software agents to search a set of merchants for the best product offering, while considering their differentiation intent. This kind of service, besides being an advantage to merchants, can also be seen as a powerful tool to shoppers as well, since they may compare the merchants' offers according to their preferences. The most notable work on defining a common language that makes the web accessible to software agents is the XML specification [3], developed by the World Wide Web Consortium (W3C) [18]. XML is a mark-up language that follows HTML (a presentation format language), making it possible to represent the semantic content of a document.

One important breakthrough in this area is the project Tête-à-Tête [17]. It provides a means for the shopper to compare, given its preferences, the multi-featured offers made by several merchants that sell a particular product. However, this system is not merely a search engine for shopper assistance. It uses software agents, on both sides of the bargaining process, which interact in order to encounter the shopper's desire.

In order to make it possible to automate the negotiation step of the shopping experience through the use of autonomous software agents, it is necessary to define both a common ontology to represent product and transaction related knowledge and a language for agent interaction. A protocol will then define what messages in that language may be exchanged in what conditions. The greatest effort in defining a general language for knowledge and information exchange between agents is KQML [7], developed by the ARPA Knowledge Sharing Effort.

Automated negotiation has been addressed in some relevant research. The Kasbah [5] marketplace allows the creation of predefined agents with simple time-dependent negotiation strategies, but considers only the price of a product when negotiating over it. In [6], negotiation is described as a process where the parties move towards

agreement through concession making. That negotiation is modelled as multilateral and multi-issued. Agent negotiation strategies are defined as combinations of tactics, including time-, resource- and behaviour-dependent ones.

Learning in negotiation is an even more recent topic of research. In [12], learning consists of finding the appropriate tactic parameters and combination in the overall negotiation strategy. Using genetic algorithms pursues this. In [20] the Bazaar system is described, a sequential negotiation model that is capable of learning, which is done by updating the agent's belief model in a bayesian way, that is, with probabilistic updates, according to its knowledge of the domain.

This paper presents a multi-layered platform, SMACE, which provides a means for testing different negotiation strategies. SMACE is a Multi-Agent System for Electronic Commerce that supports and assists the creation of customised software agents to be used in Electronic Commerce negotiations. The system has been used for testing automated negotiation protocols, mainly those based on the *continuous double auction*, supporting also a *multi-issue* approach in the bargaining process. On-line learning in Electronic Commerce negotiations is another subject also addressed within SMACE. The system includes two main types of agents: those that consider a strategy as a weighted combination of several tactics, and those that learn to combine these tactics in a dynamic way, by means of Reinforcement Learning techniques.

The rest of the paper is organised as follows. Section 2 addresses SMACE, a multi-agent platform for Electronic Commerce, describing its architecture and negotiation model employed. Section 3 describes the strategies implemented in the predefined agents available in the system. In section 4 we present experiments conducted in order to test the performance of agents enhanced with learning capabilities. Finally, in section 5, we finalise with some conclusions and we focus on some topics of our future work.

2 SMACE: A Platform for Agent-Mediated Electronic Commerce

In this paper we present SMACE, a multi-agent system for Electronic Commerce, where users can create buyer and seller agents that negotiate autonomously, in order to reach agreements about product transactions.

2.1 Negotiation Model

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

The negotiation model that we adopted is based on the one in [6]. So, we concern about multilateral negotiations over a set of issues. Multilateral refers to the ability of buyer and seller agents to manage multiple simultaneous bilateral negotiations with other seller or buyer 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). Negotiation is realised by the exchange of 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 those issues and is autonomously generated by the agent's strategy.

The proposal evaluation is based on *Multi-Attribute Utility Theory (MAUT)*. In order to do that, an agent i must take into account the preferences defined by its creator for each issue $j \in \{1, \ldots, n\}$ under negotiation:

- a range of acceptable values $[min^i_j, max^i_j]$, which must be satisfied in order to accept a proposal;
- a scoring function V^i_j: $[min^i_j, max^i_j] \rightarrow [0, 1]$, which calculates a normalised score that agent i assigns to a value for issue j inside the range of acceptable values (the higher the score, the better the agent's utility);
- a weight w^i_j, which translates the relative importance of the issue j in the overall negotiation.

Assuming normalised weights ($\sum_j w^i_j = 1$), the agent's utility 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: $V^i(x) = \sum_j w^i_j V^i_j(x_j)$. After generating a proposal, an agent will decide on sending it upon comparing its utility to the one associated with the previously received proposal. The one with highest utility will prevail.

Following [6], the sequence of proposals and counter-proposals in a two-party negotiation is referred to as a *negotiation thread*. That thread will remain active until one of the parties accepts the received proposal or withdraws from the negotiation. Because of the multilateral nature of de negotiation model (many sealed bilateral negotiations per agent), after an acceptation the agent will wait for a deal confirmation from its opponent.

2.2 Architecture

SMACE works as an open environment where buyer and seller agents meet in the marketplace, as shown in figure 1. This entity facilitates agent meeting and matching, besides supporting the negotiation process.

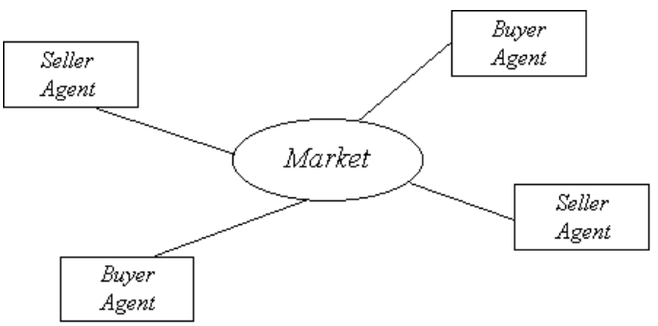


Fig. 1. The marketplace

SMACE allows users, through its web user interface, to create buyer and seller agents that negotiate under the model described in the previous section. The system was designed in layers, in order to separate the infrastructure components – that

provide the communication and negotiation protocols – from those associated with the agents' negotiation strategies. This provides both openness and easy expandability. As a supporting platform, JATLite [11] was used to provide the communication infrastructure needed. Figure 2 gives an overview of how such architecture was implemented.

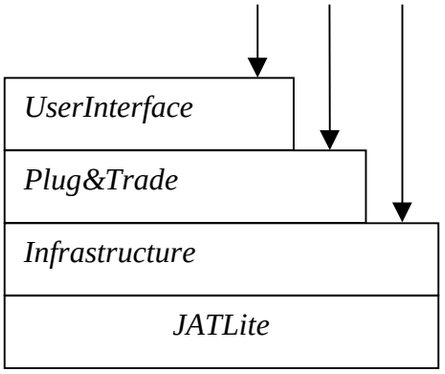


Fig. 2. SMACE architecture

SMACE consists of an API, fully implemented in Java (JDK1.1.4 API), with three layers built on top of the JATLite packages:

- *Infrastructure* – this layer contains two fundamental parts:
  - *MarketPlace*: the application that represents the marketplace, as an environment where the agents meet and trade. It includes message routing and agent brokering facilities.
  - *MarketAgent*: a template for the creation of market agents, which has already implemented the negotiation model. Building an agent with this template requires only assigning it a negotiation strategy.
- *Plug&Trade* – this layer includes ready-to-use predefined market agents, that were built using the *MarketAgent* template:
  - *MultipleTacticAgent (MTA)*: a market agent that considers a negotiation strategy as a combination of several tactics, as described in the next section.
  - *AdaptiveBehaviourAgent (ABA)*: a market agent that is able to weight several tactics in an adaptive way, using Reinforcement Learning techniques, as described in the next section.
- *UserInterface* – this layer consists of an application that provides both an HTML user interface for creating and monitoring the operation of *Plug&Trade* market agents and their persistence.

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.

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 their parameters. In order to do so, one can use the SMACE user interface. Through this interface, the user can monitor the agents' activities and change their settings. Taking another path, the user may create his own agent, with his own strategy, 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.

3 Negotiation Strategies

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 negotiation strategy will define the way to do so. There are no restrictions on the negotiation strategies that can be implemented in the market agents. However, as discussed in the previous section, SMACE assists the activation of two kinds of predefined agents.

3.1 Combinations of Tactics

The predefined agents already implemented in the SMACE system (*MTA* and *ABA*) use combinations of tactics as the underlying philosophy of implementing their strategy. A tactic is a function used to generate a proposal value, for a given issue, based on a given criterion. Tactics can be combined using different weights, representing the relative importance of each criterion in the overall strategy. The values that will be part of the proposal will be calculated by weighting accordingly the values proposed by each one of the tactics used. The tactics implemented were adopted from [6]:

- 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.
- 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 proposal variations.

Other kinds of tactics can be considered or other variants of the tactics mentioned. Whereas time-dependent tactics depend on a predictable factor, it is difficult to foresee the results of applying resource- or behaviour-dependent ones, since they depend on "run-time variations" of factors.

3.2 Adaptive Behaviour through Reinforcement Learning

The *MTA* predefined market agents are somewhat fixed, in the sense that they will use the same tactic combination, no matter what the results obtained are, unless the user specifies otherwise. However, in repeated negotiations, agents should be capable of taking advantage of their own experience. This consideration led us to the development of an agent that, enhanced with learning capabilities, can increase its

performance as it experiences more and more negotiations – the *AdaptiveBehaviourAgent (ABA)*. Tactics provide a way of adaptation, in some degree, to different situations, considering certain criteria. But it is not clear what tactics should be used in what situations. The ultimate goal of our adaptive agent is to learn just that.

The idea is to define a strategy as the way in which the agent changes the tactic combination over time. In order to do that, we used a kind of automated learning that can take place online, from the interaction with the environment: Reinforcement Learning [16]. It is also the most appropriate learning paradigm to dynamic environments, such as the one we are addressing.

By applying this kind of learning in the adaptive agents, we aimed at enhancing those agents with the ability of winning deals in the presence of competitors and increasing the utility of those deals. We intended to check if the agents adapt to a given market environment, associated with the transaction of a given type of product.

In dynamic environments, such as an electronic market, actions are *non-deterministic*, in the sense that they do not always lead to the same results, when executed in the same state. For this reason, we implemented a specific kind of Reinforcement Learning – *Q*-learning – that estimates the value of executing each action in each state (also known as the quality *Q*). In our environment, actions are weighted combinations of tactics that will be used in the proposal generation process. The characterisation of the states is a major factor to the success of the algorithm implementation, and will determine the relevance of the results obtained. In our case, we considered two main variables: the *number of negotiating agents* and the *time available for negotiation*, that is, the time left till the agent's deadline.

Updating the *Q* values associated with each action-state pair – *Q(s,a)* – consists of rewarding those actions that leaded to good results while penalising the ones that failed to achieve the agent's goal. The general *Q*-learning update formula is the following:

$$Q(s,a) = Q(s,a) + \alpha [r + \gamma max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

where α is the learning rate, representing the impact of the update in the current value; *r* is the reward obtained by executing action *a* in state *s*; γ is the discount factor, meaning the importance of future *Q* values (in future states) to the *Q* currently being updated; $max_{a'} Q(s',a')$ is the maximum *Q* value for the actions in the next state.

For the *ABA* agents, actions leading to deals are rewarded with a function depending on the deal values' utility and on the average utility obtained so far. This allows us to distinguish, from the deals obtained, those that correspond to greater utilities. Considering the average utility takes into account, when classifying the goodness of a deal, the characteristics of the environment (the difficulties) that the agent is repeatedly facing; the same deal in harder conditions should have a greater reward because it is closer to the best possible deal. Goal failure imposes penalisation (negative reward) to the last action used.

Action selection is another important aspect of the Reinforcement Learning paradigm. The simplest rule would be to select the action with the biggest *Q* value. Yet, this rule does not consider that there may exist non-executed actions that may perform better. Furthermore, in dynamic environments and therefore *non-deterministic*, actions do not always lead to the same results. In fact, to obtain a continued reward of great value, the agent should prefer actions that were considered

good in the past, but in order to find them it must try actions that were never selected before. This dilemma leads us to the need of a compromise between *exploitation* (to take advantage of good quality actions) and *exploration* (to explore unknown actions or those with less quality). To satisfy this compromise, according to the Reinforcement Learning literature, two policies are possible, and the adaptive agents developed support them both: the ε*-greedy* approach selects uniformly, with a probability ε, a non-greedy action; the *Softmax* policy uses a degree of exploration τ (the temperature) for choosing between all possible actions, while considering their ranking.

In order to make it possible for the agent to increase the utility obtained in the deals made, it is necessary that the agent does not prefer the first action leading to a deal. Before the agent tries enough actions, it has got an incomplete knowledge of the environment, that is, it might know what action to use to likely get a deal (because unsuccessful actions are penalised), but not what the best actions are (those that result in higher utilities). To enforce the agent to try all the actions available before preferring the best ones, we implemented a Reinforcement Learning technique called *optimistic initial values*. This means that all the *Q* values associated with the actions are initialised to a value greater than the expected reward for them. This measure increases, independently of the action selection parameters chosen, the initial action exploration, since the *Q* values will then be updated to more realistic lower values.

4 Experiments

The implementation of the learning algorithm intended to enhance the agents with the capacity of gaining "know how" about the market mechanics in respect to the transaction of certain kinds of products. This sensibility refers not only to the usual pattern of appearance of new agents (the market dynamics), but also to the way buyer and seller agents in a specific environment normally relax their bids. The agent should improve its performance as it experiences more negotiation episodes.

To represent these market-specific features, which work as a reference to the adaptation process, we designed four different scenarios, over which we conducted some experiences. These scenarios did not intend to simulate real-world Electronic Commerce negotiations, but to illustrate situations where it was possible to observe the results of applying the learning skills on the adaptive agents.

4.1 Scenarios' Description

The four basic scenarios are illustrated in figure 3. The negotiation was made over the single-issue price, since this option does not affect the results; the strategies implemented do not take advantage of the multi-issue support (each tactic generates a value for an issue). All agents were configured with time-dependent tactics. The *MTA* agents had single time-dependent linear tactics. The *ABA* agents had time-dependent tactics that allowed them to vary their behaviour from an anxious (early concession) to a greedy (late concession) extremes. In the third scenario, a resource-dependent tactic, depending on the number of opponents, was added to the adaptive agent, since one of the opponents was activated only after a period of negotiation time. The

adaptive agent in the fourth scenario had also a resource- and a behaviour-dependent tactics. Both kinds of agents have a set of other specific parameters whose values are not described here.

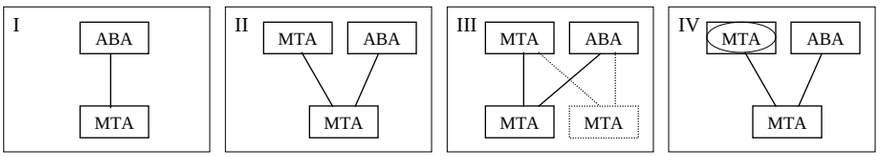


Fig. 3. Experimental scenarios

In scenario I, we intended to check if the adaptive agent was able to increase its utility, after a number of negotiation episodes in the same exact environment configuration. This would allow us to test the learning algorithm. Scenario II expanded this test to the agent's ability to win deals over its competitor, and from those deals, to increase the utility to the best possible value, which was limited by the opponent's fixed strategy. Scenario III was configured in a way that it was preferable to the adaptive agent to achieve deals with the late arriving agent. So, the *ABA* should learn to wait, instead of hurrying on making a deal before its competitor. Finally, scenario IV provided a way of testing the re-adaptation time of the adaptive agent, since its competitor was modified twice after a significant number of negotiation episodes.

The most important learning parameters of the *ABA* agents, which have an impact on their performance, are the *learning rate* (that influences the *Q* value updating), and the *degree of exploration* (which controls the action selection when using the *Softmax* approach). In highly dynamic environments, agents should use high learning rates, allowing for fast adaptation to new environment conditions. On the other hand, a high degree of exploration will force the agent to select many different actions, which may also be important in dynamic environments. However, as a consequence of that, the adaptation process slows down. The *ABA* agents were configured with a learning rate of 0.5, a middle value that allows for quick adaptations (notice that scenarios I and II are fixed, and so this parameter becomes more important in scenarios III and IV). The degree of exploration was set to a low value of 0.2, since that the initial exploration was already assured by the use of optimistic initial values (see subsection 3.2). However, exploration is still needed after the initial adaptation period (namely in scenarios III and IV). The next subsection presents the results obtained by using such values for these parameters.

4.2 Results

In general, the results obtained were satisfactory. That is, the adaptive agent performed well but, for its current implementation, in some cases it took too long to achieve a good result. The scenarios described were run over 2000 negotiation episodes.

In all scenarios illustrated, the adaptive agent tended to achieve the predicted results. Figure 4 shows the utility evolution of the adaptive agent in each one of the scenarios. In scenario I, the agent was able to continuously increase the utility obtained in the deals, by waiting for its opponent to concede. Scenario II was more

limited in terms of utility increasing, but the *ABA* could, besides winning the majority of deals over its competitor, increase the average utility of those deals very close to the highest possible in that situation. Results in scenario III allowed us to conclude that the adaptive agent learned to wait and prefer dealing with the late arriving opponent, which enabled it to achieve higher utilities. In scenario IV, we observed that despite the considerable adaptation of the adaptive agent to an initial situation, after changing the agent's competitor it readapted relatively quickly to the new environment conditions.

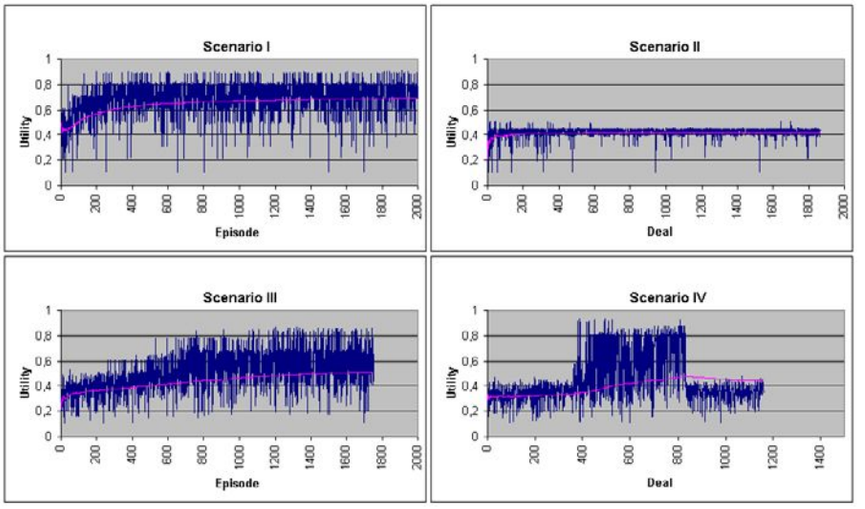


Fig. 4. Utility results

These results show us that, under some circumstances, it is possible to endow software agents with capabilities that allow them to improve their performance with their own experience. The task that now raises is to adapt this mechanism to situations closer to real Electronic Commerce transactions, where the real negotiating parties (the agents' creators) can benefit from negotiation "know how" stored in their software agents.

5 Conclusions and Future Work

Software agents can help users to automate many tasks. In our case, we focus on automating Electronic Commerce activities, namely those of buying and selling products. There exist several applications of information seeking agents applied to this domain that help users on finding the best price for a given product. As explained above, in order to be helpful, such tools should take into account the multi-issue trend of doing online business today.

The automation of the negotiation process is more critical, since it implies the usage of negotiation strategies that will determine the wins and loses of delegating shopping tasks in autonomous software agents. According to [4], the intelligence or

sophistication level that buying or selling software agents may possess is not restricted by Artificial Intelligence limitations, but by user trust considerations.

We have developed SMACE, a platform that includes an infrastructure framework over which it is possible to build agents with different negotiation strategies. The mass development of agent-mediated negotiations in Electronic Commerce will depend on the adoption of standards in this domain, related both to the ontologies used to represent semantically the objects of negotiation and to the software agent's interaction.

As negotiation strategy examples, we implemented two kinds of agents, with the assistance provided by the lower layer of the SMACE system, and made some experiments that involved interactions between these agents. Our results claim that it is possible to build negotiation strategies that can outperform others in some environments.

Directions of our future work include implementing strategies that take effective advantage of multi-issue negotiations, by correlating those issues. In respect to the agents' adaptation capabilities, we intend to refine our learning algorithm implementation. In particular, considering a new definition of a strategy (different from a combination of tactics) might help enhancing the results obtained through the learning experience. We also intend to compare our learning mechanism with other learning approaches, and to continue our research on the practical applications and effective benefits of learning processes.

In which concerns increasing the open nature of our system, we consider adopting some of the emerging standards in the Electronic Commerce domain, namely the XML specification for agent's communication content. The negotiation model that we are using can also be optimised to include support to interactions that may be beneficial, following what is described in [13] as a *critique*.

References

1 AuctionBot. URL: http://auction.eecs.umich.edu/
2 BargainFinder. URL: http://bf.cstar.ac.com/bf
3 Bosak, J. (1997), "XML, Java, and the future of the Web", Sun Microsystems.
4 Chavez, A., D. Dreilinger, R. Guttman and P. Maes (1997), "A Real-Life Experiment in Creating an Agent Marketplace", in *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*.
5 Chavez, A, and P. Maes (1996), "Kasbah: An Agent MarketPlace for Buying and Selling Goods", in *Proceedings of The First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pp. 75-90.
6 Faratin, P., C. Sierra and N.R. Jennings (1998), "Negotiation Decision Functions for Autonomous Agents", *International Journal of Robotics and Autonomous Systems*, 24 (3-4), pp. 159-182.
7 Finin, T., Y. Labrou and J. Mayfield (1997), "KQML as an agent communication language", in *Software Agents*, J. M. Bradshaw (editor), MIT Press.
8 Fishmarket. URL: http://www.iiia.csic.es/Projects/fishmarket/
9 Guttman, R.H., A.G. Moukas and P. Maes (1998), "Agent-mediated Electronic Commerce: A Survey", *Knowledge Engineering Review*.
10 Jango. URL: http://www.jango.com/
11 JATLite. URL: http://java.stanford.edu

12 Matos, N., C. Sierra and N.R. Jennings (1998), "Determining Successful Negotiation Strategies: An Evolutionary Approach", in *Proceedings, Third International Conference on Multi-Agent Systems (ICMAS-98)*, pp. 182-189, IEEE Computer Society.
13 Parsons S., C. Sierra and N.R. Jennings (1998), "Agents that reason and negotiate by arguing", in *Journal of Logic and Computation*, 8 (3), pp. 261-292.
14 Rodríguez-Aguilar, J.A., P. Noriega, C. Sierra and J. Padget (1997), "FM96.5 A Java-based Electronic Auction House", in *Proceedings of the Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*.
15 Romm, C.T. and F. Sudweeks (1998), *Doing Business Electronically*, London: Springer-Verlag.
16 Sutton, R.S. and A.G. Barto (1998), *Reinforcement Learning: An Introduction*, Cambridge: MIT Press.
17 Tête-a-Tête. URL: http://ecommerce.media.mit.edu/Tete-a-Tete/
18 World Wide Web Consortium. URL: http://www.w3.org
19 Wurman, P.R., M.P. Wellman and W.E. Walsh (1998), "The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents", in *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, K.P. Sycara and M. Wooldridge (editors), pp. 301-308, ACM Press.
20 Zeng, D. and K. Sycara (1996), "How Can an Agent Learn to Negotiate?", in *Intelligent Agents III*, J. P. Muller et al. (editors), pp. 233-244, Springer-Verlag.

Trading without Explicit Ontologies

Michael Schroeder, Julie McCann, and Daniel Haynes

Department of Computing,
City University,
London EC1V 0HB, UK
{msch,jam}@soi.city.ac.uk

Abstract. Classical approaches to traders in middleware rely on a common language of server, clients, and traders to understand each other. In these systems, pre-defined ontologies play a crucial role. But when dealing with large-scale, open systems such ontologies are no longer available. To cope with this problem, we have developed a radically different approach to trading. Rather than relying 100% on a trader, we assume that traders provide only rough matches and that clients need to make intelligent choices to find a more suited service. To this end, we introduce the notion of trust, which evolves with the client's experience. We implement a simulation of this trust-based trading system and run several test scenarios investigating the use of history and dynamic trust to discover the more suited services. Our analysis and simulation [1] indicate that intelligent clients and rough traders may considerably extend the scope of trading towards large-scale, open distributed systems.

1 Introduction

A trader is a middleware component currently used to match requests for services to services. They are typically used in relatively closed environments where a single defined language and set of semantics are used to describe requests and services. Many researchers are beginning to see a broader use of trading within a more open distributed system, such as the Web. In this environment it is expected that trading systems need to be able to carry out more fuzzy mappings between requests and services as the semantics and languages are no longer standard. Further, systems which request services (clients) need to be able to acquire a level of trust in both the traders and the services when deciding to use them in future. The client must be able to break away from a service that it notices is deteriorating, so as to obtain a better one. Further, it is also envisaged that future services, and in fact the trading service itself, will charge for usage.

Current trading systems are unable to provide this level of service; in this paper we provide an alternative which does. This system stores a history of quality of services to establish its trust, carry out fuzzy matching of services while providing the ability to dynamically rebind from one service to another. The perception of trust must also take into account that a service will cost and this cost may fluctuate also. To this end, we propose a new trading system and have built a simulation of its behaviour so as to study how metrics such as trust and costs affect client/trader selection. The structure of this

[1] The simulation is available at www.soi.city.ac.uk/homes/msch

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 108–120, 2001.

paper is as follows. Firstly, we introduce the concept of middleware trading systems. We then describe some systems currently in operation and follow this with a description of more intelligent traders that are able to carry out fuzzy mapping of service to requests. We highlight the shortcomings of current trader architectures and then describe our model of trading and the simulation model which is being used to look at their behaviour. Finally, we draw some conclusions.

2 Traders

The concept of the trader was first formally specified in the early 1990's by ANSA [Apm98] which lead to the Open Distributed Processing (ODP) trading function specification (under the ISO and ITU) and in 1997 the Object Management Group (OMG) added the Trading service to its CORBA architecture [Omg92].

The principle idea of a trading service is to have a mediator that acts as a broker between clients and servers. Previously a client needed to locate a service by naming this service. Naming services, such as the Internet Domain Name service, the X.500 directory service or the CORBA Naming service map hierarchically organised names to component identifications such as IP addresses or CORBA object references. This is called white- page-matching, similar to a telephone directory. However, often a client does not know the name of the component it wants to use. Although white-page-matching improves location transparency, it limits clients, which may not know the name of the service they want to use, but which do know a service profile only. To this end, traders enable a client to ask for service profiles rather than service names. The client changes perspective from asking 'who?' to asking 'what?'.

Thus, trading services provide a higher level of abstraction than naming services. Servers register the services they offer with a trader and clients use the trader to enquire about services. Once the trader has matched a client inquiry with a service offer, most clients and servers communicate privately without involvement of the trader (there are exceptions; some monitor QoS and allow run- time rebinding). Hence traders usually do not impose any overhead on the communication between clients and servers, they just enable clients to find the most suitable server and thus facilitate service discovery.

The trader may select a suitable service provider on behalf of the client or it may simply recommend a list of suitable services to the client. The idea is not restricted to distributed systems, but is found as well in other (non-computerised) systems. A good example is yellow pages; service providers, such as plumbers or lawyers, register with the publisher of the yellow pages: Clients who wish to use a particular service can then look up the yellow pages by service providers to find a provider that offers the required service. Other examples similar to the yellow pages are stockbrokers and insurance brokers. This is why this kind of matching is termed Yellow Pages matching of services to requests.

In current systems, there has to be a language for expressing the types of services that both the client and server understands. This language has to be expressive enough to define the different types of services that a server offers or that a client may wish to use. Moreover, the language has to be expressive enough so that a client can ask for given degree of service quality, such as performance, reliability or privacy. Servers use the

expressiveness of the language in order to advertise the quality of their services. In order to enable a trading service to act as a broker between clients and servers, the servers have to register the services they offer with the trader. They use the above language to declare the types of services they offer and their qualities during the registration.

The quality of service may be defined statically or dynamically. A static definition is appropriate (because it is simpler) if the quality of service is independent of the state of the server. This might be the case for qualities such as precision, privacy or reliability. For qualities such as performance, however, the server may not be able to ensure a particular quality of service statically at the time it registers the service with the trader. Then a dynamic definition of the quality is used that makes the trading service inquire about the quality when a client needs to know it.

After servers have registered the types of services they offer with the trader, the trader is in a position to respond to service inquiries from clients. A client uses the common language to ask the trader for a server that provides the type of service the client is interested in. Clients may or may not include specifications of the quality of service that they expect the server to provide. The trader reacts to such an inquiry of clients in different ways. The trader may itself attempt to match the clients request with the best offer and just return the identification of a single server that provides the service with the intended quality. This technique is known as service matching. The trader can also compile a list of those servers that offer a service, which matches the clients request. The trader returns the list to the client in order to enable the client to select the most appropriate server itself. This technique is known as service shopping.

3 Object Traders and Service Discovery

There are a number of Trading systems available at the moment, we describe these briefly below.

3.1 Corba Trader

The Corba trading service works in principle as the trader described above. As a common communication language it uses Corba's interface definition language IDL. Additionally, the client can specify properties, given as attribute/value pairs, which the service has to provide. Technically, a trader recommends services matched to a client's request. The client can then dynamically look up the service's interface in Corba's interface repository. With this information, the client is then able dynamically construct a request to a service.

Corba is a very popular architecture for distribution object interaction and most applications make heavy use of its name service, which is implemented by most commercial and academic Corba platforms. Corba's trading server is less widely used, since many applications are not so dynamic that they would require trading. If they do, then the Corba trader works well for in-house applications, but not for truly open systems, where service discovery requires fuzzy matching of service requests. To put it succinctly, with Corba naming the client needs to know the server name, with trading it needs to know the service name, which is still too inflexible.

3.2 ANSAware Distributed Systems Platform

The ANSAware software model [Apm98], developed at APM Ltd., is based on a location-independent object model providing uniform interaction schemes between communication. The roles of client, trader, and server, the use of attribute/value pairs to specify properties are similar to the Corba trader. Interfaces in ANSAware are defined in an Interface Definition Language and the operations import, export, and a second language - a Distributed Processing Language, describes interface implementations. If the trader finds matching candidates, an implicit binding between the peers is created. Although attribute- based matching provided by the trader enables enhanced flexibility, there is no notion of runtime adaptation and hence the same critique as to the Corba trader applies.

3.3 Aster

The Aster project [Iss98], developed at Irisa/Inria, addresses middleware reconfigurations based on software specification matching, which selects the components of the middleware customized to the application needs. The Aster environment provides three elements: the Aster type checker, which implements type checking of components described using the Aster language, the Aster selector, which retrieves middleware components that satisfy the interaction requirements and thus acts as a trader, and the Aster generator, which is responsible for interfacing the source code files with the middleware objects. The selector in the three-stage selection method processes non- functional properties described in terms of formulas of first order predicate calculus. The stages are; exact match selection, plug-in match selection (the selected component implements behaviour that satisfy the application, but does not match exactly the application's requirements), and closest match selection (the selected middleware needs to be customized through complementary components).

Although Aster presents a powerful framework for parameterized component selection enabling automated customization, it does not address dynamic runtime adaptation.

3.4 Matchmaking

The Matchmaking framework [Ram98], developed at the University of Wisconsin-Madison, is based on components describing their requirements and provisions in classified advertisements, which are matched by a designated service: the matchmaker. Classified advertisements enable components to be described in terms of parameters enhanced with arithmetical and logical operators (e.g. Type = Machine, Activity = Idle, Arch = INTEL, Rank = 10, etc.) The matchmaker compares relevant parameters of component advertisements, and notifies components; then the client contacts the server using a claiming protocol to establish a dynamic binding.

Powerful parameter-based matching resource allocation provides the required flexibility, however it does not allow user-defined service selection from a group of matching ones, nor does it support decentralization and runtime adaptation.

3.5 MAGNET

The MAGNET trader [Kos98a,b,c] caters for dynamic resource management in a object based distributed system. In particular, it deals with operating systems and mobile computing. Similar, to other object traders it implements a yellow-page-service. However contrary to most others, MAGNET monitors the quality of service and rebinds services if necessary. Technically, the Magnet trader contains a shared data repository available to all components based on the tuplespace paradigm. Structured data items (requests and service descriptions or meta-Knowledge) are placed into the information pool as tuples. Uniquely to Magnet, the shared information pool is distributed via federations thus avoiding the performance bottleneck problems experienced previous trading architectures.

3.6 Summary of Classical Traders

To summarise, current trading mechanisms fall short when dealing with a truly open distributed environment. Although they extend location transparency and provide flexible load balancing, it is still necessary to know the service name to be used. That is, most object traders provide flexible robust systems and fault tolerance at no cost, but do not support service discovery and therefore are not useful for truly open system such as the web.

3.7 Service Discovery

As argued above, currently available object traders do not cater for truly open systems, where no assumptions regarding name and signature of a service can be made in advance. While such flexibility is not necessary for many applications, it allows one to explore the wealth of components available over the Internet. And although such components have to be treated carefully as they may not be reliable or worse may be malicious, the easy and free access to them, turns service discovery into a vision for a new programming paradigm, where the “network is the computer”. Here it is envisaged that a computer program is generated on the fly from composing objects obtained on the Web.

Service discovery could be realised through regular search engines. However, currently used engines limit themselves to plain text when indexing and cannot make use of other formats. Search engines of the future may overcome this problem. The Agora [Sea98] and Webtrader [Vas99] project are two such efforts. In the Webtrader system, servers advertise their services on XML pages containing keywords, metadata (such as QoS) and the service interface. Keywords and metadata help the trader to match services against client requests; the interface allows the client to use the service. While this approach requires servers to advertise in the required XML format, Agora aims to trade existing components, which are already online. Similar to classical search engines, which parse HTML pages for indexing, Agora parses Java class files and indexes them. Preliminary results show that this method is promising for applets, JavaBeans, and ActiveX, but has limitations for Corba objects, whose interface is not directly accessible.

Agora and Webtrader are paving the way for a new computing paradigm and consequently three application domains may benefit directly [Vas99]:

1. Application level brokers, such as the shopping agent Jango, which integrates given online information sources 2. Adaptive content delivery, which enables a client with limited capabilities to obtain contents in the format it requires. 3. Application recontexting, which may be necessary due to limited resources to reconfigure thin clients on mobile devices as their environment changes.

With such service discovery systems at hand, it is paramount to deploy intelligent clients, which do not fully trust the traders recommendations and which learn to choose the best services. Service discovery being the first step, the second and more difficult step is the client's service selection process. Obviously, this choice should be informed and not be made by the trader as we opt for service shopping rather than matching. Therefore, we have to consider selection criteria for clients. One approach [Pud95] to help the client "understand" service descriptions is the use of ontologies. But the problem all ontologies face is the question, who defines them and who ensures that all parties give them the same meaning [Nwa99]. Therefore, the approach followed in [Pud95] is severely limited and the author's point out rightly that the challenge remains to have clients, servers, and traders learn the shared ontology.

4 Rough Traders and Intelligent Clients

In our approach, we want to go a step further and drop any notion of explicit meaning, which we cannot obtain. Instead we rely on meaning, which evolves implicitly. As Wittgenstein said, 'The meaning of a word is its use in language', we argue 'the meaning of a service is given by its use by the client'. That is, the client learns which services to use through experience. Rather than relying on the trader, our intelligent clients base their choice on their history of use to generate a notion of trust in services. This trust evolves over time and reflects the client and servers joint understanding of service descriptions. Recently, there has been considerable interest in the notion of trust within agent systems [Elo98, Jon99]. In particular, Jonker and Treur [Jon99] define a comprehensive framework of useful trust functions based on e.g. different window sizes of history, which can serve as sound basis for the client's adaptive selection process. Avoiding the problems of ontologies, the use of intelligent clients with a trust mechanism reduces the service selection problem to service assessment. The main difficulty for a client to update its trust in a server is for it to have a fair judgement on the service it had rendered. While many aspects may be impossible to formalise, issues such as time taken, exceptions generated, cost, and simple service specs (did the sort service return a sorted list?) can be automated.

4.1 Requirements

Let us summarise the two main ideas of adaptive service discovery and more flexible trading as outlined above: 1. traders provide rough matches to client requests based on informal service description and keyword search. 2. It is up to the client to choose the service and to learn over time, which of the services is best suited. To implement these two ideas our trading simulation has to meet the following criteria: 1. A client maintains a history of its interactions with a given service. The history of its interactions with a

service is used to update the client trust value of that service. 2. A client maintains a history of its interactions with the trader The history of its interactions with the trader is used to inform the trust value the client will associate with any future interaction with the trader. 3. The trader's recommendations for the clients is of variable 'quality' and almost never perfect. 4. Services actively pursue client interaction through advertising with a trader and providing appropriate descriptions of their service.

4.2 The Current Implementation

In our current implementation in Java, the trader maintains data (an informal service description and a server reference) on all of the services that are currently registered. Service objects can register and de-register with the trader. When registering with the trader the service object supplies all the required data. Service objects comprise a quality value and a cost value. Service quality may be static or vary over time. The client object maintains data (a history object) on its interactions with the trader and with the services it uses. At a simple level the client maintains a trust rating for the trader and a trust rating for all the services that it uses. In selecting a new service the client uses its own rating in conjunction with the traders matching rating for each service to come to a decision. After using a service the client updates its rating of the service. The Client has a test mechanism to 'test' a service. While this is ostensibly the same as using a particular service, a test mechanism would be entirely reasonable for many types of service (e.g. translation service). Testing may be expensive in terms of 'cost' (or alternatively services may offer a test mechanism at a lower/no cost) and overloading the network so it is in the interests of the client to keep testing to a minimum. Depending on the client's trust of any particular list of services (from the trader) the client can then test a number of choices before selecting a particular service. The client's default behaviour is to iterate through the entire list of services recommended by the trader. The client then selects a service from all successful test invocations by rating the service by its 'quality' and its cost. The 'quality' is an arbitrary value (between 1 worst and 100 best) associated with each service. This value may remain static or may change over time depending on the particular service. The cost value is a charge associated with each service for the use of that service. At present the cost value associated with each service is fixed over time. The client chooses the service with the lowest cost per quality ratio. It then invokes the service method on the service. If the invocation fails the client will try to invoke the service method again until a successful invocation is achieved (the method invocation failure is simulated and simply means that the service was not carried out). The client will then make a new request to the trader and repeat the process. There are two options to change the behaviour of the client. Firstly, the 'Using History' option. Here the client, after iterating through the entire trader list will retrieve any data it has on the services on that list from its own 'database' (a History object). For each service the client calculates its reliability value. This is simply the ratio of invocation failures to invocation successes for that particular service. The client then calculates an overall rating for each service based on its reliability and the cost per quality value it got from testing the service. The client then selects and uses the service with the highest overall rating. The second option is the 'Dynamic Trust' option. At present the client divides the trader list equally into four quarters and depending on its current trust rating of the trader will either iterate

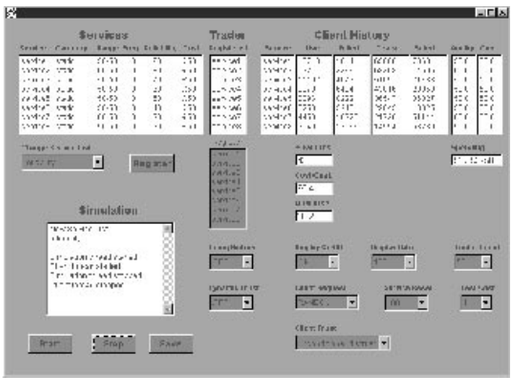


Fig. 1. Screenshot of rough trader and intelligent client simulation.

through one quarter, one half, three quarters or the whole list when conducting its testing process. (The assumption being made here is that the trader is returning an ordered list with its highest matched services at the top of the list and less likely services lower down). Whenever the client makes a selection it records where in the trader list the selection was found. Before each testing process the client checks its recent selection record. If selections were predominantly been above the mean point of its current trust rating value then the Client would increase its trust rating of the trader and thereby choose to test a smaller portion of the list of service. If recent selections have been below the current trust rating mean point then the client will lower its current trust rating of the Trader. At present the 'Dynamic Trust' option effects the overall costs acquired by the client when testing. The cost associated with testing a particular service can be changed by adjusting the 'Test Cost' setting. While this cost is presently a monetary cost it could equally be thought of and changed accordingly to become a value associated with use of a network in a distributed system – a 'network cost'

4.3 Simulation

Consider Figure 1. Among others, the GUI shows the failure rate in percent, the client's cost/quality, quality/use, and the overall spending. The history option allows to enable/disable the use of experience with the service in the client's decision making. Thus, this option implements the history-based trust functions described in [Jon99]. Besides rating the services, the client can rate the trader which is enabled by activating 'Dynamic Trust'. Values can range from unconditional distrust to conditional trust. The 'Random' setting instructs the trader to return a list containing all registered services in a randomised order. Test cost alters the cost incurred by the client when testing a particular service. A setting of '1' means the testing cost is the same cost as using a service. A higher cost setting reduces the cost of testing in relation to the cost of using a service.

Figure 1 shows in particular the eight services that are registered with the trader. All these services oscillate their 'quality' value between the range shown and at the frequency

shown. In this test the cost of each service is fixed to reflect the average quality a service delivers over an extended period of time. The trader returns a list of these services at random to the client upon each request The client history window shows how many times each service was used during the test run. It also displays the average quality it received from each service and the average cost incurred for each service (this is in fact the fixed cost, as the cost value does not alter). In this test run the client has used service1 (for a maximum of 301157 times) and has obtained an average quality from that service of 73.6. The average service quality service1 produces is 50. The overall 'cost per quality' is 0.73. Because of the parity between the average quality of each service and their fixed cost a client randomly selecting services would expect a 'cost per quality' of 1.00. In this set-up the client trust (trust of the trader) is set at '8'. The client tests all eight services it receives from the trader and chooses the service with the lowest 'cost per quality'. A client that had more 'trust' in the trader may set its trust value to, for example. '3' and only test the three most highly matched services it receives from the trader. After selecting and using a service the client, as configured in this set-up, will continue to use the service providing the 'cost per quality' value remains the same or is not less than its previous use of the service. The test cost variable provides the ability to change the cost to the client of testing a service. The service-reset rate provides the ability to reset all the services at regular intervals during a test run. Initial service settings are randomised between the accepted range of each service. The client does not use any of the history data that it gathers to aid its selection choice. It does not alter its trust value (of the trader) and it does not alter its request to the trader during a test run. This functionality will be added to the client to enhance its 'intelligence'. It is intended that the prototype will be enhanced further by allowing multiple clients (differently configured) and allowing different 'types' of services which will register and de-register (or become unavailable) during a simulation Three example tests are included in this paper to demonstrate the functioning of the system. Each test is configured to demonstrate the effectiveness of the client selection process with differently configured services. These tests are primarily to demonstrate the potential benefits that can be gained from an 'intelligent' client using a rough trader to maximise its service selection efficiency.

Reliability Test. This simulation run demonstrates the clients' capabilities when presented with a list of services with differing reliabilities. There are eight services, all providing a static quality of service of 50. The services cost is also fixed at 50. Service1 has a reliability factor of 90. This equates to a service failure rate of 10 %. Service8 is the least reliable service with a failure rate of 80%. Figure 2 shows the results from running the client without history/trust, with history, with trust, with both. The history mechanism in the client effectively reduces the number of service use failures. In this test the dynamic trust mechanism has no effect on the clients efficiencies because the cost per quality of each service is the same and is fixed, so the client rates all the services equally, and therefore cannot make a judgement on the 'trustworthiness' of the trader list.

Oscillating Service Quality Test. This simulation run demonstrates the clients capabilities when presented with a list of services whose quality oscillates between set values

Reliability	Default	History	Trust	Both
Failures(%)	44	11	44	19
Cost/Quality	5.40	5.40	5.40	4.07
Quality/Use	27.99	44.21	28.10	40.15

Fig. 2. Reliability test results

over time. All these services are 100% reliable and thus do not produce any invocation failures. The cost of each service is fixed. There are eight services all providing a service whose quality oscillates (a sine wave) over the stated range. Service1 produces a service whose quality ranges from 55 to 95 and whose cost is fixed at 75. Service8 produces a service whose quality ranges from 20 to 60 and whose cost is fixed at 40. Figure 3 shows the results of the simulation.

Reliability	Default	History	Trust	Both
Failures(%)	0	0	0	0
Cost/Quality	7.64	7.65	4.95	4.90
Quality/Use	66.79	66.7	66.4	66.44

Fig. 3. Oscillating test results

Consider Figure 4. The graph shows the clients view of the services it uses in the oscillating test. Different colours represent different services. So each spline is part of its service's complete quality curve, which is, for clarity sake, not shown. Around time point 350, the client selects e.g. service 8, whose quality steadily increases. As it declines at around time point 400, the client switches to service 3, with much higher quality. At first glance, it appears the client should always select the highest quality, but in fact it optimises quality per cost, so that it does not simply stick with the best quality. As one may expect, the client tends to stick with a service as the service quality improves and drops it as it declines. The table and graph show that the history mechanism has no effect on the average cost per quality incurred by the client. This is simply because all services are 100% reliable and thus the history mechanism offers no extra information to the client on top of what it gains from running tests on the services. The trust mechanism effectively reduces the clients cost per quality because the client is able to alter its judgement as the 'trustworthiness' of the trader list and is thus able to reduce the amount of service testing that it does from the default.

Mixed Test. This simulation run demonstrates the clients capabilities when presented with a list of oscillating services with differing reliabilities. There are eight services all providing a service that varies. The services cost is fixed at 50. Service1 has a reliability factor of 92. This equates to a service failure rate of 8 %. Its quality varies between 55 and 95. Service4 has a failure rate of 2%. The history mechanism reduces the amount

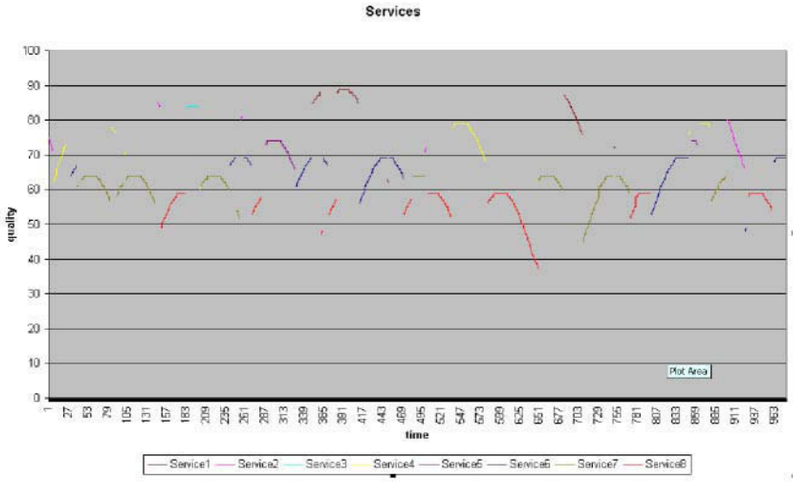


Fig. 4. Clients perspective of selected services with oscillating quality (y axis) over time (x axis). Different colours represent different services. The splines correspond to the fragments of service quality curve that were selected by the client.

of service failures but increases the cost per quality. The increase in the cost per quality is due to the fixed nature of the cost associated with each service and the fact the more unreliable services are offering a generally higher quality of service. The dynamic trust mechanism reduces the cost per quality, and both mechanisms operating together offer a better failure rate and a better cost per quality than the default run.

Mixed	Default	History	Trust	Both
Failures(%)	6	2	5	2
Cost/Quality	5.19	6.95	3.58	4.7
Quality/Use	77.71	60.46	73.42	58.96

Fig. 5. Mixed test results

Conclusions from Testing. The results show that both client mechanisms, using history and dynamic trust, are effective in enabling the client to select and use services in a more efficient manner. In particular, they achieve a failure reduction and lower cost per quality during operation. Additionally, the two mechanisms enable the client to use and judge the quality of the information it receives from the trader. 4.3.5 Further Enhancements to the Model The next step for the simulation is to move from a simulated trader to a real online

service discovery tool such as Agora [Sea98] and Webtrader [Vas99]. Additionally, it would be interesting to implement a feedback loop for the trader, which would amount to a joint learning of best service provision by all parties involved. Regarding, clients it would be interesting to consider the ability of clients to reformulate requests, if the trader does not provide satisfactory results.

5 Conclusions

The concept of traders to realise flexible and fault-tolerant open distributed systems is around for over 10 years. Large bodies such as ANSA and OMG developed and integrated traders into their platforms. Nonetheless, trading solutions are not so widely adopted, which is partly due to the inability to deal with newly available services, which were not envisaged at design time of the system. All classical approaches such as the Corba trader [Omg92], AnsaWare [Apm98], Aster [Iss98], Matchmaker [Ram98], and Magnet [Kos98a,b,c] rely on a common language and understanding of services and clients, which is a severe limit when dealing with an open system. In most approaches the common language is simply predefined and joint understanding between clients and servers is tackled through joint ontologies. Although the research into ontologies and common understanding has been active for a few years, there is not yet a groundbreaking solution to this difficult problem [Nwa99]. Our approach is dramatically different in that it does not rely on any pre-defined ontology or notion of meaning. We follow Wittgensteins idea that “the meaning of a word is its in language” and allow a joint understanding between clients and servers to evolve over time and experience. To this end, we introduce a trust function [Jon99] into our clients, which enables the client to learn the best suited service profiles and adapt its service selection accordingly. Motivated by [Jon99], we compare clients taking decisions based on no history and full history, respectively. Additionally, we compare and vary the client trust into the trader ratings. We implemented this scenario and ran tests regarding failure rate, oscillating service quality, and a mix of both. Our simulation indicates that the use of the client’s experience (history) reduces the failure rate substantially. The dynamic trust option, which allows the client to rate the traders recommendations, are valuable to reduce the cost, as the client does not have to test services constantly, when it can rely on its perceptions of the trader’s ratings. To take the results of our simulation further, future work will concentrate on making the trader more intelligent and replacing it by an operational service discovery tool such as Agora [Sea98] or Webtrader [Vas99] and embedding it into middleware solutions such as Magnet [Kos98a,b,c]. To summarise, the contributions of this paper are fourfold. First, we review trading platforms and service discovery as currently used in middleware and outlines their strength and weaknesses. Second, we introduce a radically different approach to trading, which does not rely on any pre-defined ontology or other explicitly defined meaning. Third, to underpin our novel ideas, we implemented a simulation of an intelligent client dealing with unreliable traders. For the client decision making process we focused on two aspects: the use of previous experience and the client’s rating of the trader’s recommendations. Forth, we evaluated the simulation and came to the conclusion that history data can substantially reduce the service failure rate and the rating of the trader can reduce the costs, as less testing of services is required.

Acknowledgements. We'd like to thank Patty Kostkova, who contributed substantially to the trader in middleware survey.

References

[Apm98] A.P.M. Ltd. The ANSA Reference Manual Release 01.00. APM Cambridge Limited, UK, March 1989.

[Elo98] G. Elofson. Developing Trust with Intelligent Agents: An Exploratory Study. In Proceedings of the First International Workshop on Trust, 1998

[Dec97] K. Decker, K. Sycara, and M. Williamson. Middle- Agents for the Internet. In Proc. of IJCAI97, Nagoya, Japan, August 1997.

[Des92] J.-P. Deschrevel. A Brief Overview of the ANSA Trading Service. ISA Project technical report APM/RC.324.00, February 1992

[Iss98] V. Issarny, C. Bidan, T. Saridakis. Achieving Middleware Customization in a Configuration-Based Development In Proc. of Intl. Conf. on Configurable Distributed Systems, p207-214, 1998.

[Jon99] C. M. Jonker and J. Treur. Formal Analysis of Models for the Dynamics of Trust Based on Experiences. In Proc. of MAAMAW99, Springer-Verlag, LNAI 1647. 1999

[Kos98a] P.Kostkova, MAGNET: QoS-based Dynamic Adaptation in a Changing Environment. Technical Reprt. City University. Feb 1998. ftp://ftp.cs.city.ac.uk/users/patty/qos- abstract.html

[Kos98b] P.Kostkova, T.Wilkinson: MAGNET: A Virtual Shared Tuplespace Resource Manager. In Intl. J. on Parallel and Distributed Computing. 13, September 1998.

[Kos98c] P. Kostkova, J. S. Crane, J. A. McCann, T. Wilkinson: MAGNET: A Dynamic Information Broker for Mobile Environments. Technical Report. City University. March 1998. ftp://ftp.cs.city.ac.uk/users/patty/broker-abstract.html

[Pud95] A. Puder et al, AI-based Trading in Open Distributed Environments. In IFIP International Conference on Open Distributed Processing. Brisbane, Australia, 1995

[Omg92] Corba - The Object Management Group, OMG The Common Object Request Broker: Architecture and Specification. Wiley. 1992

[Nwa99] H. Nwana and D. Ndumu. A perspective on software agents research. In Knowledge engineering review. 14:2125- 142. Cambridge University Press, 1999

[Ram98] R. Raman, M. Livny, M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In Proc. of IEEE Intl. Symp. on High Performance Distributed Computing, Chicago, USA, July 1998.

[Sea98] R. C. Seacord, S. A. Hissam, K. C. Wallnau. Agora: A Search Engine for Software Components. CMU/SEI-98-TR-011. August 1998

[Vas99] V. Vasudevan and T. Bannon, WebTrader: Discovery and Programmed Access to Web-based Services. In Proc.of Intl. WWW Conf. Toronto, Canada, May 1999

Customer Coalitions in Electronic Markets

Maksim Tsvetovat[1], Katia Sycara[1], Yian Chen[2], and James Ying[2]

[1] Robotics Institute,
Carnegie Mellon University,
Pittsburgh PA 15213, US
[2] Institute For Information Networking,
Carnegie Mellon University,
Pittsburgh PA 15213, US

Abstract. [1] In the last few years, the electronic marketplace has witnessed an exponential growth in worth and size, and projections are for this trend to intensify in coming years. Yet, the tools available to market players are very limited, thus imposing restrictions on their ability to exploit market opportunities. While the Internet offers great possibilities for creation of spontaneous communities, this potential has not been explored as a means for creating economies of scale among similar-minded customers. In this paper, we report on coalition formation as a means to formation of groups of customers coming together to procure goods at a volume discount ("buying clubs") and economic incentives for creation of such groups. We also present a flexible test-bed system that is used to implement and test coalition formation and multi-lateral negotiation protocols, and show use of the test-bed system as a tool for implementation of a real-world "buying club".

1 Introduction

A coalition is a set of self-interested agents that agree to cooperate to execute a task or achieve a goal. Such coalitions were thoroughly investigated within game theory [9,10,14,11]. There, issues of solution stability, fairness and payoff disbursements were discussed and analyzed. The formal analysis provided there can be used to compute multi-agent coalitions, however only in a centralized manner and with exponential complexity. DAI researchers [11,14] have adopted some of the game-theoretical concepts and upon them developed coalition formation algorithms, to be used by agents within a multi-agent system. These algorithms concentrate on distribution of computations, complexity reduction, efficient task allocation and communication issues. Nevertheless, some of the underlying assumptions of the coalition formation algorithms, which are essential for their implementation, do not hold in real-world multi-agent systems.

In this paper, we report on coalition formation as a means to achievement of economies of scale among customer agents. In particular, we concentrate on

[1] This material is based on work supported in part by MURI contract N00014-96-1222 and CoABS Darpa contract F30602-98-2-0138 and NSF grants IRI-9612131 and IRI-9712607

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 121–138, 2001.

formation of "buying clubs" - groups of customers coming together to procure goods at a volume discount. The paper is organized as follows: We begin by presenting the economic models that show how both suppliers and customers can benefit from advent of such buying clubs (i.e. *incentives* to create buying clubs), which are critical in any real-world system. In section 5, we discuss the issues that need to be addressed in creation of a coalition protocol, and provide and critique several distinct coalition models. We proceed in section 6 to describe test-bed that can be used to test different coalition formation protocols and coalition models, as well as a real-world system that was implemented using the test-bed. We conclude by describing future experiments that will be conducted using the test-bed system.

2 Prior Work

Auctions are the predominant electronic commerce models on the Internet but they are not the most suited for the wholesale marketplace. Research centers like MIT Media Labs have been trying to address the issue of cooperative market models (Tete-a-Tete system [6]) but there is still a lot of research that needs to be done. Currently there are no implementations of a wholesale agent market where agents collaborate and do many-to-many multi-attribute negotiations on behalf of buyers and sellers. However, research is being done on some issues of this market. Some systems, such as MAGNET ([3]) attempt to address aggregation of multiple suppliers in order to form a supply chain and complete a task. Other examples of electronic markets are Fast Parts Trading and Kasbah [2]. Fast Parts Trading Exchange is a product that allows wholesale sellers to meet and exchange offers. The main functionality of Fast Parts Trading is to act as an electronic exchange and it does not incorporate automated negotiation nor does allow collaboration between buyers. Kasbah is an agent marketplace that allows agents to negotiate the price of goods on behalf of consumers. There are several differences between the proposed framework and Kasbah. The first is that Kasbah only allows negotiation based on price while our framework implements multi-attribute preferences. Also, Kasbah does not allow buyers to form coalitions. Most important, Kasbah uses only one specific protocol and strategy for negotiation, while our work provides a flexible test-bed where different strategies can be experimented with.

A lot of research has been devoted to studying game-theoretic properties of coalitions[9,11]. The main topics of this work has been coalition stability, fairness, payoff distribution, methods for efficient formation of coalitions, as well as methods for manipulating results of coalition processes. However, most of this work has concentrated on theoretical aspects of coalitions, and thus there is currently no work or implemented system in the context of buyer coalition formation.

However, recently there has also been some commercial work in the area. Accompany.com (*www.accompany.com*, [1]) allows customers to join together during a "buying cycle", and obtain a volume discount dependent on the size of

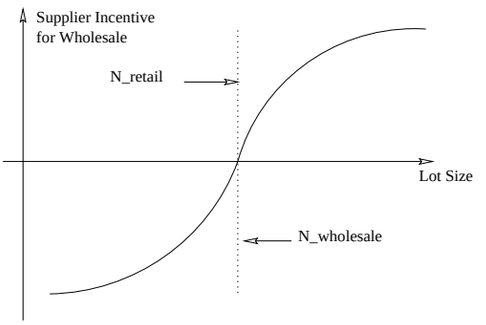


Fig. 1. Supplier's Incentive to Sell Wholesale

the group. However, the company functions as a retailer (i.e. is responsible for breaking up volume shipments and shipping retail-sized packages to individual customers) - which significantly affects the costs and benefits of buyer coalitions. Also, Accompany.com functions by pre-negotiating volume discounts with a handful of suppliers, thus leaving the user out of the bidding process.

3 Incentives for Customer Coalitions

When one studies an electronic commerce system, especially one dependent on a novel approach such as buyer coalitions, one has to consider the incentives involved in this system. In short - what would be a compelling reason for a person to move to a new commerce paradigm. Such incentives are usually monetary - reduction of cost, or increased profit, although they can include less tangible benefits such as reduction of risk (or allowing someone else to assume the risk), or increase in market size or market share.

In this section, we will outline the economic incentives that could compel both suppliers and customers to join an electronic commerce system based on buyer coalitions.

3.1 Supplier Incentive to Sell Wholesale

Let us assume that suppliers are (a) rational and (b) self-interested, i.e. they will attempt to sell their goods at maximum profit. As new opportunities present themselves, seller agents will conduct some sort of cost-benefit analysis and take advantage of all opportunities to raise their profits.

As a simplifying assumption, let us also say that the manufacturing cost of one item is constant and, above some threshold, independent of the amount of units sold. This is true if the supplier's production facility is working at or near capacity.

Now, let us suppose that an agent is selling its goods retail.

Let:

- U_{item} be the utility (profit) of selling one item retail.
- P_{item} be the sale price of the item (or reservation price in an iterative negotiation)
- $C_{retail-marketing}$ be the cost of retail marketing, per item (in an electronic marketplace, this could be related to the cost of submitting advertisements or the cost of making bids or a per-transaction charge imposed by the market)
- C_{item} be the cost of manufacturing one item.

The utility an agent receives from selling one item can be expressed as

$$U_{item} = P_{item} - C_{retail-marketing} - C_{item}$$

The utility of selling n items would be

$$U_n^{retail} = P_{item} * n - C_{retail-marketing} * n - C_{item} * n$$

The utility of selling a lot of n items *wholesale* (i.e to one buyer instead of many) can be expressed as

$$U_n^{wholesale} = P_{item} * n - C_{wholesale-marketing} - C_{item} * n$$

An agent would have incentive to sell at wholesale if it receives greater utility from such sale, or

$$U_n^{wholesale} - U_n^{retail} > 0$$

From the above expressions, we then infer that the agent has incentive to sell wholesale if

$$C_{wholesale-marketing} < C_{retail-marketing} * n$$

Hypothesis:

Since marketing to one buyer is usually less expensive then marketing to multiple buyers, the incentive to sell wholesale will usually be present. However, wholesale marketing to one customer will be more expensive then retail marketing to one customer, due to more protracted negotiation and other factors. Up to some lot size N_{retail}, the supplier has a negative incentive for selling at a wholesale price, as marketing costs will be nearly identical to retail and lowering the price to wholesale level is not justified (see figure 1) As the size of the lot increases past that point, selling wholesale lots becomes beneficial for supplier.

This pattern can repeat as the lot size increases, resulting in multiple price breakpoints. For example, one could purchase an item at retail quantities (by pack), by case (12 packs), by box (50 cases), pallet of boxes or truck-load. All of these packaging options have different costs for the supplier, and are likely to result in different wholesale prices.

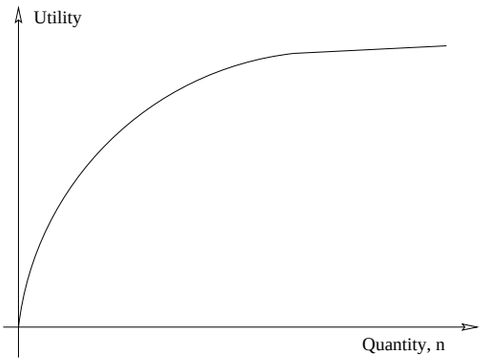


Fig. 2. Customer Total Utility

3.2 Customer Incentive to Buy Wholesale

The customer utility curve (shown on figure 2) is commonly known in the field of economics and illustrates the law of decreasing returns. It illustrates the fact that utility of each unit acquired is smaller than that of the previous unit.

However, a more realistic representation of the utility curve (fig. 3) shows that there is range of acceptable quantities of the good, after which the utility of each additional unit drops sharply. This is due to the additional costs associated with storage or management of surpluses.

Customer Utility $U = Benefit - Price_{unit} - Cost_{storage}$

Let us define the Maximum Utility Range (MUR) as

$$MUR = (n_{min}, n_{max})$$

, a range in which the utility is high while management costs remain low.

If the supplier's optimal size of wholesale lot $n_{wholesale} \in MUR$, then the customer can purchase a wholesale lot.

HOWEVER: If the price of purchasing goods in larger lots remains the same as retail price, the customer has no incentive to buy such lots and might just as well buy through retail channels at a higher marketing/packaging cost to supplier, thus lowering supplier's per-unit profit margin.

Thus, supplier has an incentive to sell wholesale lots, it must give the customer an incentive to buy wholesale - which is commonly done by lowering per-unit price at when the requested lot size is large enough.

These price decreases are usually represented by a step function such as the one on figure 4.

ASSUMPTION: Customer's utility of an item is higher if the price of the item is lower while all other factors remain constant, or

$$\Delta U_{customer} = \Theta(\Delta P)$$

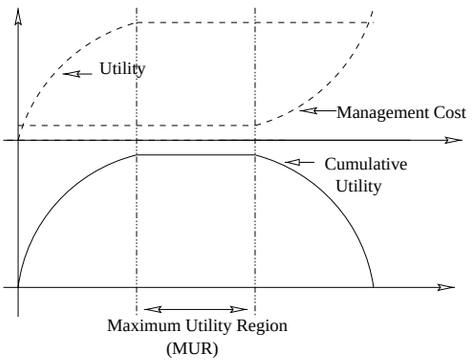


Fig. 3. Customer Per-Unit Utility and Costs

Thus, if the supplier lowers its price for larger lots, the customer has an incentive to buy wholesale.

4 Coalitions and Wholesale Purchasing

In the real world, a single customer rarely wants to buy large enough quantities of goods to justify wholesale purchasing, or

$$N_{wholesale} \notin MUR$$

In order to lower the purchase price (and, therefore, increase utility), self-interested customer agents can join in a coalition such that

$$N_{wholesale} \in MUR_{coalition} = \sum MUR_i$$

where MUR_i is the MUR of each member of the coalition.

This would enable the coalition to buy a wholesale lot from the supplier, break it into sub-lots and distribute them to its members, thus raising the utility of each individual member.

However, the formation and administration of coalitions, as well as distribution of sub-lots has its costs, represented as $C_{coalition}$.

$C_{coalition}$ consists of number of different costs, closely related to the real-world situation where the coalition is formed. In particular, such costs include the cost of administering coalition membership, cost of collecting payments from individual members, and the cost of distributing items to the members when the transaction is complete. In some cases (such as distributing copies of software) some of the costs can be very small, and in other cases may rise to be prohibitively large.

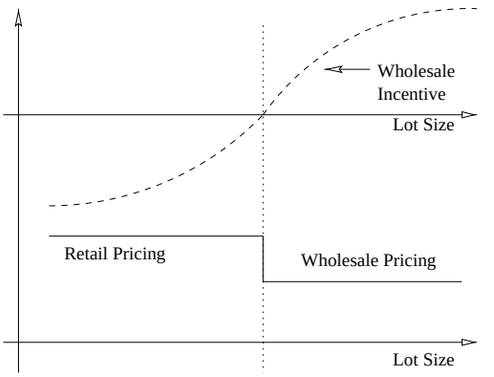


Fig. 4. Wholesale and Retail Pricing

A coalition is only viable if the increase in group's total utility from wholesale purchases is greater than the cost of creating and running a coalition, or

$$\Delta U > C_{coalition}$$

5 Coalition Models

It is possible to construct a number of coalition models and protocols, all of which would have different properties and requirements. In general, all coalition models include several stages:

- **Negotiation:** The coalition leader or another representative of the coalition negotiates with one or more suppliers to provide the good or service. The protocol must address issues such as the choice of suppliers, and evaluation of competitive bids.
- **Coalition Formation:** The coalition leader solicits new members to join his coalition. It is important to note that the coalition must have some admission constraints (such as geographical proximity of the members or their ability to pay for the goods.).
- **Leader Election/Voting:** The members elect a coalition leader or cast direct votes for or against certain bids. Not all coalition formation protocols make use of this stage.
- **Payment Collection:** The coalition leader (or elected treasurer, as defined by the protocol) collects the payments from coalition members and is responsible for conveying the full amount to the supplier
- **Execution/Distribution stage:** As a transaction is executed and the purchased goods arrive, they must be distributed to the members of the coalition.

In design of coalition protocols, the following issues have to be taken into account:

- **Coalition Stability:** Are members of a coalition allowed to leave? If yes, what would be their incentive for leaving? Would a member leaving a coalition incur any costs or penalties - or would these penalties be incurred by the coalition as a whole or the supplier?
- **Distribution of Gain:** How are the gains from the difference between retail and wholesale prices of a good distributed to the members of the coalition?
- **Distribution of Costs and Utility:** Who bears the cost of goods distribution and how to arrange the logistics of such? If there is a reward for creating a larger coalition, how is this reward distributed?
- **Distribution of Risk:** Who bears financial risks as the transaction is executed and how large are they? Are there any uncertainties and which parties experience them? What are the strategies for minimizing such risks?
- **Trust and Certification:** There are three levels of trust required for the coalition leader - trust in the negotiation stage, payment collection and in the distribution stage. Is such trust critical in the protocol? Is it possible to design a protocol that would not require such trust or minimize the number of stages where trust is required? How can the coalition deal with a breach of trust?

In this section, we will discuss a number of protocols for forming customer coalitions and address some of the issues raised above.

Most coalition protocols can be divided into two classes (*pre-negotiation and post-negotiation*), based on the order in which negotiation and coalition formation happen. In pre-negotiated coalitions, the coalition leader negotiates a deal with one or more suppliers using an estimated coalition size or order volume, and then advertises the creation of the coalition and waits for other members to join. In another scenario, the group is formed first, based on some admission criteria. Then, a group leader negotiates with suppliers, and offers the resulting deal to the group.

One of the chief differences between the two scenarios is in the distribution of risk between the parties of the negotiation. In a pre-negotiation protocol, the coalition leader must estimate the group size in order to be able to sign a deal with the supplier. If such estimate is wrong, the coalition leader must absorb the loss or through some mechanism make the coalition members pay a higher price.

In the post-negotiated mechanism, the group must be able to trust its leader to negotiate on its behalf. Unless the group is formed by a number of people who know each other through other channels (i.e. a group of students in a class), there would have to be an explicit leader selection/verification mechanism, or a mechanism for collective negotiation.

5.1 Post-negotiation

A simple post-negotiation protocol involves the following parties: the coalition leader (L), a set of suppliers $S = \{s_1, s_2, ..., s_n\}$, a set of potential coalition members $M = \{m_1, m_2, ...m_n\}$, and a coalition advertising server CS.

1. $L \rightarrow CS$: Advertise creation of a coalition with specified parameters (such as item to be procured, location of the leader, etc.). The coalition is open to members for a limited period of time or until a specified group size is reached.
2. $CS \rightarrow M$: The coalition server supplies the coalition advertisement to potential coalition members
3. Each $m_i \in M$ considers whether to join the coalition
   $m_i \rightarrow L$: A "Join the Coalition" message
4. At the expiration of the coalition deadline/size limit, the leader enters the negotiation with the suppliers $s_i \in S$ using its private protocol/strategy and decides on a deal in the best interest of the group.
5. Coalition Leader L collects money from group members, and arranges for the shipping and distribution of goods.

This protocol requires an immense amount of trust in the coalition leader. In fact, the protocol is wide-open to representatives of suppliers ("shills") starting coalitions that would act in the behalf of a given supplier rather than groups of customers.

If there is no implicit trust in the coalition leader that can be inferred from other sources (such as previous relationships between coalition leader and its members), the protocol must feature mechanisms that help the coalition members establish this trust or conduct transactions without having to trust the leader.

Trust in the coalition leader can be established in a number of ways. Leaders can be elected from the general membership of the group before the negotiation starts. The group could also appoint a trusted third party to conduct negotiations on its behalf. Also, the coalition leader could be compelled to open every step of the negotiation to the scrutiny of group members.

It is also possible to conduct negotiation by having the members of the group vote on bids. In this mechanism, the result of the negotiation would be acceptable to the majority of members of the coalition, and it would take a large number of "shills" for a supplier to sway the opinion of the coalition.

The approaches that utilize voting may not be practical due to the long time it would take to conduct a multi-round negotiation and the amount of communication that is necessary to decide on the outcome of a negotiation. However, they can be very useful if the bid is awarded through a single-round auction. Also, it has been shown that voting systems can be manipulated in a number of ways [5,9] and thus have to be approached with some caution.

5.2 Pre-negotiation

The simplest pre-negotiation protocol is defined as following:

1. The coalition leader L conducts a negotiation session with a set of suppliers$S$, using his private parameters such as reservation prices, bid evaluation and concession strategies.
2. $L \rightarrow CS$: After the negotiation stage is complete, the coalition leader opens the coalition to new members, disclosing the details of the deal struck in the negotiation stage.
3. $CS \rightarrow M$: The coalition server supplies the coalition advertisement to potential coalition members
4. Each $m_i \in M$ considers whether to join the coalition
   $m_i \rightarrow L$: A "Join the Coalition" message
5. After a certain period of time elapses, or the coalition gains some minimum number of members, the coalition leader closes the coalition to new members and executes the transaction.

In this protocol, the coalition leader carries a number of risks. In order to give volume discounts, suppliers must have some idea of the number of members expected to join the coalition (or, alternatively, expected quantity of goods to be sold). While this number can be estimated, the coalition leader carries a risk of not being able to find enough members to join the group. In this case, the deal must be re-negotiated, resulting in a higher price and, possibly, more members leaving the coalition - a vicious cycle that can, in the worst case, completely destroy the coalition.

Another risk factor is inherent in the fact that the coalition leader uses a private reservation price and negotiation strategy - which may be very different from the reservation price that the majority of the target population has.

Since the details of the discounts are revealed before people join the coalition, the coalition members do not have to trust the coalition leader in the negotiation stage. However, the trust in the payment collection and goods distribution stages is still required.

The main problem with this protocol is the risk that the coalition leader has to take while estimating the group size. However, this protocol could be altered to remedy this problem in the following way:

In the negotiation stage, the coalition leader presents not his estimated group size, but a range of sizes. In response the supplier bids with a step function $Price = F_{bid}(quantity)$ (see figure 5). This function can later be revealed to the coalition members if this supplier is awarded a bid. The bid evaluation strategies for operating on step functions are very similar to ones operating on singular bids.

When step function bidding is used, most risk in the transaction is shifted onto the coalition members due to the price uncertainty. For example, the potential buyer can have a reservation price that is between the maximum and minimum prices of the step function ($Pmax_{coalition} \leq P_{reservation} \leq Pmin_{coalition}$). In this case the decision on whether to join the coalition depends on the buyer's estimate of the probability that the final price will be lower than his reservation

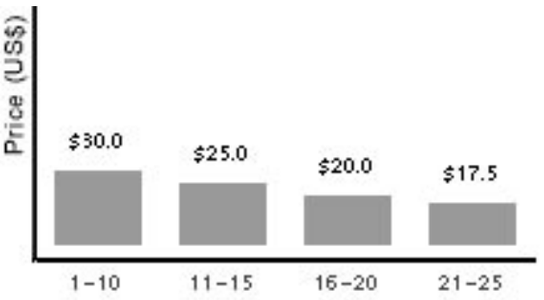


Fig. 5. A Step-function Bid from a Supplier

price, which essentially is an estimate of the final size of the group. As a result, in this case a dominant strategy for a buyer would be to wait until the coalition is almost closed for new members - which could result in a more accurate estimate of the final coalition size. However, if many buyers use this strategy, the overall behavior of the system may become non-deterministic.

5.3 Distribution of Costs and Utility

The coalition leader can operate on several different principles:

- **Non-Profit**: $C_{coalition}$ is distributed either equally among all participants or on the sub-lot size basis.
  These coalitions can be formed spontaneously for negotiating one particular deal or be stable "buyer's clubs" that exist over time.
- **For-Profit**:
  - **Consolidator**: Pre-negotiates a deal with the supplier given an estimated group size, and then re-sells the items individually, keeping enough of the savings (ΔP) to cover the $C_{coalition}$ and make a profit.
    Customer's savings $\Delta P_i = \Delta P_{coalition} - Markup$ where $Markup > \frac{C_{coalition} * n_i}{n_l ot}$ (or customer's share of the coalition costs)
    This is the business model used by airline consolidators or concert ticket distributors. They can usually obtain fairly accurate estimates of demand given the statistical data (i.e. popularity of certain air routes during a certain season), and absorb any losses resulting from not being able to sell the predicted number of seats.
    Similar approach has been used by a number of group-buy commercial sites, in particular Mercata [8] and Accompany.com [1]. These sites play both the role of a coalition facilitator (a service that helps buyers form coalitions) and a coalition leader (responsible for negotiation of volume deals and distribution of product).
  - **Rebater**: Sells the items at retail price minus a small rebate, and keeps the rest of the savings. Additional profits can be gained by delaying rebates, thus improving the cash flow of the company.

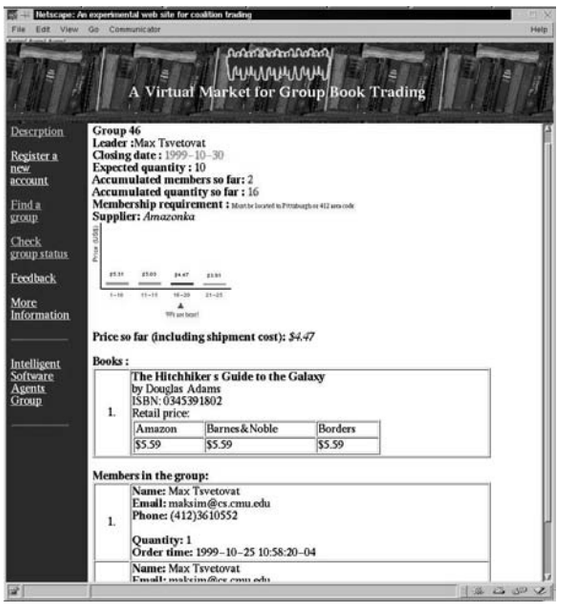


Fig. 6. Screen-shot of the WWW Interface

6 Customer Coalitions for Volume Discounts - An Implementation

In order to verify the abovementioned hypotheses, we have designed a flexible test-bed that can be used to evaluate different coalition creation protocols, as well as determine the real-world feasibility of automated agent-based coalition formation and negotiation protocols.

As an initial problem domain, we chose collective book purchasing. Often, in the university setting, one sees large number of students that are enrolled in the same class purchasing large quantities of the same book. This seemed to be a natural application of a coalition-based commerce for a number of reasons. Firstly, the group of students enrolled in the same course has the same or very similar requirements for the book, so the issue of matching customers to correct groups is greatly simplified. Secondly, the distribution and payment collection are much easier given the fact that the coalition members have to physically gather in a classroom several times a week. Thirdly, it would provide us with a large base of potential users once the system is ready for real-world tests.

6.1 Coalition Protocol

The system we have implemented uses a pre-negotiation protocol with step-function bids, as described in section 5.2. The coalition leader specifies the pro-

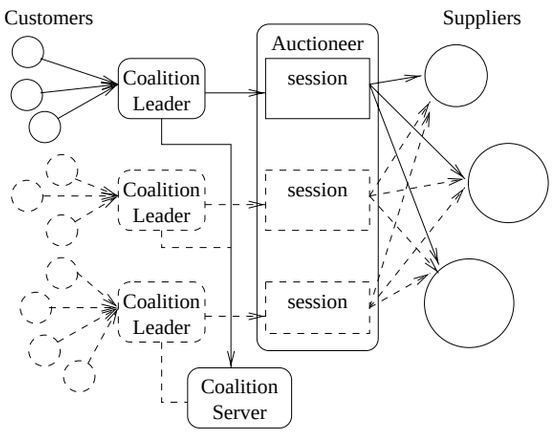


Fig. 7. System Architecture

duct to be bought through a search in the books-in-print database, and submits a request for bids to a set of supplier agents.

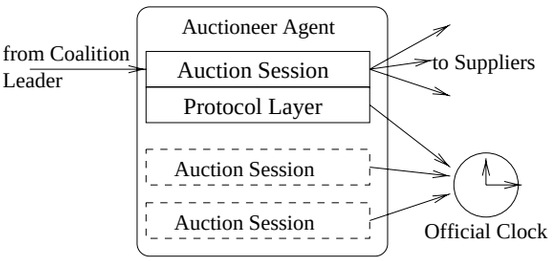


Fig. 8. Auctioneer Agent

The auctioneer agent (figure 8) implements and enforces a simple first-price sealed bid auction protocol [12] for negotiation between suppliers and the coalition leader. We chose this protocol for its simplicity and ease of comprehension for first-time users. We have already defined more advanced additional protocols (such as open-bid first price auction, Vickrey auction and a simple leveled commitment protocol [13,12]) and will incorporate them into the system to experimentally study their comparative performance.

Supplier agents (figure 9) respond to the request for bid by searching in their catalog databases, and applying a pricing policy to each item. The pricing policy

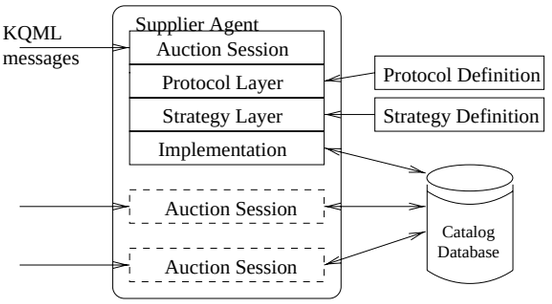


Fig. 9. Supplier Agent

can be specified through configuration files or a simple interface, thus making the agents easily configurable. Applying the pricing policy to an item generates a price-vs-quantity step function as described in section 5.2, which is then included in the bid.

Before computing the bid price, supplier agent queries an information agent, which acts as an electronic catalog. In order to run our system using real-world data, each of the information agents queries a well-known Internet book retailer (Amazon.com, Borders and Barnes & Noble). In our system we used Retsina information agents ([16,15]), which can be easily adapted to serving data from any web-site. The information retrieved from the agent is then cached by the supplier agent, which greatly speeds up future access.

All bids are submitted to the auctioneer agent (figure 8), which waits for the auction period to expire and then releases the bids to the coalition leader. coalition leader can then apply its private bid evaluation strategy to determine the winner of the auction.

After the supplier has been determined, the coalition leader notifies the winning supplier of a tentative accept, and the coalition is opened for new members for a certain period of time (which is pre-set during the initialization stage). During this time, the coalition is advertised by the coalition server and people are allowed to join it.

The final price of the item is determined after the coalition membership is closed. At this time, a message confirming the quantity and delivery date is sent to the supplier, who responds with a confirmation and executes the transaction.

6.2 Testbed Architecture

The testbed system (see figure 7) consists of a coalition server, an auctioneer agent, set of supplier agents, and a web-based interface (figure 6) for end users.

The coalition server is essentially a database that allows coalition leaders to advertise their coalitions to potential members, and allows customers to search for coalitions given their criteria and join a coalition. Both customers and

coalition leaders can be either human using the web interface or agents communicating to the server in KQML [4].

6.3 Implementation of Agents

All agents in the system are implemented using a multi-layer approach that separates the definition of a protocol from definition of the negotiation strategy of an agent and its implementation. This is accomplished through use of a protocol definition language to define the high-level interactions of the agents, while a built-in Scheme interpreter is used to define the agent's strategy and script lower-level behaviors.

The protocol definition language is based on the I/O Automata formalism commonly used for definition and analysis of cryptographic and communication protocol [7]. However, it also includes a number of enhancements designed specifically for definition of negotiation protocols.

Input/Output Automata model [7] is a very general mechanism, suitable for describing almost any type of asynchronous concurrent system. The model itself provides very little structure, which allows it to be used for modeling many different types of distributed systems. Simple I/O Automata can be combined to form larger automata that represent concurrent systems.

An I/O Automaton is a model of a distributed system component that interacts with other system components. It is a simple state machine in which state transitions are associated with *actions*. The actions are classified as *input*, *output*, or *internal*. The inputs and outputs are used to communicate to other entities in the automaton's environment.

An example of a simple I/O Automaton is a process in an asynchronous distributed system. The automaton is initialized by an *internal start* input, conducts a set of speech acts with the outside world using *external send* output and *receive* input, and returns a result via its *internal return* output when it terminates.

Formally, an I/O Automaton A can be specified by its interfaces, a set of states and transitions and a set of actions associated with particular transitions:

- $signatureS(A) = input(A), output(A), internal(A)$
- *external interface* $ext(A) = input(A), output(A)$
- $states(A)$, a set of states
- $start(A) \in states(A)$, a non-empty set of states known as *start states*
- $trans(A)$, a *state-transition* relation such that for every state s and for every input action π, there is a transition $(s, \pi, s') \in trans(A)$
- $actions(A)$, a set of actions that the automaton can execute. This includes *send* and *receive* actions, as well as all *internal* actions (figure 10)

An execution of an I/O Automaton is either a finite sequence, $s_0, \pi_1, s_1, \pi_2, ..., \pi_f, s_f$, or an infinite sequence $s_0, \pi_1, s_1, \pi_2, ..., \pi_f, s_f, ...$, of alternating states and actions of A such that $(s_k, \pi_k + 1, s_k + 1) \in trans(A)$ for every $k \geq 0$.

The composition operation allows and automaton representing a complex system to be constructed by composing automata representing individual system

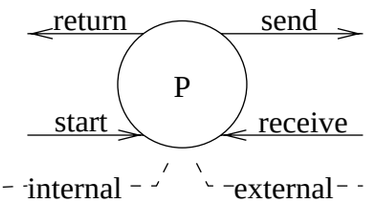


Fig. 10. A simple I/O Automaton

components. A composition of I/O Automata C is defined as having a signature

$$S(C) = \Pi_{i \in I} S_i = \{input(C), output(C), internal(C)\}$$

$$input(S) = \cup_{i \in I} input(S_i),$$

$$output(S) = \cup_{i \in I} output(S_i),$$

$$internal(S) = (\cup_{i \in I} internal(S_i) - \cup_{i \in I} output(S_i))$$

6.4 PiPL Language

PiPL (PiPL Is a Protocol Language) Language was designed based on the I/O Automata formalism presented above. It encapsulates definition of an I/O Automaton or a composition of I/O Automata into a set of productions that can be later shared over a network. The syntax of PiPL is very similar to LISP, making PiPL files easy to parse with existing parsers.

The protocol definition in PiPL consists of two main parts, the agent society definition and a composition of I/O Automata representing agents in this society.

The agent society definition breaks down an agent group into a set of roles. A role is usually defined by a set of tasks an agent is capable of performing or is allowed to perform in a given interaction. Thus, an agent playing a role of *auctioneer* in an e-commerce scenario is capable of administering an auction and is (presumably) trusted to do so.

An admission criteria could be specified to find which agents can play a given role. Also, an agent can play more than one role in an interaction, if it can satisfy admission criteria for all roles. Therefore, admission criteria have to be designed to check for conflicts of interest (i.e. an auctioneer cannot also be a bidder)

Each role definition also specifies the number of agents allowed to play this role in a given interaction (for example, there can be only one auctioneer, but there could be many bidders), and a start state for the I/O Automaton for this role (see figure 11).

After roles of agents participating in the interaction are defined, the specification defines I/O Automata for each of the role. Each automaton A is a set of states $states(A)$. A state of A $s_i(A) \in states(A) = \{\pi, T = \{trans(A, s_i)\}\}$, where π is an implemented action of the agent, and T is a set of transitions from a given state to the next state (figure 12). Action π is executed when an agent enters the state from any other state.

```
(role <role name>
      :cardinality <max.number of agents>
      :admission_cond <expression>
      :start <start state>)
```

Fig. 11. Definition of a Role

```
(state <state name>
      :roles <roles allowed in this state>
      :action <executed upon entering>
      :final <is this a stop state?>
      <list of transitions>)
```

Fig. 12. Definition of a state

A transition (figure 13) is defined as $T = \{\pi_{cond}, \pi, s'\}$ where π_{cond} is a condition that has to be satisfied for a transition to happen, π is an implemented action of the agent and s' is the state that the I/O Automaton should transition to.

```
(transition
      :roles <roles allowed>
      :cond <condition to be satisfied>
      :action <executed upon firing>
      :goto <next state>)
```

Fig. 13. Definition of a Transition

As a result, the agents in the system are not limited to use of hard-coded protocols and it is very easy to implement and add new protocols to the system.

7 Conclusions and Future Work

In this paper, we have presented the economic incentives that drive coalition formation among self-interested agents, concentrating on formation of buyer coalitions and obtaining volume discounts. We have also discussed variety of coalition models and their properties, and presented an implemented system using one of such models. The system is currently available on the Web and will be used for practical evaluation of coalition models and collection of real-world data.

A number of experiments using the coalition formation testbed are planned in the near future. Firstly, we will use a set of distinct coalition formation and negotiation protocols to conduct automated tests of performance of such protocols. Secondly, we would like to conduct a test of the real-world capabilities of the system by inviting groups of students to run through the system in a structured

experiment. Finally, we plan to extend the formalisms used in creation of the protocol definition language to allow protocol definitions to be shared by multiple agents in the marketplace, thus eliminating many compatibility problems.

References

1. Accompany.com. Accompany, inc. to revolutionize commerce: Buyers come together for best value, 1999.
2. Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.
3. J. Collins, M. Tsvetovat, B. Mobasher, and M. Gini. Magnet: A multi-agent contracting system for plan execution. In *Proceedings of Workshop on Artificial Intelligence and Manufacturing: State of the Art and State of Practice*, pages pp 63–68. AAAI Press, August 1998.
4. Tim Finin, Yannis Labrou, and James Mayfield. Kqml as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, 1997.
5. A. Gibbard. Manipulation of voting schemes: General result. *Econometrica*, 41(4):587–602, 1973.
6. R. Guttman and P. Maes. *Cooperative vs. competitive multi-agent negotiations in retail electronic commerce*. Berlin:Springer-Verlag, 1998.
7. Nancy A. Lynch. *Distributed Algorithms*, chapter pages 200-231. Morgan Kaufman, 1996.
8. Mercata.com, 1999.
9. B. Peleg. *Game Theoretic Analysis of Voting in Commities*. Cambridge University Press, 1984.
10. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, Cambridge, MA, 1994.
11. T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence Journal*, 1999.
12. Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts, 1996.
13. Tuomas W. Sandholm and Victor Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of 1st International Conference on Multiagent Systems*, pages 328–335, 1995.
14. O. Shehory and K. Sycara. Multi-agent coordination through coalition formation. In *Proceedings of the Agent Theories, Languages and Methodologies (ATAL97)*, 1997.
15. K. Sycara and K. Decker. Designing behaviors for information agents. In *Proceedings of the First International Conference on Autonomous Agents*, Feb 1997.
16. K. Sycara, K. Decker, and M. Williamson. Intelligent adaptive information agents. In *Working Notes of the AAAI-96 Workshop on Intelligent Adaptive Agents*, Aug 1996.

Algorithm Design for Agents which Participate in Multiple Simultaneous Auctions

Chris Preist, Claudio Bartolini, and Ivan Phillips[1]

Hewlett-Packard Labs
Filton Road
Stoke Gifford
Bristol BS34 8QZ, UK
cwp@hplb.hpl.hp.com

Abstract. In this paper, we discuss the design of algorithms for agents to use when participating in multiple simultaneous English auctions, aiming to purchase multiple goods. Firstly, we present a coordination algorithm, which ensures the agent places appropriate bids in the different auctions to buy exactly the right number of goods. Secondly, we combine this with an algorithm to determine what maximum bid an agent should place in an auction that is about to terminate. This algorithm combines a belief-based model of the auctions with a utility analysis. This analysis is to trade off the certain outcome of the terminating auction against the possible outcomes of the remaining auctions, and hence to place appropriate bids in each.

1 Introduction

Electronic commerce [1] is having a revolutionary effect on business. It is changing the way businesses interact with consumers, as well as the way they interact with each other. Electronic interactions are increasing the efficiency of purchasing, and are allowing increased reach across a global market.

E-commerce is not a static field, but is constantly evolving to discover new and more effective ways of supporting business. Initially, e-commerce involved the use of EDI and Intranets to set up long-term relationships between suppliers and purchasers. This increased the efficiency and speed of purchasing, but resulted in lock-in in the relationship. Both suppliers and purchasers had to invest significantly up-front in the relationship, so were not easily able to move their business elsewhere. The technological relationship between the parties was a friction factor, preventing free competition in the longer-term [2]. Often, the relationship was (and still is) beneficial to both parties. However, the lock-in effect also meant that when the relationship became less beneficial to one party, they couldn't easily move elsewhere.

The second phase of e-commerce aimed to address this problem. With the increasing availability of the web, a more open e-commerce environment is developing, al

[1] Visiting student from Department of Computing, Imperial College, London.

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 139-154, 2001.

lowing businesses to trade more flexibly with each other. Some of this openness is achieved by competition between web portals, while some competition occurs within a single web portal, acting as a marketplace for buyers and sellers to meet. Some of the efficiencies of EDI can now be achieved in a more open environment, where relationships no longer need to be long-term.

However, there is a benefit of the EDI approach that is often lost in this new phase. Price negotiation was carried out in advance in the EDI world, so purchasing can be entirely automated. When a manufacturing planning and forecast system identifies the need for a purchase, it can initiate it automatically without any human involvement, increasing speed and efficiency. In phase two, each purchase may involve interaction with a new supplier, and so may involve new negotiation of terms. As a result of this, many of these purchases can't be made automatically and instead require human interaction, mediated by the web.

The third phase of electronic commerce is just beginning. It aims to address this issue, allowing automated business interactions to take place in a fluid environment. Technology will no longer be a friction factor to changing supplier or customer. Long-term relationships will still play an important role, but they will persist because of the choice of both parties rather than technological lock-in. The key building blocks of this new world, e-services, will be able to interact dynamically with each other to create short-term or long-term trusted trading relationships to satisfy the needs of different business partners. Many technologies are involved in this development – distributed systems, encryption and PKI, XML and associated business ontologies, economic analysis and game theory, to name just a few. As automation and distribution are central to the vision, agent technology provides a fundamental role in this.

In Section 2, we discuss the role of auctions in electronic commerce, present the problem of multiple auction participation and give an overview of tools used to support participation in auctions currently. In section 3, we present a basic coordination algorithm used to bid effectively in multiple auctions that terminate simultaneously. In section 4, we extend this algorithm by providing a belief-based learning component, and use utility analysis to determine whether it is worth making a purchase in an auction about to terminate. In section 5, we describe the implementation of the system. In section 6, we present related work, and in section 7 we conclude with future directions for this work.

2 Auctions in Electronic Commerce

In phase two of the e-commerce revolution, auctions have become increasingly important both for business to business transactions and for consumer purchasing. Many different auction designs are possible. Game theory [3] can be applied to study how best to design them for specific situations [4]. Among the most popular designs are English, Dutch and Vickrey auctions. In an English auction, a seller offers a good for sale and buyers bid the price they are willing to pay. Each bid announced must be greater than the previous bid, and the item is sold to the highest bidder. In a Dutch auction, the process runs in reverse. The seller announces a proposed price, and buyers

can accept it if they choose. As time progresses, the seller decreases the proposed price until a buyer accepts. In a Vickrey auction, bidders place their bids in a sealed envelope and submit them to a trusted third party. At a certain time, the party opens the envelopes, and the good is sold to the highest bidder at the second highest price. Under certain conditions, these three auction formats can be shown to produce identical revenue for the seller.

As a result of this explosion in popularity, more and more companies are offering auction sites. Because of this, if you want to purchase a particular good, there are often many auction sites that are offering it. If you really want to get the best price, you must monitor all of these auctions using your web browser, and place bids appropriately. Care must be taken, to ensure you don't make more than one purchase! If there are a large number of auctions, this can be quite a daunting task, requiring your undivided attention for a period of time. Furthermore, if you wish to purchase more than one item, (as is often the case in B to B trading,) it becomes almost impossible.

As a result of this, auctions are beginning to offer support tools to make your job easier. Search tools such as Auction Beagle (http://www.auctionbeagle.com/) and Auction Octopus (http://www.auctionoctopus.com/) allow you to locate and monitor auctions selling specific goods, thus eliminating the need to have 20 browsers open at once. Auction Rover (http://www.auctionrover.com/) provides price trend information on popular items, to aid you in deciding what maximum price to bid. These help, but still leave a large amount of work for you to do. Particularly in the B-to-B context, this can result in an unacceptable overhead. To eliminate this, it is necessary to design automated agents able to carry out the task on your behalf.

Some sites running English auctions, such as Auction Sales (http://www.auctionsales.com/), do offer a simple bidding agent. This agent resides on the auction site, and bids on your behalf. You enter the maximum you are willing to pay, and it places the lowest possible bid on the web site (Either the reservation price, or if bidding has already started, it bids just above the current highest bid.) If all bidders in an auction use such an agent, the auction becomes a Vickrey style auction instead of an English auction. (The sale is made at second price plus the minimum bid increment.)

However, such agents are not able to participate in multiple auctions, either on the same web site or across different ones. As a result, once you use the agent, you are committed to making a purchase on the site if you can. Locally, you may pay the lowest price you can to win the auction. However, from a global perspective, that particular auction is unlikely to have been the best place to make a purchase.

These bidding agents have an additional disadvantage. To use them, you must reveal the highest price you are willing to pay to the auction site. This gives the auction site information that could be used to cheat you. Furthermore, as auction sites often receive percentage commission on sales made, they also have an incentive to cheat you. To do this, they would take note of the highest price you are willing to pay, and enter a mythical bid in the auction (a 'shill') just under this price. Your agent would then place your maximum bid, and the seller has made a sale to you at the best possible price (assuming you are the highest bidder.) The auctioneer would therefore col-

lect their maximum possible commission.[2] This second disadvantage could be overcome by placing them locally on the auction participant's machine, but this would not overcome the first, more serious, problem.

Hence, neither the auction search facilities nor the existing bidding agents provided by current technology are adequate to meet the needs of a B-to-B auction trader. In this paper, we propose a solution to this problem. We present algorithms that can be used by an agent to participate in multiple auctions on behalf of a trader, and can lead to optimal or near-optimal purchase decisions being made.

The agent aims to purchase one or more identical goods on behalf of its user. It can participate in many auctions for this good, and coordinates bids across them to hold the lowest bids. As auctions progress, and it is outbid, it may bid in the same auction or choose to place a bid in a different auction. The algorithm consists of two parts. Firstly, it has a coordination component, which ensures it has the lowest leading bids possible to purchase the appropriate number of goods. Secondly, it has a belief-based learning and utility analysis component, to determine if it should deliberately 'lose' an auction in the hope of doing better in another auction later.

3 The Coordination Algorithm

The basic algorithm uses the coordination aspect only. If all auctions terminate simultaneously, it will place optimal bids, and make the required purchase at the lowest price. However, if some auctions terminate later than others do, and new auctions may come into being, then its behaviour can be non-optimal, and needs to be supported by belief-based and economic techniques to be discussed in the next section.

An *auction house* may run one or more *auctions* for a given good. Each auction a_i offers $n(a_i)$ goods for sale. Auctions are assumed to be English auctions in format, with *bidders* placing bids at the price they are currently willing to pay for the good. A bidder may place more than one bid in a given auction. The $n(a_i)$ goods offered in the auction are sold to the bidders making the $n(a_i)$ highest bids, for the price they bid. In case of two equal bids, the item goes to the earliest bidder. Hence the auction is *discriminatory* – some buyers will pay more than others for the same good. Different auctions impose different rules covering how a bid may be entered or retracted. For the purposes of this paper, we assume that a buyer may not retract a bid, and a buyer may enter a bid provided it is at least a certain minimal increment δ above the $n(a_i)$th highest bid.

We will define the algorithm, and simultaneously present an example to illustrate the definitions. The illustrative paragraphs, provided for clarification, are indented.

Our agent participates in many auctions selling similar goods, spread out between many auction houses. It wishes to purchase m goods in these auctions, and is given a valuation of v on each good by its user. To do this, it monitors the set of auctions currently progressing. For each auction a_i, it observes the $n(i)$ highest bids. In other words, it observes the values of the bids which, if the auction terminated immediately,

[2] While this is possible, I make no claims that it actually happens.

would result in a successful purchase. Let these bids be labeled $\{b_1^i,..,b_{n(i)}^i\}$, where b_1^i is the highest bid in the auction, and $b_{n(i)}^i$ is the lowest bid which would currently succeed. We refer to these as the currently *active* bids. To represent the reservation price r, we assume that the seller places $n(i)$ bids of value $r-\delta$, where δ is the minimum bid increment.

Let us assume our agent is attempting to purchase 5 discount PCs of a given specification, and is willing to bid up to 150 for each. There are 3 auction houses, each running one auction to sell PCs of this specification. Auction a_1 is selling 4 PCs, auction a_2 is selling 3 PCs and auction a_3 is selling 2 PCs. Assume all auctions have a minimum bid increment of 5.

> Auction a_1 currently has the following bids registered (Underlined bids are held by our agent);
> 100 95 90 90 <u>80</u> 60
>
> As the auction is for 4 items, the agent observes the 4 highest bids, $\{b_1^1=100, b_2^1 = 95, b_3^1 = 90, b_4^1 = 90\}$
>
> Auction a_2 has the following bids registered;
> 95 85 <u>85</u> 80 70
>
> The agent observes the bids $\{b_1^2=95, b_2^2 = 85, b_3^2 = \underline{85}\}$
>
> Auction a_3 has the following bids:
> 100 <u>95</u> 95 80
>
> Our agent holds 2 active bids, and so needs to place bids to gain an additional 3.

Let L be the number of currently active bids that are held by our agent. (Initially, L will be zero.) To ensure it makes m purchases, it needs to make new bids that result in it having an additional $(m-L)$ active bids. As we shall see, this may require it to make more than $(m-L)$ bids, as it may need to outbid itself.

If the agent is to hold j active bids in auction a_i, it must place bids that beat the lowest j of the currently active bids. We define the *beatable-j list* for auction a_i to be the ordered set of these bids – namely bids $\{b^i_{n(i)-j+1},..,b^i_{n(i)}\}$. To beat the bids in this list, the agent must place j bids of value $b^i_{n(i)-j+1}+\delta$ where δ is the minimum bidding increment. The *incremental cost* to the agent of placing these bids, if successful, above the cost that it would have incurred in auction a_i previously, is $j*b^i_{n(i)-j+1}+\delta$ - {sum of previous bids in a_i}. The beatable-0 list of any auction is defined to be the empty set, and has incremental cost of zero. Obviously, an auction for q goods has no beatable-j lists for $j>q$.

> In the above example, the beatable-1 list of auction a_2 is {85}, with incremental cost 5 (as it already holds that bid). The beatable-2 list is {85, 85},

with incremental cost 95. Similarly the beatable-3 list is {95,85,85}, with incremental cost 195.

The agent now constructs potential *bid sets*. A bid set is a set of beatable-j lists that satisfies the following criteria;

1. The set contains exactly one beatable-j list from each auction.
2. The beatable-j lists contain, in total, exactly $(m-L)$ bids made by parties other than our agent.

In other words, each bid set represents one possible way of placing bids to ensure that our agent will gain an additional $(m-L)$ active bids, and therefore will hold exactly m active bids. We define the incremental cost of each of these bid sets to be the sum of the incremental costs of the beatable-j lists in it.

Returning to our example, our agent needs to find bid sets containing exactly 3 bids made by parties other than it. An example bid set satisfying the above criteria would be;
[{90}, {85, 85}, {100, 95}]
This set is made from the beatable-1 list of auction a_1, and the beatable-2 lists of auctions a_2 and a_3. Its incremental cost is 300.

The agent must generate the bid set with the lowest incremental cost. In addition, it must avoid generating bid sets that contain a bid equal to or greater than its valuation of the good, v. Various algorithms can be used to do this. The simplest is to generate all possible bid sets, filter out those containing bids greater than v, and select the one with lowest cost. However, this is clearly inefficient. We have adopted a depth first strategy through the space of possible bid sets, pruning areas of the search space which are higher cost than the best solution found so far. This strategy is presented on the following page in pseudocode as a function returning the cheapest bid set.

If there is more than one bid set with identically lowest cost, the agent chooses one arbitrarily. If no such bid sets exist, the agent relaxes condition 2 and finds the smallest i such that at least one bid set exists which contains $(m-L-i)$ bids made by parties other than the agent. Given this i, the agent chooses the bid set with the lowest incremental cost.

Having generated the bid set with the lowest cost, the agent places bids in each auction. For each beatable-j list $\{b^i_{n(i)-j+1},..,b^i_{n(i)}\}$ in the bid set, the agent places j bids of value $b^i_{n(i)-j+1}+\delta$ in the corresponding auction a_i.

In our example, the bid set with lowest cost is [{90, 90}, {85, 85}, {}]
This set has cost 285. The agent therefore places two bids of 95 in auction a_1 and two bids of 90 in auction a_2.

```
cheapestBidSet(N, MaxCost, AuctionList) {

/* Return the cheapest bid set to gain N active bids in
auctions in (a non empty) AuctionList. MaxCost sets an up-
per bound on the cost of the bid set, and is used for prun-
ing redundant branches of the search space. */
  /* Vars for the best solution so far, and its cost */
  BestSolutionSoFar := 'undef';
  BestCostSoFar := MaxCost+1;

  Auc1 := head(AuctionList);

  /* If there is only one auction, then return the
     beatable-N set for this auction, if it exists and is
     cheap enough. */
  IF length(AuctionList) = 1 {
   IF numberOfGoods(Auc1) ≥ N
    AND cost(beatable_jList(N,Auc1) =< MaxCost
    {RETURN beatable_jList(N,Auc1)}
   ELSE {RETURN 'undef'}}

  ELSE {
   /* Try all possible beatable-j lists from Auc1, one at
      a time. */
   FOR i = 0 to min(N, numberOfGoods(Auc1)) {

    /* If the beatable-i list for Auc1 is cheap enough,
  then recursively generate the cheapest bid set to buy N-i
  goods from the remaining auctions. */
    IF cost(beatable_jList(i,Auc1)) =< MaxCost {
      SubSolution :=
            cheapestBidSet(N-i,
              BestCostSoFar – cost(beatable_jList(i,Auc1)),
              tail(AuctionList)); }
      IF SubSolution <> 'undef' {
       TrialSolution :=
            union(SubSolution,beatable_jList(i,Auc1));
       /* Check if the new solution is cheaper
          than current best solution, and update. */
       IF cost(TrialSolution) < BestCostSoFar {
        BestSolutionSoFar := TrialSolution;
        BestCostSoFar := cost(TrialSolution); }
      }}
  }}
```

The agent continues to monitor the auction, and repeats its analysis if other parties place new bids. In this way, the agent ensures it maintains m active bids at the least possible cost to itself, unless doing so requires it to place bids above its valuation of the good. Providing all auctions terminate simultaneously, this will result in it buying the goods at the best price possible, given the competition in each auction.

4 Dealing with Different Auction Deadlines

Now, we consider the case where auctions terminate at different times. In such a situation, the algorithm above will not necessarily behave optimally. Imagine a situation where an auction starts every half-hour, and lasts for an hour. The agent would always monitor two auctions, one that is nearer closing than the other. Inevitably, bids will be higher in the auction that is nearing completion. Hence the agent would switch bidding to the newer auction, and withdraw from the auction about to close. If this continued, the agent would never make a purchase, but would simply switch bids to a new auction every half-hour.

The agent needs a mechanism for determining whether to remain in an auction which is about to close, even when there are other auctions with lower current bid prices. To do this, it must be able to make a trade-off in terms of expected value between the relative certainty of remaining in an auction about to close, against the risk of participating in a newer auction. The newer auction may result in a lower purchase, or may result in a far higher purchase price above the agent's valuation of the good. In this section, we propose a mechanism for doing this.

The mechanism we use combines simple learning with utility theory. The agent uses learning to build a model of the spread of valuations held by participants in different auction houses. Then, based on its beliefs about these valuations, it calculates the utility of likely participation in persisting auctions, and compares this with the certain outcome in the terminating auction. If the terminating auction has a higher utility, it remains a participant and makes the purchase. If the remaining auctions have higher expected utility, it withdraws from the terminating auction and continues participation elsewhere.

4.1 Learning Mechanism

The agent generates a model of the potential outcome of auctions by creating a model of each auction house. For a given auction house and a given type of good, it creates a belief function $B(x,q)$ representing the probability that x bidders value the good with a valuation greater than q in a given auction for that good. It builds up this function by monitoring auctions for the good conducted by the auction house. Various possible learning techniques can be used to generate this function. The exact choice will depend on the underlying dynamics of the demand for the good under consideration. We present three possibilities here.

(a) Static Demand

If the demand for the good is unchanging, a simple function which gives equal weight to evidence from each auction will suffice. It can be specified iteratively – The initial beliefs after one auction $B_1(x,q)$ are defined, and the beliefs after the t+1th auction $B_{t+1}(x,q)$ are defined in terms of the beliefs $B_t(x,q)$ held prior to the auction.

$B_1(x,q)$ = 1 if x or more bidders have placed a bid of q or greater in the first auction observed, 0 otherwise.

$B_{t+1}(x,q)$ = $((t\ B_t(x,q) + 1) / (t + 1)$ if x or more bidders have placed a bid of q or greater in the t+1th auction observed,
= $t\ B_t(x,q) / (t + 1)$ otherwise

(b) Drifting Demand

If the demand for the good changes slowly with time, it is necessary to reduce or eliminate the contribution of older auctions on the belief function. This can be done either by specifying a rolling window of time and discarding evidence from auctions earlier to this, or by using a time discount factor that reduces the weight given to older auctions.

(c) Semi-Static Demand

In a semi-static environment, the demand remains fixed for a period of time, and then suddenly alters to a new level (for example, because of the arrival of a new group of buyers because of a publicity campaign.) In such an environment, it is necessary for the learning algorithm to identify when such a change has occurred, and to discard evidence from auctions prior to the change. This can be done by observing predictions from the belief model, and how they compare with the actual outcomes. If they are radically different over several auctions, a reset should take place. Further details of this approach (in double-auction markets) are discussed by Vulkan and Preist[14].

Using this function, we can estimate the probability that a bid of a certain value will be successful in an auction by a given auction house. Consider an auction for n goods, in which our agent wishes to purchase one. The probability that a bid of q by our agent will be successful can be estimated to be $1 - B(n,q)$; i.e. 1 minus the probability that n other bidders are prepared to outbid our agent.

There is a flaw in this model, which must be taken into account if it is to be successful. Unlike a Vickrey or Dutch auction, an English auction reveals nothing about the valuations of successful bidders. In other words, if a bidder makes a successful bid of x, we cannot be sure how much higher they may have been willing to bid. To take account of this, it is necessary to add some kind of heuristic weighting to the belief function – we must increase the value of a successful bid by a certain amount, to reflect this possible willingness to bid higher. One possibility is to add a small random amount to each successful bid. In some domains, it may be possible to use econometric data to determine accurately the range that this should be drawn over, while in other domains it may be necessary to use a heuristic estimate.

4.2 Utility Analysis of Leaving an Auction

We now consider how this belief function can be used to compare the expected payoff of an auction that is about to terminate with the less certain outcome of other auctions that terminate later. For the sake of clarity and brevity, we present the technique assuming our agent wishes to purchase a single good.

The expected payoff from the terminating auction is simple to calculate. Assuming our agent is holding an active bid q, or is able to place one at the last moment, then the payoff will be $(v\text{-}q)$. If the agent is unable to place a bid because all active bids are beyond its valuation of the good, then payoff will be zero and the agent is forced to participate in other auctions.

> Again, we will introduce an example to illustrate the principles being presented. Assume our agent is purchasing one good from one of two auctions. Auction a_1 is nearing completion, while auction a_2 is continuing. Each auction is for 2 goods, and is run by separate auction houses. The active bids are as follows;
>
> Auction a_1: 130 125
> Auction a_2: 115 110
>
> Our agent values the good at 200, so could continue bidding in auction a_1. However, should it, or should it switch to the other auction where prices are lower?

The expected payoff of continuing to participate in the non-terminating auctions is more complex to calculate. To do this, we use the belief function to calculate the probability our agent will be able to make a purchase at various possible bid prices. Recall that, for a given bid price q, the probability our agent will make a successful bid in an auction run by a given auction house is $1\text{-}B(n,q)$, where n is the number of goods being sold. Similarly, the probability that our agent will be able to make a successful bid at a lower price, $q\text{-}1$, is $1\text{-}B(n,q\text{-}1)$. Hence, the probability that our agent will succeed with a bid of q and no lower is $B(n,q\text{-}1)\text{-}B(n,q)$. The utility of this outcome will be $(v\text{-}q)$. Hence, we can calculate the expected utility of participating in a given auction as;

$$\sum_{q=0}^{v} [B(n,q-1) - B(n,q)](v-q)$$

Of course, as the auction may already be in progress, it is necessary to take into account the current active bids in that auction. The general belief function $B(x,q)$ for the auction house is therefore adapted for this particular auction a_n to give $B(a_n,x,q)$. If the good being traded is a private value good, and hence all buyers have valuations independent of each other, this is defined as follows;

Let p be the value of the xth highest bid in auction a_n
Then $B(a_n,x,q) = B(x,q)/B(x,p)$ for all $q \geq p$
1 for all $q < p$

To return to our example, let us assume that our agent has built up the following belief function for the auction house running auction a_2;

q:	105	110	115	120	125	130	135	140
B(2,q):	1	0.8	0.7	0.6	0.6	0.5	0.3	0

As there are already bids of 110 and 115, this becomes;

q:	105	110	115	120	125	130	135	140
B(2,q):	1	1	0.875	0.75	0.75	0.625	0.375	0

As our agent has a valuation of 200, we can calculate the expected utility of this auction to be;

(0.125*85)+(0.125*80)+(0.125*70)+(0.25*65)+(0.375*55) = 66.25

Given an expected utility on the remaining auctions, the agent must decide whether to place higher bids in the auction that is about to terminate, or withdraw from it. If we assume that the agent is risk neutral, then it will be willing to bid up to a value where the actual utility of the terminating auction is the same as the highest expected utility among the remaining auctions. In other words, it is prepared to make a maximum bid b_{max} of;

$$b_{max} = v - \sum_{q=0}^{v} [B(n,q-1) - B(n,q)](v-q)$$

In our example, assuming that a_3 has a lower expected utility than a_2, our agent will be willing to bid up to (200-66.25) = 133.75 in auction a_1. Hence, it will place a bid of 130, and will withdraw if this is outbid. In this case, it will hope to make a better purchase in auction a_2.

In this way, the agent is able to make informed decisions about whether to continue bidding in an auction or to switch. If it is making multiple purchases, it may purchase some in the terminating auction, and choose to switch others to continuing auctions. Extensions of the algorithm to handle this case will be dealt with in a future paper.

5 Implementation

In this section, we describe the implementation of a distributed prototype system for simulating activity of agents that negotiate in multiple auctions, using the algorithms described above.

The architecture is based on Java Message Server (JMS), part of the Java 2 Platform Enterprise Edition (http://java.sun.com/products/jms/docs.html). JMS is a messaging infrastructure that supports both point-to-point (message queuing) and publish-subscribe styles of messaging. In the point-to-point paradigm, clients of the JMS framework know how to work with queues: find them, send messages to and read messages from them. In the publisher-subscriber paradigm, clients publish messages to and subscribe messages from topics. The choice of JMS allows the prototype system to be fully and truly distributed, with many traders running in parallel on different machines.

The distributed architecture has three types of actor; a single infrastructure provider, auction houses and trader agents. The infrastructure provider administers the messaging infrastructure, and providing a notification service when new actors enter. The auction house is responsible for managing auctions throughout their lifecycle, and providing data about previous auctions. Each trader is given a mailbox that is implemented through a JMS queue. The infrastructure provider communicates via an admin board, while auction houses provide one messaging board per auction. The boards are implemented as JMS topics.

Initially, a trader signs up with the infrastructure provider to obtain access to the admin board. It may then sign up with any auction house, which provide it with information on any auctions which have already started, and data about previously completed auctions. Whenever an auction house creates a new auction, it sets up a blank message board associated with it. The auction house then notifies the traders of the auction by passing time, number of goods for sale, minimum increment, de-committal penalty, etc. It does this by sending a message to the infrastructure provider, which places the information on the admin board.

A trader places bids by sending bid lists to auction houses. A bid list is a data structure containing a variable number of records, representing bids. Each bid specifies a value and the identifier of the auction that the bid is being placed in. Bid placement is completely asynchronous. The trader doesn't receive notification of acceptance by the auction house, but instead observes updates on the message board associated with the auction to determine if a bid is accepted.

The auction house periodically updates the traders subscribing to its messaging board by sending out information about the current status of an auction. The auction house chooses the frequency of the updates, balancing the traders' need for frequent updates and the available bandwidth. Ideally, an update would take place whenever a bid was accepted, but in practice it is more efficient to process all bids arriving within a certain (small) interval before sending out an update.

Implementing the Agent Algorithm

Initially, the agent monitors the auctions available, and selects those that are selling the good it is interested in, and which close prior to the deadline it has for purchasing the good. It signs up with the corresponding auction houses of these auctions. It then constructs two sets of auctions – those about to terminate, and those continuing. It calculates the expected utility of the non-terminating auctions, and uses the coordination algorithm to place bids up to this value in the terminating auctions. If it is unable

to place some bids, it will place them in the non-terminating auctions instead. It re-executes the coordination algorithm whenever other participants outbid some of its bids. However, it only need re-execute the utility analysis if data about the non-terminating auctions has changed (e.g. if a new auction has started.)

Because of the distributed nature of the system, the agent algorithm must base its computation on data that has not been confirmed as valid. The agent holds a local data structure that mirrors the messaging boards of all the auctions it takes part in. The local information is updated on receiving updates from the auction house, and also after the agent places a bid list. This local knowledge is what the agent uses to make his decisions about which best next bids to place. If the local knowledge is proved wrong, because an agent's bid was not accepted due to another higher bid arriving simultaneously, it is sufficient simply to run the algorithm again and place more bids as necessary.

6 Related Work

Research into automated negotiation has long been an important part of distributed AI and multi-agent systems. Initially it focused primarily on negotiation in collaborative problem solving, as a means towards improving coordination of multiple agents working together on a common task. Laasri, Lassri, Lander and Lesser [5] provide an overview of the pioneering work in this area. As electronic commerce became increasingly important, the work expanded to encompass situations with agents representing individuals or businesses with potentially conflicting interests. The contract net [6] provides an early architecture for the distribution of contracts and subcontracts to suppliers. It uses a form of distributed request-for-proposals. However, it does not discuss algorithms for determining what price to ask in a proposal. Jennings et.al. [7] use a more sophisticated negotiation protocol to allow the subcontracting of aspects of a business process to third parties. This is primarily treated as a one-to-one negotiation problem, and various heuristic algorithms for negotiation in this context are discussed in [8]. Vulkan and Jennings [9] recast the problem as a one-to-many negotiation, and provide an appropriate negotiation protocol to handle this. Other relevant work in one-to-one negotiation includes the game-theoretic approach of [10] and the logic-based argumentation approach of [11].

As much electronic commerce involves one-to-many or many-to-many negotiation, the work in the agent community has broadened to explore these cases too. The Michigan AuctionBot [12] provides an automated auction house for experimentation with bidding algorithms. The Spanish Fishmarket [13] provides a sophisticated platform and problem specifications for comparison of different bidding strategies in a Dutch auction, where a variety of lots are offered sequentially. The Kasbah system [14] featured agents involved in many-to-many negotiations to make purchases on behalf of their users. However, the algorithm used by the agents (a simple version of those in [8]) was more appropriate in one-to-one negotiation, and so gave rise to some counter-intuitive behaviours by the agents. [15] and [16] present adaptive agents able to effectively bid in many-to-many marketplaces. [17] demonstrates how these can be

used to produce a market mechanism with desirable properties. Park et.al. [18][19] present a stochastic-based algorithm for use in the University of Michigan Digital Library, another many-to-many market.

Gjerstad et. al. [20] use a belief-based modeling approach to generating appropriate bids in a double auction. Their work is close in spirit to ours, in that it combines belief-based learning of individual agents bidding strategies with utility analysis. However, it is applied to a single double auction marketplace, and does not allow agents to bid in a variety of auctions. Vulkan et.al. [21] use a more sophisticated learning mechanism that combines belief-based learning with reinforcement learning. Again, the context for this is a single double auction marketplace. Unlike Gjerstad's approach, this focuses on learning the distribution of the equilibrium price. Finally, the work of Garcia et.al. [22] is clearly relevant. They consider the development of bidding strategies in the context of the Spanish fishmarket tournament. Agents compete in a sequence of Dutch auctions, and use a combination of utility modeling and fuzzy heuristics to generate their bidding strategy. Their work focuses on Dutch rather than English auctions, and on a sequence of auctions run by a single auction house rather than parallel auctions run by multiple auction houses. However, the insights they have developed may be applicable in our domain also. We hope to investigate this further in the future.

7 Conclusions and Future Work

By interleaving the application of two algorithms of the form described above, our agent can effectively participate in multiple English auctions. It will use the coordination algorithm to place lowest possible bids across auctions. It will use the bid withdrawal algorithm to determine when it is worth bidding higher in an auction that is about to terminate, as opposed to transferring to other auctions where the active bids are currently lower.

In this paper, we have sketched out the structure of appropriate algorithms to do this. The specifics of these algorithms, particularly the bid withdrawal mechanism, may need refinement and specialisation to operate in specific market applications. Furthermore, the richness of the model may be increased. Specifically;

- A more sophisticated learning mechanism could be used. This may be generic, or could be tailored to the specific dynamics of a particular market.
- The algorithm presented assumes that the buyer is risk-neutral. This should be generalised to allow the agent to adopt other risk attitudes as appropriate.
- The algorithm does not take into account time discounting or deadlines. Again, this should be generalised.
- The algorithm assumes that the agent values all goods equally. This should be generalised to allow the agent to receive a demand curve from its user.
- The algorithm assumes that the beliefs about auctions are accurate. By measuring the deviation of actual auctions from the predictions, it would be possible to give a measure of confidence in the belief. This could be used to

moderate the agent's decision to switch auctions, taking into account the agent's attitude to risk.

- The algorithm only takes into account the expected payoff of existing auctions. It may be appropriate to also model the possibility that an auction house may bring new auctions into being, and the potential payoff of such auctions.
- Throughout this paper, we have assumed that all auctions are English in format. Research is required to generalise this to cover other forms of auction, such as Dutch, Vickrey and Double auctions.

We hope to address some of these issues in a future paper.

References

1. R. Kalakota and A. Whinston. Frontiers of Electronic Commerce.. Addison-Wesley 1996.
2. C. Shapiro and H. Varian. Information Rules – A Strategic Guide to the Network Economy. Harvard business school press, 1999.
3. K. Binmore Fun and Games.. D. Heath and Co. 1992.
4. Paul Klemperer, "Auction theory: a guide to the literature" J. of Econ. Surveys, Vol. 13, No. 3, July 1999, pp. 227-286
5. B.Laasri, H.Laasri, S. Lander and V. Lesser. A Generic Model for Intelligent Negotiating Agents. International Journal of Intelligent and Cooperative Information Systems, 1(2) 1992 pp291-317.
6. R.G. Smith . The contract net protocol: high-level communication and control in a distributed problem solver.. IEEE Trans. Comput., 29, 1104-1113, 1980.
7. N.R. Jennings, P. Faratin, M.J. Johnson, P.O. O'Brien and M.E. Wiegand. Using Intelligent Agents to Manage Business Processes. In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96), 345-360, April 1996
8. P. Faratin, C. Sierra and N. Jennings. Negotiation decision functions for autonomous agents. Robotics and Autonomous Systems 24(3-4),1998, pp159-182.
9. N.Vulkan and N.Jennings. Efficient Mechanisms for the Supply of Services in Multi-Agent Environments, Proceedings of the 1st International Conference on the Internet, Computing and Economics, ACM Press 1998.
10. J. Rosenschein and G. Zlotkin. Rules of Encounter. MIT Press, 1994.
11. S. Parsons, C.Sierra and N.Jennings. Agents that reason and negotiate by arguing.
12. P.Wurman, M.Wellman and W.Walsh. The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In Proc. Second Conference on Autonomous Agents, 1998.
13. J. Rodriquez-Aguilar, P. Noriega, C. Sierra and J.Padget. Fm96.5: A Java-based e,ectronic auction house. Proc. Second International Conference on the Practical Application if Intelligent Agents and Multi-Agent Systems, 1997, pp207-224.
14. A. Chavez, D.Dreilinger, R.Guttman and P.Maes. A Real-Life Experiment in creating an Agent Marketplace. Proc. Second International Conference on the Practical Application if Intelligent Agents and Multi-Agent Systems, 1997.

15. Cliff, D. and Bruten, J. Less than Human: Simple adaptive trading agents for CDA markets. Proceedings of the 1998 Symposium on Computation in Economics, Finance, and Engineering: Economic Systems
16. Preist, C. and van Tol, M. Adaptive Agents in a Persistent Shout Double Auction. Proceedings of the 1st International Conference on the Internet, Computing and Economics, ACM Press 1998.
17. C.Preist. Commodity Trading using an Agent-Based iterated Double Auction. Proc. Third Conference on Autonomous Agents, 1999.
18. S. Park, E.Durfee and W.Birmingham. Emergent Properties of a Market-Based Digital Library with Strategic Agents. Proc. International Conference on Multi Agent Systems, 1998.
19. S. Park, E.Durfee and W.Birmingham. An Adaptive Agent Bidding Strategy based on Stochastic Modelling. Proc. Third Conference on Autonomous Agents, 1999.
20. Gjerstad, S. and Dickhaut, J. Price formation in double auctions. Games and Economic Behaviour, 22(1), pp1-29, 1998.
21. N. Vulkan and C. Preist. Automated Trading in Agents-based Markets for Communication Bandwidth. Proc. UKMAS (1999.
22. P. Garcia, E.Giminez, L. Godo and J. Rodriguez-Aguilar. Possibilistic-based design of bidding strategies in Electronic Auctions. Proc. 13th Biennial European Conference on Artificial Intelligence, 1998.

Agent Strategies on DPB Auction Tournaments

Javier Béjar and Ulises Cortés

Departament de Llenguatges i Sistemes Informàtics.
Universitat Politècnica de Catalunya.
c/ Jordi Girona 1-3. 08034 Barcelona, Spain.
{bejar,ia}@lsi.upc.es
Phone: 34 3 4015653 Fax: 34 3 4017014

Abstract. In this work we present the experience of an electronic tournament between trading agents developed at the Technical University of Catalonia (UPC) as a course work for an artificial intelligence applications course. Using the *Fishmarket* platform as implementation basis, fourteen different agents were developed and confronted in a set of Downward Bidding Protocol (DBP) auctions in order to measure the performance of their strategies.
We present the different groups of strategies of the agents, their architectures and their relationship to the success of the agents.

Keywords: Autonomous Agents, Multiagent Systems, Electronic Institutions, e-Auctions

1 Introduction

This article studies the strategies used by the trading agents, participating on an electronic tournament, developed by our students as course work for the graduate course Applications of Artificial Intelligence, the autumn term of 1999, at the Barcelona School of Informatics [5] at the Technical University of Catalonia (UPC).

The tournament was developed using the *Fishmarket* platform (FM1.00)[1], that provides a development environment for autonomous trading agents participating in e-Auctions. During the past 5 years we have been playing these tournaments in Barcelona [6] and the last year we also had it in Lausanne [3]. In those experiences we have been able to collect all the participating agents, having 10 to 14 different agents each year. Each group of students provided us the original code and, the explanation of their strategies. These experiences served for tuning the development of the *Fishmarket* platform. In this paper we will survey the dominant strategies among UPC agents.

In the tournament that we will describe here (see §3), we had a total of fourteen agents and those were randomly divided for competition in three groups.

[1] The *Fishmarket* is a software platform developed at the Institut d'Investigació en Intel.ligència Artificial, (IIIA-CSIC).

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 155–172, 2001.

The best two agents of each group were confronted in a final tournament to find the agent with best strategies.

This article is organized as follows. First, in section 2, we will briefly describe the development platform and the characteristics of the kind of auctions provided by the *Fishmarket* . Section 3 will be devoted to the characteristics of the tournament, and the description of the different conditions of the auctions used for the competition. Section 4 presents the different groups of strategies used by the agents and their characteristics. Section 5 will present the results obtained by the different agents and the success of the different strategies. In the last section we conclude the work by examining the results of the competition and discussing the students' projects, the pedagogical virtues of this exercise and the scope of future tournaments.

2 The Fishmarket Platform

The *Fishmarket* [10] platform is an electronic auction house that uses the traditional fish market metaphor as described in [11]. It follows the different scenes that a traditional fish auction involves, as registration of the sellers goods in the auction house by a seller manager, register of the buyers by an admitter agent, bid for goods directed by a seller manager, etc.

This platform defines an electronic institution that is managed by a central agent. All the interactions between the other agents and the institution are managed by this manager agent. Each agent is given an identification and an interaction protocol that defines which ilocutions can be used and its meaning.

This platform has been developed in JAVA. It provides a set of classes to implement JAVA agents, that provides all the data structures that define the auction house and the state variables needed for the bidding. To implement bidding agents it is only necessary to implement the strategy, the processing of the ilocutions between seller manager and the agents and the updating of the auction variables are done by the provided classes. The agents had only one ilocutions allowed to send to the seller manager, the intention to bid to the actual price.

We will center our study on the central scenario of the auction, the auction itself, in which buyers, agents in this case, bid for boxes of fish that are presented by an auctioneer who calls prices. So, we will continue describing the elements and the parameters that define the auction: the rounds of an auction, the bidding protocol (in our case the DBP[2] protocol) and the description of the goods.

2.1 The Auction

Auction rounds aim at identifying and characterizing the ontological elements involved in each bidding round.

[2] DBP stands for *downward-bidding protocol* (or "Dutch" protocol)

Auction Round

For a given round r of auction i we define the *auction round* $\mathcal{A}_r^i$ as the 5-tuple

$$\mathcal{A}_r^i = \langle \mathcal{B}_r^i, g_r^i, \mathcal{C}_r^i, d_r^i, \mathcal{I}_r^i \rangle$$

where

- $\mathcal{B}_r^i$ is a non-empty, finite set of buyers' identifiers such that $B_r^i \subseteq \mathcal{B}$, the set of all participating buyers.
- $g_r^i = \langle \iota, \tau, p_\alpha, p_{rsv}, s_j, p_\omega, b_k \rangle$ is a good where ι stands for the good identifier, τ stands for the type of good, $p_\alpha \in \mathbb{N}$ stands for the starting price, $p_{rsv} \in \mathbb{N}$ stands for the reserve price, $s_j \in \mathcal{S}$—the set of all participating sellers—is the seller of the good, $p_\omega \in \mathbb{N}$ stands for the sale price, and $b_k \in \mathcal{B}_r^i$ is the buyer of the good. Notice that g_r^i is precisely the good to be auctioned during round r of auction i, and that p_ω and b_k might take on empty values when the round is over, denoting that the good has been withdrawn.
- $\mathcal{C}_r^i : \mathcal{B}_r^i \longrightarrow \mathbb{R}$ assigns to each buyer in $\mathcal{B}_r^i$ his available credit during round r of auction i.
- d_r^i stands for an instance of a bidding protocol dynamics descriptor.
- $\mathcal{I}_r^i$ is a set of information functions available for the agents during the round. It contains those functions labeling some of the events occurring during the round. Thus, the contents of this set will depend on the bidding protocol governing each round. For instance, for the downward bidding protocol, functions for labeling offers, sales, fines, expulsions, collisions, and withdrawals must be provided within this subset. For example, the auction catalogue could be included as an element of this set.

Fishmarket lets the tournament designer decide on the degree of transparency to be attached to *auction rounds* . In other words, the designer will have to decide what information about *auction rounds* is to be conveyed to the contenders, whether these should be informed about the participating buyers, and the subset of the set of information functions to be transmitted.

Auction is defined from the definition above.

Auction

We define an auction $\mathcal{A}^i$ as a sequence of *Auction rounds*

$$\mathcal{A}^i = [\mathcal{A}_1^i, \ldots, \mathcal{A}_{r_i}^i]$$

2.2 The Bidding Protocol

When auctioning a good, one could choose among a wide range of bidding protocols (DBP,UBP[3], etc.). Each of these protocols can be characterized by a set of parameters that we refer to as *bidding protocol dynamics descriptors*, so that

[3] UBP stands for *upward-bidding protocol* (or "English" protocol)

different instantiations of such descriptors lead to different behaviors of their corresponding bidding protocols. As a particular case, we will concentrate on the downward bidding protocol (DBP) since it is the one utilized in the *Fishmarket* tournaments.

[**Step 1**] The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter.

[**Step 2**] With a chosen good g, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, (p_α) previously fixed by the sellers' admitter, as long as these price quotations are above a *reserve price* (p_{rsv}) previously defined by the seller.

[**Step 3**] For each price called by the auctioneer, several situations might arise during the open round:

Multiple bids Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price. Nevertheless, the auctioneer tracks whether a given number of successive collisions (Σ_{coll}) is reached, in order to avoid an infinite collision loop. This loop is broken by randomly selecting one buyer out of the set of colliding bidders.

One bid Only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Whenever there is an unsupported bid the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is punished with a fine, and he is expelled out from the auction room unless such fine is paid off.

No bids No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price which is obtained by decreasing the current price according to the price step. If the reserve price is reached, the auctioneer declares the good *withdrawn* and closes the round.

[**Step 4**] The first three steps repeat until there are no more goods left.

Six parameters that control the dynamics of the bidding process are implicit in this protocol definition. We shall enumerate them now, and require that they become instantiated by the tournament designer as part of a tournament definition.

DBP Dynamics Descriptor

We define a Downward Bidding Protocol Dynamics Descriptor $\mathcal{D}_{DBP}$ as the 6-tuple $\langle \Delta_{price}, \Delta_{offers}, \Delta_{rounds}, \Sigma_{coll}, \Pi_{sanction}, \Pi_{rebid} \rangle$ such that

- $\Delta_{price} \in \mathbb{N}$ (price step). Decrement of price between two consecutive quotations uttered by the auctioneer.
- $\Delta_{offers} \in \mathbb{N}$ (time between offers). Delay between consecutive price quotations.

- $\Delta_{rounds} \in \mathbb{N}$ (time between rounds). Delay between consecutive rounds belonging to the same auction.
- $\Sigma_{coll} \in \mathbb{N}$ (maximum number of successive collisions). This parameter prevents the algorithm from entering an infinite loop as explained above.
- $\Pi_{sanction} \in \mathbb{R}$ (sanction factor). This coefficient is utilized by the buyers' manager to calculate the amount of the fine to be imposed on buyers submitting unsupported bids.
- $\Pi_{rebid} \in \mathbb{R}$ (price increment). This value determines how the new offer is calculated by the auctioneer from the current offer when either a collision, or an unsupported bid occur.

Note that the identified parameters impose significant constraints on the trading environment. For instance, Δ_{offers} and Δ_{rounds} affect the agents' time-boundedness, and consequently the degree of situatedness viable for bidding strategies.

2.3 The Goods

The goods to be auctioned in the *Fishmarket* are boxes of different kind of fishes. Each agent receives the description of the goods to be auctioned at the start of the auction. Each good is described by the following set of attributes:

- The name of the good
- The name of the seller
- The starting price, the initial price to be used by the bidding protocol.
- The reserve price, the lower price that the seller is going to accept.
- The resale price, the price that the buyer could sell the good that has acquired.

All but the reserve price are known by the buyers.

3 The Autumn Tournament at UPC

As said before, there were a total of 14 agents that were divided in three groups (5, 5 and 4 agents). The distribution of the groups was randomly chosen. The agents competed first in a tentative tournament in order to adjust their strategies previously to the definitive tournament. The agents were also allowed to compete in private tournaments organized by their owners. We do not keep records from the private tournaments but our perception of them is very positive. In those tournaments the co-operative teams –those who participate the most– got as reward a better information about other participant's strategies

An interesting variation of those private tournaments are the *single confrontations* where two agents played against each other. Many times this has been used to confront the agent against himself of against a radical variation of itself to test extreme policies. This provoked in some *ecological evolutions* of some agents' bidding policies [4].

Each agent had an initial credit of 20.000 Euros and this was a known fact for all participants. Although this unrealistic constraint this gives clues about the agents' spending policies.

Three different scenarios were used in order to test the performance of the agents:

- **Short Auctions:** Auctions with a limit of 10 goods and a total initial price in the range of 20.000 Euros to 35.000 Euros
- **Medium Auctions:** Auctions with a limit of 20 goods and a total initial price in the range of 50.000 Euros to 80.000 Euros
- **Large Auctions:** Auctions with a limit of 30 goods and a total initial price in the range of 90.000 Euros to 120.000 Euros

The objective of these models of auction was to compare the agents' strategies performance within very different scenarios. The general characteristics of the market where: A lot of money on display for a few goods (short auctions), an equilibrium among global market amount of money and the initial global price of the goods (medium auctions), and not enough global money to buy all the goods to its initial price (large auctions).

These different kinds of auctions were meant to force the agents to have to face very different scenarios that need different kind of approaches in order to obtain a good performance. The agents also have to to *adapt* their responses to be tolerant to different kinds of strategies that can be expected from the rival agents.

The set of boxes of goods to be auctioned were randomly generated from a set of eight different goods with a variable range of starting prices (see table 1). The prices of the goods were chosen in order to present to the agents a diversity of purchase opportunities. This diversity ranges from expensive goods with a good expected benefit and an expected high reserve price and cheap goods with an initial low benefit, but with an expected low reserve price.

The calculation of the price of a good is as follow:

- The starting price is the base price of the good plus a fixed percent of the base price multiplied by a random factor between 0 and 1.
- The reserve price is the starting price of the good minus a fixed percent of the starting price multiplied by a random factor between 0 and 1.
- The resale price is the starting price of the good plus a fixed percent of the starting price multiplied by a random factor between 0 and 1.

In table 1 the different goods auctioned and the factors used to calculate the prices are shown.

Each group of agents participated on 5 round of auctions of the referenced three types, using DBP protocol with a time between offers of 250ms (Δ_{offers}), a price step of 10 euros (Δ_{price}), a time between rounds of 2 seconds (Δ_{rounds}), a maximum number of 3 collisions (Σ_{coll}), a sanction fine of 25% ($\Pi_{sanction}$), and a price increment after collisions of 25% (Π_{rebid}). The auctions were randomly

Table 1. Values for the calculation of the prices of the goods

Good	Base Price	Initial Price	Reserve Price	Resale Price
Squid	2000	.3	.3	.25
Baby-fish	1000	.2	.2	.3
Hake	2500	.2	.15	.2
Sardine	2200	.2	.25	.2
Monkfish	3000	.3	.1	.5
King Prawn	5000	.3	.2	.5
Codfish	3500	.25	.15	.35
Gilthead	4000	.25	.2	.4

generated and were the same for the three groups of agents. The objective of each agent was to maximize its benefit, buying at least one good.

For more detailed information, we address the reader to the tournament web page[4].

4 The Agents

4.1 Their Architectures

The fourteen agents can be, generally speaking, characterized in three types depending of its architecture:

- Rule based (8 agents)
- Automata based (3 agents)
- Multi-Agent Systems (3 agents)

The Rule based agents were the simplest. In those agents a set of rules controlled the parameters that determined the price of the bid. The different conditions of the state of the auction were used to tune different agent's state variables that determine its interest on a given good. The complexity of these agents is variable and goes from a few generic rules that allow to tune some parameters, to too many particular rules, one for each particular scenario that can arise during the auction. Obviously, there exists a correlation between the different situations characterized by the rules and the agent's success.

The Automata based agents were defined as a set of *behaviors* (states) that determines agent's actions. Each *behavior* has an effect on the value of the parameters used to calculate the bidding price. The transition from one state to another is usually determined by the agent's previous performance, the auction state variables and the interest on the auctioned good. Frequently, these agents have a *panic state* that is fired if the agent has not bought at least one good

[4] http://www.iiia.csic.es/Projects/fishmarket/

and it provokes a *very aggressive* behavior in order to compete for any available good.

The Multi-Agent systems were the most complex. In this tournament three of the competing agents were built as a pool of subagents that individually examine the state of the auction. Each subagent gives a price to bid. The prediction of each subagent is evaluated. The subagents are demoted or promoted if their bid wins or loses the actual bidding. The predictions of the subagent that performs the best are preferred.

The subagents used were very different among them, some were rule based (few rules) others only calculated a function based on the auction state.

Curiously enough, one of the Multi-Agent System used as subagents the example agents provided by the *Fishmarket* platform. These example agents use different functions depending of the market money, resale price and market benefits as basic price bid. These functions determined the strategy, for example, with a sinusoidal function the strategy using the number of rounds as semi-period, starts with low bids, increment the bids until the middle-round and decrease the bids until the end of the auction. Among the used functions there were also the sigmoid function, the logarithmic function, and the arctangent function. Those functions had free parameters that were experimentally determined. Unfortunately, this multi-agent was the worst among the competing agents.

4.2 The Strategies

Very different strategies were developed for the agents ranging from the most elaborated, trying to introduce the modelation of the competitors and all the available information from the auction, to very simple, just using the prices of the goods and the global market money. Nevertheless, the complexity of the strategy of an agent was not always correlated with its success.

Almost all agents were parameterized by the size of the auctions, changing radically its behavior in each scenario. Some agents were, in fact only three agents, implemented with a different strategy for each scenario. In the short auctions the problem was approached in most of the cases by bidding almost to the starting price not allowing competitors to bid first. This *eager* behavior does not pay on the long run but proves to be effective in very short auctions, where time is too short to allow some kind of reaction or retaliation.

Some agents gave priority to the long auctions, waiting to the end of the auction to acquire the goods, when the competitors have spent almost all their money, expecting to obtain a better price. This strategy can be thought as *wait-and-see*. Due to that, the long auctions allow to obtain better benefits, because almost all the credit can be spent. The use of this strategy make the difference in performance among agents.

From all the available information in the auction, the most frequent state information used by the strategies of the agents was:

- The benefit of the other agents (global benefit or that of the best agents)
- The own actual benefit

- The own last benefit
- The global market money
- The competitors' remaining credit
- The starting or resale price
- The number of remaining rounds

All agents used some of these values to calibrate their parameters in order to approximate the bidding price. The exact implementation has shown a lot of variations, usually depending on values heuristically obtained from the private tournaments.

The rule based agents used these state values to determine the different scenarios. The automata based agents used these values to change from one state to another. The multi-agent systems used the values from different perspectives on each subagent and try to combine the available information.

Almost half of the agents explicitly used the behavior of their competitors as an additional parameter. In doing this there was also a lot of variability. Some observed all the agents, others just the one or two best. The effect of this parameter was to decrease the own target benefit to allow bids at a slightly higher price than the best competitor in order to overcome it.

Some of the agents tried to assign an initial evaluation of the *interest* of each good of the auction, in order to plan the bids beforehand. This seems *a priori* a good strategy, but none of the agents that used it were among the best agents. Probably the reasons for these bad results were due to: (a) the random distribution of the goods, (b) the length of the auctions and (c) the number of competitors. The sum of all these factors did not allow agents to build reliable plans in advance.

The agent's interest on a good could change dynamically with the market conditions, the competitors behavior, the own performance and the length of the auction. This is also a reason because an initial plan has to be dynamically changed during the auction.

The objective of all the agents was to achieve the greater benefit. This can be achieved also by buying more goods with medium benefit, and not only by buying the best goods. In the best goods there are a bigger competition.

5 Evaluation of the Strategies

The evaluation of the agents' strategies was measured by the total benefit obtained after all the auctions were performed. Due to that the auctions were independent, and none of the agents decide to keep memory of previous tournaments, there was no parameter in order to promote learning or adaptation between auctions.

The first temptative tournament allowed to define better strategies for the definitive tournament. Some of the competitors changed radically their strategy in between the two phases, and some that performed well in the first tournament were outperformed by other agents.

Table 2. Results of the final tournament

Name	Money Spent	Total Benefit	Benefit ratio	Goods Bought	Mean benefit per good
FishdelCastro	199028	52743	1.18	63	837.19
Boqueron	250189	51342	1.17	81	633.85
duHast	199163	46075	1.15	58	794.40
EncaraBelluga	235290	41970	1.14	72	582.92
superagente86	142358	41107	1.14	38	1081.76
tnt	153260	40914	1.14	46	889.43

The private tournaments allowed to refine and to determine heuristic parameters to improve their strategies. Almost all the agents competed in those private tournaments, and all the participant agents faced many of the different strategies implemented.

The groups for the second tournament were maintained, so each group knew beforehand who would be their competitor. They also knew that the two better agents from each group would qualify for an additional competition.

From the three groups the two agents with the greater benefit were chosen for an additional tournament. The luck of the winners was differently distributed, the fist group had two well performing agents and three agents with poor strategies, the second group had three good agents and two poor agents, and the third group was the most competitive, all four agents were very good, the second place was almost a tie between three agents.

From the analysis of the agents it can be seen also that the time of response had a big influence on the success. The faster the calculation of the bid, the higher benefit they got. So, more complicated architectures were not very successful.

Proportionally, all architectures were successful, four rule based, one automata based and one multi-agent based qualified for the last tournament.

The winners of the first group were the agents named **duHast**, and **EncaraBelluga**, the winners of the second group were the agents named **tnt**, and **superagente86**, the winners of the third group were the agents named **boqueron**, and **FishdelCastro**.

5.1 Participants in the Final Tournament

In the final tournament the number of goods was increased to maintain the proportion of 2 goods per agent for the short auctions, 4 goods per agent for the medium auctions and 6 goods per agent for the large auctions. The six agents that competed in the final tournament had the following characteristics:

FishdelCastro: It is a rule based agent. The agent follows the benefits of the two best competitors and combine this information with its own evaluation of the auction. The rules identify the actual scenario and tune the values of a set of parameters that determines the benefit to be pursued in the current round.

Boqueron: It is a rule based agent, it chooses between different heuristically calculated parameters using the number of rounds and the number of competitors. The initial bid is calculated from the starting price adjusted by the chosen parameters. The bid is corrected by a parameter calculated from the agent performance. If the auction is long the expected benefit is increased with the number of rounds in order to do lower bids at the end of the auction.

DuHast: It is an automata based agent. It has a four dimensional matrix indexed by a discretization of four variables that describe the agent state, the market state, the length of the auction and the quality of the good. The market state and the agent state are calculated from the benefit of the agent and their competitors and from the market money. The quality of the goods are calculated from the ratio between the starting price and the resale price. The discretization is based on a fixed set of thresholds. Each position of the matrix has a value that is used to calculate the bidding price. The value obtained from the matrix is corrected by an additional parameter that depends on the performance of the agent.

EncaraBelluga: It is a rule based agent that characterizes the auction using the state variables, calculating a base bid price. If the auction is short, it bids at the starting price if the difference between the starting price and the resale price is greater than a fixed quantity. It estimate the reserve price. It tries to modelize the benefit pursued by the competitors and bids slightly higher.

superagente86: It is a rule based agent. The value of the parameters are different depending if the auction is long or not. The bidding price is approximated from the market money. Less money implies to pursue greater benefit. An additional parameter allow the shift between a normal state and a *panic* state (no good bought yet), this parameter reduces the benefit to allow bidding to a higher price.

tnt: It is a multi-agent system. It uses five different strategies to evaluate the current state, each agent returns a value that measures one of five different circumstances: the good is a bargain, panic, our benefit is good, wait for a while and be more aggressive. The evaluation is summed up to a parameter that determines the bidding price. Each subagent uses a set of rules to determine how near is the actual moment of the auction to its target scenario.

The total results of the six better agents are shown in table 2. The values of the table are the money spent by the agents, the benefit obtained calculated as the difference between the price payed and the resale price, the benefit ratio calculated as the quotient between the initial credit plus the benefit and the initial credit, the number of goods acquired and the mean benefit per good.

In the figure 1 it is shown the total benefit of the agents for each group of auctions. It can be seen that the different parameters used by some agents for each group of auction yields a great variability in their performance. The regularity of the behavior of the agents differentiate between more general strategies from length auction specific strategies.

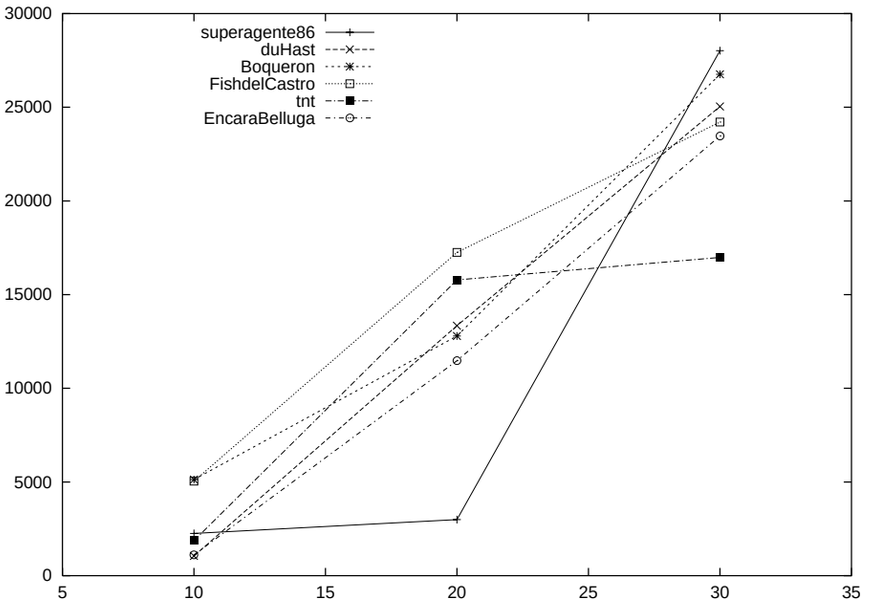


Fig. 1. Total benefit of the agents for each group of auctions

The more general the strategy the better performs the agent in each kind of auction and, so, the better global results are obtained. As can be seen in figure 1, there are two agents that only performed well in some kind of auctions, so, their global results were lower.

In the figure 2 it is shown the total spent money of the agents for each group of auctions. It demonstrates that to spent a lot of money in order to obtain better results it is not a good strategy. This strategy in not good even when there are a lot of money to spent as in the 10 goods auctions. Comparing the figures 1 and 2 can be seen that in the first kind of auctions the first and second agents obtained almost the same benefit, but the difference of spent money between them is more than twice.

From the total results it can be seen that the final benefits were not very different. But the target goods of the agents were different. For example, **superagente86** had preference for goods with better ratio between starting price and resale price and takes advantage of long auctions. This is the explanation for its lower number of goods but grater benefit per good.

Others, like **encarabelluga** or **Boqueron** had preference for goods with a lower ratio between starting price and resale price, but were more successful against its competitors winning a greater proportion of bids, but spending more money. This two agents were very aggressive in the short auctions, pursuing to buy the greater quantity of goods in spite of the high price they had to pay. In

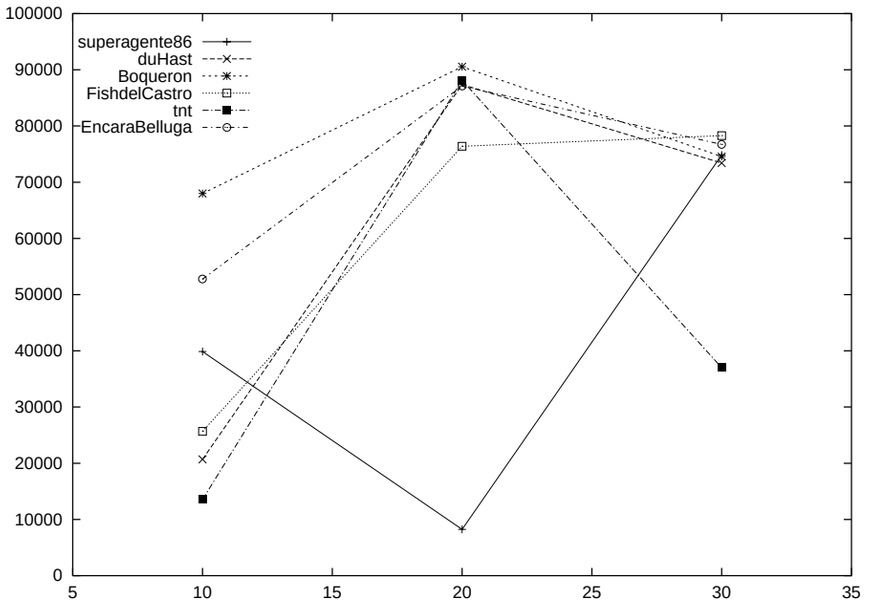


Fig. 2. Total spent money of the agents for each group of auctions

this kind of auctions, apparently, this seems a good policy because the available money is enough to buy almost all the goods in the auction. Evidently, this is not a very realistic scenario and in general this kind of strategy is not a good one.

The winner, **FishdelCastro**, had not a defined preference for a specific kind of goods, performing well against all their competitors and bidding for the better goods depending the circumstances. Its generality helped to perform well in the different scenarios.

The success of a general strategy against strategies focused on some special characteristics of the auction proofs that is better to implement adaptable agents able to analyze the auction information than to rely on special scenarios that sometimes can give a good benefit.

As a hint of how competition can allow to obtain better results in this kind of domains, it is curious that the two agents from the more competitive group were the two best.

5.2 The Influence of the Bidding Time

The tournament was designed under very restrictive time bounds in order to give preference to smart, small and fast agents. This could penalize more complex architectures. The hypothesis is that more complex strategies obtain worst

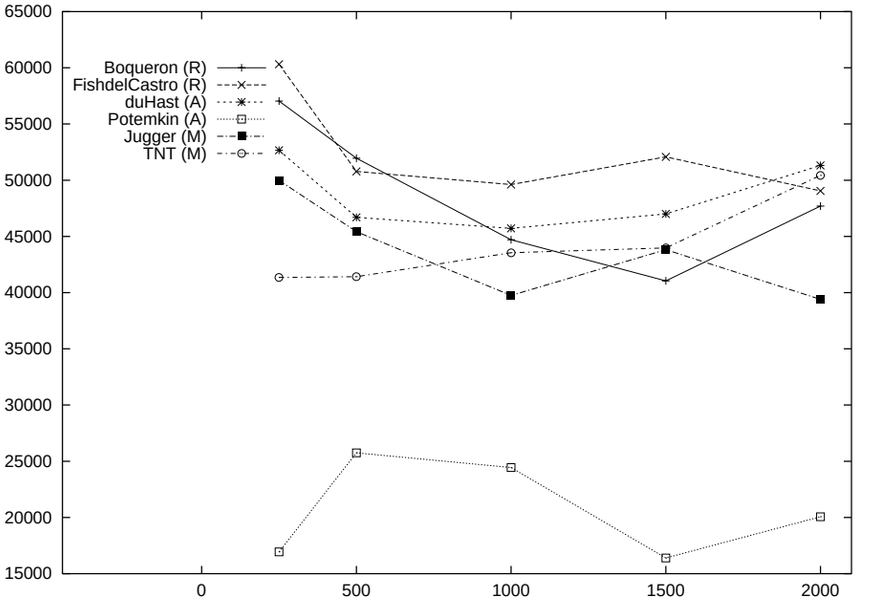


Fig. 3. Total benefit of the agents

results not because they can not model the auction, but because they had no time to end their deliberations before the smaller agents.

In order to test this hypothesis, another tournament was designed modifying the time between offers (Δ_{offers}). This amount of time is the most restrictive because the slower agents have very little time in order to decide if the current price is acceptable or not and then put forward their bids on time. In this tournament auctions were restricted to be medium and large size auctions, this is because the short auctions do not allow a true deliberative behavior [1] and the only practical strategy in that setting is compulsive bidding.

The agents confronted were the two best of each kind of architectures. It is supposed that the rule based agents are faster than the automata based agents and this two architectures are in turn faster than the multi-agents.

The chosen agents were the following:

Rule based agents: FishdelCastro and Boqueron, the two best agents of the tournament.

Automata based agents: duHast and Potemkin, were the third and the eleventh agents of the tournament.

Multi-Agents: tnt and jugger, were the fourth and the ninth agents of the tournament.

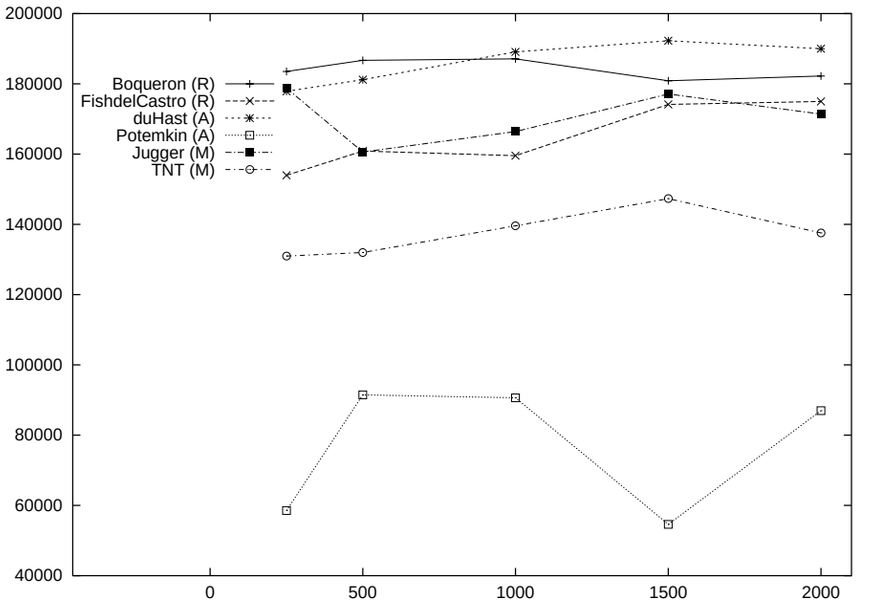


Fig. 4. Total spent money of the agents

In the figures 3 and 4 the total benefit and the spent money of the agents are presented. The time between consecutive offers are 250 ms, 500 ms, 1000 ms, 1500 ms and 2000 ms, respectively.

From these results, it seems like that some of the most complex agents take benefit from the existing extra time between offers. The benefit of one automata agent (duHast) and one multiagent (tnt) increases until finally outperform by a small difference all the rule based agents, but only when the deliberation time is eight times bigger than the original amount of time of the original tournament.

The number of collisions (simultaneous bids), as shown in figure 5, is increased. This is the reason because the agents total benefit decreases and the money spent on the market increases. Agents have to pay more for the goods in order to outperform their competitors. The agents ability to recalculate their bids after a collision is now a characteristic that becomes very important.

A conclusion drafted from this tournament is that given enough time any kind of implementation can be successful in this kind of auctions, but does not seems reasonable to use a complex architecture if a simple but effective agent can be implemented. This confirm some results already obtained at the EPFL Tournament [3].

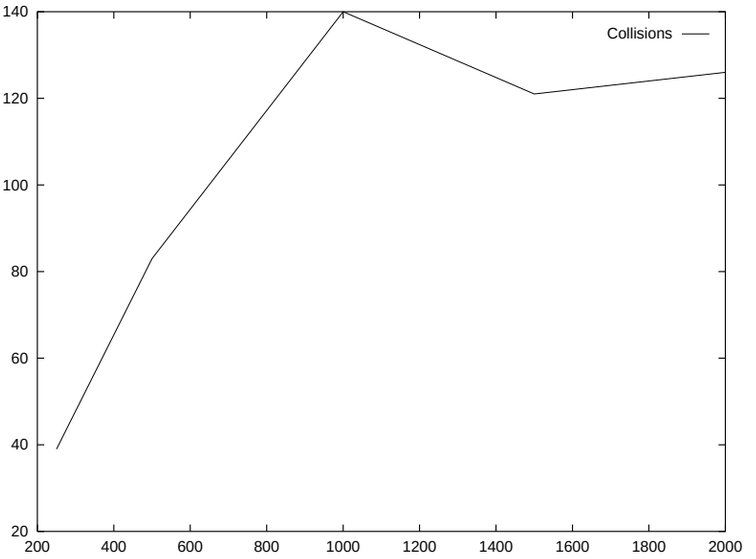


Fig. 5. Number of simultaneous bids for increasing time between offers

6 Conclusions

This competition turned out to be highly levelled and very interesting with respect to the variety of strategies developed by the contendants. Some conclusions can be drawn from the tournaments. The specific architecture of the agent is of a little importance for its success given enough time for deliberation. The strategy and background knowledge about the contest are more valuable and enable agents to accomplish their goals in enviroments with bounded resources. In this framework the main constraint is the response time. The calculations made in order to take a decision have to be done inside the time assigned. Without the time bound more deliberative strategies seem to work slightly better.

An outstanding feature of some agents in this tournaments was their attempt to model their rivals [3] and use this information to modify the own behavior.

The strategies that tried to plan beforehand it participation in an auction seem not to be very successful due to the very nature of the process. Three agents choose this strategy and were among the five worst participants. The dynamic nature of this kind of auction does not allow to plan ahead, in complex environments as *Fishmarket* plans may grow very large, and the decisions must be taken very fast with the information obtained at the present moment, taking in account only the immediate future and the agents' targets. Adaptability is a key feature in this environment.

From what has been observed there are different ways to perform well:

1. To direct the bidding towards a specific kind of good is as successful as to distribute the bidding equally to all kind of goods. The interest of a good is relative, not absolute.
2. To use specific knowledge about extraordinary circumstances in order to take advantage from it, is sometimes successful. For example, to wait-and-see on long auctions or to calculate the remaining credit of the competitors, allows to raise benefits or to be able to recognize *bargains*. But, it seems more interesting to try to follow the direct competitors' performance. For example, creating models of the best agents and trying to bid to a price slightly higher.
3. To recognize different scenarios in an auction helps agents to modify their targets and behavior.

Acknowledgments. We want to thank the IIIA (CSIC) for allow us to use the *Fishmarket* platform for the tournaments. Also we want to acknowledge to all the competitors of this autumn tournament, all did a great job.

This work is partially supported by UPC grup precompetitiu PR99-09. U. Cortés is partially supported by the European Union project A-TEAM IST-1999-176101. The views in this paper are not necessarily those of A-TEAM consortium.

References

1. C. Castelfranchi, F. Dignum, C. M. Jonker, and J. Treur. *Deliberative Normative Agents: Principles and Architecture*, volume 1757 of *LNAI*, pages 364–378. Springer-Verlag, 2000.
2. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, 1996.
3. U. Cortés and J. A. Rodríguez-Aguilar. Trading agents in auction-based tournaments. *INFORMATIQUE*, 1(1):39–50, 2000.
4. J. P. Delahaye, P. Mathieu, and B. Beaufils. *The Iterated Lift Dilemma*, chapter 10, pages 202–223. Volume 1 of Müller and Dieng [8], 2000.
5. Facultat d'Informàtica de Barcelona. http://www.fib.upc.es.
6. The FishMarket Project. http://www.iiia.csic.es/Projects/fishmarket.
7. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1995.
8. H. J. Müller and R. Dieng, editors. *Computational conflicts: Conflict modeling for distributed Intelligent Systems.* Springer-Verlag, 2000.
9. P. Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor.* Number 8 in IIIA Monograph Series. Institut d'Investigació en Intel.ligència Artificial (IIIA), 1997. PhD Thesis.
10. J. A. Rodríguez-Aguilar, F. J. Martín, F. J. Giménez, and D. Gutiérrez. Fm0.9beta users guide. Technical report, Institut d'Investigació en Intel.ligència Artificial. Technical Report, IIIA-RR98-32, 1998.

11. J. A. Rodríguez-Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. Competitive scenarios for heterogeneous trading agents. In *Proceedings of the Second International Conference on Autonomous Agents (AGENTS'98)*, pages 293–300, 1998.
12. J. A. Rodríguez-Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 11(1):5–19, 1998.
13. J. A. Rodríguez-Aguilar, P. Noriega, C. Sierra, and J. Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pages 207–224, 1997.
14. S. Sen, editor. *Negotiation: Settling Conflicts and Identifying Opportunities*, number WS-99-12 in AAAI Workshop. American Association for Artificial Intelligence, AAAI Press, 1999.

To Bid or Not To Bid
Agent Strategies in Electronic Auction Games

Javier Béjar[1] and Juan A. Rodríguez-Aguilar[2]

[1] Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
c/ Jordi Girona 1-3. 08034 Barcelona, Spain
Phone: 34 3 4015653 Fax: 34 3 4017014
bejar@lsi.upc.es
[2] Institut d'Investigació en Intel·ligència Artificial
Consejo Superior de Investigaciones Científicas
08193 Bellaterra, Spain
jar@iiia.csic.es

Abstract. This paper presents the results and analysis of the *Fishmarket* tournament held this spring at the Technical University of Catalonia (UPC) by a group of undergraduate students as a course work for an artificial intelligence applications course.
In the tournament participated sixteen different agents that competed in a three phase eliminatory competition. The agents were divided in groups of four and competed in a number of Downward Bidding Protocol (DBP) auctions for boxes of fish.
We present the information analyzed by the students in order to build their agents, what information was considered relevant, and the different strategies of the agents.

Keywords: Autonomous Agents, Multiagent Systems, Electronic Institutions, e-Auctions

1 Introduction

This work presents the results of the spring auction tournament held at the Technical University of Catalonia. The participants of the tournaments are students from the undergraduate course on applications of the artificial intelligence from the Barcelona School of Informatics.

This kind of tournaments have been held during the past five years with very fruitful results. The agents implemented has been used as a test for the Fishmarket platform and had aid to tune and extend its possibilities.

This competition consists in set of auctions of goods (fish boxes) using the Downward Bidding Protocol (DBP) as auction protocol. The goal of the agents is to pursue the greater benefit.

Notice that our initiative shares many commonalities with the *Double auction* tournaments held by the Santa Fe Institute[1] where the contendants competed for developing optimized trading strategies.

F. Dignum and U. Cortés (Eds.): AMEC 2000, LNAI 2003, pp. 173–191, 2001.

The agents used in this last tournaments, a total of sixteen, were developed in groups of three students to introduce issue about electronic markets and their relationship with autonomous agents.

The fishmarket platform provides all the implementations needs (data structures, market information, communication, etc.), so the only problem to solve is the strategy to deal with the auctions. The students had all the available information about how the market works and the parameters that the platform provides. As way to stimulate competition among the different groups, a part of the mark of the course is related to the performance of their agents in the tournament.

This article is organized as follows. In section 2 we will briefly describe the development framework and the characteristics of the auctions that can be held with the Fishmarket platform. Section 3 will be devoted to the characteristics of the spring tournament, its parameters, the different scenarios that the agents had to face and the goals that were pursued. In section 4, the different agents will be analyzed, describing its strategies and the different market information that were used. Section 5 will summarize the results of the tournament and the explanation of the success of the different strategies. Finally, the section 6 will summarize all the conclusions drawn from the tournament.

2 The Experimental Framework

In order to obtain an auction tournament environment, more functionality has been added to the FM96.5 agent-mediated electronic auction house[14] to turn it into a *domain-specific* test-bed that models and simulates an e-auction house that henceforth we shall refer to as *FM*. A distinguishing feature of the resulting test-bed is that it is *realistic* since it has been built out of a complex real-world application. Being an extension of FM96.5, FM inherits *interagents*, the mechanism of interaction between trading agents and the market. As introduced in [7]interagents are a particular type of facilitators conceived as autonomous software agents devoted to mediating the interaction among agents in an agent society in general and in an agent-mediated institution in particular. Thus, interagents constitute the unique mean through which agents interact within a multi-agent scenario as depicted in Figure 1. Interagents are all *owned* by the institution but *used* by external agents. As a major role, interagents are responsible for guaranteeing the enforcement of institutional norms to external agents.

Consequently FM shows a crisp distinction between agents and the simulated world, a desirable requirement for any multi-agent test-bed. Furthermore, the use of interagents permits also to consider FM as an *architecturally neutral* environment since no particular agent architecture (or language) is assumed or provided. However, some support for agent developers is provided by including a library of agent templates in various languages (C, Java, and Lisp) for building agents. Furthermore, the test-bed also offers the possibility of generating customisable *dummy agents* at the aim of providing agent developers with contenders for training purposes.

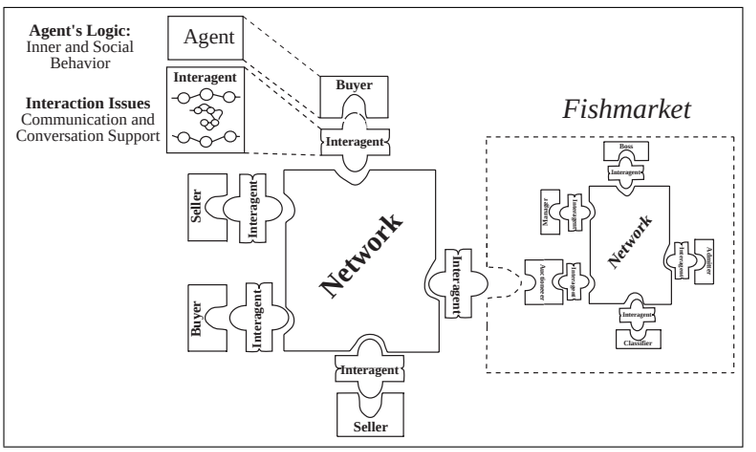


Fig. 1. Interagents in an Agent-mediated Institution.

FM inherits also all the auction protocols included in FM96.5, namely Dutch, English, First-price sealed-bid and Vickrey. All these auction protocols are classified as *single-sided* since bidders are uniformly of type buyer of uniformly of type seller[1]. *Double-sided* auctions admit multiple buyers and sellers at once. Figure 2 depicts a possible taxonomy for a small part of the auction space. The classification is made on the basis of whether the auction is single or double, bids are sealed (SB) or public (outcry), and prices are called in either ascending or descending order. FM contains the auction protocols hanging along the left branch, i.e. the classic auction types. Consequently FM can be classified as a multi-agent test-bed for classic auctions. As to the systematisation of our experiments, the complete parametrisability of FM allows for the generation of different market scenarios. This capability of *scenario generation* appears as a fundamental feature of any multi-agent test-bed if it intends to guarantee the *repeatability* of the experiments to be conducted. Concretely, the customisability of FM allows for the specification, and subsequent activation, of a large variety of market scenarios: from simple toy scenarios to complex real-world scenarios, from carefully constructed scenarios that highlight certain problems to randomly generated scenarios useful for testing trading agents' average performance. Figure 3 displays a snapshot of the graphical display provided by FM to specify the particular features of a tournament scenario.

As to the matter of evaluating a trading agents' performance, FM keeps track of all events taking place during a experimental session, so that a whole auction can be audited step-by-step, and the evolving performance of all the agents involved in a tournament can be traced, calculated, and analysed. On the one hand, FM records all information produced during an experimental session

[1] Particularly single auctions have been the main focus of theoretical studies of auction [8].

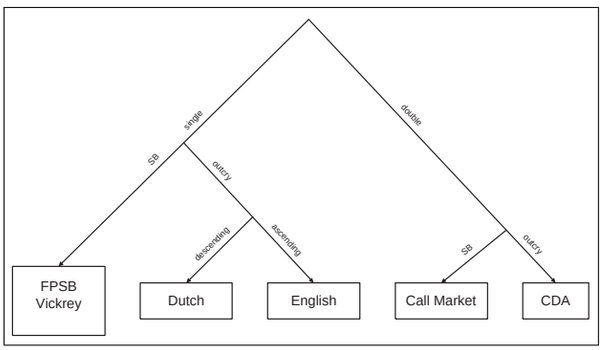


Fig. 2. A classification of classic auction types[15].

onto a database. On the other hand, FM counts on monitoring capabilities. A monitoring agent receives all the events distributedly coming about in the marketplace thanks to interagents, which collect and convey *carbon copies* of all external and institutional agents' utterances so that the monitoring agent can order them to reconstruct the dynamics of a market session.

Lastly it is worth mentioning a very important feature that seems to be somewhat skipped by test-bed designers: the problem of scalability. When running multi-agent experiments, an experimenter usually faces serious resource limitations that may prevent him from having all agents up and running. We say that FM is *scalability-aware* in the sense that it provides support for distributing an experimenter's agents across several machines in a network. This does not mean that all agent involved in a tournament must belong to the very same user. Tournament designers are free to define *open tournaments* accessible to agents owned by multiple users.

Notice that the resulting environment, FM, thus constitutes a multi-agent testbed where a very rich variety of experimental conditions can be explored systematically and repeatedly, and analysed and reported with lucid detail if needed. Table 1 summarises the features of FM.

3 The Tournament Scenario

A trading scenario will involve a collection of explicit conventions that characterise an artificial market. Such conventions define the bidding conditions (timing restrictions, increment/decrement steps, available information, etc.), the way goods are identified and brought into the market, the resources buyers may have available, and the conventions under which buyers and sellers are going to be evaluated. Next we introduce the elements needed to make precise specifications of actual tournament scenarios in general and of the actual UPC tournament scenario. In general terms, a tournament scenario specification is intended to

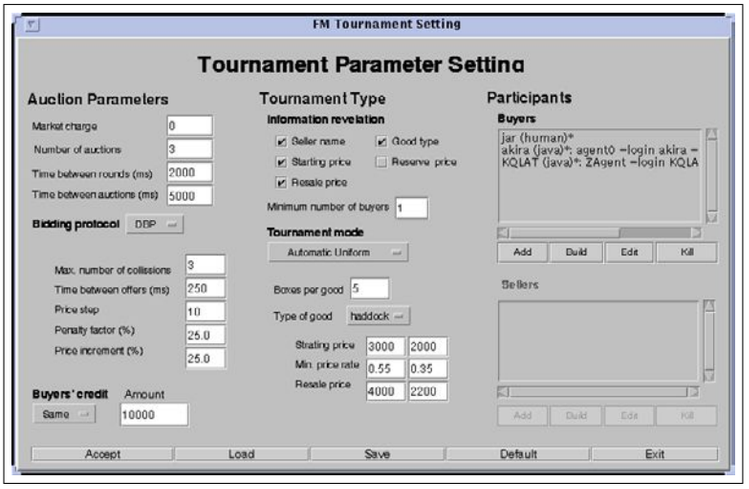


Fig. 3. FM Tournament Definition Panel

comprise all the information necessary for a trading agent to participate in a tournament along with the way they are to be evaluated.

We shall start by studying the characterizing parameters of auction protocols. In particular, although FM supports the classic auction protocols (Vickrey, First-price Sealed-bid, English and Dutch), we shall solely consider a slight variation of the Dutch bidding protocol —henceforth referred to as Downward Bidding Protocol or DBP for shorter— since it was the unique auction protocol employed in the UPC tournament[2]. Each auction protocol can be characterised by a set of parameters that we refer to as *bidding protocol dynamics descriptors*, so that different instantiations of such descriptors lead to different behaviours of their corresponding bidding protocols.

With a chosen good g,the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, (p_α), as long as these price quotations are above a *reserve price* (p_{rsv}) previously defined by the seller. For each price called by the auctioneer, several situations might arise during the bidding round:

- *Proper sale.* When a single buyer submits a bid that his credit can support, it is turned into a sale.
- *Unsupported bid.* When a buyer submits a bid that his credit cannot guarantee. The buyers' manager fines this bidder and the round is restarted by the auctioneer who calculates the new starting price by increasing by some percentage $\Pi_{sanction}$ the price within the bid.
- *Collision.* When two or more buyers simultaneously submit the same bid. The auctioneer declares a collision and restarts the round. Again, the new

[2] A thorough characterization of the rest of bidding protocols is provided in [10].

Table 1. Features of the FM test-bed.

Test-bed Features
■ domain-specific
■ realistic
■ architecturally neutral
■ scenario generation and reapeatibility capabilities
■ monitoring and evaluation facilities
■ library of agent templates (C,Java,Lisp)
■ dummy agents
■ scalability aware
■ open (multi-user) and closed (single-user) tournaments
■ market scenarios as tournament scenarios

starting price is calculated by increasing by some percentage Π_{rebid} the collision price.
- *Expulsion.* When a buyer is overdrawn and cannot back up a fine, he is sent off the market and the round is restarted as usual
- *Withdrawal.* Each good is assigned a minimum price when passing through the sellers' admitter office. If minimum prices are reached, the round is restarted as usual.

The algorithm in Figure 4 codifies the downward bidding protocol. The description helps us to explicitly identify the parametrisation of the bidding protocol.

Six parameters that control the dynamics of the bidding process are implicit in this protocol definition. We shall enumerate them now, and require that they become instantiated as part of a tournament scenario definition.

Definition 1 (DBP Dynamics Descriptor).
We define a Downward Bidding Protocol Dynamics Descriptor $\mathcal{D}_{DBP}$ *as a 5-tuple* $\langle \Delta_{price}, \Delta_{offers}, \Sigma_{coll}, \Pi_{sanction}, \Pi_{rebid} \rangle$ *such that*

- $\Delta_{price} \in \mathbb{N}$ *(price step). Decrement of price between two consecutive quotations uttered by the auctioneer.*
- $\Delta_{offers} \in \mathbb{N}$ *(time between offers). Delay between consecutive price quotations.*
- $\Sigma_{coll} \in \mathbb{N}$ *(maximum number of successive collisions). This parameter prevents the algorithm from entering an infinite loop as explained above.*
- $\Pi_{sanction} \in \mathbb{R}$ *(sanction factor). This coefficient is utilized by the buyers' manager to calculate the amount of the fine to be imposed on buyers submitting unsupported bids.*
- $\Pi_{rebid} \in \mathbb{R}$ *(price increment). This value determines how the new offer is calculated by the auctioneer from the current offer when either a collision, or an unsupported bid occur.*

```
Function round (B_r^i, g_r^i, p, coll, D_DBP) =
let Function check_credit(b_i) =
 if C_r^i(b_i) ≥ p then
  update_credit(b_i, p);
  sold(g_i^r, b_i, p);
 else if C_r^i(b_i) ≥ p * Π_sanction then
        update_credit(b_i, p * Π_sanction);
        round(B_r^i, g_r^i, p * (1 + Π_rebid), 0, D_DBP);
       else
        round(B_r^i − {b_i}, g_r^i, p * (1 + Π_rebid), 0, D_DBP);
in
 offer(g_r^i, p);
 wait(Δ_offers);
 let B = {b_i|bid(b_i) = true, b_i ∈ B_r^i} in
  case
    ||B|| = 0 :  if p = p_ω then withdraw(g_r^i);
                 else round(B_r^i, g_r^i, p − Δ_price, 0, D_DBP);
    B = {b_i} :  check_credit (b_i);
    ||B|| > 1 :  if coll < Σ_coll then
                  round(B_r^i, g_r^i, p * (1 + Π_rebid), coll + 1, D_DBP);
                 else check_credit(random_select(B));
  end case
 end
end

DBP(B_r^i, g_r^i) = round(B_r^i, g_r^i, p_α, 0)
```

Fig. 4. Downward bidding protocol

Note that the identified parameters impose significant constraints on the trading environment. For instance, Δ_{offers} and Δ_{rounds} affect the agents' time-boundedness, and consequently the degree of situatedness viable for bidding strategies.

By *auction round* we shall refer to the ontological elements involved in each bidding round.

Definition 2 (Auction Round).
For a given round r of auction i we define the auction round $\mathcal{A}_r^i$ *as a 4-tuple* $\langle \mathcal{B}_r^i, g_r^i, \mathcal{C}_r^i, d_r^i \rangle$ *where*

- $\mathcal{B}_r^i$ *is a non-empty, finite set of buyers' identifiers such that* $B_r^i \subseteq \mathcal{B}$*, the set of all participating buyers.*
- $g_r^i = \langle \iota, \tau, p_\alpha, p_{rsv}, s_j, p_\omega, p_{rsl}, b_k \rangle$ *is a good where* ι *stands for the good identifier,* τ *stands for the type of good,* $p_\alpha \in \mathbb{N}$ *stands for the starting price,* $p_{rsv} \in \mathbb{N}$ *stands for the reserve price,* $s_j \in \mathcal{S}$*—the set of all participating sellers—is the seller of the good,* $p_\omega \in \mathbb{N}$ *stands for the sale price,* p_{rsl} *stands*

for the expected resale price, and $b_k \in \mathcal{B}_r^i$ *is the buyer of the good. Notice that* g_r^i *is precisely the good to be auctioned during round* r *of auction* i*, and that* p_ω *and* b_k *might take on empty values when the round is over, denoting that the good has been withdrawn.*
- $\mathcal{C}_r^i : \mathcal{B}_r^i \longrightarrow I\!R$ *assigns to each buyer in* $\mathcal{B}_r^i$ *his available credit during round* r *of auction* i*.*
- d_r^i *stands for an instance of a bidding protocol dynamics descriptor.*

Each auction is devoted to the auctioning of a particular *lot* of goods by opening an auction round for each item within the lot. Typically a tournament session (and a market session too) will be composed of a sequence of auctions.

Definition 3 (Auction). *We define an auction* $\mathcal{A}^i$ *as a sequence of* auction rounds $\mathcal{A}^i = [\mathcal{A}_1^i, \ldots, \mathcal{A}_{r_i}^i]$

On the basis of these definitions, we are ready to determine what elements and parameters are necessary to wholly characterise a tournament scenario, i.e. all the relevant information needed by an agent to participate in an auction-based tournament, compiled in the definition of *tournament descriptor*. A tournament descriptor is intended to be the sole information on which trading agents count prior to the starting of a tournament session.

Definition 4 (Tournament Descriptor). *We define a Tournament Descriptor* $\mathcal{T}$ *as the 11-tuple* $\mathcal{T} = \langle n, \Delta_{auctions}, \Delta_{rounds}, \mathcal{D}, \mathcal{P}_B, \mathcal{P}_S, \mathcal{B}, \mathcal{S}, \mathcal{F}, C, M, \epsilon, E \rangle$ *such that:*

- n *is the tournament length expressed either as the number of auctions to take place during a tournament or the closing time.*
- $\Delta_{auctions}$ *is the time between consecutive auctions.*
- $\Delta_{rounds} \in I\!N$ *(time between rounds) stands for the delay between consecutive rounds belonging to the same auction.*
- $\mathcal{D}$ *is a finite set of bidding protocols' dynamics descriptors.*
- $\mathcal{P}_B$ *is the conversation protocol that buyer agents must employ in their interaction with their interagents.*
- $\mathcal{P}_S$ *is the conversation protocol that seller agents must employ in their interaction with their interagents.*
- $\mathcal{B} = \{b_1, \ldots, b_p\}$ *is a finite set of identifiers corresponding to all participating buyers.*
- $\mathcal{S} = \{s_1, \ldots, s_q\}$ *is a finite set of identifiers corresponding to all participating sellers.*
- $\mathcal{F} = [\mathcal{F}^1, \ldots, \mathcal{F}^n]$ *is a sequence of supply functions. A supply function* $\mathcal{F}^i$ *outputs the lot of goods to be auctioned during auction* i*.*
- $C : \mathcal{B} \rightarrow I\!N$ *is the credit initially endowed to each buyer. For some tournaments, all buyers are assigned the same credit, while for others they may either have assigned different credits or alternatively declare themselves the credit they want to have available.*

- $M = \langle b, s, r, r' \rangle$ *where* $b, s, r, r' \in \{0, 1\}$ *is the information revelation mask. It determines whether the identity of buyers (b) and sellers (s) is revealed to the contenders, and whether the reserve price (r) and expected resale price (r′) of a good are revealed too.*
- ϵ *stands for the fees charged to an agent for participating in a bidding round.*
- $E = \langle E_b, E_s \rangle$ *is a pair of evaluation functions that permit to calculate respectively the score of buyers and sellers.*

From the definition follows that a tournament descriptor contains:

- all the relevant parameters that characterise the dynamics of the auctioning process;
- the procedural information that allows trading agents to participate in the market by means of their interagents;
- the degree of information revelation (transparency) (i.e. the degree of uncertainty concerning the identity of traders and some particular, relevant features of goods); and
- the way the performance of trading agents is evaluated.

It is the task of the tournament designer to conveniently set up the parameters of the tournament descriptor in order to generate the desired type of tournament scenario. For this purpose, FM provides the graphical configuration tool shown in Figure 3 to assist the tournament designer to configure tournament scenarios.

Additionally FM incorporates the so-called *tournament modes* that constrain the type of tournament descriptor that can be defined. The purpose of this standard tournament modes is to allow an experimenter to define tournament scenarios of different degrees of complexity: from toy scenarios where, for instance, the same lot of goods is repeated over and over with complete information to actual-world auction scenarios. Thus, in FM tournament designers can choose among the following standard tournament modes:

One Auction (data set). This mode permits a tournament designer to specify a fixed set of goods to be repeatedly auctioned a finite number of times. Notice that no sellers are involved in this type of tournament.

Automatic. The lots of goods to be auctioned are artificially generated by the sellers' admitter based on supply functions of arbitrary complexity specified by the tournament designer in the set $\mathcal{F}$. Notice that likewise *one auction (data set)* no sellers are allowed to participate in these tournaments.

This tournament mode allows to artificially generate a large variety of markets. For instance, markets with more demand than supply or the other way around, markets with high quality goods more appropriate for restaurant owners, or markets with large supply of low-quality goods more appropriate for wholesale buyers[3]. In general, this tournament mode allows to create tournaments focusing on particular market scenarios.

[3] Note that for all the examples we consider fishmarket-like tournament scenarios.

Uniform. This mode is a particular case of the preceding tournament mode. Lots of goods are randomly generated by the sellers' admitter based on uniform distributions in $\mathcal{F}$ defined by the tournament designer. Notice that again no sellers are involved in the resulting tournaments either. Table 2 shows some examples of uniform distributions that can be employed for generating lots of goods.

This tournament mode is intended to generate scenarios wherein the average performance of buyer agents can be tested. Along with *one auction (data set)* it must be considered as a mode to generate game-like scenarios.

One Auction (with sellers). Once all participating sellers have submitted their goods, the same auction is repeated over and over with the same lot of goods. This tournament mode is particularly useful to test the adaptivity of trading agents to an actual market scenario.

Fishmarket. The mode closest to the workings of an actual auction house[4]. The tournament designer simply specifies the starting and closing times. During that period of time buyers and sellers can enter, submit goods, bid for goods, and leave at will. *Fishmarket* is the more realistic mode, standing for an actual market scenario.

Depending on the tournament mode chosen by the experimenter, some features of the tournament descriptor will be either enabled or disabled in the *parameter setting panel* at Figure 3. Notice that all parameters identified as part of the tournament descriptor lie down on the *parameter setting panel*.

Finally the UPC tournament can be fully characterised by the tournament descriptor in Table 2. Some comments apply to the resulting scenario:

- All buyer agents were assigned the same credit (17.500 EUR) at the beginning of each auction of the tournament.
- Because the tournament mode was set to *uniform*, the number of fish boxes for each type of fish (τ) were randomly generated for each auction $\mathcal{A}^i$, and the starting price (p_α), resale price (p_{rsl}), and reserve price (p_{rsv}) of each box were also randomly generated according to the uniform distributions in Table 2.
- As to information revelation, whereas the identity of buyers and the expected resale price of each good were made publicly available, the reserve price was kept as private information.
- The chosen evaluation function (E_b) calculates the performance for each buyer at round number r of auction number k based on the accumulated benefits ($B_k(b)$) of buyer b at auction k. The goal of this evaluation function is to weigh higher the fact of winning the auctions which are closer to the end of the tournament. In this way, bidding strategies that learn to improve an agent's performance as the tournament goes by are more valued.

[4] We name it *fishmarket* for historical reasons, though the term must not be misleading since under this mode goods can be auctioned through several auction protocols.

Table 2. UPC'2000 Tournament Descriptor

n	10
$\Delta_{auctions}$	*8000 msec*
Δ_{rounds}	*4000 msec*
$\mathcal{D}$	$\{d_{DBP}\} = \{\langle 50.0EUR, 500msec, 3, 0.25, 0.25\rangle\}$
P_B	P_{DBP}
P_S	$\emptyset$
$\mathcal{B}$	$\{b_1, b_2, b_3, b_4\}$
$\mathcal{S}$	$\varnothing$
$\mathcal{F}^i (i = 1 \ldots n)$	

τ	$\#Boxes$	p_α	p_{rsl}	p_{rsv}
cod	$U[1, 15]$	$U[1200, 2000]$	$U[1500, 3000]$	$U[0.4, 0.5]$
tuna fish	$U[1, 15]$	$U[800, 1500]$	$U[1200, 2500]$	$U[0.3, 0.45]$
prawns	$U[1, 15]$	$U[4000, 5000]$	$U[4500, 9000]$	$U[0.35, 0.45]$
halibut	$U[1, 15]$	$U[1000, 2000]$	$U[1500, 3500]$	$U[0.4, 0.6]$
haddock	$U[1, 15]$	$U[2000, 3000]$	$U[2200, 4000]$	$U[0.35, 0.55]$

C	$C(b) = 17.500EUR\ \forall b \in \mathcal{B}$
M	$< 1, 0, 0, 1 >$
ϵ	0
E	$\langle E_b, E_s \rangle = \langle \sum_{k=1}^{n} ln(k+1) B_k(b), \varnothing \rangle$

At this point, it is time to make explicit how trading agents and interagents interact in practice and the conversation protocol that they employ. An interagent works as a Java process which uses its standard input and standard output to communicate with trading agents via pipes. In adopting such a simple convention, software agents written in any programming language can interact with the auction house via interagents. Thus, a trading agent firstly spawns the interagent received from the auction house as a child process and subsequently plug to it. Thereafter trading agent and interagent communicate in a rather straightforward way by exchanging string–based illocutions according to the protocol depicted in Figure 5 as an FSM. Tables 3 and 4 list respectively the possible contents of the illocutions labelling the arcs in Figure 5, while Table 5 lists their intended meanings. In Figure 5 numbers followed by / stand for a buyer's agent utterance while messages following / stand for a buyer's agent reception.

Table 3. Messages that (software) buyer agents can utter during a tournament

.

#Message	**Predicate**	**Parameters**
1	*admission*	*buyerlogin password*
2	*bid*	
3	*exit*	

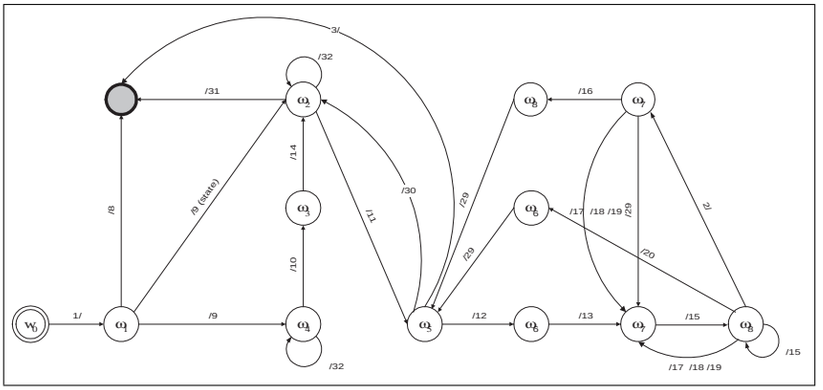


Fig. 5. Protocol used by buyer agents to interact with interagents.

4 The Agents

As said before, a total of sixteen agents participated in the tournament. All the students had time to study the environment, and to experiment with toy agents provided by the platform and agents from previous tournaments. The code of the agents from previous tournaments was not available, so, they only could observe their behavior against other agents.

From their study of the platform and private tournaments, the different groups observed the information that could be helpful in the problem. They reported the following possibly relevant information:

From the market: Number of rounds, number of boxes, initial market money, remaining market money, last benefit.

From the goods: Starting price, resale price, reserve price if the good is retired from the market, name of the good, ratio between buying price and resale . price, name of the buyer,

From the competitors: Mean benefit, benefit of the best agent, remaining credit, behavior of the agent.

From the agent state: Own benefit, remaining credit, number of boxes bought.

Due to the time restrictions, not all this information could be used during the tournament. Each group reduced the information available to just what they though could be relevant on deciding the bidding price for a good.

There was a great consensus between the agents about what information had to be considered. The first of it was the length of the auction. Almost all the agents considered a classification of the auctions from its length. The number of classes ranged from two to four, but the most used value was three. The classification characterized short sized auctions (approximately 20 boxes), medium sized auctions (approximately 45 boxes) and long sized auctions (up to 75 boxes.

Table 4. Messages that (software) buyer agents can receive during a tournament.

#Message	Predicate	Parameters
8	*deny*	*deny_code*
9	*accept*	*admission*
10	*open_auction*	*auction_number*
11	*open_round*	*round_number*
12	*good*	*good_id good_type starting_price resale_price auction_protocol*
13	*buyers*	$\{buyerlogin\}^*$
14	*goods*	$\{good\_id\ good\_type\ starting\_price\ resale\_price\}^* protocol$
15	*offer*	*good_id price*
16	*sold*	*good_id buyerlogin price*
17	*sanction*	*buyerlogin fine*
18	*expulsion*	*buyerlogin*
19	*collision*	*price*
20	*withdrawn*	*good_id price*
29	*end_round*	*round_number*
30	*end_auction*	*auction_number*
31	*closed_market*	*closing_code*
32	*tournament_descriptor*	$auction\ n\ \Delta_{auctions}\ \Delta_{rounds} \in bidding\_protocols\ dbp\ \Delta_{price}\ \Delta_{offers}\ \Sigma_{coll}\ \Pi_{sanction}\ \Pi_{rebid}\ UBP\ \Delta_{price}\ \Delta_{offers}\ \Pi_{sanction}\ \Pi_{start}\ FPSB\ B_t\ \Pi_{sanction}\ vickrey\ B_t\ \Pi_{sanction}$ (*buyers* $\{buyerlogin^*|\#buyers\}$ *credit* $\{credit^*|credit|unkown\}$ *sellers* $\{sellerlogin^*|Market\}$ *mode* $\{automatic, uniform\ one\_auction\_data, one\_auction\_sellers, fishmarket\}$

It is a strategy observed in this tournament and previous, to classify the auction by length. Each kind of auction lead to a different strategy:

- In short auctions an aggressive strategy is used, trying to buy almost at starting price. If the credit is enough, this is an admissible strategy because the total money is more than the cost of all the goods. There are no time to consider the characteristics of the goods, because probably not all the money could be spent. The better good is that with a better ratio between starting price and resale price.
- In medium auctions a more deliberative strategy is necessary. The total money of the agents is almost enough to buy all the goods, so the agents had to be selective and compete for the best goods. The last goods of the auction can be interesting because their price can be lower.
- In long auctions the planning is very important. The cost of the goods are more than the total market money. The agent has to decide what goods are interesting because its price and its position in the auction. It could be

Table 5. Semantics of the messages exchanged between a buyer and the auction house.

Predicate	**Semantics**
exit	Leave the marketplace.
admission	Request for admission.
bid	Bid at the current price.
deny	Refuse requested action.
accept	Accept access to scene.
open_auction	The auctioneer opens a new auction.
open_round	The auctioneer opens a new bidding round.
good	Features of the good in auction.
buyers	List of participating buyers.
goods	Lot of goods to be auctioned.
offer	Current offer called by the auctioneer.
sold	The good in auction has been sold.
sanction	Sanction imposed on a given buyer.
expulsion	Buyer expelled out of the market.
collision	Multiple bids at the same price (DBP).
withdrawn	Reserve price reached. Good withdrawn.
end_round	Bidding round over.
end_auction	Auction over.
closed_market	End of market session.

an interesting strategy to wait until all the competitors had spent all their money in order to obtain better prices. In this kind of auctions an accurate estimation of the reserve price is very important.

The other information from the auction that had almost all the consensus was the quotient between the total resale value of the goods of the auction and the total market money. This value can be uses as an estimation of the mean expected benefit. To outperform or underperform this value is an indicator of the performance of the agent. This measure is correlated to the behavior of the auction and allow to not to observe individually to each competitor.

This expected benefit can be updated during the auction by the bought of the agents. This allow to change the behavior of the agent because the raise or fall of the expected benefit.

Almost all the agents used this ratio as base value in order to decide its bid. If the initial benefit of the good is lower than the mean benefit, then the good is not interesting and, either the bid is not done, or the agent wait until the price drops to a more interesting one.

The agents used other complementary values to correct the bidding price obtained from the calculation of the mean benefit. For example, the benefit of the best agent, the remaining credit of the competitors and heuristical factors obtained by experimentation during the private auctions that were held before the official tournament.

Due to that in long auctions to wait until almost the end of the auction is a profitable policy, the estimation of the reserve price becomes important. Every agent has a way to estimate the reserve price. Some agents do the estimation dynamically, trying to learn this price from the auction, others used a constant percentage from the initial price. Obviously, the agents that try to estimate the reserve price dynamically obtained better results.

The strategies to determine the reserve price were diverse, but based on statistical estimation. Because the real reserve price is unknown, the difference between starting price and the lower price payed for the goods is a good initial estimation. This estimation can be corrected using the price observed when a good goes out of market, circumstance that can be observed in long auctions. Some agents tried to estimate the reserve price for each kind of good. Due to the relative shortness of observations those estimations were less accurate that those from the agents that tried to estimate a global reserve price.

Planning and learning were rare among the agents of the tournament. Some agents tried to plan beforehand the goods more attractive, estimating the optimal bid and distributing the available money among them. All allowed a dynamical redistribution of the bids if the chosen goods were bought by another competitor.

Just two agents tried to use learning between auctions to improve their performance. The first, used the comparison between the benefit obtained and the benefit of its competitors in order to reduce or increase the bidding price in the next auction. the second used a more sophisticated learning mechanism based on reinforcement learning. This strategy used Q-learning in order to decide the optimal benefit for each good from the own actions and the actions of its competitors.

5 The Results

The competition was organized in three eliminatory rounds. The first round divided the agents randomly in four groups. Each group competed in a tournament as specified in section 3. From each group only the two best were chosen.

In this round the agents with a weak strategy obtained a significant less performance than the more elaborated agents. This year, in contrast with previous tournaments, the level of cooperation between the groups were very low. Only a small number of agents participated on private tournaments. Most of the agents that were eliminated in this round were the non cooperative ones. This gives an idea of how important is cooperation and interaction during the developement of agents.

The second round paired the winning agents of the first and second group and the agents of the third and fourth group. In this round also the two best of each group passed to the final round.

In this round the competition was hardest. In the first group the difference among the three firsts agents were very short. In the second group there was a clear difference between the first two agents and the other two competitors.

The strategies of the winners of this round were not significantly different from the rest, but included some the agents that used some kind of learning and adaptation.

Surprisingly, the winner of the final round was the agent with the simplest strategy of the four competing agents. Those are the four agents of the final round and their strategies:

HumbleJES: This is the winner agent. The basis of this agent is the ratio between the resale price of the remaining goods and the total credit of the agents. This ratio is weighted using a value that is an estimation of the desired benefit. This expected benefit is a constant that is not changed during the competition.

This value is used to estimate the bid for the actual good. This price is corrected with the information about the money available for the other agents. If this value is greater than the price that can be paid by their competitors, it is adjusted to a little more than this quantity. If the competitors can not buy the good, then the price is adjusted to the estimated reserve price.

Garsa: This is the second agent. The basis of this agent is also the expected benefit obtained as a ratio of the resale price of the goods and the money available, but in this case, this ratio is calculated at the beginning of each auction. This value is modified using the behavior of the other agents. If the rest of agents bid to a price higher that the calculated, the value is not touched. If the other agents bid to a lower price, the benefit is adjusted to obtain a bid slightly higher that the bid of the competitors, increasing the own benefit.

This agent detects when the competitors have not enough money to buy more goods. When this happens, the bid is adjusted to a statistically estimated reserve price.

The Pretender: This is the third agent. This is the more sophisticated agent, it uses reinforcement learning (Q-learning)in order to learn what is the better price for a good. It uses a probability matrix indexed by resale price and expected benefit. This matrix stores the probability distribution of the optimal benefit for a given resale price. The matrix was initialized with a priori probability distributions obtained from the private tournaments.

Three different reinforcements are used during the auction. A positive reinforcement if the current bid is successful and is considered a good bid, a negative reinforcement if it is considered that the actual bid benefit has to change and a negative reinforcement if the actual did benefit of the agent is high. A set of rules allow to decide what kind of reinforcement is necessary. These rules evaluate different information, as the number of remaining rounds, the performance of the competitors or the number of competitors with enough money. The learning is done in each auctions, so the information of the previous auctions is not maintained.

This probability matrix adapts to the behavior of the market, and predicts the most probable benefit that the competitors desire to obtain. This information allow to advance the bid and to buy before than the competitors.

TokOchons: This is the fourth agent. The strategy of this agent uses two information. The first is a variation of the ratio between the resale price of the remaining goods and the remaining market money. This information allow to guess the expected benefit. The second source of information is a function that give a measure of how interesting is a good. This function combines the relative and absolute benefit obtained for a given bid.
This bid is corrected using different parameters. The more interesting is a value that measures the proportion of the market money that the agent owns. If the proportion is great, this means that the agent almost has not competitors, so, the expected benefit can be raised.
This agent stores the past auctions in order to analyze them. If the current auction has a similar number of rounds that a past auction, its information is recalled. If in this past auction some money was not spent, the bids are raised in order to spent all the money, increasing the benefit by buying more goods. If the winner of this past auction obtained a benefit higher than ours, the expected benefit for the current auction is raised in order to pay less for the goods.

In the figure 6 can be seen the evolution of the objective function (see section 3) that measured the performance of the agents. It can be seen that the agent **HumbleJES** performs significatively better that the others from the start of the competition, the other tree agents are in a tie until auction number seven, in this point the agent **garsa** starts outperforming the other two agents. It seems that the learning procedures of this two agents are not a real advantage against the other two strategies.

6 Conclusions

Some conclusions can be drawn from this tournament. First of all, that more sophisticated strategies has not evident advantage against simple ones. The best agents use an strategy based on market information without neither trying to model the other agents not use learning from experience to improve their performance. This does not means that this characteristics are not desirable. An adequate learning policy could overperform simple strategies in a more dynamic environment.

The other conclusion is the significance of competition in the developement of this kind of agents. At has been said, only the agents from the people that decided to share their knowledge and competed in private tournaments were successful. The need to test a strategy are crucial for its developement. It is difficult to have success without interaction.

Acknowledgments. We want to thank the IIIA (CSIC) to allow us to use the *Fishmarket* platform for the tournaments. Also we want to acknowledge all the competitors of this spring tournament.

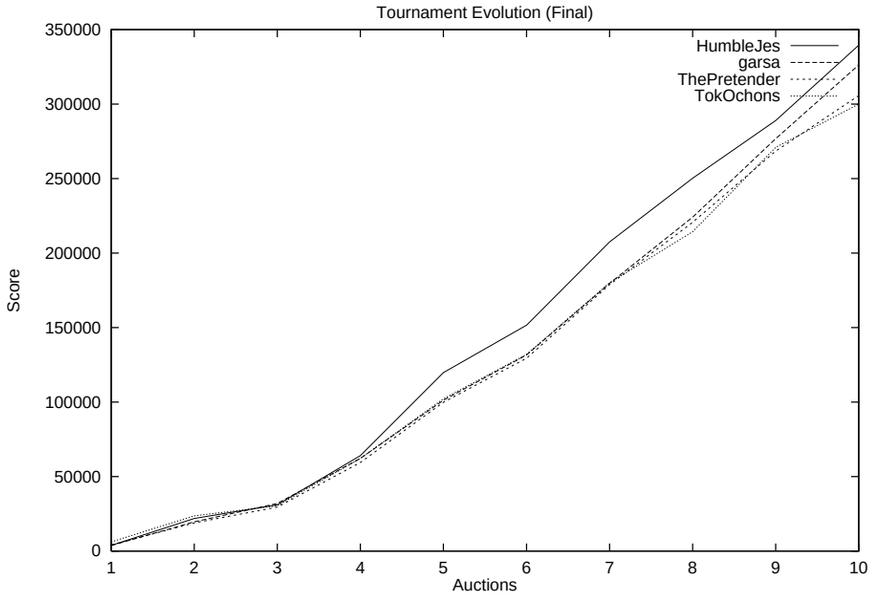


Fig. 6. Evolution of the last round of the tournament

References

1. M. Andrews and R. Prager. *Genetic Programming for the Acquisition of Double Auction Market Strategies*, pages 355–368. The MIT Press, 1994.
2. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, 1996.
3. U. Cortés and J. A. Rodríguez-Aguilar. Trading agents in auction-based tournaments. *INFORMATIQUE*, 1(1):39–50, 2000.
4. Facultat d'Informàtica de Barcelona. http://www.fib.upc.es.
5. The FishMarket Project. http://www.iiia.csic.es/Projects/fishmarket.
6. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1995.
7. F. J. Martín, E. Plaza, and J. A. Rodríguez-Aguilar. An infrastructure for agent-based systems: An interagent approach. *International Journal of Intelligent Systems*, 15(3), 2000.
8. R. P. McAfee and J. McMillan. Auctions and bidding. *J. Ec Lit.*, XXV:699–738, jun 1987.
9. P. Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. Number 8 in IIIA Monograph Series. Institut d'Investigació en Intel.ligència Artificial (IIIA), 1997. PhD Thesis.
10. J. A. Rodríguez-Aguilar. *On the Design and Construction of Agent-Mediated Institutions*. PhD thesis, Institut d'Investigació en Intel.ligència Artificial, 2000. (forthcoming).

11. J. A. Rodríguez-Aguilar, F. J. Martín, F. J. Giménez, and D. Gutiérrez. Fm0.9beta users guide. Technical report, Institut d'Investigació en Intel.ligència Artificial. Technical Report, IIIA-RR98-32, 1998.
12. J. A. Rodríguez-Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. Competitive scenarios for heterogeneous trading agents. In *Proceedings of the Second International Conference on Autonomous Agents (AGENTS'98)*, pages 293–300, 1998.
13. J. A. Rodríguez-Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 11(1):5–19, 1998.
14. J. A. Rodríguez-Aguilar, P. Noriega, C. Sierra, and J. Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pages 207–224, 1997.
15. P. R. Wurman, , M. P. Wellman, and W. E. Walsh. The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In *Second International Conference on Autonomous Agents (AGENTS'98)*, pages 301–308, May 1998.

Author Index