
Multi-Agent Implementation of Asymmetric Protocol for Bilateral Negotiations

James E. Hanson, Gerald Tesauro, Jeffrey O. Kephart and Edward C. Snible
IBM T. J. Watson Research Center
19 Skyline Dr., Hawthorne NY, 10532

Abstract

We present a practical implementation of a FIPA-compliant multi-agent system, written in Java using the ABLE agent platform, in which the agents can negotiate the exchange of multi-attribute goods in reasonable time and with high efficiency. The architecture employs separate modules for messaging and decision logic. Negotiation protocols are expressed as conversation policies in XML and are highly reconfigurable; agents can also conduct a meta-negotiation to decide which protocol to use. We believe this provides a flexible and powerful platform for both real-world negotiation of complex goods, and for research into protocol design and strategy design.

We test the system using an asymmetric protocol to negotiate a two-attribute good, and using the simplest possible “Zero-Intelligence” (ZI) negotiating strategies. A key aspect of the protocol is a facility to obtain mutually beneficial modifications to an initial agreement. We find that the ZI agents reliably reach final agreements in a moderate number of rounds that are surprisingly close to Pareto-optimal contracts. These results, combined with ongoing studies of more sophisticated strategies, suggest that the protocol may be more generally useful in negotiating realistic goods with intelligent strategies.

1 Introduction

Many researchers have recognized the potential for much of electronic commerce to be conducted by intelligent software agents. Commerce involving simple fixed goods is effectively carried out by posted-price or auction mechanisms. On the other hand, for configurable goods with many attributes, or complex contracts involving multiple issues, direct negotiation between buyer and seller is the prevalent mechanism. There has been considerable analysis of models of agent-based negotiation [6], as well as implementations of systems such as Kasbah [2] and MARI [3]. Recently, negotiation scenarios in FIPA-compliant multi-agent systems have been investigated by Faratin et al [7].

Economists and game theorists have been working for decades on formulations and analyses of a number of negotiation models [16, 17], and this continues to be an active area of research. The two major topics addressed in these studies are *protocol design* and *strategy computation*. Research in protocol design aims to develop a set of rules for agent behavior during a negotiation that reliably leads to efficient

agreements. (In this paper we equate efficiency to Pareto-optimality, i.e., no player's payoff can be improved without harming another player.) A further aim is the development of "incentive-compatible" protocols that eliminate the need for agents to engage in strategic reasoning. However, it has been shown that, for two-sided negotiation, no incentive-compatible protocol can exist that simultaneously achieves efficient agreements and is budget-balanced [15]. Therefore, in general one expects that in any realistic negotiating scenario, it will be beneficial for agents to behave strategically.

The problem of strategy computation is, given the protocol and information set available to the agent, how to compute the negotiating decision that maximizes the agent's payoff. In theory, the Bayes-Nash equilibrium provides the correct agent strategy, but computation of such equilibria is generally feasible only for the smallest toy problems. Research in game theory of computationally limited agents attempts to address this limitation [14]. However, there are important practical aspects of real-world agent negotiations that are difficult to address with current theory, including: the ability to represent user preferences and negotiate over goods with many attributes, uncertain knowledge of the opponent's utility function and strategy, and the availability of outside deal-making opportunities beyond the current negotiating partner. We therefore expect that heuristic strategies will inevitably be employed in real-world negotiations for the foreseeable future.

This paper presents an implementation of a multi-agent system, written in Java atop the ABLE agent platform [5] and compliant with the emerging FIPA standard [8], that can be widely adopted for practical use for negotiation of multi-attribute goods. Informed by the above theoretical considerations, our system is capable of employing a wide variety of negotiating protocols, expressed as conversation policies in XML. The protocol is easily reconfigurable, and in fact there is a capability for the agents to have a meta-negotiation regarding which protocol to use. A detailed description of the system implementation is provided in section 2. While our system can implement many different protocols, in this paper we investigate the use of a particular asymmetric protocol, described in section 3, that appears to offer a number of practical and theoretical advantages. We have devised a scenario, described in section 4, for testing the effectiveness of our system, in which agents use simple "Zero-Intelligence" (ZI) strategies to negotiate a two-attribute, and CES utility functions are used to model the agents' preferences. The issue of strategy computation under a given protocol is beyond the scope of this paper. However, we suggest that ZI concept is easily coded and may be generally useful in testing the validity of any given protocol. Our test results are presented in section 5, followed by a summary discussion in section 6.

2 Multi-Agent System Implementation

2.1 ABLE as a multi-agent platform

We chose to use the ABLE agent platform [5] to implement our negotiating agents. ABLE provides support for multiple agents running in a distributed, Java-bean-based environment. ABLE has recently been upgraded to encompass the latest Java Agent Services (JAS) API for FIPA multi-agent platform services [13].

By adopting ABLE as a base platform, we were able to take advantage of a preexisting body of code for the workaday details of agent plumbing, such as lifecycle, agent activation scheduling, intra-agent subcomponent interconnection, visual beanbox program assembly. In addition, ABLE provides native implementations of a rich set of sophisticated AI tools, including genetic algorithms, rule-based inferencing

with both forward and backward chaining, and several kinds of machine learning. This opens up a wide set of possibilities for future enhancements.

ABLE's incorporation of JAS-based messaging adds a further degree of configurability, because JAS naturally encapsulates details of wire-protocol selection and message-transport plumbing, permitting the agent as a whole to be written in a transport-independent fashion.

For our purposes here, however, the most important feature of ABLE is its general-purpose support for *conversations* between agents. By this we mean bilateral, peer-to-peer exchanges of multiple, correlated messages in an explicit conversational context. The general outlines of conversation support architectures have been discussed elsewhere [12]. In brief, the architecture consists of three distinct layers: messaging support; conversation support; and decision logic. We believe this layering represents a natural, general-purpose, and powerful approach to agent design, which brings significant benefits in flexibility, interoperability, ease of coding and maintenance, and reusability of agent software.

2.2 Conversation support architecture

In ABLE conversational agents, each agent maintains a separate thread for each conversation it is carrying on at any give time. (For our experiments, only one conversation was needed.) The fundamental building blocks of an ABLE agent are AbleBeans, which are an enhanced version of standard Java beans.

Figure 1 shows the high-level architecture of an ABLE conversational agent. The agent acts as a messaging endpoint, bound to the JAS message transport service, which is provided by the ABLE agent platform and which takes care of receiving and sending messages between agents. Inbound messages are fed for processing into a Conversation Manager, described below, which is also responsible for generating outbound messages. As the conversation progresses, the Conversation Manager will from time to time call on the agent to make a decision (such as, whether to accept or reject an offer). This it does by sending a decision request to one or another decision logic module, which carries out the decision-making process and sends its result back to the Conversation Manager. The Conversation Manager and the decision logic modules are implemented as AbleBeans, an enhanced version of standard Java beans.

Though the figure only shows a single agent, the ABLE agent platform can support any number of agents running in a container, and any number of containers running on any number of servers. The JAS messaging service automatically routes messages between agents within the same container or in different containers.

The function of the conversation management system is runtime support for *conversation policy* execution. A conversation policy (CP) is a specifications of a message-exchange protocol, giving the message formats, sequencing constraints and (optionally) timeouts that define the protocol.

Conversation policies have been the subject of research within the software agents community for years (e.g., see ref. [4] and references therein). However, much of that work has tended to blur the useful distinction between protocol management and decision logic.

Conversation policies used by ABLE agents, on the other hand, specify *only* patterns of message exchange. They model constraints on message sequencing in terms of a finite-state machine in which transitions correspond to the sending of a message by one or another of the participants. States correspond to the different stopping

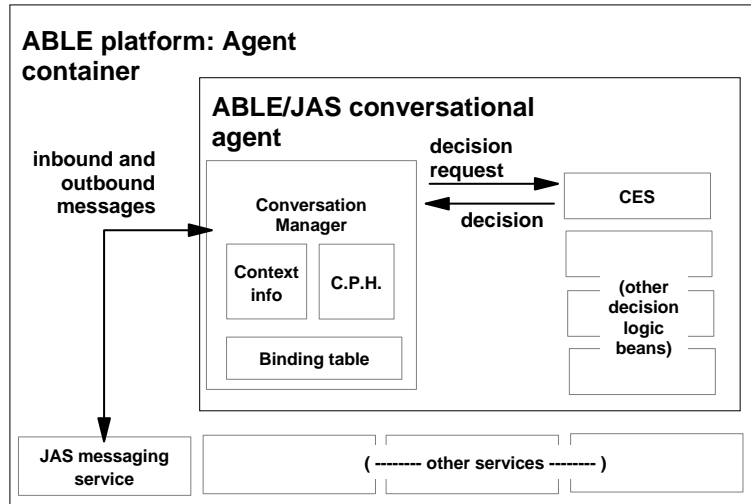


Figure 1: Conversational ABLE/JAS agent architecture, including CES utility bean.

points within a conversation, points where one of the agents needs to perform some computational activity, such as evaluating an offer just received, generating a counteroffer, etc.

Any particular conversation traces a path through the CP's state-transition graph, with the particular transition taken at each state selected by the decision logic of one or the other agent.

CPs are nestable, though this feature is not used extensively here. There is one level of nesting that is extremely useful, however: we implemented, as a conversation policy, the protocol governing both the initial setup of the conversational session and the process of selecting a conversation policy to use.

In ABLE, the conversation management is done by a Conversation Manager bean. Its function is to provide the connection between the Conversation Policy Handler (described below) and the agent as a whole. As shown in Fig. 1, the Conversation Manager contains, in addition to a Conversation Policy Handler (labelled "C.P.H."), a store of certain facts relevant to the conversation (labelled "Context") and a bindings that specify the way in which the decision logic is to be accessed.

The Context stores the agent's conversation ID, and that of its counterpart, as well as counterpart's name, message-delivery address, and possibly other information such as whether the other agent's identity has been authenticated, etc.

The Bindings table consists, in essence, of routing information which specifies, for each point in the conversation at which the agent can make a decision, which decision logic module should be called on, and the way it should be called. ABLE provides very general and flexible tools for representing binding information, extensively exploiting the Java introspection mechanism.

The execution of a conversation policy itself is carried out by a Conversation Policy Handler embedded in the Conversation Manager. The Conversation Policy Handler implemented in ABLE is a general-purpose finite-state machine, which can be

configured to execute any conversation policy specified in cpXML, an XML specification language for state-machine-based conversation policies which we are currently developing [11].

The Conversation Policy Handler operates as a sequential event processor, in which events such as inbound messages and decision-logic outputs are sequentially fed into it for processing. Each input is tested against the set of transitions from the conversation policy's current state, and if a transition matching the input is found, that transition is taken. Typically, an output event is generated when the transition is taken, such as an outbound message to be sent via the JAS messaging service or a decision request to be forwarded to the appropriate decision logic module. Since the bindings to decision logic and to the messaging service are stored in the Conversation Manager, the Conversation Policy Handler was implemented independently of the binding mechanism. This permits it to be deployed as-is in a wide range of agent architectures.

2.3 A typical conversational session

The first phase of a conversation is the setup of a session. Since conversations are inherently peer-to-peer, session management apparatus developed for client-server or web-based applications (such as J2EE session beans) is insufficient. Instead, the two agents much engage in a handshaking protocol in which each agent first creates a conversation identifier for its own use, and then exchanges that identifier with the other agent. Thereafter, whenever either agent sends a message, it includes the *recipient's* conversation identifier with the message. This permits the recipient to sort the message into the correct conversational inbox.

Once a conversation has been setup, the two agents exchange information about which conversation policy to use for the next part of the conversation. In its most simple form, the initiator of the conversation sends a proposal containing the name of a conversation policy and the role, defined within that policy), that the sender wishes to adopt. The other agent can then accept or reject the proposal. This information can be send along with the session setup information, or, as we have done here, in a separate message.

Finally, the two agents begin executing the policy they have agreed upon, each one taking the role it has agreed to. Each agent executes its side of the conversation policy in its own Conversation Manager, generating messages and making decisions as defined by the policy's state-transition structure. Typically, any given state is a decision-point for only one of the agents, i.e., the CP designates one of the roles as the sender of the next message, and determines the set of possible messages to send. By selecting one of the transitions exiting a decision point, the agent causes the Conversation Policy Handler to generate the appropriate message and pass it out to the Conversation Manager, which forwards it to the JAS messaging service for delivery.

The other agent, when it receives the message, feeds it in to the Conversation Policy Handler, which identifies the message that was sent, matches it to one of the transitions in the conversation policy, and takes that transition. Depending on the policy, this frequently causes this second agent's conversation policy to arrive at a decision point, which thereby causes the Conversation Manager to issue a decision request to the decision logic. Then the second agent makes a decision, causing its policy to make a transition and, typically, create and send a message to the first agent. This process repeats until the policy reaches a terminal state, with the order of which agent is "active" determined by the policy they are executing.

As stated above, we can take advantage of the nestability of conversation policies to implement the setup phase (both of the session and of the CP) as a conversation policy in its own right, that calls, in the appropriate place, the agreed-upon policy as a sub-conversation.

3 Asymmetric Protocol

In any human negotiation, there is at least an implicit protocol governing the form and sequence of communications between the parties, when agreement is reached, and when the negotiation terminates without agreement. For agent negotiations, the protocol needs to be explicit, and understood by both parties. As described previously in section 2, we advocate for agent negotiations the use of conversation policies, represented in the form of finite-state machines, that are agreed upon by the agents before beginning a negotiation.

A wide variety of negotiation protocols have been studied theoretically or used in practice. The most commonly used protocols are variants of the “alternating-offers” model, in which either party can propose an initial offer, and the parties then take turns responding to each other. A response to an offer can consist of: accepting the offer, rejecting the offer, proposing a counter-offer, or ending the negotiation.

Symmetric protocols such as alternating-offers may be appropriate when there is a fairly equivalent and symmetric relationship between buyer and seller. However, there are many situations in on-line commerce where the buyer and seller have highly asymmetric relationships. For example, Dell has a highly optimized business model, with precise knowledge of its profit function for any PC configuration it offers. It can also devote huge resources to modeling the behavior of PC buyers, and optimizing its negotiating strategy based on the buyer population model. On the other hand, many PC buyers would have difficulty in articulating their own utility function, let alone in developing a complex optimized bargaining strategy. While buyers may be effective at shopping for bargains amongst competing sellers, one wouldn’t expect them to devote the resources necessary to develop sophisticated negotiating strategies.

For situations as described above, we advocate the use of the following asymmetric negotiating protocol. In each round, the seller proposes a set of one or more offers to the buyer. The buyer then either accepts one of the offers, or rejects the entire set. After each round, one or both parties can decide to quit the negotiation. If neither party quits, the negotiation continues to the next round. Once an offer has been accepted, it is binding for both parties. However, the protocol also allows for further rounds of negotiation after the offer acceptance in an attempt to find mutually agreeable revisions. The seller can continue to propose offer sets to the buyer. If the buyer accepts any of these, it becomes the new accepted offer, otherwise the original accepted offer is still binding. Note that the protocol is parameter free in that neither the number of rounds nor the number of offers per round is specified in advance.

The above protocol is less onerous on buyer agents than requiring an ability to generate counter-offers. Buyer agents need only be able to distinguish acceptable from unacceptable offers, and to choose the most desirable offer from several acceptable alternatives. Combined with bargain-hunting amongst several sellers, this should enable buyers to do relatively well using fairly simple negotiating strategies. Seller agents need more sophisticated strategies, including modeling of the buyer population, but it should be worthwhile for them to invest in the development of such strategies.

We also point out a theoretical merit of the offer-improvement phase of this protocol. If the negotiation is required to terminate after rejection of an offer set in this phase, it then becomes incentive-compatible for the buyer to always accept any improving offer. It can then be shown that, with a sufficiently intelligent seller agent strategy, the protocol is guaranteed to find a Pareto-optimal agreement [10]. While we do not currently implement this protocol variant, due to limited intelligence of our seller agents, it can easily be implemented once more intelligent agents become available.

3.1 The conversation policy for the asymmetric protocol

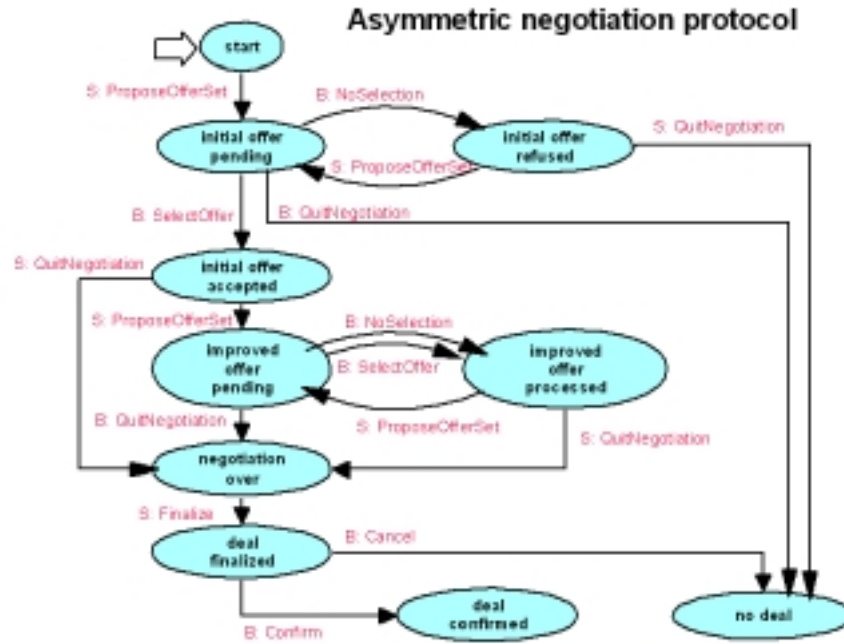


Figure 2: A conversation policy for asymmetric negotiation.

The conversation policy that implements the asymmetric negotiation protocol is shown in Fig. 2. Transitions between states in the conversation policy correspond to messages sent by one or the other agent, identified by *role*: either that of a buyer (labelled “B”) or seller (“S”). The states themselves can be thought of as labels identifying distinct points in the progress of the conversation. According to the sequence in which an agent sends and receives messages, the Conversation Policy Handler is driven along a path in the state-transition graph.

Depending on the role the agent is playing, any given state may or may not be a point at which the agent needs to make a decision. For example, when the conversation is in the state labelled “initial offer pending” the agent acting as buyer needs to decide whether to select one of the offers—and thereby send a “SelectOffer” message and make the transition to the “initial offer accepted” state—or alternatively to select none of them, send a “NoSelection” message and go to the “initial offer refused” state. (Note that the transition labels shown in the figure are shorthand for the detailed specifications of message formats defined in the actual conversation policy.)

At all such decision points, conversation policy therefore only defines the set of

options that constitute “legal” messages to send. It is the agent’s decision logic that selects which option to choose, and also supplies any additional data that must be sent in the body of the outgoing message.

4 Test scenario

4.1 Two-attribute goods and CES utility functions

We now consider negotiation of a two-attribute good described by a pair of integers (x_1, x_2) ranging from 0 to N . We assume that buyer valuations increase monotonically with attribute values, whereas seller valuations decrease monotonically. The attributes can be thought of as, for example, a quality parameter and a percentage discount off a list price, so that buyers prefer high quality and large discounts, whereas sellers prefer to offer low quality and small discounts.

We model buyer and seller preferences using CES (“Constant Elasticity of Substitution”) utilities: this is a parameterized family of functions widely used by economists to model a broad range of user preferences. For two attributes, CES buyer utilities take the form:

$$U = [\alpha_1 X_1^\rho + \alpha_2 X_2^\rho]^{1/\rho} - o$$

where $X_1 = x_1/N$ and $X_2 = x_2/N$ are normalized attribute values lying in the unit interval, o is a positive offset parameter, α_1 and α_2 are positive constants, and the exponent ρ is determined by a constant $r \in [0, 1)$ according to $\rho = (1 - 2r)/(1 - r)$. For seller agents, CES utilities have a similar form:

$$U = [\alpha_1(1 - X_1)^\rho + \alpha_2(1 - X_2)^\rho]^{1/\rho} - o$$

CES utilities have a number of important desirable properties, including the ability to model both substitutable ($r \rightarrow 0$) as well as complementary ($r \rightarrow 1$) attributes. Without loss of generality we may assume that $\alpha_1 + \alpha_2 = 1$.

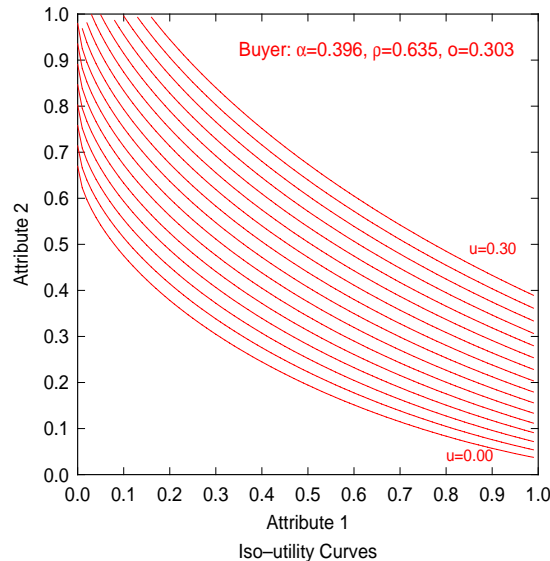


Figure 3: Iso-utility curves for a sample buyer with α, ρ, o values as indicated.

In the following experiments we generate a distribution of buyer and seller agent utility functions using uniform random values of $\alpha (= \alpha_1) \in [0, 1]$, $r \in [0, 1]$. The offset o is uniform random in $[0, 1.25]$ for buyers: this results in most buyers having positive utility for (N, N) , whereas no buyer has positive utility for $(0, 0)$. For sellers, o is uniform random in $[0, 0.25]$; this results in a theoretically acceptable deal existing for most buyer-seller pairs.

An example CES buyer utility function is plotted in figure 3.

4.2 Zero-Intelligence Negotiating Strategy

Computation of game-theoretic equilibrium strategies for our asymmetric protocol is intractable, and thus the agents must be programmed to employ heuristic strategies. We have chosen to implement the simplest possible “Zero-Intelligence” (ZI) negotiating strategies. We do not seriously advocate the use of ZI for real negotiations; indeed, we expect ZI to be crushed by opponents with greater intelligence. Rather, the purpose of using ZI is to test the viability and usefulness of our asymmetric protocol. In the continuous double auction, ZI agents were found to obtain surprisingly high efficiency and reliable convergence to equilibrium trading [9]. This suggested that desirable behaviors seen in many financial markets may to a large extent be due to market rules such as spread-improvement, rather than due to the intelligence of human traders. We likewise suggest that if ZI agents perform well in a given negotiation protocol, then the protocol should also give good results using more intelligent negotiating strategies.

ZI buyer agents in our asymmetric protocol have extremely simple behavior: they reject any offer with negative or zero utility, and accept any offer with utility $u_b > 0$ if there is no accepted offer, or $u_b > u_b^*$ if there is a current accepted offer with utility u_b^* . If there are multiple acceptable offers within an offer set, the buyer will choose the one with highest utility. The only adjustable parameter in the ZI buyer strategy is a patience parameter M indicating the maximum number of rounds that the agent will continue negotiating.

Seller agents send an offer set containing s offers each round to the buyer. If there is no accepted offer, ZI seller agents generate random offers uniformly from the entire offer space, eliminating those with negative or zero utility. Once there is an accepted offer, with utility u_s^* , the seller generates offers by perturbing each attribute of the accepted offer by a uniform random change in $[-\delta, +\delta]$. Any perturbed offer with utility $u_s > u_s^*$ is then sent to the buyer. We assume for convenience that the seller’s patience parameter is greater than or equal to the buyer patience, so that the maximum negotiation length is governed by the buyer. The ZI seller strategy thus has two adjustable parameters: the size of offer set s , and the perturbation radius δ .

An example of negotiating behavior obtained with ZI buyers and sellers is shown in figure 4. The buyer and seller iso-utility curves are superimposed; the mutually acceptable offers are enclosed by $u_s = 0$ curve for the seller and the $u_b = 0$ curve for the buyer. The dark line is the contract curve, i.e. the set of Pareto-optimal deals. The accepted offers in two sample negotiations are plotted with light blue dots. In each negotiation, the initial accepted offer is at the lower right, and the sequence of improved deals moves in small steps up and to the left, towards the contract curve. The final accepted offers end up extremely close to the contract curve.

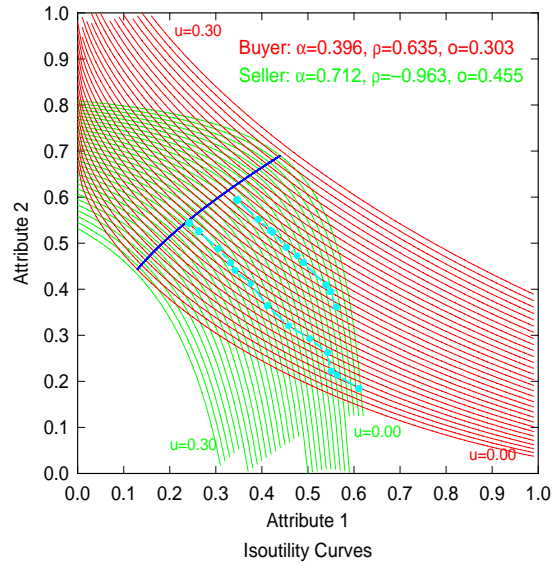


Figure 4: Trajectories of accepted offers in two sample negotiations between a specific ZI buyer and ZI seller, plotted with light blue dots. The trajectories start at the lower right and move towards the Pareto-optimal deal set, indicated by the dark blue line.

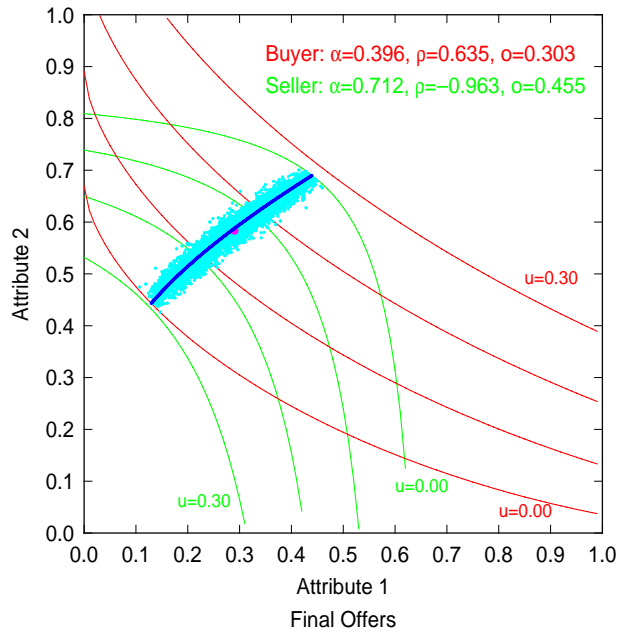


Figure 5: Distribution of final accepted offers for a particular ZI buyer and ZI seller. The dark line indicates Pareto-optimal offers.

5 Results

An illustration of typical negotiating results using ZI agents is shown in figure 5, which plots the distribution of final agreements for a particular buyer-seller pair. All of the final agreements are close to the Pareto-optimal contract curve, and a significant fraction of agreements lie directly on the curve.

We have performed a number of experiments with ZI agents, varying the patience parameter M and the offer set size s . In examining our results, we are interested issues such as: how often the agents actually reach theoretically achievable deals, how close to Pareto-optimality the agents' deals are, and how well buyers do relative to sellers. Each of the experimental data points represents an average of 1000 negotiations, using a different randomly generated buyer and seller agent in each negotiation. The seller's revision radius parameter was set to $\delta = 0.05$.

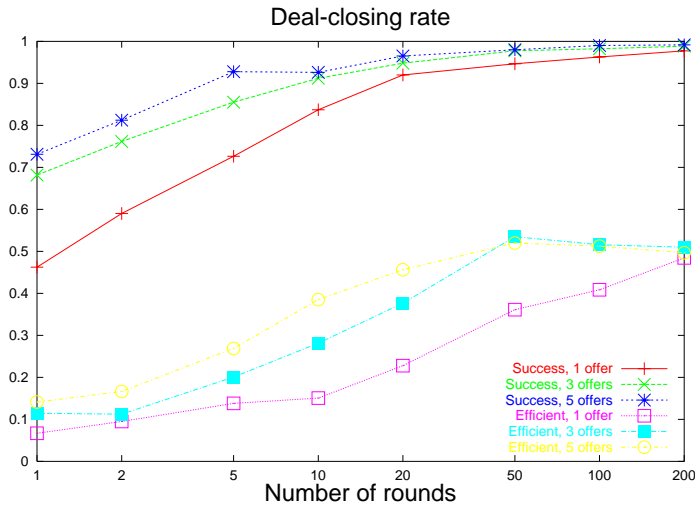


Figure 6: Fraction of theoretically achievable deals that were actually made (upper curves), and made at Pareto-optimal points (lower curves), vs. number of rounds and number of offers per round, as indicated.

Figure 6 examines the agents' deal-closing rate as a fraction of total theoretically achievable deals. The upper curves plot all agent deals, while the lower curves plot only Pareto-optimal deals. We note that both rates improve as M increases, and as s increases. For 20-round negotiations, the deal-closing rate exceeds 90%, while at $M = 50$ and $s \geq 3$, about half the deals are Pareto-optimal.

For further insight into the quality of agent deals, we define a “loss of efficiency” measure as follows. If the deal is Pareto-optimal, the efficiency loss is zero. For non-Pareto-optimal deals, or for deals that could have been made but were not, we define the loss of efficiency as the maximum total improvement in buyer utility plus seller utility that could have been made over the Pareto-optimal set. Loss of efficiency results are plotted in figure 7. For comparison we plot the maximum efficiency loss that would be obtained if no deals were ever reached. On this scale we see that the actual efficiency losses are quite small for $M \geq 50$ rounds and $s \geq 3$ offers per round.

Finally we plot the relative performance of buyers and sellers in figure 8. The per-

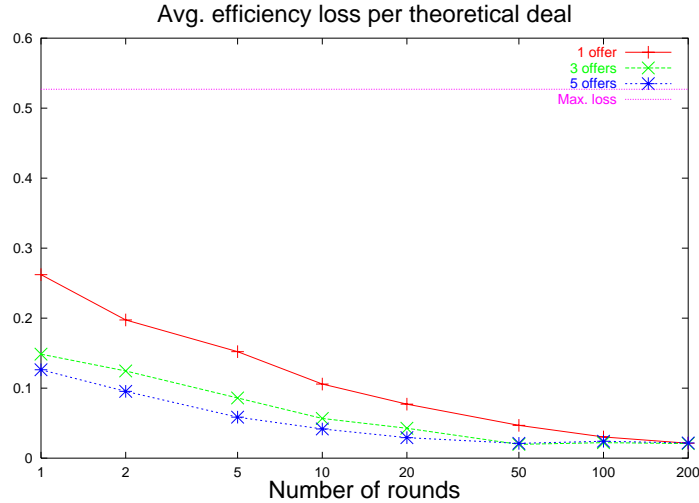


Figure 7: Loss of efficiency, as defined in the text, as a function of number of rounds and number of offers per round. The upper line indicates the maximum possible efficiency loss obtained when no deals are made.

formance measure for buyers is the total buyer utility obtained in the experiments, divided by the maximum utility that could have obtained had every agreement occurred at the Pareto-optimal contract point with maximum buyer utility. A similar measure is used for sellers. We see that both buyers and sellers increase their utility as the number of rounds increase. Sellers achieve the best results with one offer per round, while buyers do better with five offers per round. This is as we expected, since with more offers per round, buyers have greater opportunity to increase their utility, and this tends to decrease seller utility since the good’s attributes are purely competitive.

6 Discussion

We suggest that our implemented system for agent negotiation may be of practical use in two ways. First, the ABLE/JAS/CP architecture conveniently automates and modularizes many important aspects of real-world negotiations. The negotiation protocol is expressed as data in machine-readable XML; the Conversation Manager automates message handling given the protocol, and the agent can incorporate a variety of negotiating strategies, expressed as decision logic beans.

By adopting the JAS messaging API, this architecture insulates the agents from the details of message transport plumbing. By setting communications into an explicit conversational context, it provides general support for stateful multi-step interactions. By providing a general-purpose execution engine into which any conversation policy can be fed, it permits the evolution of de-facto standard conversation policies for general use, while still supporting the needs of specialized sub-populations of agents to use custom conversation policies adapted to their particular needs. Finally, by keeping the decision logic separate from protocol management, it avoids limiting the scope of innovation in the decision-making software—though because it is built on ABLE, a wide range of advanced AI tools is available for use.

Second, our system may also be useful as a research platform for developing and

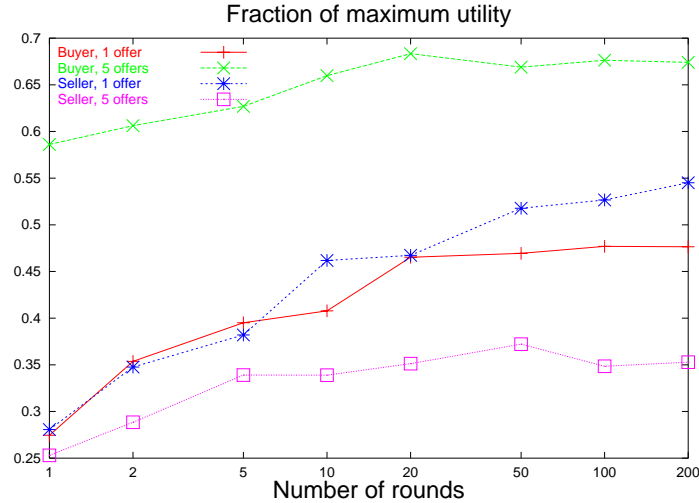


Figure 8: Average buyer and seller utility, divided respectively by the maximum obtainable utility per deal, as a function of number of rounds, and number of offers per round.

testing new negotiating protocols and strategies. As yet there is no single protocol that stands out as being clearly superior to all others on theoretical grounds. Thus, to a large extent, evaluation of the merits any given protocol may need to be performed empirically. Users may also have other motivations for adopting a given protocol beyond theoretical principles such as Pareto-optimality. It seems likely that a modular platform where many different protocol and decision logics can easily be plugged in and tried out would be of great use to many researchers in multi-agent systems. We have therefore made our system available for downloading for research use as part of the ABLE 1.0 download, available from the IBM Alphaworks web site [1].

In future work, we plan to carry out further tests of the asymmetric negotiating protocol, using more intelligent agent strategies. Recent studies of a more sophisticated seller agent based on Bayesian inferencing suggest that highly efficient agreements can be achieved on much shorter timescales [18]. These agreements are highly skewed in favor of the seller, and it is an open question whether efficient agreements can still be reached quickly when both buyer and seller employ sophisticated strategies.

References

- [1] Alphaworks. <http://www.alphaworks.ibm.com>.
- [2] Kasbah. <http://www.ecommerce.media.mit.edu/Kasbah>.
- [3] MARI. <http://www.media.mit.edu/gtewari/MARI>.
- [4] Workshop on agent languages and conversation policies, 2002. At the 2002 Autonomous Agents and Multiagents Systems conference (AAMAS-02), Bologna, Italy. <http://www.csc.liv.ac.uk/mph/ALC2002>.
- [5] ABLE: Agent Building and Learning Environment Project. <http://www.research.ibm.com/able>.

- [6] F. Dignum and U. Cortes, editors. *Agent-Mediated Electronic Commerce III*. Springer, Berlin, 2001.
- [7] P. Faratin, N. R. Jennings, P. Buckle, and C. Sierra. Automated negotiation for provisioning virtual private networks using FIPA-compliant agents. In *Proceedings of 5th Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM-2000)*, pages 185–202, Manchester, UK, 2000.
- [8] Foundation for Intelligent Physical Agents. <http://www.fipa.org>.
- [9] D. K. Gode and S. Sunder. Allocative efficiency of markets with zero intelligence traders: Market as a partial substitute for individual rationality. *Journal of Political Economy*, 101(1):119–137, Feb. 1993.
- [10] A. Gulcu, R. Mohan, and A. Zarovnyi. Efficiency of contracts in multi-attribute two-party negotiations. Technical report, IBM Research, 2002.
- [11] J. E. Hanson and P. Nandi. cpXML: Conversation Policy XML. in preparation, 2002.
- [12] J. E. Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. In *Proc. 6th IEEE International Conf. on Enterprise Distributed Object Computing (EDOC 2002)*, 2002. to appear.
- [13] Java Agent Services specification. <http://www.java-agent.org>.
- [14] K. Larson and T. Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001.
- [15] R. B. Myerson and M. A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29:265–281, 1983.
- [16] J. F. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.
- [17] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):155–162, 1982.
- [18] G. Tesauro. Efficient search techniques for multi-attribute bilateral negotiation strategies. In *Proceedings of the Third International Symposium on Electronic Commerce (ISEC-02)*, pages 30–36. IEEE Computer Society, 2002.