Collaborative Interface Agents

Yezdi Lashkari
MIT Media Laboratory,
Cambridge, MA 02139
yezdi@media.mit.edu

Max Metral
MIT Media Laboratory,
Cambridge, MA 02139
memetral@media.mit.edu

Pattie Maes
MIT Media Laboratory,
Cambridge, MA 02139
pattie@media.mit.edu

Abstract

Interface agents are semi-intelligent systems which assist users with daily computer-based tasks. Recently, various researchers have proposed a learning approach towards building such agents and some working prototypes have been demonstrated. Such agents learn by 'watching over the shoulder' of the user and detecting patterns and regularities in the user's behavior. Despite the successes booked, a major problem with the learning approach is that the agent has to learn from scratch and thus takes some time becoming useful. Secondly, the agent's competence is necessarily limited to actions it has seen the user perform. Collaboration between agents assisting different users can alleviate both of these problems. We present a framework for multi-agent collaboration and discuss results of a working prototype, based on learning agents for electronic mail.

Introduction

Learning interface agents are computer programs that employ machine learning techniques in order to provide assistance to a user dealing with a particular computer application. Although they are successful in being able to learn their user's behavior and assist them, a major drawback of these systems is the fact that they require a sufficient amount of time before they can be of any use. A related problem is the fact that their competence is necessarily restricted to situations similar to those they have encountered in the past. We present a collaborative framework to help alleviate these problems. When faced with an unfamiliar situation, an agent consults its peers who may have the necessary experience to help it.

Previous interface agents have employed either end-user programming and/or knowledge engineering for knowledge acquisition. For example, (Lai, Malone, & Yu 1988) have "semi-autonomous agents" that consist of a collection of user-programmed rules for processing information related to a particular task. The problems with this approach are that the user needs to recognize the opportunity for employing an agent, take the initiative in programming the rules, endow this agent with explicit knowledge (specified in an abstract language), and maintain the rules over time (as habits change etc). The knowledge engineered approach on the other hand, requires a knowledge engineer to outfit an interface with large amounts of knowledge about the application and the domain and how it may contribute to the user's goals. Such systems require a large amount of work from the knowledge engineer. Furthermore, the knowledge of the agent is fixed and cannot be customized to the habits of individual users. In highly personalized domains such as electronic mail and news, the knowledge engineer cannot possibly anticipate how to best aid each user in each of their goals.

To address the problems of the rule-based and knowledge-engineered approaches, machine learning techniques have been employed by (Kozierok & Maes 1993; Maes & Kozierok 1993; Hermens & Schlimmer 1993; Dent *et al.* 1992) and others. In the Calendar Agent (Kozierok & Maes 1993), memory-based reasoning is combined with rules to model each user's meeting scheduling habits. Results described in (Kozierok & Maes 1993) show that the learning approach achieves a level of personalization impossible with knowledge engineering, and without the user intervention required by rule-based systems. It is also interesting to note that the addition of rules provides the flexibility to explicitly teach the agent, and shows that the rule-based and learning approaches can successfully coexist.

While the learning approach enjoys several advantages over the others, it has its own set of deficiencies. Most learning agents have a slow 'learning curve' ; that is, they require a sufficient number of examples before they can make accurate predictions. During this period, the user must operate without the assistance of the interface agent. Even after learning general user behavior, when completely new situations arise the agent may have trouble dealing with them. The agents of different users thus have to go through similar experiences before they can achieve a minimal level of competence, although there may exist other agents that already possess the necessary experience and confidence.

We propose a collaborative solution to these problems. Experienced agents can help a new agent come up to speed quickly as well as help agents in unfamiliar situations. The framework for collaboration presented here allows agents of different users, possibly employing different strategies (rule-based, MBR, CBR, etc.) to cooperate to best aid their individual users. Agents thus have access to a much larger body of knowledge than that possessed by any individual agent. Over time agents learn to trust the suggestions of some of their peers more than others for various classes of situations. Thus each agent also learns which of its peers is a reliable 'expert' vis-a-vis its user for different types of situations.

A Single User's Agent

This paper describes experiments conducted with implemented interface agents for the electronic mail domain for a commercial email application, Eudora (Dorner 1992). This section describes an individual email agent.

Each user's interface agent learns by continuously "looking over the shoulder" of the user as the user is performing actions. The interface agent monitors the actions of the user over long periods of time, finds recurrent patterns and offers to automate them. For example, if an agent notices that a user almost always stores messages sent to the mailing-list "genetic-algorithms" in the folder *AI Mailing Lists*, then it can offer to automate this action next time a message sent to that mailing list is encountered. The agent can also automate reading, printing, replying, and forwarding as well as assign priority to messages.

We have chosen Memory Based Reasoning (Stanfill & Waltz 1986) as the algorithm which attempts to capture user patterns. Our implementation of MBR is based upon the concepts of situations and actions. In the electronic mail domain, we choose mail messages along with some context information to represent situations and the user's handling of the messages as actions. At any particular point in time, the user may be presented with a number of messages. When the user takes an action, it is paired with the corresponding situation and the situation-action pair is recorded in the agent's memory. For example, if the user reads a message $\mathcal{M}$, the pair $< \mathcal{M}'$, read-action$>$ is memorized, where $\mathcal{M}'$ contains details about the message $\mathcal{M}$ and relevant context information (for example that $\mathcal{M}$ was read n^{th} out of a total of k unread messages). When new situations occur, they are compared to the situations previously encountered. After gathering the closest matching situations in memory, the agent can calculate a prediction for an action in the new situation. In addition, the agent can calculate a confidence in its prediction by considering such factors as the number of situations in its memory and the proximity of the culled situations to the new situation. For a more detailed description, see (Kozierok & Maes 1993).

A situation is specified in terms of a set of fields. MBR measures situation proximity by applying a weighted sum of the distance between the corresponding fields of two situations. In the e-mail domain, appropriate fields would be the originator of the message, the subject, etc. The values of fields may be of any type. In previous systems, these fields were mainly strings or other static values. In our implementation, field values can also be objects. These objects can in turn have fields, which may be used in predicting actions. For example, the originator of a message is a *Person* object, which contains fields such as that person's position in an organization and their relation to the user. Object-based MBR is much less brittle than traditional MBR systems and can also use extra knowledge present in the objects if it finds it to be useful. For example, let's say that Mary always reads all messages from her boss Kay. If Mary were to suddenly receive a message from Kay's boss (therefore also Mary's boss), the system will correctly suggest that Mary read the message, since it uses the knowledge that Mary reads everything from her boss (and therefore probably her boss's boss too) although it has never previously received a message from Kay's boss. [1] Thus object-based MBR allows the same situation to be viewed differently depending on what information is available. In contrast, a string-based MBR system does not possess the same flexibility since we cannot extract more features from the string.

After predicting an action for a given situation, the agent must decide how to use that prediction. For each possible action, the user can set two confidence thresholds: the *tell-me* threshold and the *do-it* threshold. If the confidence in a prediction is above the tell-me threshold, the email agent displays the suggestion in the message summary line. If the confidence is above the do-it threshold, the agent autonomously takes the action.

The agent's confidence in its predictions grows with experience, which gives the user time to learn to trust the agent. During this period, it is especially useful to give the user the opportunity to see exactly what the agent is doing. This feedback is accomplished in three ways: an activity monitor, an explanation facility, and an interface to browse and edit the agent's memory. The activity monitor presents a small caricature to the user at all times. The caricature depicts states such as alert, thinking, and working, similar to (Kozierok & Maes 1993). An explanation facility provides English descriptions of why the agent suggested an action.

An agent starts out with no experience. As messages arrive and its user takes action, its memory grows. Only after a sufficient number of situation-action pairs have been generated, is the agent able to start predicting patterns of behavior confidently and accurately.

[1] Information about Kay and her boss are retrieved from a knowledge base of the kind maintained by most corporations or university academic departments.

However, when it encounters a new situation that is unlike anything it has in its memory, it is still unsure of what to do. This is because the machine learning algorithm used requires the training examples to cover most of the example space to work effectively.

A Framework For Collaboration

We propose a collaborative solution to the problems above. While a particular agent may not have any prior knowledge, there may exist a number of agents belonging to other users who do. Instead of each agent re-learning what other agents have already learned through experience, agents can simply ask for help in such cases. This gives each agent access to a potentially vast body of experience that already exists. Over time each agent builds up a trust relationship with each of its peers analogous to the way we consult different experts for help in particular domains and learn to trust or disregard the opinions of particular individuals.

Collaboration and communication between various agents can take many different forms. This paper is only concerned with those forms that aid an agent in making better predictions in the context of new situations. There are two general classes of such collaboration.

Desperation based communication is invoked when a particular agent has insufficient experience to make a confident prediction. For example, let us suppose that a particular agent A_1 has just been activated with no prior knowledge, and its user receives a set of new mail messages. As A_1 doesn't have any past experience to make predictions, it turns in desperation to other agents and asks them how their user would handle similar situations.

Exploratory communication, on the other hand, is initiated by agents in bids to find the best set of peer agents to ask for help in certain classes of situations. We envisage future computing environments to have multitudes of agents. As an agent has limited resources and can only have dealings with a small number of its peers at a given time, the issue of which ones to trust, and in what circumstances, becomes quite important. Exploratory communication is undertaken by agents to discover new (as yet untried) agents who are better predictors of their users' behaviors than the current set of peers they have previously tested.

Both forms of communication may occur at two orthogonal levels. At the situation level, desperation communication refers to an agent asking its peers for help in dealing with a new situation, while exploratory communication refers to an agent asking previously untested peers for how they would deal with old situations for which it knows the correct action, to determine whether these new agents are good predictors of its user's behavior. At the agent level, desperation communication refers to an agent asking trusted peers to recommend an agent that its peers trust, while exploratory communication refers to agents asking peers for their evaluation of a particular agent perhaps to see how well these peers' modelling of a particular agent corresponds with their own. Hence agents are not locked into having to turn for help to only a fixed set of agents, but can pick and choose the set of peers they find to be most reliable.

For agents to communicate and collaborate they must speak a common language as well as follow a common protocol. We assume the existence of a default ontology for situations in a given domain (such as electronic news, e-mail, meeting scheduling, etc). Our protocol does not preclude the existence of multiple ontologies for the same domain. This allows agent creators the freedom to decide which types of ontologies their agents will understand. As the primary task of an agent is to assist its particular user, the protocol for collaboration is designed to be flexible, efficient and non-binding. We briefly present the protocol below.

- **Registration:** Agents wishing to help others register themselves with a "Bulletin Board Agent" whose existence and location is known to all agents. While registering, agents provide information regarding how they can be contacted, what standard domains they can provide assistance in, what ontologies they understand and some optional information regarding their user. Every agent registering with a bulletin board agent is given a unique identifier by the bulletin board.
- **Locating peers:** Agents wishing to locate suitable peers may query bulletin board agents. An agent querying a bulletin board agent need not itself register with that bulletin board. Queries to a bulletin board agent can take many different forms depending on the type of information required. This allows agents to locate suitable peers in the most convenient way. For example, an agent's user may explicitly instruct it to ask a specific user's agent for help in dealing with certain types of situations.
- **Collaboration:** Collaborative communication between agents occurs in the form of request and reply messages. An agent is not required to reply to any message it receives. This leaves each agent the freedom to decide when and whom to help. Any request always contains the agent's identifier, the agent's contact information (for replies), the ontology used in the request, and a *request identifier* (*reqid*) generated by the agent issuing the request. The *reqid* is necessary since an agent may send out multiple requests simultaneously whose replies may arrive out of order.

  Analogously every reply always contains the replying agent's identifier and the reqid used in the request.

  The types of requests and their associated replies are presented below.

  - **Situation level collaboration:** When a situation occurs for which an agent does not have a good prediction, it sends off a Request-for-Prediction message to its peers. A prediction request contains all the features of the situation which the agent issuing the request wishes to divulge. This allows the requesting agent the freedom to withhold sensitive or private information.

An agent receiving a prediction request may choose to ignore it for any of a variety of reasons. It may not have a good prediction for the specific situation, it may be too busy to respond, the agent issuing the request may not have been very helpful in the past, or the agent may not be important enough. If however, an agent decides to respond to a prediction request, it sends back a response containing its prediction and its confidence in this prediction (a normalized value).

Note that the prediction request is used by agents for both desperation and exploratory communication. An agent receiving a prediction request does not know whether the request originated via exploratory or desperation based behavior on the part of the agent issuing that request. The distinction is made by the agent issuing the request. Replies to requests sent in desperation are used to predict an action for a particular situation, while replies to requests sent in exploratory mode are compared with the actions that the user *actually took* in those situations, and are used to model how closely a peer's suggestions correspond with its user's actions.

– **Agent level collaboration:** An agent may send its peers a Request-for-Evaluation request. An evaluation request is sent when an agent wants to know what some of its peers think about a certain agent in terms of being able to model their users in particular classes of situations. An evaluation request contains the identifier of the agent to be evaluated (designated as the *target agent*) and the particular class of situations for which the evaluation is needed.

  In any domain and ontology there exist different classes of situations. Certain agents may model a particular user's behavior in a particular class of situations very well and fail miserably in other classes. Note that we expect the domain ontologies to define these classes. For example an email agent may discover that peer agent A_1 is a very good predictor of its user's actions for messages sent to a mailing list, while being quite useless in predicting what its user does with any other type of message. On the other hand peers A_2 and A_3 are excellent predictors of its user's behavior with regards to email forwarded by her groupmates. This enables agents to locate and consult different 'expert' peers for different classes of situations.

  An agent that chooses to respond to an evaluation request sends back a normalized value which reflects its *trust* in the target agent's ability to model its user's behavior for that particular class of situations.

  An agent may also ask trusted peer agents to recommend a peer who has been found to be useful by the trusted peer in predicting its user's behavior for a particular class of situation. A Request-for-Recommendation contains the situation class for which the agent would like its trusted peer to recommend a good agent. Replies to recommendation requests contain the identifier and contact information of the agent being recommended.

Agents model peers' abilities to predict their user's actions in different classes of situations by a *trust* value. For each class of situations an agent has a list of peers with associated trust values. Trust values vary between 0 and 1.

The trust values reflect the degree to which an agent is willing to trust a peer's prediction for a particular situation class. A trust value represents a probability that a peer's prediction will correspond with its user's action based on a prior history of predictions from the peer. Agents may start out by picking a set of peers at random or by following their user's suggestion as to which peer agents to try first. Each previously untested peer agent gets has its trust level set to an initial value. As a peer responds to a prediction request with a prediction p, and an agent's user takes a particular action a, the agent updates the trust value of its peer in the appropriate situation class as follows:

$$trust = \quad clamp(0, 1, trust + \delta_{p,a} * (\gamma * trust * conf))$$

where

$$\delta_{p,a} = \begin{cases} +1 & \text{if prediction p = user action a} \\ -1 & \text{if prediction p} \neq \text{user action a} \end{cases}$$

and *trust* represents the trust level of a peer, *conf* represents the confidence the peer has in this particular prediction, γ is the trust learning rate, and $clamp(0, 1, c)$ ensures that the value of c always lies in $(0, 1]$. The rationale behind the modelling above is as follows. An agent's trust in a peer rises when the peer makes a correct prediction and falls for incorrect predictions. The amount it rises and falls by depends on how confident the peer was in its prediction. That is, a peer who makes an incorrect prediction with a high confidence value should be penalized more heavily than one that makes an incorrect prediction but with a lower confidence value.

When an agent sends out a prediction request to more than one peer it is likely to receive many replies, each with a potentially different prediction and confidence value. In addition, the agent has a trust value associated with each peer. This gives rise to many possible strategies which an agent can use to choose a prediction and a confidence value for this prediction. We believe that both trust and peer confidence should play a role in determining which prediction gets selected and with what confidence. Each predicted action is assigned a *trust-confidence sum* value which is the trust weighted sum of the confidence values of all the peers predicting this action. The action with the highest trust-confidence sum is chosen. The confidence associated with the action chosen is currently that of the most confident peer suggesting this action. We are exploring more sophisticated trust-confidence combination strategies using decision theoretic and Bayesian strategies.

Experimental Results

The concepts above have been implemented for a commercial electronic mail handler (Eudora) for the Apple Macintosh. The agent, implemented in Macintosh Common Lisp, communicates with the mail application using the AppleEvent protocol. The MBR Engine is domain independent, and can be easily adapted to calendar applications or news readers. Furthermore, all of these applications can share fields and actions. As more applications implement an AppleEvent interface, the agent should be able to aid the user with these applications as well. Currently, several users are actively making use of the agent on their actual mail. While the computations are intensive, we have achieved satisfactory performance on most high end Macintoshes.

The performance of MBR in interface agents has been documented in (Kozierok & Maes 1993). We wish to show that multi-agent collaboration strictly improves upon results obtained from single agent systems. Namely, multi-agent collaboration should steepen the learning curve and improve the handling of entirely novel messages.

To illustrate this, we set up the following scenario using the actual e-mail of two graduate students over a three day period (approximately 100 messages per user).

Calvin and Hobbes are two graduate students in the Intelligent Agents group.

1. **Hobbes :** Hobbes has been around for some time and hence his agent is quite experienced. Hobbes' agent has noted the following trends in its user's behavior. All messages to 'bpm', a music mailing list are refiled to a folder called bpm for later reading. Messages directly addressed to Hobbes are read by him and then deleted, as are messages to other mailing lists.
2. **Calvin :** Calvin is a new graduate student in the group. Calvin's agent starts out with absolutely no experience. Calvin also refiles all messages from the 'bpm' list for later perusal. He deletes subscription requests sent to the list. Calvin reads messages directly addressed to him, and then refiles them to appropriate folders. The rest of his mail, such as messages to other lists, he reads and deletes.

We plotted the confidence of Calvin's agent's suggestions as Calvin takes actions on about 100 actual mail messages. Figure 1 shows the results obtained. The x-axis indicates the growing experience of Calvin's

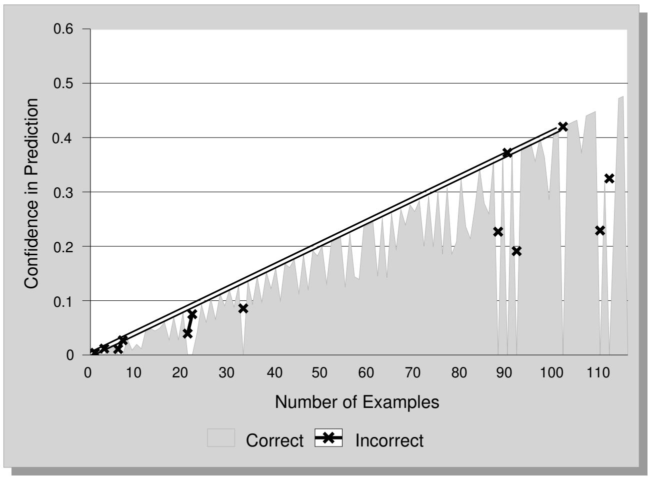

Figure 1: Performance without Collaboration

agent as Calvin takes successive actions on the mail messages and the number of situation-action pairs in the memory increases. The thick rising *trend line* indicates how Calvin's agent's performance (in terms of confidence in predictions) rises slowly with experience. The numerous pockets show new user behavior being modeled. The agent makes several mistakes very early, which is to be expected, since the situations it has in its memory early on do not effectively capture all of Calvin's behavior patterns. Towards the end, we see several more mistakes, which reflect a new pattern occuring. With the tell-me threshold for all actions set at 0.1, the graph shows that it will take approximately 40 examples for the agent to gain enough confidence to consistently have suggestions for the user.

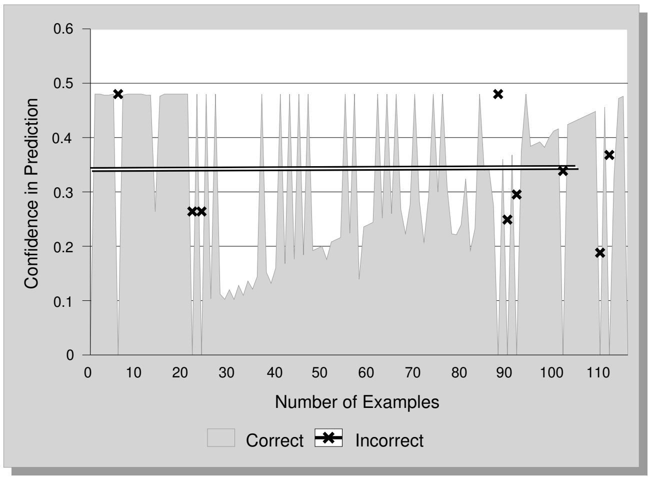

Figure 2: Performance with Collaboration

Figure 2 shows the level of confidence of Calvin's agent in its suggestions with multi-agent collaboration. [2] It may be noted that the confidence levels of all correct suggestions are always greater than the confidence levels generated by Calvin's agent alone at any point. The thick horizontal *trend line* indicates that multi-agent collaboration enables an inexperienced agent to make accurate predictions with high confidence as soon as it is activated as well as fill in gaps in even an experienced agent's knowledge. Note that trust modelling of Hobbes' agent is taking place inside Calvin's agent with each action Calvin takes on his mail. Space restrictions preclude the presentation of results regarding trust modelling of multiple peers in this paper.

Related Work

Various types of learning interface agents have been implemented (Kozierok & Maes 1993; Maes & Kozierok 1993; Hermens & Schlimmer 1993; Dent *et al.* 1992). All of them are essentially designed to act in a stand-alone fashion or engage in restricted task specific communication with identical peers. Our agents not only come up to speed much faster, but also discover which of a large set of heterogeneous peers are useful consultants to know in particular domains.

Multi-Agent Systems research has concentrated on negotiation and cooperation strategies that are used by autonomous agents who must compete for scarce resources. Various formal protocols and frameworks have been proposed to model agent's intentions, domains and negotiation strategies (Zlotkin & Rosenschein 1993; Rosenschein & Genesereth 1985) based on various game-theoretic, logical, economic and speech-act models. While the analytic frameworks above are important, most are based on restrictive assumptions about the domain or the agents' capabilities and assume that the reason agents cooperate is because they need access to a shared resource or have multiple overlapping goals.

The Ontolingua tools (Gruber 1993) and the work on the KQML Agent-Communication Language (Finin *et al.* 1993) provide a way for agents using different ontologies to communicate effectively with each other and may be used to implement our collaborative architecture. Our research represents an actually implemented system in a real domain that shows the benefits of collaboration amongst agents.

Conclusions

We have implemented a learning interface agent for a commercial application in a real world domain, and have tested it with real world data. Results have shown that multi-agent collaboration steepens the agent's learning curve, and helps in new, unseen situations. Trust modeling allows each agent to build a model of each agent's area of expertise, and consult only those agents which will be useful for each area.

[2] Hobbes takes no actions on his mail for the duration of this experiment, hence his agent's confidence remains unchanged.

Acknowledgments

This research was sponsored by grants from Apple Computer Inc. and the National Science Foundation under grant number IRI-92056688.

References

Dent, L.; Boticario, J.; McDermott, J.; Mitchell, T.; and Zabowski, D. 1992. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 96–103. San Jose, California: AAAI Press.

Dorner, S. 1992. *Eudora Reference Manual.* Qualcomm Inc.

Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzson, R.; McKay, D.; McGuire, J.; Pelavin, R.; Shapiro, S.; and Beck, C. 1993. Specification of the KQML agent-communication language. Technical Report EIT TR 92-04 (Revised June 15, 1993), Enterprise Integration Technologies, Palo Alto, CA.

Gruber, T. 1993. A translation approach to portable ontology specification. *Knowledge Acquisition* 5(2):199–220.

Hermens, L., and Schlimmer, J. 1993. A machine learning apprentice for the completion of repetitive forms. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications*, 164–170. Orlando, Florida: IEEE Press.

Kozierok, R., and Maes, P. 1993. A learning interface agent for scheduling meetings. In *Proceedings of the ACM SIGCHI International Workshop on Intelligent User Interfaces*, 81–88. Orlando, Florida: ACM Press.

Lai, K.; Malone, T.; and Yu, K. 1988. Object lens: A spreadsheet for cooperative work. *ACM Transactions on Office-Information Systems* 5(4):297–326.

Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 459–465. Washington D.C.: AAAI Press.

Rosenschein, J., and Genesereth, M. 1985. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 91–99. Los Angeles, CA: Morgan Kaufmann.

Stanfill, C., and Waltz, D. 1986. Toward memory-based reasoning. *Communications of the ACM* 29(12):1213–1228.

Zlotkin, G., and Rosenschein, J. 1993. A domain theory for task oriented negotiation. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 416–422. Chambery, France: Morgan Kaufmann.