Kasbah: An Agent Marketplace for Buying and Selling Goods

Anthony Chavez Pattie Maes
MIT Media Lab
20 Ames Street
Cambridge, MA 02139
asc/pattie@media.mit.edu

Abstract

While there are many Web services which help users find things to buy, we know of none which actually try to automate the process of buying and selling. Kasbah is a virtual marketplace on the Web where users create autonomous agents to buy and sell goods on their behalf. Users specify parameters to guide and constrain an agent's overall behavior. A simple prototype has been built to test the viability of this concept.

Introduction

Kasbah is a Web-based system where users create autonomous agents that buy and sell goods on their behalf.

Our original idea was to reinvent the classified ads. We observed that there are many sites on the Web that post ads. Some of these sites allow users to perform keyword searches on the ads (Seattle Times/PI 1996). Some have their ads nicely categorized, making it easier to find ones of interest (ADWorld 1996). Other sites have more advanced searching capabilities. For example, Stanford's Infomaster (Infomaster 1995) (Geddis et al. 1995) allows users to fill out a form that precisely describes the kind of apartment they are looking for. Infomaster searches the ad databases of several local newspapers and returns those which describe apartments that match the user's specification.

These "classified ad" sites all provide tools to help the user find ads of interest. Certainly, such tools are useful. Yet they only assist with one step in the multi-step process of buying and selling, that of finding ads which match what one is looking for. The idea behind Kasbah is to help users with the other major step in the process, the negotiations between buyer and seller, by providing agents which can autonomously negotiate and make the "best possible deal" on the user's behalf.

Overview of Kasbah

Kasbah is a Web site where users go to buy and sell things. They do this by creating *buying* and *selling* agents, which then interact in the *marketplace*. Kasbah is thus a multi-agent system. The marketplace is designed to handle any type of agent that supports the appropriate protocol (discussed later), though the current prototype only has a single kind of relatively simple buying and selling agents. It is these agents that will be described in the remainder of the paper. The agents themselves are not tremendously smart (although they are completely autonomous); they do not use any AI or Machine Learning techniques, though we are currently working on agents which do. What makes Kasbah fundamentally interesting is its multi-agent aspect — the interaction and competition between the agents in the marketplace.

Selling and Buying Agents

Think of a selling agent as being analogous to a classified ad. When a user creates a new selling agent, they give it a description of the item they want it to sell. Unlike the traditional classified ad, though, which sits passively in its medium and waits for someone to notice it, Kasbah's selling agents are pro-active. Basically, they try to sell themselves, by going into the marketplace, contacting interested parties (namely, buying agents) and negotiating with them to find the best deal.

A selling agent is autonomous in that, once released into the marketplace, it negotiates and makes decisions on its own, without requiring user intervention. The user does have high-level control of its behavior. When the user creates a new selling agent, they set several parameters to guide it as it tries to sell the specified item. These parameters are:

- *Desired date to sell the item by.* People usually have a deadline by which they want to sell something. For example, a graduating student might want to sell their bicycle before they leave school, because they cannot take it with them.

- *Desired price.* This is the price the user would *like* to sell their good for.

- *Lowest acceptable price.* This is the lowest price the user *will* sell their good for. If the user has junk in their basement that they want to get rid of, they may set the desired price rather high, hoping someone might be willing to pay it, and also set the lowest acceptable price to a more realistic level. On the other hand, a person willing to accept nothing less than their asking price would set the lowest acceptable price to be the desired price.

The above parameters can be changed by the user at any time after the agent has been created.

These parameters define the agent's goal: to sell the item in question for the highest possible price — ideally, the desired price, but as low as the lowest acceptable price, if that is what it takes to attract buyer interest in the time frame given. Exactly how to achieve this goal is left to the agent. The appropriate metaphor here is that of a personal assistant (Maes 1994). You tell your personal assistant what you would like to be done ("sell this for the best possible price"), and trust it to figure out how to accomplish this task, freeing your time and energy for more interesting pursuits. In addition, we hope that agents might be able to sell (and buy) goods better (e.g. at a higher price) than the user would be able to, by taking advantage of their edge in processing speed and communication bandwidth.

While an agent is "free" in terms of how to achieve its objective, the parameters described above suggest how it works. The crude heuristic used by the agents in the current version is: begin by offering the item at the desired price. If there are willing buyers, great. If not, as time goes along, lower the asking price to entice more interest. When the desired date to sell the good by rolls around, the asking price should be about the lowest acceptable price. Of course, all the interesting action is in the subtleties and nuances of how the selling agent goes about lowering the price. It is possible that there will be no buyers (perhaps the lowest acceptable price is too high, or no one interested in what the agent is selling). In this case, the agent fails to achieve its goal.

The user has some control over the agent's negotiation "strategy". The user can specify the "decay" function the agent uses to lower the asking price over its given time frame. The user has three choices: linear, quadratic, and cubic. Figure 1 shows the shape of the asking price as a function of time for each type of decay function. More sophisticated agents would have different parameters or "knobs" allowing the user to tune its selling strategy as desired.

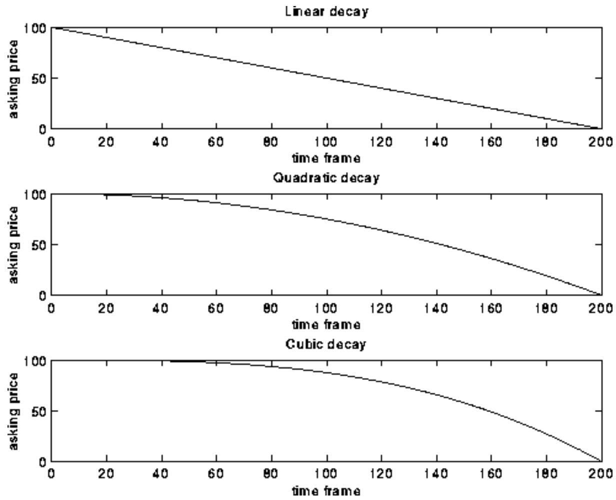

Figure 1: Shape of a selling agent's asking price over time, for the three possible decay functions: linear, quadratic, and cubic. (Note: values on the axes are arbitrary)

The user can check on its selling agents, see which other agents they have talked to, and what prices they have been offered. This information might prompt the user to do something like lower an agent's price parameters, if they see that offers are coming in much lower than expected.

The user always has final control over their agents. When a selling agent reaches an agreement with a buying agent, their users may want to give the ok, so to speak, before the agents "shake hands" on the deal. The user has a couple of controls they can set when creating both selling and buying (discussed below) agents:

- *Get user approval before finalizing deal.*
- *Send email notification when agreement reached.* The user might not be logged into Kasbah when the agent reaches a tentative agreement, and sending email is a convenient way to alert them.

The six parameters given above are by no means exhaustive. One can imagine many more controls a user might want to set on their agents, depending on the complexity of these agents. For instance, there might be a parameter that tells the agent to only negotiate with agents whose users are in a certain physical region (e.g., within the city of Boston).

Figure 2 is a screen shot from the Kasbah Prototype showing the HTML form the user fills out to create a selling agent that sells a playing card (playing cards are the prototype's test domain). For agents that sell things other than playing cards, the part of the form that describes the item to sell will be different.

Thus far we have discussed selling agents. There are also buying agents. They are essentially the symmetric opposite of selling agents. Their job is to buy goods on behalf of users. One can think of a buying agent as a want ad which actively seeks to find and buy what it's looking for. When creating a buying agent, the user describes their item of interest. Alternatively, they could specify a set of selling agents already in the marketplace and direct their buying agent to buy from one of them. Like for selling agents, the user also sets parameters to guide the agent's negotiations.

- *Date to buy the item by.*
- *Desired price.* What the user wants to pay for the good.
- *Highest acceptable price.* The highest price the user is willing to pay for the good.

Once created, the buying agent is released into the marketplace, where it negotiates with selling agents, trying to make the best possible deal.

As with selling agents, the user is also able to specify the shape of the agent's price "raise" function from among three choices: linear, quadratic, and cubic. A plot of the asking price versus time for these three functions would be those shown in Figure 1 but flipped about a line parallel to the x-axis.

Once a buying and selling agent have reached agreement on a price and gotten their respective user's approval, then the physical transaction of the good can occur. At this point, the humans must take over. In the future, agents may be able to do this too, using electronic cash, and if the goods in question are things which do not require a physical medium, such as information and knowledge.

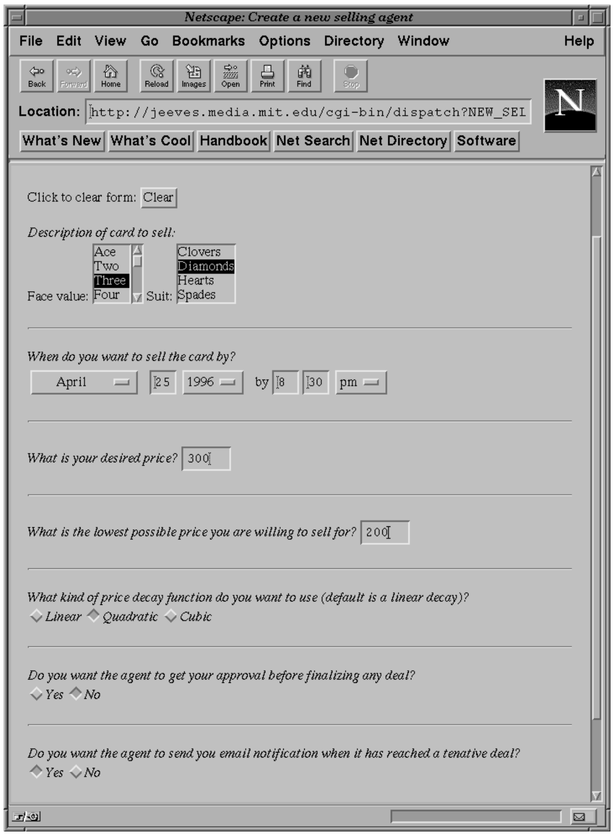

Figure 2: Form that allows users to create a new selling agent.

Users can currently only choose from a standard set of agents developed by us. In the future, though, people may want to use Kasbah for a variety of market applications — not just buying and selling goods a la the classifieds, but for things like stock markets, auctions, vendor-based markets, electronic malls, etc. For each of these types of market, there will need to be specialized agents. An agent that buys in a classifieds Kasbah will have a completely different strategy than one that buys, say, in an auction Kasbah.

In future versions, Kasbah's architecture will allow agents that speak the appropriate language to be easily "plugged" into the marketplace, thus allowing outside parties, including end-users, to build and use agents that suit their own specialized needs. We are considered using Telescript, a mobile agent language developed by General Magic (White 1996) to implement this architecture. Telescript allows agents to transport themselves over the Internet and to send messages to one another. Using Telescript, a user could create their own buying or selling agents on their machine. The agents would then be shipped over the network to the machine housing the Kasbah marketplace, where they would do their thing.

We think Kasbah's buying and selling agents will prove useful to the everyday end-user. Here are some of the services and benefits which these agents will provide:

- Spare the user from having to find, negotiate, and in general deal with buyers and sellers. Kasbah will be eliminating human-human contact, which might sound like a bad thing. But a lot of people don't like talking to strangers, which is generally required when buying or selling something via the newspaper classifieds. Language barriers and misunderstandings are often a problem. With agents doing the talking, though, the process is depersonalized. Here technology is not increasing human interaction, but reducing it, freeing people to pursue their more meaningful relationships.
- Know who are prospective buyers and sellers. The buying and selling agents we are building remember everyone they have talked to. This information can be accessed by their creators, which can be very useful. Suppose that a potential buyer asks you to clarify the description of an item you are selling. You respond to their request, but you may receive several similar requests, and answering them as they come in is annoying. Armed with information about which users' buying agents have contacted your selling agent, you can pre-emptively send out a clarification to all of those users.
- Enable better pricing. In addition to recording who they have talked to, the agents also record the content of their conversations (e.g., Agent 14 made an offer of $60.) This allows users to gather price information. They may see that they are asking too much, or alternatively, asking too little.

The Marketplace

Buying and selling agents meet and negotiate in the Kasbah marketplace. The marketplace's job is to facilitate interaction between the agents. There are many roles the marketplace could play depending on the type of market. At a minimum, the marketplace needs to ensure that the agent's participating in it speak a common language. Kasbah's marketplace also matches up agents interested in buying or selling the same kinds of things. When a selling agent is created, the marketplace asks what the agent is interested in selling. The marketplace then sends the agent a list of all the potential buyers of that particular item. The marketplace also sends messages to all of these potential buyers informing them of the existence of the new selling agent. The same thing happens when a buying agent is created. When an agent leaves the marketplace, the marketplace notifies all of its potential customers (if it is a selling agent) or all the agents for whom it was a potential customer (if it is a buying agent). Think of this as the marketplace forming "tents" of interest within itself. For example, there might be a tent for cars, a tent for apartments in Cambridge, a tent for stereo equipment, etc. The marketplace also determines the terminology spoken, that is, how goods are described. In future versions of Kasbah, this terminology will be extendible by users.

We can consider more complex marketplaces. We might want a marketplace which regulates the activities that occur within it. As agents become more intelligent, we can easily imagine malicious and deceitful agents, trying to rip off honest ones. In such a world, there will need to be some kind of law enforcement. We might have regulator agents roaming the marketplace to ensure that no illegal activity occurs. We have not yet considered in depth the social issues associated with complex agent communities, but we are aware of them, and a lot of theoretical research has been done in this area (Rosenschein and Zlotkin 1994).

Very important is that Kasbah's marketplace architecture is agent independent. As long as an agent speaks the common marketplace language, i.e., supports the appropriate protocols, then it can participate in the marketplace.

Kasbah Prototype

To test the validity of what has been described, we have built a Kasbah Prototype.

Implementation

The Kasbah Prototype is implemented in CLOS using Harlequin Lisp. As is standard in CLOS, everything is an object (an instance of a class) — the marketplace, the agents, the item descriptions, etc.

The marketplace language is implemented by requiring agents to support certain methods that can be called on them. All of the following methods can be called on both buying and selling agents.

- `accept-offer?(agent, from-agent, offer)`
  This method is used to ask `agent` whether or not they accept the offer of `offer` from `from-agent`. `agent` returns either "accepted" or "rejected".

- `what-is-price?(agent, from-agent)`
  This method is called by `from-agent` to ask `agent` what its offering price is, which is returned. If `agent` is a buying agent, then its price is how much it is *at that moment* willing to pay. If `agent` is a selling agent, then its price is how much it is *at that moment* willing to sell for.

- `what-is-item?(agent, from-agent)`
  This method is called by `from-agent` to ask `agent` what item it is trying to sell or buy, depending on whether `agent` is a buying or selling agent. An item description is returned.

These methods allow an agent or the marketplace to talk directly to another agent. There is no way for an agent to "overhear" a conversation between two other agents. In the future, we will implement methods that allow an agent to broadcast a message to all interested parties in parallel.

A marketplace object contains buying and selling agents. Agents are added to the marketplace by calling the methods `add-sell-agent` and `add-buy-agent`. When an agent is added to the marketplace, it is notified of agents who are interested in buying (selling) the item it is selling (buying). In addition, those agents are notified of the existence of the new agent. The marketplace does this notification by calling the following two methods, which must be supported by all agents:

- `add-potential-customers(sell-agent, potential-customers)`
  Notifies `sell-agent` that the buying agents specified by `potential-customers` want to purchase the type of item it is selling.

- `add-potential-sellers(buy-agent, potential-sellers)`
  Notifies `buy-agent` that the selling agents specified by `potential-sellers` wish to sell the type of item it wants to buy.

In addition, buying and selling agents must also support the methods `remove-potential-sellers` and `remove-potential-customers`, which the marketplace calls to notify agents that other agents are no longer of interest (because they have terminated or already reached a deal with someone else).

Agents also need to be able to send messages to the marketplace. There are two methods which the marketplace supports:

- `agent-terminated(marketplace, agent)`
  Called by `agent` to notify `marketplace` that it has ceased to exist.

- `deal-made(marketplace, sell-agent, buy-agent, item, price)`
  Notifies `marketplace` that `sell-agent` and `buy-agent` have agreed to transact `item` for the given `price`.

The items that are bought and sold are described by feature vectors. These vectors consists of (feature, feature value) pairs. Describing items in this way makes it easy to determine if two item descriptions match.

Conceptually, buying and selling agents in the marketplace are constantly talking to one another, all at the same time. Because we cannot really run the agents in parallel, the marketplace simulates this by implementing a simple scheduling algorithm. The algorithm works as follows. Each agent is allowed exactly one "slice" of execution time per marketplace "cycle". During this slice, an agent can do whatever — talk to other agents, do internal computations, etc. There needs to be a mechanism that limits how much processing time an agent can consume per slice, but this has not been implemented. The order in which the agents execute per cycle is determined randomly. To execute an agent during its slice, the marketplace calls the method `do_thing`, which all agents must support.

We will now describe how selling and buying agents work. We refer to selling agents; buying agents work in the obviously symmetrical way. (While it is possible to build buying and selling agents which use different strategies, we chose for convenience to have ours use the same framework.)

An agent consists of the following components: control parameters, negotiation history, and internal state. Each of these will be described in turn. An agent also stores a timestamp of when it was created and a description of what it is selling (or buying).

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. It consists of a list of (date, event) pairs, where each event describes the conversation that occured on that particular date. An example conversation is "I offered Agent 3 a price 100. They rejected the offer.", or "Agent 14 asked my selling price. I replied 91." Recording all the conversations that an agent has had can provide useful information to the user.

The internal state of an agent contains information that the agent uses to decide what it will do during its slice, i.e. when `do_thing` is called. The internal state stores a list of "potential contacts", which are those agents interested in buying (selling) what that agent is selling (buying). With each potential contact is recorded its last known offering price (i.e., what that agent is willing to buy or sell the item for), and whether it has been asked this round (explained later). The internal state also stores the agent's own current asking price.

The strategy an agent uses to decide what to do each slice is very simple, and is described below.

1. *Determine the current asking price.*
   The agent lowers (increases) its asking price according to the specified price decay (raise) function. When the agent is created, its asking price is set to the desired price. By the date 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/raise function.

2. *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 to a given potential contact until it has first talked to all other potential contacts. If a potential contact makes an offer to the agent, this is considered equivalent to the agent having talked to that contact during the current round. 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 considered agents, pick one that has *never* been contacted, or, if all agents under consideration have been contacted, then pick the one whose last know offering price is the highest (lowest). The idea is to first talk to those agents which seem the most promising — first those who have never been spoken to, and then agents who have indicated a willingness to pay a higher (or sell for a lower) price.

3. *Talk to the potential contact.*
   The agent offers to sell (buy) 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, and its asked-this-round flag is set to true.

Agent ID: 1 — type sell
agent created on: Wed 24 Jan 21:20:14 1996
the control parameters:
Sell by: Wed 24 Jan 21:24:00 1996
Desired price: 100
Lowest possible price: 50
Price decay function: linear
the item description: Ace of Spades

Agent ID: 2 — type buy
agent created on: Wed 24 Jan 21:20:14 1996
the control parameters:
Buy by: Wed 24 Jan 21:24:00 1996
Desired price: 50
Highest possible price: 85
Price raise function: linear
the item description: Ace of anything

Agent ID: 3 — type buy
agent created on: Wed 24 Jan 21:20:14 1996
the control parameters:
Buy by: Tue 24 Jan 21:25:00 1996
Desired price: 70
Highest possible price: 110
Price raise function: linear
the item description: Anything of Spades

Figure 3: Important features of Agents 1, 2, and 3.

A Simple Example

To illustrate how the agents work, we give a simple example: a marketplace containing three agents buying and selling playing cards.

Agent 1 wants to sell an Ace of Spades. Agent 2 wants to buy any card with an Ace face value. Agent 3 wants to buy any cards of the Spades suit. So, Agent 1 is a potential seller for Agents 2 and 3, which are potential customers for Agent 1. Figure 3 shows the important features of these agents.

Agents 1, 2, and 3 were added to the marketplace, which was then run for several cycles, each about twenty seconds apart. Eventually, Agent 3 agreed to buy the Ace of Spades from Agent 1 for a price of 79. Figure 4 shows the entire negotiation history of Agent 1. The histories for Agents 2 and 3 are similar and thus not shown. Note that agents record both conversations initiated by themselves ("I offered Agent 3...") and those initiated by other agents ("Agent 2 offered...").

Experimental Results

To test the Kasbah Prototype, we conducted a simple experiment where users bought and sold playing cards amongst themselves by creating buying and selling agents. The goal for each user was to maximize the "quality" of their hand as in poker. The medium of exchange was virtual "play" money, of which each user was given a fixed amount at the start of the experiment, so there was incentive for users to create selling as well as buying agents. Although this test domain was not a realistic one, it provided insight into what users expect from agents that negotiate on their behalf. Here are our major qualitative observations:

- In general, feedback was positive. Participants in the experiment seemed to think it was quite fun. But we cannot easily extrapolate from this to assume that people would use Kasbah to buy and sell real goods for real money.
- Users expect their agent not to do clearly stupid things. Even though most participants knew the details of how the agents worked, they were disappointed when the agents did "dumb" things that a human would never do, such as accept an offer when a better one was available. This happened because the agents always accept the first offer which meets their asking price; however, there might be another offer which also meets the asking price but is even better. If Kasbah's agents are to serve as intelligent assistants to the user, then they will have to be made free of such "bugs".
- Users want agents that reason more like people and are more pro-active in terms of making decisions. When people buy and sell things, there are many factors they take into account, e.g. the competition, the level of buyer interest, etc. It is doubtful, though, if the normal person could explicitly delineate the strategy they use to buy or sell. When they turn the the task of buying and selling over to someone else, be it a real-life agent or one of Kasbah's, they don't want to have to give it precise instructions. Rather, they trust that the agent will consider the factors that they themselves would consider. Any direction given the agent should be at a higher, more motivational level. You don't tell your real estate agent, "It's taking an awfully long time to sell my house. Switch from a cubic price decay to a linear one." Instead you say, "It's taking an awfully long time to sell my house" and leave it to the agent to work out the nuances of adjusting the price. We found that even specifying a desired price is something that many users would rather have their agent decide, by looking at agents buying/selling comparable items.

Our experiment demonstrated the need and desire for "smarter" agents whose decision-making processes more closely mimic those of people and which can be directed at a more abstract, motivational level. Future work on Kasbah is headed in this direction.

the negotiation history:
Date: Wed 24 Jan 21:20:15 — Event: Offered agent 3 100. Rejected.
Date: Wed 24 Jan 21:20:15 — Event: Asked agent 3 their price. Replied 70.
Date: Wed 24 Jan 21:20:15 — Event: Agent 2 offered 50. I rejected.
Date: Wed 24 Jan 21:20:15 — Event: Agent 2 asked my price. I replied 100.
Date: Wed 24 Jan 21:20:16 — Event: Agent 3 offered 70. I rejected.
Date: Wed 24 Jan 21:20:16 — Event: Agent 3 asked my price. I replied 100.
Date: Wed 24 Jan 21:20:34 — Event: Offered agent 3 96. Rejected.
Date: Wed 24 Jan 21:20:34 — Event: Asked agent 3 their price. Replied 73.
Date: Wed 24 Jan 21:20:34 — Event: Agent 3 offered 73. I rejected.
Date: Wed 24 Jan 21:20:34 — Event: Agent 3 asked my price. I replied 96.
Date: Wed 24 Jan 21:20:35 — Event: Agent 2 offered 53. I rejected.
Date: Wed 24 Jan 21:20:35 — Event: Agent 2 asked my price. I replied 95.
Date: Wed 24 Jan 21:20:53 — Event: Agent 3 offered 75. I rejected.
Date: Wed 24 Jan 21:20:53 — Event: Agent 3 asked my price. I replied 91.
Date: Wed 24 Jan 21:20:53 — Event: Agent 2 offered 56. I rejected.
Date: Wed 24 Jan 21:20:53 — Event: Agent 2 asked my price. I replied 91.
Date: Wed 24 Jan 21:20:54 — Event: Offered agent 3 91. Rejected.
Date: Wed 24 Jan 21:20:54 — Event: Asked agent 3 their price. Replied 76.
Date: Wed 24 Jan 21:21:12 — Event: Agent 2 offered 59. I rejected.
Date: Wed 24 Jan 21:21:12 — Event: Agent 2 asked my price. I replied 87.
Date: Wed 24 Jan 21:21:12 — Event: Agent 3 offered 78. I rejected.
Date: Wed 24 Jan 21:21:12 — Event: Agent 3 asked my price. I replied 87.
Date: Wed 24 Jan 21:21:13 — Event: Offered agent 3 87. Rejected.
Date: Wed 24 Jan 21:21:13 — Event: Asked agent 3 their price. Replied 78.
Date: Wed 24 Jan 21:21:31 — Event: Agent 3 offered 81. I rejected.
Date: Wed 24 Jan 21:21:31 — Event: Agent 3 asked my price. I replied 83.
Date: Wed 24 Jan 21:21:31 — Event: Offered agent 2 83. Rejected.
Date: Wed 24 Jan 21:21:31 — Event: Asked agent 2 their price. Replied 62.
Date: Wed 24 Jan 21:21:32 — Event: Agent 2 offered 62. I rejected.
Date: Wed 24 Jan 21:21:32 — Event: Agent 2 asked my price. I replied 83.
Date: Wed 24 Jan 21:21:50 — Event: Offered agent 3 79. Accepted.

Figure 4: Negotiation history of Agent 1.

Related Work

Much work has been done over the past few years on software agents (Maes and Kozierok 1993) (Lashkari et al. 1994). The overall goal of this work is to help people deal with "information and work overload", and that it is what we are trying to do with Kasbah.

The notion of autonomous agents is not a new one. It appears extensively throughout computer science literature, in several different contexts. In the field of Distributed AI, agents are entities which collaborate to solve a specific common problem (Demazeau and Muller 1990). In Decentralized AI, the focus is more on the interactions of agents with different motivations. The underlying notion, though, is that the agent interaction should further some organizational goals (Demazeau and Muller 1990).

This notion of agents is somewhat different from the one we take: Kasbah's agents not only don't share common goals, they often have diametrically opposing aims. This contrasts to a system such as Firefly (Shardanand and Maes 1995), where agents serve individual users (to make music recommendations), yet cooperate and exchange information in a mutually beneficial fashion.

A lot of work has also been done in the area of agent communication. KQML is perhaps the most notable attempt to design a general purpose agent language (Labrou and Finin 1994). We have chosen not to use KQML thus far, since our agents are all locally built and thus can be made to communicate via a predefined set of methods. We are considering using KQML in the future to easily allow agents developed by outside parties to participate in the marketplace.

As Kasbah's agents evolve and become more advanced, they will require a richer semantics of communication to enable more complex and subtle negotiations. The field of speech acts (Winograd and Flores 1986) has investigated such theoretical issues in depth.

Conclusion

Kasbah is a system where users create agents to negotiate for the purchase and sale of goods on their behalf. We have built a simple prototype to test the basic concepts and feasibility and conducted some simple experiments. Future work is focussed on making smarter agents which are directable at a more natural level for users. Though we have only just scratched the surface in terms of making a truly useful system, we are excited about this work and think it has the chance to fundamentally change the way people buy and sell goods and services in the not-too-distant future.

References

[AdWorld 1996] AdWorld, 1996.
http://www.scbe.on.ca/int/adworld.html

[Seattle Times/PI 1996] Seattle Times/PI, 1996.
http://webster.seatimes.com/classified/index.html

[Infomaster 1995] Infomaster, 1995.
http://infomaster.stanford.edu:4000/ASK/RENTAL

[Demazeau and Muller 1990] Demazeau, Y., and Muller, J. 1990. Decentralized Artificial Intelligence. In: Decentralized AI. Eds. Demazeau and Muller. Elsevier Science Publishers, North Holland.

[Geddis et al.] Geddis, D., Genesereth, M., Keller, A., and Singh., N. 1995. "Infomaster: A Virtual Information System". In Proceedings of CIKM 95 Workshop on Intelligent Information Agents. Baltimore, Maryland.

[Labrou and Finin 1994] Labrou, Y., and Finin, T. 1994. "A Semantics approach for KQML — a general purpose communication language for software agents." In Proceedings of CIKM 94. New York: ACM Press.

[Lashkari et al. 1994] Lashkari, Y., Metral, M., and Maes, P. "Collaborative Interface Agents." 1994. In Proceedings of the 12th National Conference on Artificial Intelligence. Seattle, Washington: AAAI Press.

[Maes and Kozierok 1993] Maes, P., and Kozierok, R. 1993. "Learning Interface Agents." 1993. In Proceedings of AAAI 93 Conference. Washington, DC: AAAI Press.

[Maes 1994] Maes, P. 1994. Agents that Reduce Work and Information Overload. *Communication of the ACM* Vol. 37, No. 7. 31-40.

[Rosenschein and Zlotkin 1994] Rosenschein, J., and Zlotkin, G. 1994. *Rules of encounter: designing conventions for automated negotiation among computers.* Cambridge, Mass.: MIT Press.

[Shardanand and Maes 1995] Shardanand, U., and Maes, P. 1995. "Social Information Filtering: Algorithms for Automating Word of Mouth". In Proceedings of CHI 95 Conference, Denver, Colorado.

[White 1996] White, J. 1996. Telescript Technology: Mobile Agents. General Magic White Paper.