34

Kasbah: An Agent-based Marketplace for Buying and Selling Goods

by

Anthony S. Chavez

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 10, 1997

Copyright 1997 Anthony S. Chavez. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author ______________________________
Department of Electrical Engineering and Computer Science
Februrary 10, 1997

Certified by ______________________________
Professor Pattie Maes
Thesis Supervisor

Accepted by ______________________________
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

OF TECHNOLOGY

OCT 2 9 1997

Kasbah: An Agent-Based Marketplace
For Buying and Selling Goods

By

Anthony S. Chavez

Submitted to the
Department of Electrical Engineering and Computer Science

February 10, 1997

In Partial Fulfillment of the Requirement for the Degree of
Bachelor of Science in Computer Science and Engineering
and Masters of Engineering in Electrical Engineering and Computer Science

Abstract

Kasbah is a Web-based marketplace in which software agents play a novel role: as buyers and sellers of goods on behalf of users. A user gives an agent a description of the good to buy or sell, as well as pricing instructions. The agent then proceeds to find other agents in the marketplace who are interested parties, i.e. potential buyers or sellers, and negotiates with them in an attempt to find the best deal. This negotiation takes place in a simple manner which the user can understand and has control of through several parameters. We conducted two experiments with Kasbah to assess its viability. By automating the finding and negotiating aspects of conducting transactions, Kasbah frees the end-user from much of the work of buying and selling; this opens the way for the formation of new markets which were previously prohibited by high transaction costs.

Thesis supervisor: Pattie Maes
Title: Associate Professor, Sony Corporation Professor of Media Arts and Sciences, MIT Media Lab

Table of contents

Abstract 1

Table of contents 2

Chapter 1: Introduction 5

- Motivation 5

Chapter 2: Functional Overview 8

- Using Autonomous Agents to Buy and Sell 8
- Preliminaries 9
- Creating an agent 10
- Describing the good of interest 11
- Instructing the agent 13
- Agent behavior 15
- Modification of existing agents 18
- Agent communication with user 19
  - Should I be negotiating? 20
  - Status report 22
  - Deal made 24
- Finding agents 25
- Making a deal 25
- Sample scenario 27

Chapter 3: Related Work 30

- Software agents 30
  - Eager assistants 30
  - Multi-agent matchmaking systems 32
  - Collaborative filtering agents 33
- Market-Based Systems 35
- Electronic Commerce on the Web 38

Classified Ads Style Sites 38
Price Comparison Agents 41
AdHound 42
Other related work 43
Sumary of related work 44
Chapter 4: Architecture and Implementation 45
Overall Architecture 45
Physical topology 46
Marketplace engine 46
Selling and buying agents 47
Agents as objects 48
Agent Communication 48
Item Descriptions 50
Buyer against seller 50
Seller against seller 51
Buyer against buyer 51
Agent Behaviors 52
Finding Interested Parties 53
Agent's current asking price 54
Passive and Pro-Active Behaviors 54
Marketplace 57
The Scheduler 57
Front-end interface 59
Creating and modifying agents 60
Communicating with the end user 60
Backing up the System 61
Chapter 5: Experimental Results 63
Initial experiment 64

Experiment in creating an Agent Marketplace 68
Experimental Setup 68
Results from Agent-Marketplace Experiment 73
Qualitative analysis 78
Chapter 6: Future Work and Conclusion 80
Future work 80
Item description problem 80
Improved architecture 81
New types of agents 82
Collaborative agents 82
Social consequences 83
End-user testing 84
Conclusion 84
References 85

Chapter 1: Introduction

Software agents help people with time-consuming activities [Maes94]. Software agents have in the past been used in many diverse roles, such as making recommendations, prioritizing email, and matchmaking [Maes95][Bradshaw97]. In the research described in this thesis, we explore a new role for software agents: that of agents assisting people with buying and selling. We describe Kasbah, a Web-based system which allows users to create agents that negotiate to buy and sell on their behalf.

Traditionally, there are just a few ways that people can buy and sell goods directly to one another (by "directly" we mean not involving commerical stores): through the newspaper classifieds, by having garage sales, by posting flyers, etc. The rise of the Internet and especially, the World-Wide Web (WWW), has ushered in a new way that people can buy and sell -- the field of electronic commerce. Today, there are a proliferation of web sites where one can purchase all types of goods. There are many different types of sites -- electronic malls which house individual stores, mail-order sites, classifieds-style sites where one can post ads, to name just a few. The model of commerce that we are concerned with is that of parties selling and buying directly to one another. Our emphasis is on when these parties are end-consumers, but the model also supports business-to-business and business-to-end-consumer transactions.

Motivation

The primary motivation for our work is the observation that, today, transacting goods directly among end-consumers is a labor-intensive and time-consuming activity. We can break this process down into three main stages: finding interested parties (potential buyers or sellers), negotiating with these parties to make the best deal, and finally, consummating the transaction, i.e. exchanging money for the good or service. As far as finding interested parties goes, today the primary means at our disposal are: posting an ad in the newspaper classifieds, using word of

mouth, holding a yard sale, or going to a flea market. All of these means require a non-trivial amount of effort to utilize; perhaps this is why many of us accumulate so much unwanted and unused junk in the attic. As for negotiating to find the best deal, today this burden rests entirely on the end-consumer. Some people may choose to haggle and negotiate with lots of parties in an attempt to find the best deal, while others are content to make a deal with the first person who has a minimally acceptable offer. The extent to which we are willing to negotiate depends on the value of the good in question: the more valuable the good, the more we inclined we are to negotiate. Finally, when it comes to consummating the transaction, how this is done really depends on the good or service in question.

We believe that software agent technology can make the process of buying and selling goods significantly easier on the end-consumer, by automating some of the activities described above.

Today on the Web there are many sites where users can do things like post ads, browse existing ads, and search for goods of interest [SeaTimes][InfoMaster]. All of these sites help people find interested parties (potential buyers or sellers), but they are essentially like newspaper classifieds on-line with additional searching capabilities. The user must still do the bulk of the work. These sites also only partially help with this first stage of transacting goods because they do not support the notion of a "standing query". For example, suppose you want to be notified about any ads selling Mozart records. Today, you are forced to return to the site daily and re-issue your search query. Kasbah, on the other hands, has "finding" agents which automate this task, continually looking for new buyers of sellers which the user may be interested in contacting.

More interesting are a two sites, BargainFinder and Fido which do comparison shopping for the user [BF] [Fido]. Both of these sites make use of a personal agent metaphor: you tell your agent what good you want to buy; it then goes out and searches a set of web sites, and returns to you the site which has the best price for that particular good.

While all of these sites are interesting and useful, none of them attempt to automate the second stage of the buying/selling process: negotiating to find the best deal.

The Kasbah system which we developed uses software agents to automate, to a large extent, both the activities of finding interested parties and negotiating to find the best deal. The user creates a Kasbah agent,describes to it the good they wish to buy or sell, and specifies parameters which guide the agent as it negotiates with other agents in the Kasbah marketplace on the user's behalf. If the goods being transacted are digital in nature, e.g. information services, then Kasbah could also automate the third stage of transacting goods, namely the exchange of the good for payment through use of "digital" cash. To our knowledge, this is a novel application of software agent technology.

We built several prototype Kasbah systems that were derived from the same functional core. The basic architecture consisted of a back-end marketplace "engine", and a front-end Web-based interface. This decomposition allowed for different Kasbah sites to be easily set up. We describe experiments which we performed with two Kasbah systems. The goal of these experiments was to understand how people would react to concept of agents buying, selling and negotiating on their behalf, and to assess the appropriate level of user-agent interaction. The results of the experiments confirmed user acceptance of the Kasbah concept. They also indicated that the agent behaviors should be kept simple so that users can understand what their agents are doing and thus be able to retain direct and simple control over them.

The organization of this thesis is as follows. In Chapter 2 we present a functional overview of the Kasbah system and give an example end-user scenario. In Chapter 3, we discuss related work. In Chapter 4, we describe the architecture and implementation of our prototype Kasbah systems. In Chapter 5, we discuss two experiments which were conducted using Kasbah and results obtained. In Chapter 6, we present suggestions for future work and research directions and briefly conclude.

Chapter 2: Functional Overview

Kasbah is a Web site where people go to buy and sell various goods. As noted in the Introduction, its current model of commerce is that of end-consumers selling directly to one another, without going through a third-party or broker. However, there is no fundamental reason why it could not support business-to-business or business-to-end-consumer transactions. In this chapter, we give an overview of the functionality of the system and an example end-usage scenario.

Using Autonomous Agents to Buy and Sell

In Kasbah, users buy and sell goods by creating autonomous agents which do the bulk of the work of buying and selling for them: namely, finding interested parties and negotiating with these parties to find the best deal. Kasbah thus moves away from direct-manipulation paradigm of traditional end-user applications, to the instructible-agent paradigm [Maes94]. In this paradigm, the user instructs their personalized agent about the task they want to accomplish. The agent then goes out and accomplishes the task on its own, without requiring constant oversight and detailed manipulation on the part of the user. This requires, of course, that the agent be autonomous and intelligent enough to do the task successfully. For this reason, software agents are often referred to as autonomous, or intelligent agents. The goal of the instructible-agent paradigm is to free the user of the burden of tedious and time-consuming activities by having agents do them instead.

In Kasbah, when a user wants to buy or sell something, they create an agent to do it for them. The agent that they create is owned by the user and is working on their behalf. A user can have multiple agents working for them. Kasbah can be thought of as a marketplace which consists of many agents, each trying to find the best deal on behalf of some user. The agents do this by

interacting and negotiating with one another. Each agent can be seen as selfishly trying to maximize its own utility (actually, the utility ofits owner). This makes Kasbah a classic example of a marketplace. Kasbah is thus a multi-agent, market-based system.

Preliminaries

Kasbah is a Web site which is intended to be accessible to the general public. To use Kasbah, you must have an account. Anyone can get an account by going to the main Kasbah page and clicking on the "new account" link that is located there. Once a person has logged into Kasbah, they are presented with a personalized home page from which they do any one of the number of things listed below, by clicking on the appropriate links or portion of the menu bar which runs across the top of every screen. Figure 1 shows a typical user's home page.

- create a new selling, buying, or finding agent
- modify existing selling, buying, or finding agents
- see and respond to messages from your agents
- browse the agents in the marketplace
- create an agent to find items of interest in the marketplace

How the functionality listed above works is explained in detail in the sections that follow. Once the user has done all that they wish to do in their session with Kasbah, they log off by clicking on "Exit" portion of the menu bar.

Figure 1: Typical user home page

Creating an agent

In Kasbah users can create two kinds of agents: buying agents and selling agents. (There are also finding agents, but they just have a subset of the functionality of a buying or selling agent, and so will be discussed separately.) What kind of agent a user creates depends on whether they want to buy or sell something. In this section, we explain how a user creates an agent to buy or sell something on their behalf.

Creating an agent consists of two main steps: (1) telling the agent what item you want to buy or sell, and (2) instructing the agent about how it should go about buying or selling that item.

Describing the good of interest

The first step in creating an agent in Kasbah is describing to it what good you are interested in either buying or selling. In our prototype Kasbah systems, we limited the goods that could be transacted so as to avoid the very difficult general problem of how to describe an arbitrary good in such a way that a computer program or software agent can accurately assess the similarity between two good descriptions. By constraining the domain of goods, this problem becomes much easier. Figure 2 shows how a user would describe to their agent a musical recording that they would like to buy or sell. (Our main Kasbah prototype, where this screen shot, and all those that follow in this Chapter, is taken from, was limited to transacting musical recordings.) One can see from Figure 2 that we came up with a very simple structured format for describing musical recordings. To describe a musical recording to a Kasbah agent, the user specifies:

- the type of recording: either CD, tape, DAT, record, or 8-track tape. This field is mandatory, i.e. must be specified.
- for selling agents, the musical genre: we provide a variety of choices, including rock, pop, alternative, blues, etc. This field is optional, i.e. the user can choose to leave it blank.
- The name of the artist: this is a text field that the user fills in
- The name of the album: this is also text field that the user fills in.
- For selling agents, a human-readable (natural language) description of the musical recording. The purpose of this field is to allow users to describe aspects of the recording that are not expressable through the other fields (e.g., "this is a crystal clear recording!"). This field is used by the agents in doing matching, and is optional.
- For buying agents, a list of keywords that the user thinks are relevant to precisely describing the recording but are not expressablethrough the other fields (e.g., "classical"). This field is used by the agents in doing matching, and is optional.

As an example, suppose a user has U2's "The Joshua Tree" CD which they never listen to anymore and wish to sell for a few dollars. They would create a selling agent and describe their U2 recording as by specfying the title field to be "The Joshua Tree" and the artist field to be "U2".

Further suppose that there is another user of Kasbah who is interested in buying the Joshua Tree CD for a good price. He creates a buying agent and describes the recording of interest by specifying the title field to be "joshua tree" and the artist field to be "u2".

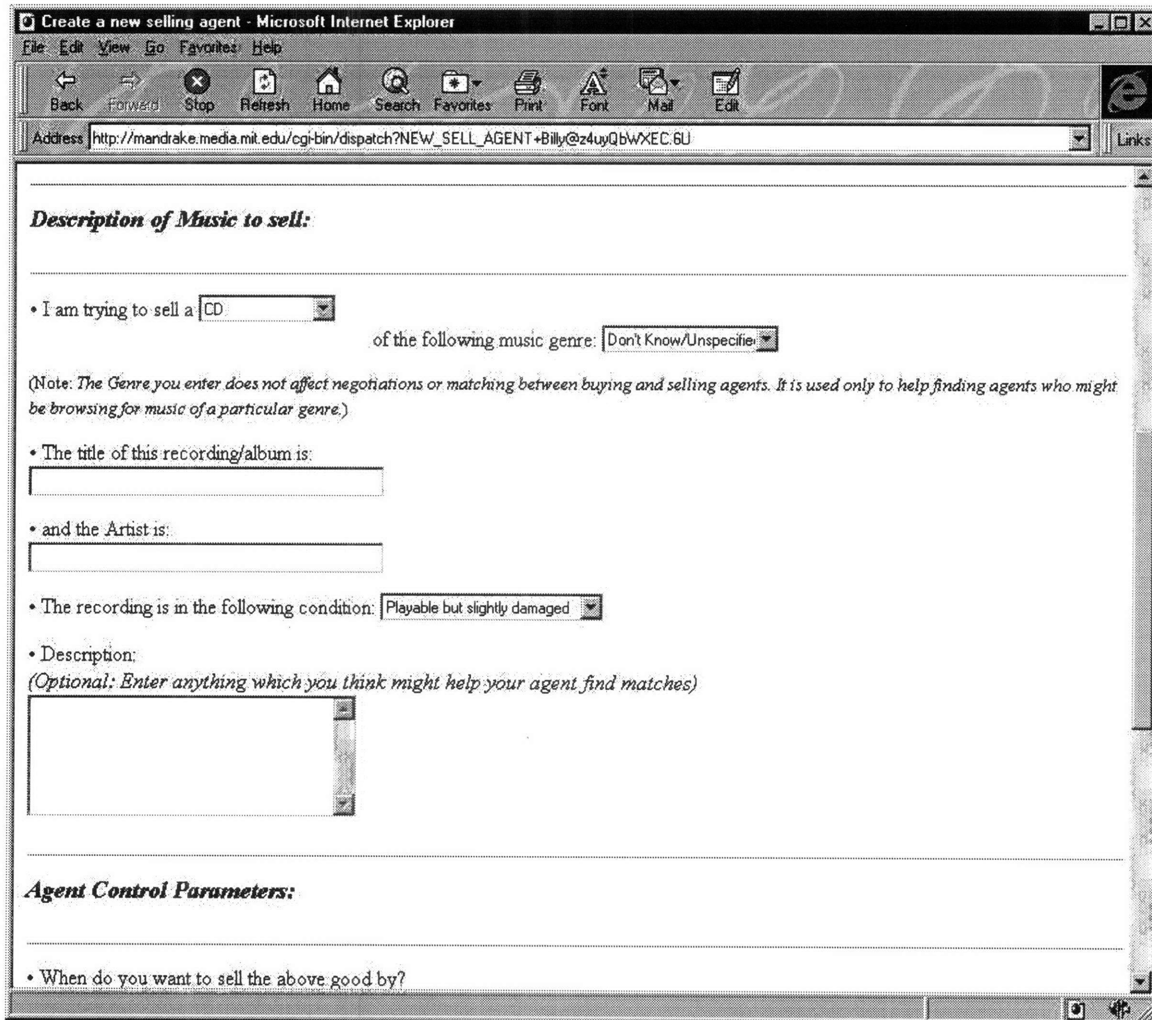


Figure 2: form used to describe music to sell

Note that these descriptions of the Joshua Tree CD are not precisely identical. For the seller, the name of the album is "The Joshua Tree", and for the buyer, the name of the album is "joshua tree". Despite this syntatic difference in descriptions, it is clear that the seller and the buyer are referring to the same album. However, if the buyer had entered "Rattle and Hum" as the name of the album, then it would be equally clear that they weren't referring to the same album. Kasbah's agents are designed to do what we call "fuzzy" or "soft" matching, i.e. two item descriptions can be considered "matches" despite minor differences in syntax. Indeed, it is very important that we

provide soft matching; if agents are to buy and sell on our behalf, we except them to display at least some minimal degree of human competence, one aspect of which is understanding that two album titles or artist names which only differ by a couple of characters are most likely referring to the same album or artist.

Fundamentally, this problem enters into the area of machine learning and artificial intelligence. Devising a general-purpose descriptive language and associated matching rules is a very difficult problem, and one which our research does not attempt to address. We used ad hoc techniques that worked for our limited domain of musical recordings.

Instructing the agent

Once the item of interest has been described, the next step in creating a selling or buying agent is to instruct it as to how to go about buying or selling that item. In the instructible-agent paradigm, the idea is that instructing the agent about doing the task in question is less work for the user than actually doing the task themselves. For this to be the case, the agent must already have a basic level of knowledge about how to accomplish the task. We don't want to have the user program the agent from scratch. If the agent already essentially knows how to do the task, then instructing it can become just a matter of setting various parameters which, while not fundamentally altering the agent's core behaviors, allow the user to control its actions at a higher level. This is precisely the approach that Kasbah's buying and selling agents take.

Kasbah's buying and selling agents have a set of built-in behaviors which enable them to negotiate with other agents to find the best deal on behalf of their owner. The agents expose a set of control parameters to the user which allows them to tune the agent's behaviors to suit their specific desires. We describe below these control parameters and what affect they have on agent behavior. The control parameters for selling agents and buying agents are symmetrical. Buying agent parameters are given in parentheses where necessary.

- date to sell (buy) by: the latest date by which the user wishes to have sold or bought the item of interest. The agent will attempt to buy or sell by this date.

- desired price: the price the user would like to sell (buy) the item of interest for. The agent will attempt make a deal at this price (or higher if a selling agent, and lower if a buying agent)
- lowest (highest) acceptable price: for selling agents, the lowest price that the user is willing to sell the item of interest for; for buying agents, the highest price that the user is willing to pay for the item of interest. The agent will never make a deal at a lower (higher) price.
- pricing strategy: the strategy that agent should use as it adjusts its asking price over time (this will explained in full later). The agent will use the specified pricing strategy. In the prototype implementation we give the user three choices of strategy.

Figure 3 shows the HTML form that user sets these control parameters for the agent that they are creating.

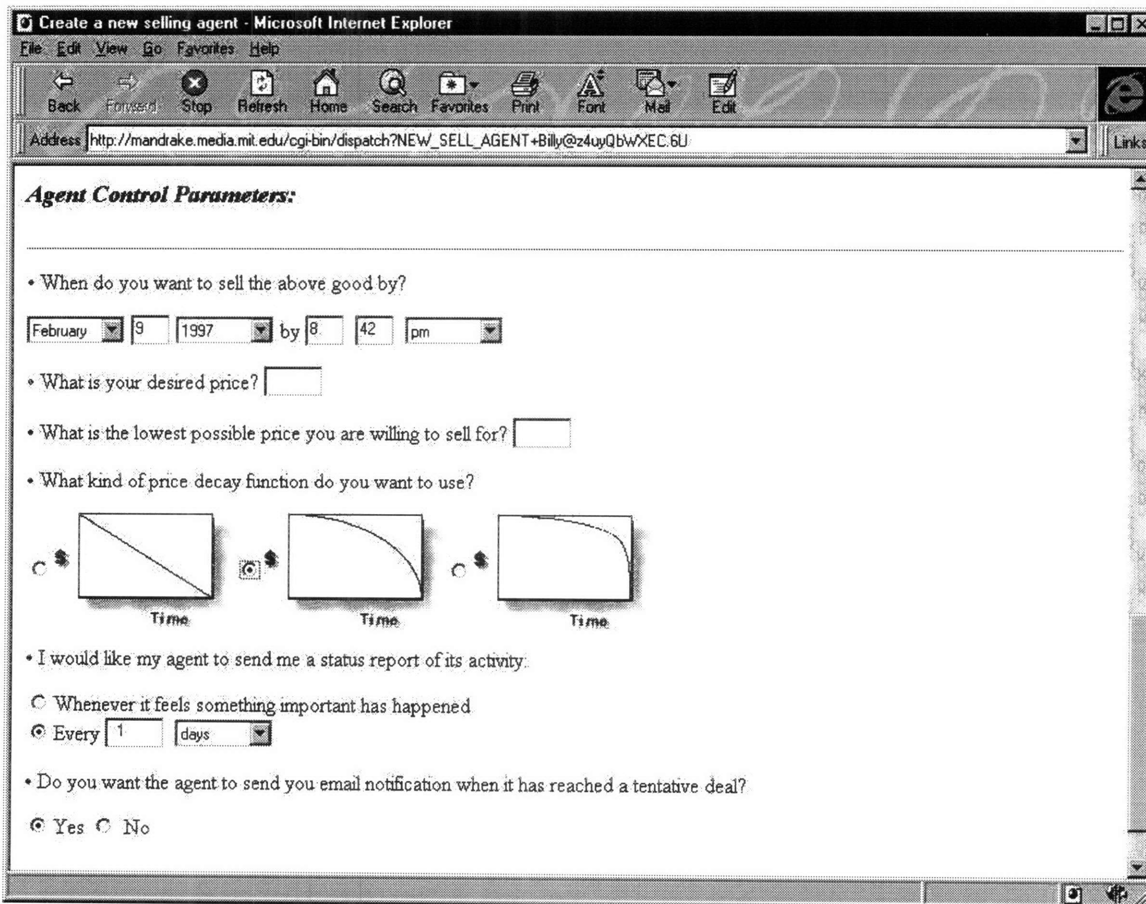


Figure 3: how the user specifies the agent's control parameters

In setting these control parameters, the user is instructing the agent, at a high-level, about how it should behave. The user is not concerned with the low-level details of the agent's behavior, e.g. how it finds other interested parties and communicates with them; the user can assume the agent is this competent. Rather, the user wants to tell the agent about the overall considerations it should take into account, much as a person would instruct a human agent like a real-estate agent. Examples of such considerations would be things like, "I really need to sell this by the 14th of March" (captured by the "date to sell by" parameter). Or, "I would like to buy this for 12 dollars, but I'm willing to pay as much as 22 dollars if I really have to" (captured by the desired and highest acceptable price parameters).

Agent behavior

The behavior of Kasbah's agents may be described as follows. (We describe the behavior of selling agents; the behavior of buying agents is intuitively symmetric.)

Once created, a selling agent goes into the marketplace and finds the other agents who are interested in buying the item that it is selling (these agents can be considered "potential customers"). The agent will continue to look for potential customers as new agents are added to the marketplace over time. The agent begins negotiations with the potential customers by offering the item for the user-specified desired price. If the agent finds a willing customer at that price (or higher), it makes the deal and notifies its owner. If the agent is not able to find a customer willing to pay the desired price, then it will lower its asking price. This decrease in asking price does not happen instantaneously, but over time, just as a real person would negotiate.

The rate at which the agent lowers its asking price over time is determined by the user-specified pricing strategy. The agent then checks if there is a willing customer at the new, lowered asking price. If there is, the agent makes the deal. If not, then the agent will again lower its asking price. This process continues until either the agent makes a deal, or the date to sell by has been

reached, at which the point the agent's asking price will be the user-specified lowest acceptable price (or very close to it). The negotiation strategy described above that the agents use to find the best deal is rather simple. One could easily envision more complex and dynamic strategies that take into account current market conditions, the observed behaviors of other agents, etc., rather than the straightforward deterministic approach which our agents use. In fact, the simplicity of the current negotiation strategy was a deliberate design decision. There are limits to how complicated one should make an agent stemming from user trust. In order for a user to be able to "trust" their agent to do the right things on their behalf, they must be able to predict how it will be behave. An agent with very complex and subtle behaviors will be very difficult for the user to understand, and thus, to trust.

The core of the our agent's negotiation strategy is how it adjusts its asking price over time. This is completely determined by the user-specified control parameters: the date to sell by, the desired price, the lowest acceptable price, and the pricing strategy. Figure 4 gives the mathematical formula showing how these parameters determine the agent's "pricing curve" --- its asking price over time. At any given moment, an agent's asking price is the minimum (maximum) offer it would accept to sell (buy) the item in question, and this price continuously changes over time. The end points of the price curve are determined by the date the agent was created, the date to sell (buy) by, the desired price, and the lowest (highest) acceptable price. The shape of the curve between these end-points is determined by which "pricing strategy" the user has chosen.
In the prototype Kasbah, there are three different strategies to choose from:

- linear (anxious seller/buyer): this is shown in Figure 5 for selling agents (buying agents are the symmetric opposite flipped along the y-axis). The agent lowers (raises) its asking price in a linear fashion over time. We call this the "anxious" strategy, because of the three available, this is the one in which the price changes most significantly at the earlier time, indicating that the agent wants to sell (buy) the item in question as soon as possible.
- Quadratic (cool-headed seller/buyer): this is shown in Figure 6 for selling agents. The agent lowers (raises) its asking price in an inverse-quadratic fashion over time. We call this the

"cool-headed" strategy, because its pricing curve lies in between that of the "anxious" and "frugal" (see below) strategies.

- Cubic (frugal seller, greedy buyer): this is shown in Figure 7 for selling agents. The agent lowers (raises) its asking price in an inverse-cubic fashion over time. We call this the "frugal" strategy, because of the three available, this is the one in which the price changes most significantly at the later time (and least signficantly at the earlier times), indicating that the agent wants to hold out lowering (raising) its price as long as possible to try and get the best deal. One can think of the agent as "driving a hard bargain".

For selling agents:
P(t) = P_desired – FACTOR * (P_desired – P_lowest)
where:
P(t) is the current asking price of the agent as a function of time
t is the time elapsed from when the agent was created
P_desired is the user-specified desired price
P_lowest is the user-specified lowest acceptable price.
FACTOR is given by:
(t / time_to_live) ^ N
where:
time_to_live is the time that the user has given the agent to make a deal
N is either 1 (anxious strategy), 2 (cool-headed strategy), 3 (greedy strategy)

For buying agents:
P(t) = P_desired + FACTOR * (P_highest – P_desired)
where:
P(t) is the current offering price of the agent as a function of time
P_highest is the user-specified highest acceptable price
all other variables are as above

Figure 4: formula describing how agents adjust price over time

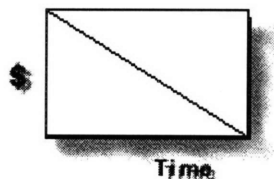


Figure 5: anxious strategy for selling agents

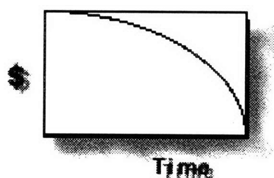


Figure 6: cool-headed strategy for selling agents

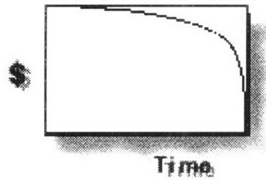


Figure 7: greedy strategy for selling agents

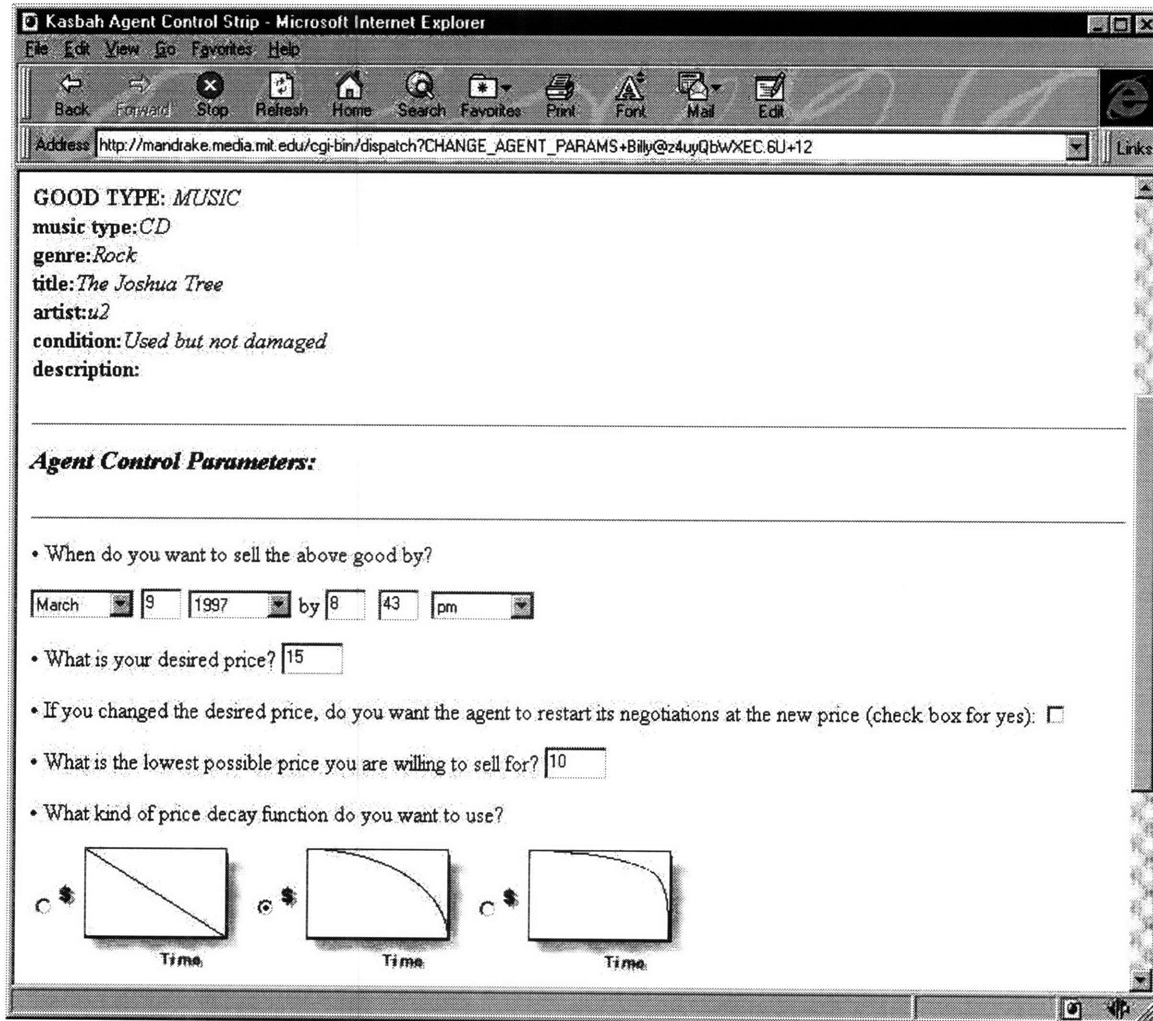
Modification of existing agents

Once an agent has been created and goes off into the marketplace and begins negotiating with other agents, the user still retains control over its behavior --- the agent is not completely independent, after all, but working on behalf of its owner.

A user can delete an agent that they created. Doing this means that the agent ceases to exist --- it breaks off all negotiations it is currently engaged in. A typical scenario where a user deletes an agent would be if the user suddenly decides that they no longer want to purchase something, or that they are no longer willing to sell something.

A user can modify any of the control parameters they have set on an existing agent (date to sell/buy by, desired price, highest/lowest acceptable price, and pricing strategy). Figure 8 shows the HTML form used to make these modifications. By modifying any of these parameters, the user changes the pricing curve which the agent is using.

Figure 8: how the user modifies an agent's control parameters



Agent communication with user

The purpose of using agents in Kasbah is to free the user of much of the burden involved with buying and selling. The agent, once created, is autonomous, and negotiates with other agents and changes its asking price without requiring the intervention of its owner. The user still has control over the agent's behavior, i.e. the user can delete the agent or modify its control parameters. In addition, agents communicate with their owners to keep them informed about the status of their negotiations, and also to ask for input when necessary. Agent communication with

the user happens via two channels in Kasbah: through email and over the Web. In our prototype Kasbah (the one for transacting music recordings), the user can only respond to agent communications through the Web (via HTML forms); email is used to notify the user of the content of the communication and, when appropriate, to tell them to log into Kasbah to respond.

The model of agent communication is the following: An agent sends a message to its owner. This message is either a "status report", notifying the user of how the agent's negotiations are progressing, or it is a request for input from the user. The different types of agent communications will depend on the nature of the agent and the marketplace, as well as the kinds of goods being transacted. In the our prototype Kasbah, there are three kinds of possible agent communications.

Should I be negotiating?

An agent sends this message when it is unsure about whether it should be negotiating with a particular agent. By unsure, we mean that there was not a 100 percent match between the two agent's item descriptions. The agent sends the user a textual representation of the item description of the other agent, and asks the user whether or not it should be negotiating with this agent. The user responds with either "yes" or "no". If the response is "yes", the agent will negotiate with the agent in question; if the response is "no", the agent will cease negotiations with that agent. The purpose of this communication is to avoid the situation, which is possible when doing "fuzzy" matching of item descriptions, where an agent makes a deal with the "wrong" agent. For example, suppose you create an agent to buy U2's "The Joshua Tree" CD and your agent goes and makes a deal to buy U2's "Rattle and Hum" CD. You would not be pleased. Actually, this is distressing only to buyers, who definitely care about what they're buying, but not to sellers, who don't really care who they sell their item to. Thus this communication is only sent by buying agents, and not selling agents. Furthermore, this communication is only relevant if the notion of "fuzzy" or "soft" matching applies to the item descriptions. For item descriptions which perfectly match, this communication is never sent. It is up to particular implementations of

Kasbah to determine how matching works. In our prototype Kasbah for musical recordings, perfect matches occur when the title and artist strings to match exactly (case-insensitive). Soft matches are determined by an ad hoc technique which is described in Chapter 4. On a perfect match, the buying agent assumes that the matched item is what its owner wants to purchase. On a soft match, the buying agent sends a "should I be negotiating?" message to its owner, since it is not entirely sure that the other agent is indeed selling what its owner is interested in buying. Figure 9 shows both the Web version of this message for our prototype Kasbah.

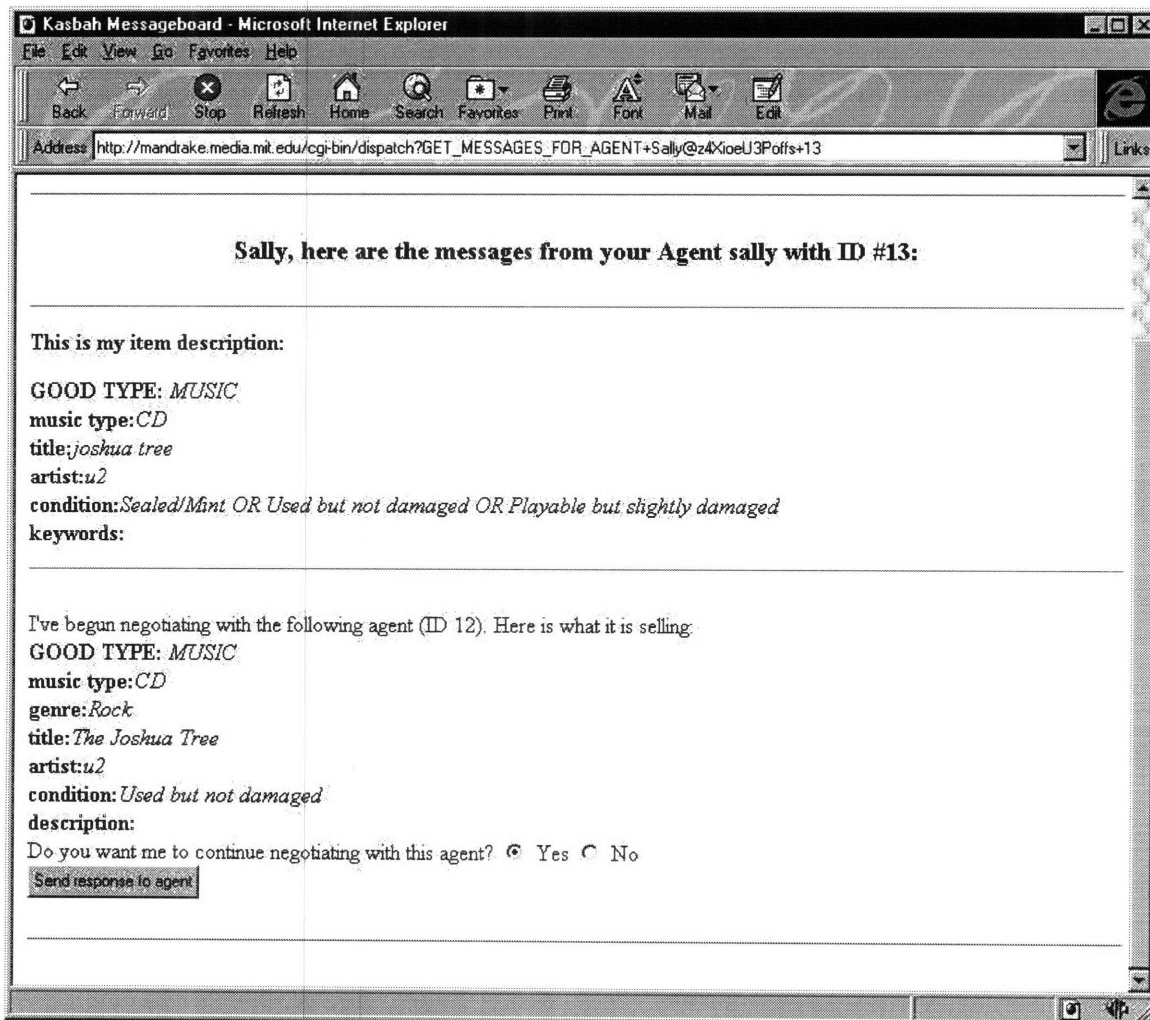


Figure 9: "Should I be negotiating?" message

Status report

Status reports are used to keep users informed and up-to-date about the progress of its agents. They are sent by every agent to its owner on a regular basis, the frequency of which is configurable by the user. A status report can contain a variety of information that a user might find useful. Based upon a status report, for instance, a user might decide to modify their agent's control. The information contained in the status reports of our agents is:

- current asking price: what the agent is currently asking for this item
- current average asking price of sellers of this item: if the agent is selling, this information helps the user gauge what the "competition" is up to, i.e. see what other sellers of the item of interest are currently asking; if the agent is buying, this information helps the user assess what the current going rate is for the item of interest.
- current average offering price of buyers of this item: if the agent is buying, this information helps the user to gauge what the "competitors" for the item of interest are willing to pay; if the agent is selling, this information helps the user assess what price the market is currently willing to bear for the item of interest.
- current minimum asking price for sellers of this item: the asking price of the seller of the item of interest who is currently asking the least. If the agent is buying, this information indicates how close the agent is to making a deal (because once this price reaches the agent's current asking price, a deal may be made). If the agent is selling, this information indicates the disparity between the agent's own asking price and the current best deal in the marketplace for the item of interest.
- current maximum offering price for buyers of this item: the offering price of the buyer of the item of interest who is currently willing to pay the most. If the agent is selling, this information indicates how close the agent is to making a deal (because once this price reaches the agent's current asking price, a deal may be made). If the agent is buying, the information indicates the disparity between the agent's own asking price price and current best deal in the marketplace for the item of interest.

- number of buyers of this item: If the agent is selling, this is the number of potential customers for the item of interest. If the agent is buying, this is the number of "competitors" who are trying to buy the same item.
- number of sellers of this item: If the agent is buying, this is the number of potential sellers to purchase the item of interest from. If the agent is selling, this is the number of competitors. who are trying to sell the same item.
- current average transaction price of this item: This indicates what price the item of interest has historically been transacted for, i.e. the current average of all transaction prices for the item of interest (back to some cutoff date).

All of this information is useful to buyers and sellers in terms of helping them making pricing decisions, i.e. deciding what their desired price, lowest/highest acceptable price, and pricing strategy should be. Based upon the current average asking price of its competitors, for example, a user might decide to lower or raise an agent's current asking price (as explained above, this would be done by changing the desired price parameter and "resetting" the price curve). Our intention is to provide the user with useful marketplace information, without overloading them with raw data, and allowing them to do with it what they will. These data on the marketplace (with the exception of the agent's current asking price) are actually kept track of by the marketplace itself, and not the individual agents. This was done to reduce the computational and communication burden on the agents. The marketplace serves, in this sense, as a trusted third-party observer of the conditions and events in the marketplace, which agents can query when desired. The agents in turn pass this information on to their owners in status reports. Actually, the marketplace data could be presented to the user directly, and not go through the agents at all. In one the experiments we conducted involving a slightly modified version of Kasbah (described in the Chapter 5), this is indeed what we did: the user had access to all of the aforementioned data through visual graphs, and would refer to them when giving pricing instructions to their agents. Status reports are a one-way communication; that is, the user does not respond other than acknowledging having read it.

Deal made

An agent sends this message when it has successfully made a deal, i.e. it has found an agent who is willing to meet its current asking price. The message consists of a statement that the agent has made a deal, the name of the other party (i.e. the owner of the other agent) and contact information (right now, this is just a email address, but it could in the future include things like a phone number). Figure 10 shows both the HTML version of this type of message. Like status reports, deal made communications are one-way. Once the message has been acknowledged, the agent is terminated and removed from the marketplace.

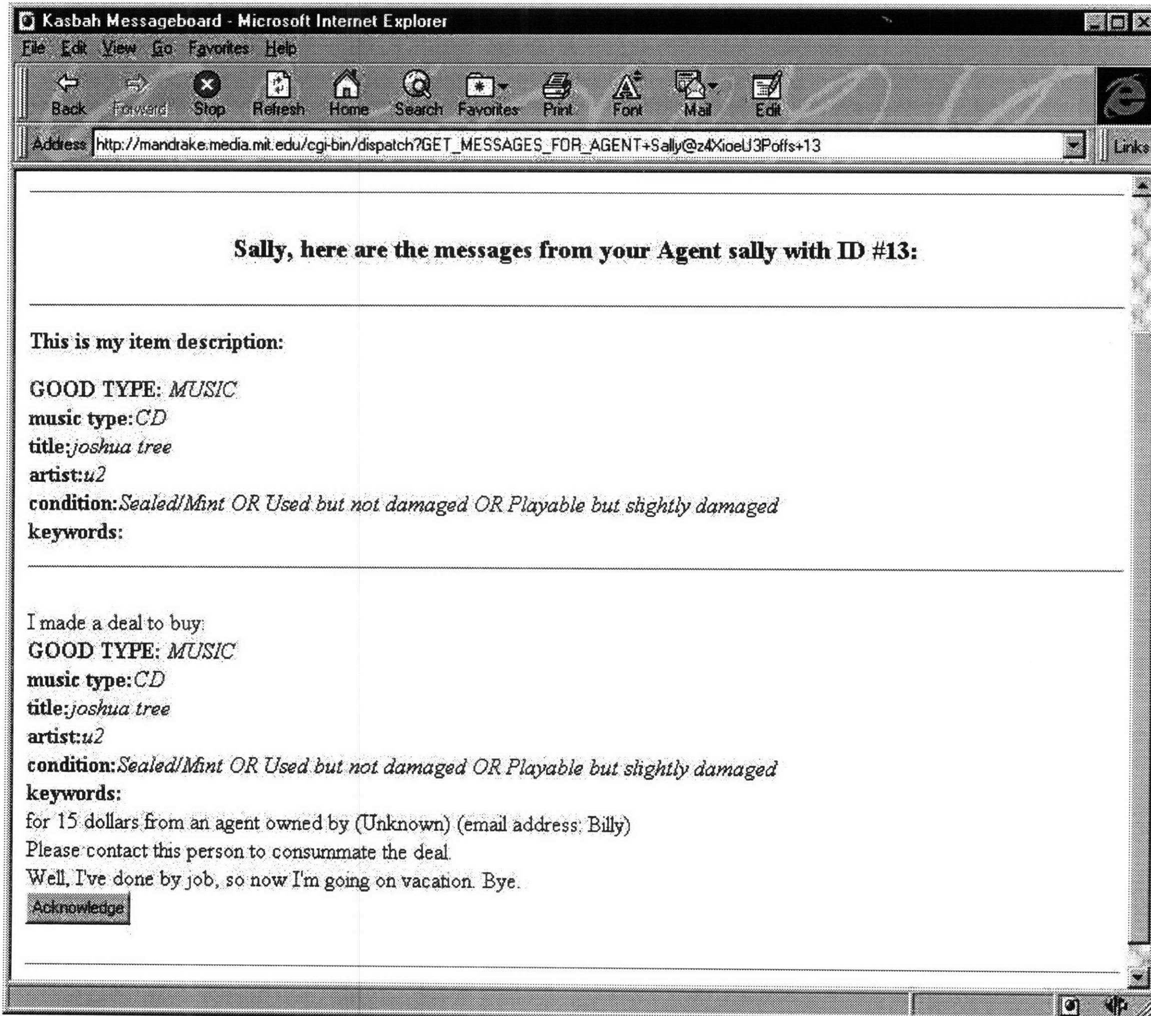


Figure 10: "deal made" message

Finding agents

Finding agents are a special derivative of buying and selling agents. They are used when a user only wants to search for certain goods in the marketplace, for possibly an extended period of time. To create a finding agent, a user specifies an item description just as they would for a buying or selling agent, except that they do not specify any control parameters, other than for how long the finding agent should exist, and whether it should look for agents buying the item, selling the item, or both. Once created, a finding agent searches the marketplace for agents that match the user's description. The agent accumulates a list of these agents, and on a daily basis, sends the list to the user through email.

Making a deal

As stated previously, the task of Kasbah agents is two-fold: one, to find interested parties, and two, to negotiate with these parties on behalf of the owner to find the best possible deal. Once a deal has been struck between two agents, both send deal made messages to their owners (discussed in the previous section). At this point, with both users notified, the agents' task is deemed complete, and they are terminated. It is left up to the users to actually consummate the deal, i.e. exchange the good or service for monetary payment. Kasbah does not provide a mechanism for electronic payment or shipping of goods (although future versions may do so for digital goods). We see Kasbah simply as a useful tool to relieve end-consumers of the bulk of the burden involving in buying and selling.

An important matter is the extent to which a deal between two agents is considered binding, both in a legal and social sense. What happens if one of the users decides that they don't like the deal that their agent made for them? Can they then back out it? What happens to the other party, who might happen to *like* the deal that their agent made? Is there any recourse for them to take against the user which backed out of the deal? These questions raise important issues, ones which will have to be addressed as agents become more and more prevalent. Fundamentally, to what extent does the delegation of a task to an agent imply accepting responsibility for the

actions that the agent might take on the user's behalf. For example, can I say that my agent didn't work the way I expected it to (or the way it was promised to me to work), therefore I'm not obligated to accept any deal that it made on my behalf?

Our approach to these issues in the current version of Kasbah is to make clear to users that there is NO legal obligation to accept the deal that their agents make for them. To answer the questions posited above, if the user doesn't like a deal that their agent made, they are free to not consummate it. If this offends the other party, that is too bad. There is no official recourse they can take. This may seem a somewhat harsh and cavalier attitude, but we believe a laissez-faire approach is the most realistic. After all, if you contact someone through the newspaper classifieds, agree on a deal, and then they back out of it, what recourse do you have other than to be angry and vow to never deal with that person again. The newspaper is not responsible for the possible reprehensible actions taken by those who advertise in it. The assumption we make is that *most* of the people who use Kasbah, like those who use the newspaper classifieds, are essentially honest and not out to defraud others. For future versions of Kasbah, it has been proposed that a "blacklist" functionality be added that would allow a user to instruct their agents NOT to negotiate with any agents owned by certain users. If a user has had a bad experience dealing with certain people, they can blacklist them. One important benefit of a blacklisting mechanism is that it allows users to choose not to deal with certain people, without in any way publicly tarnishing the reputation of these people. This is in contrast to other approaches for dealing with malicious users, such as a "Better Business Bureau". Blacklisting would work not only for dealing with people that back out of deals, but those who engage in fraudulent advertising through their agents, i.e. selling something which they don't have (or isn't in the condition they say it is), or attempting to buy something which they have no intention of paying for. One disadvantage of personal blacklisting as opposed to a “Better Business Bureaus” approach is that it doesn’t provide a way for users to share information about malicious users.

The consummation of a deal is the responsibility of both users (the buyer and the seller). Contact information is provided, but they must agree upon a meeting place and terms of the good exchange and payment. These terms will depend on a number of factors. For large-sized goods, like furniture, in most cases the buyer will come and pick up the good. At the time of the pick-up, the payment would be made. In these cases, then, it may be required that the buyer and seller live within a reasonable distance of each other. For smaller kinds of goods, such as compact discs, both parties may choose to mail the good and payment to each other. This kind of arrangement happens all the time, particularly for low-value items like CDs. In such cases, buyer and seller don't care about how close they live to each other.

In future versions of Kasbah, we would like to add the notion of geographic locale. That is, when signing up for an account, a user would indicate where they live (via an address, city, state, zip code, etc.). A user could instruct their agents to only negotiate with agents whose owners live within a specified proximity (e.g., same city, same state, 100 miles, whatever) of themself. This would be a convenient feature when a user only wants to make a deal with someone that lives nearby.

Sample scenario

So far in this chapter, we have described the many pieces of Kasbah's functionality. In this section, we attempt to pull it all together by presenting a fictional (and semi-humorous) scenario of how Kasbah would be used by a typical end-user. The Kasbah system discussed is the prototype we built for transacting musical recordings --- the one we have used in the examples and Figures.

Suppose we have a user named Mike. Mike is a poor graduate student, and one of his big pleasures in life is listening to music. Mike has a CD which he bought long ago and never listens to anymore, Prince's "Purple Rain" album. He would really like to buy a CD recording of Mahler's

4th Symphony. Because Mike is a poor student, he can't afford to buy the Mahler CD. Mike, if he could, would sell his Prince "Purple Rain" CD and use the money from that to buy the Mahler. He has tried to find a friend who likes Prince and who wants an old "Purple Rain" album, but alas, Prince is not too popular these days. Mike is pretty sure that there must be people on earth who are big Prince fans and who would buy his CD, but how can he find these people? He's certainly not going to spend a lot of time on it, because he's busy. However, if there were an easy way to find buyers for his Prince CD and sell it to one of them, Mike would definitely take advantage of it. Fortunately for Mike, there is Kasbah.

Mike goes to the Kasbah web site for selling music, logs in (creating an account if he already doesn't have one), and selects "create new selling agent". He fills out the information necessary to create an agent to sell his CD: the artist and title, the date to sell by (say, one week from today), his desired price for the CD (say 15 dollars), the lowest price he is willing to sell it for (say 7 dollars), and the desired pricing strategy (say "Anxious" because Mike wants to buy that Mahler CD as soon as possible). Having specified this information, Mike clicks on the "create agent" button, and voila, his agent is created and begins working for him. The next day, Mike gets an email status report message from his agent. The report says that the agent is currently negotiating with four agents interested in buying the Prince CD, its current asking price is 14 dollars, the average price being offered by the four potential buyers is 8 dollars, and that the best price being offered is 10 dollars. The report also says that there are two other agents in the marketplace that are also selling the same Prince CD, and that their average asking price is 13 dollars, with the best price being 12 dollars. Based upon this information, Mike decides to drop down his agent's asking price to 11 dollars, as he is concerned that the other agents selling Prince CD's may take away his most promising buyers. Dropping the price also increases the likelihood of making a deal soon, and Mike is getting impatient. Mike logs into the Kasbah site, goes to the "Change agent parameters" page, and change the agent's current asking price to 11 dollars. A few hours later, Mike gets an email from his agent saying that it has made deal with an agent owned by a person named Drew for a price of 11 dollars. The message gives Drew's

email address as well as his phone number. It turns out that Mike and Drew work in the same building but had never met. (Note: one might question the likelihood of such fortuity, but Kasbah systems can be set up for specific, localized communities, e.g. MIT. This is in fact the plan for future versions of Kasbah.) Mike fires off an email to Drew saying that he accepts the deal his agent made, and asks if Drew also accepts. Drew responds with an email saying that he does accept the deal and proposes that they meet at location X at time Y to consummate the deal. Which they do -- Mike gets his 11 dollars so he can buy the Mahler CD, which he does using Kasbah of course, Drew gets his long-desired Prince CD, and both are pleased because their Kasbah agents did most of the hard work for them.

Chapter 3: Related Work

Kasbah falls into three classes of related systems: software agents, market-based system, and electronic commerce systems. In this chapter, we will discuss representative examples from each class, and how Kasbah compares, both in terms of similarities and differences.

Software agents

The selling and buying agents in Kasbah, described in the previous chapter, are software agents, in that they perform some specific task on behalf of users. Software agents can be further subdivided into three classes of systems: eager assistants, matchmaking agents, and collaborative filtering agents. We will provide examples of agents from each of these categories.

Eager assistants

Eager assistants are agents that watch what you do, and based upon what they observe, make pro-active efforts to help you by automating tasks on your behalf. There have been many eager assistants built [Cypher91] [Dent92] [Maes93] [Kozierok93]. We will now describe one such system in detail.

Maxims is an agent-based email prioritization system that was developed at the MIT Media Lab [Lashkari94]. The purpose of Maxims is to help people deal with the problem of email overload, something that many people who use email on a daily basis can attest too. What Maxims does is to "watch over the user's shoulder", and learn their email reading habits.

Once it has learned these habits to a sufficient degree of confidence, the Maxims agent will make recommendations as to what the user should do with incoming email. These recommendations are: "read immediately", "delete", or "file for later reading". The Maxims agent also has an anthrophomorphic representation, i.e. a very simple face cariacture that has several different

poses. Based upon the agent's level of confidence in its recommendations, the agent's face will assume the appropriate pose. For example, if the agent is very confident, then its face will light up brightly. If the agent is not so confident, its face will have a doubtful expression. Maxims is a good example of an agent that is highly *pro-active* (eager). In other words, the user doesn't have to do anything to use a Maxims agent. After the agent is installed, the user just reads their email like they normally do. The agent silently watches what the user does until it has detected statistical patterns in the user's habits, at which point it will begin to make its recommendations, which the user is free to accept or ignore. For example, my Maxims agent might notice that I always delete mail that comes from the "Fresh Trout Lovers" mailing list that I subscribed to long ago and never got around to removing myself from. My agent might also notice that whenever I get email from my thesis advisor Pattie, I read it immediately. Based upon this evidence, the agent will recommend that I "delete immediately" mail that comes from the "Fresh Trout Lovers" mailing list. It will also recommend that I "read immediately" mail that comes from my advisor. These recommendations are indicated by the agent putting an appropriate symbol next to the email entries in my inbox. This prioritization of email can be useful to those who have the experience of sitting down at their desk, firing up their mail reader, and being overwhelmed by hundreds of new messages in their inbox. The Maxims agent provides a way to help deal with this information overload. Maxims has been deployed and found useful within a limited community of users.

A Maxims agent is similar to a Kasbah agent in that it performs some useful task for a user - with Maxims it is prioritizing email, with Kasbah finding a good deal. However, how the agents are instructed is different. The Maxims agent requires no explicit instruction from the user, instead it implicitly learns their email-reading patterns by watching their behavior. The Maxims agent is entirely autonomous. A Kasbah agent, on the other hand, requires the user to explicitly instruct it - namely, describing the good to sell or buy, and setting the control parameters. This difference is natural and to be expected, given the different nature and purpose of the two systems. Maxims and Kasbah also differ in that Maxims is strictly a single-agent system (one agent per user), while

Kasbah is fundamentally multi-agent. Not only may a Kasbah user have multiple agents making deals on their behalf at the same time, in order for *any* deal to be made requires there to be multiple agents present in the marketplace. In fact, the success of Kasbah hinges on there being a large number of agents present, so that the marketplace may be considered viable. After all, if only a handful of agents are present in the Kasbah marketplace, who would use to find a good deal on something?

Multi-agent matchmaking systems

Matchmaking systems are those that pair up users with similar interests. Examples of such systems are Yenta and work in [Kuokka95]. We will now describe Yenta in detail.

Yenta is a multi-agent system for matchmaking that is currently being developed at the MIT Media Lab [Foner96]. The purpose of Yenta is to help users find other people with similar interests on the Internet. It is motivated by the observation that although the Internet makes it easy to contact millions of other people, it doesn't help you find the much smaller set of people who you share interests with, and who you would actually be interested in contacting. Yenta is an agent that runs as a background process on a user's workstation. The agent learns a profile of the interests of its user by examining various contents on their machine - mail folders, saved news articles, documents, etc. For example, a Yenta agent sitting on my machine would learn, by looking at the files in my directory, that I am interested in software agents, sports, and writing my thesis. Armed with this knowledge of my interests, my Yenta agent goes out on the Internet and tries to find other Yenta agents whose users have some of the same interests that I do. This matching up of Yenta agents whose users have similar profiles is done in a completely decentralized fashion, where each agent first contacts its neighbor Yentas and through a "word-of-mouth" referral technique is directed towards agents with matching interests. Once your Yenta has located other users that you share interests with, it provides a way of doing introductions and "flirtations" that allow you to get acquainted and exchange information , while

still protecting privacy. These methods are described in detail in [Foner96]. The goal of the Yenta project is to have thousands, and perhaps millions, of Yentas deployed across the Internet.

Yenta is a more autonomous system than Kasbah. Similar to Maxims, it implicitly "learns" the tastes and interests of its user. It then goes off and tries to find other Yentas whose users have similar interests. All of this happens without need for user interaction. The only direct interaction with the user occurs when their Yenta notifies them that it has found people with similar interests. At this point, the user may use the Yenta's capacity to facilitiate "introductions" with these people. As with Maxims, the difference between how Yenta and Kasbah agents are instructed is due to the different nature of their tasks. A Kasbah agent has a very specific, one-time job that requires explicit instructions as to precisely what the user's wishes are (expressed through the control parameters), while a Yenta agent has a much broader, longer-range directive - learn the user's interests and search for people with the same ones - that does not require explicit user parameterization.

Unlike Maxims, but like Kasbah, Yenta is a fundamentally multi-agent system. A Yenta agent cannot succeed in its mission (to find other Yentas whose users have the same interests) unless there are other Yentas on the Internet, and the more Yentas, the better. Kasbah and Yenta are dependent for their success on the existence on the proliferation of their agents.

Collaborative filtering agents

There is whole class of work on collaborative filtering (CF) agents. Examples include GroupLens, Ringo, and HOMR, to name just a few [Resnick94] [Shardanand94] [Shardanand95]. We have chosen to discuss the Firefly system as a representative example.

Firefly is a collaborative-filtering system for making personalized recommendations from a catalogue. Firefly also has a showcase site for recommending movies and music [Firefly].

Firefly is a commercial site and enjoys extensive public use - over a million people have used Firefly, and there are tens of thousands of regular users. Firefly may be considered a software agent that makes recommendations for users, although some have argued that Firefly does not possess enough of the characteristics of a software agent - namely, it is not autonomous or long-lived enough - to be truly considered one [Foner95].

The way Firefly works is by having users rate things in some domain (for example, music albums and movies) on a scale from 1 to 7, with 1 indicating extreme dislike and 7 indicating extreme like. Once the user has rated enough items to form a profile of their tastes, they can ask their Firefly agent to recommend other items (other music albums or movies) which they would like. The Firefly agent makes these recommendations based upon statistical techniques. Namely, the agent looks for the agents of other users whose tastes most closely match those of their owner - these are the user's "nearest neighbors". The agent then recommends to its user the items which their "nearest neighbors" have rated highly but they have not yet rated. The essential principle at work here is this: "if I like A and you like A, then if I like B, you will probably like B too". This simple yet intuitive technique can make surprisingly good recommendations. As the user rates more and more items over time, and more users are added to the system (increasing the number of potential nearest neighbors to choose from), their set of "nearest neighbors" will change, and their agent will make different recommendations - hopefully ones which better reflect the user's actual taste.

In some respects, Firefly is similar to Yenta. It is a multi-agent system that learns its user's tastes, and based upon this helps them find others with similar tastes. However, there are major differences. For starters, the user has to explicitly tell the Firefly "agent" what their tastes are. The Yenta agent autonomously learns the user's interest profile by examining contents on their workstation. Also, the Firefly agent only "runs" - that is, makes recommendations - when the user explicitly asks it to. In this sense, it is not autonomous like the Yenta agent, which silently works

in the background. Another difference between the two systems is that Firefly recommends items from some domain, while Yenta just recommends people.

The relevance of Firefly to Kasbah is its demonstration that a multi-agent system which requires many users, can be successful. Firefly and Kasbah both lure users with the promise of doing something useful or interesting for them - in Firefly's case it is recommending items while in Kasbah's case it is finding the best deal. Unless enough people can be enticed to use the system, though, this promise cannot be fulfilled. Firefly has shown that if the agents do what they purport to - make good recommendations - and are simple and easy enough to use, then they can succeed in the real world.

Market-Based Systems

Since Kasbah matches up willing buyers and sellers in an on-line marketplace, Kasbah can be considered a market-based system. Indeed, a multi-agent system like Kasbah fundamentally lends itself to modelling a marketplace. The basic tenet of market-based systems is this:
Each agent acts in its self-interest, i.e. follows a set of rules designed to maximize its own utility. The net result of the interaction of these agents is hoped to be better than what could be achieved through a centralized solution.
The motivating force behind the growing popularity of market-based systems is the observation that a decentralized approach is the best, if not only way, for addressing certain classes of problems. One such class of problems that market-based techniques have been extensively applied to is that of resource allocation. The field is known as market-based control [Clearwater96]. Resource allocation problems are known to be very difficult, if not impossible, to solve directly. Instead, the problem is modelled as a collection of agents each trying to buy, sell, exchange, or trade something in order to maximize its happiness, just like what is presumed to happen in real-world markets (i.e. people are greedy). The exact nature of the market model (what the agents buy and sell, and how - trade, auction, with money or not) is determined by the particular problem in question. Although each agent is acting selfishly, the net result can be, if the

problem is modelled correctly, an optimal, or near-optimal, global allocation of the resource in question. This can be a quite startling result - after all, the agents are not acting in concert for the "common good". [Clearwater96] provides many impressive examples of the effectiveness of market-based control for dealing with a wide range of problems.

Challenger is a multi-agent system that uses market-based control to address the problem of efficient allocation of processor resources in a collection of networked workstations [Chavez97]. This work is motivated by the real-world observation that loads between machines are often highly variable, i.e. you may be running many compute-intensive jobs on your machine, slowing down the response time of your editor, say, while the loads on the machines of your colleagues down the hall are near zero. It would be nice if you could run some of your jobs on these currently unloaded machines. And when your machine's load is light, you colleagues could run some of their heavy jobs on your machine. The goal is to better harness the global processing power of the community.

The Challenger model makes a number of assumptions. One is that a task originating on one machine can be run on any of the other machines. Another is that all of the tasks are batchable, i.e. they do not require any user interaction to run. Many tasks fall into this category, and with the advent of Java, the interoperability assumption may no longer be a concern. The Challenger model has the following high-level features:

- for each machine in the network, there is an associated Challenger agent
- whenever a task is originated on a machine, the associated agent decides whether that task should be run locally or if it should be put up "for bid"
- if the agent decides to put the task up "for bid", it sends out a message to all of the agents in the network, asking them to make bids on the task. Included in this message is the agent's estimated completion time of the task (which is why they need to be batch tasks) (Also, how the agent makes this estimation is dependent on the nature of the task. If it is an image

processing task, for example, the estimate may be linearly proportional to the size of the image being processed.)

- the other agents receive the request for a bid, and return a bid giving their estimate of when they would complete the task, if it were to be assigned to them
- the originating agent collects the bids, selects the best one, and assigns the task to that agent. Note that the agent may assign the task to itself.
- the "winning" agent receives the task, runs it, and returns the result to the originating agent.

It turns out that, for a wide range of conditions (different machine loads, network delays, task lengths, network configurations, etc.) the Challenger system yields better mean task completion time, often significantly better, than running all jobs locally. This is not such a surprising result, and has been shown in previous work [Malone88]. However, in this previous work, under certain network conditions the "bidding" method of assigning tasks could produce much worse results than just running the jobs locally.

The main contribution of Challenger is to show that by improving the "smarts" of the agent, i.e. the rules that it uses to assign tasks, it is possible for the system to *never* yield results any worse than the case where all jobs are run locally. In other words, in the worst case, the Challenger system performed no worse than running the tasks locally; in all other conditions, Challenger performed better.

The main observation to take away from this is the critical importance of the behavior of the individual agent in a decentralized, market-based system. The better the agent performs, the much more likely one is to achieve an optimal global solution.

Kasbah is similar to Challenger and other market-based control systems in that its goal is also to efficiently allocate resources - after all, Kasbah creates a marketplace, and the definition of an "effective" market is one which is efficient. However, we do not approach Kasbah from the perspective of trying to create the optimal marketplace. We are really not trying to maximize societal utility, though if this results that is welcome. Rather, our perspective is end-user and

task-oriented. Kasbah provides a service that people will wish to use because it benefits them, that maximizes their own utility. It would be difficult to convince people to use Kasbah just because it creates an efficient global marketplace. People will want to use Kasbah because it finds them the best deal and saves them valuable time and energy.

The focus on performing a task for a real end-user makes Kasbah very different from Challenger and systems like it. Market-based systems usually model an abstract problem, e.g. processor resource allocation. The agents in these systems are conceptual entities, much like a procedure or method, which are a useful way of decomposing a complex problem. They typically do not have any interaction with a human end-user, like Kasbah agents do. This difference deeply effects how the agents are designed. Kasbah agents must be able to interact with and understand the desires of their human owners. This puts additional requirements and constraints on the design of these agents.

Electronic Commerce on the Web

Electronic commerce is becoming a major buzzword. All over the web, there are sites where people can buy and sell things, the range and diversity of which probably now exceeds that found in traditional newspaper classifieds. Electronic commerce has also become a focus of many commercial software companies, such as Netscape and Microsoft. Since Kasbah is an electronic commerce as well as an agent system, we now examine a few of these systems that exist today.

Classified Ads Style Sites

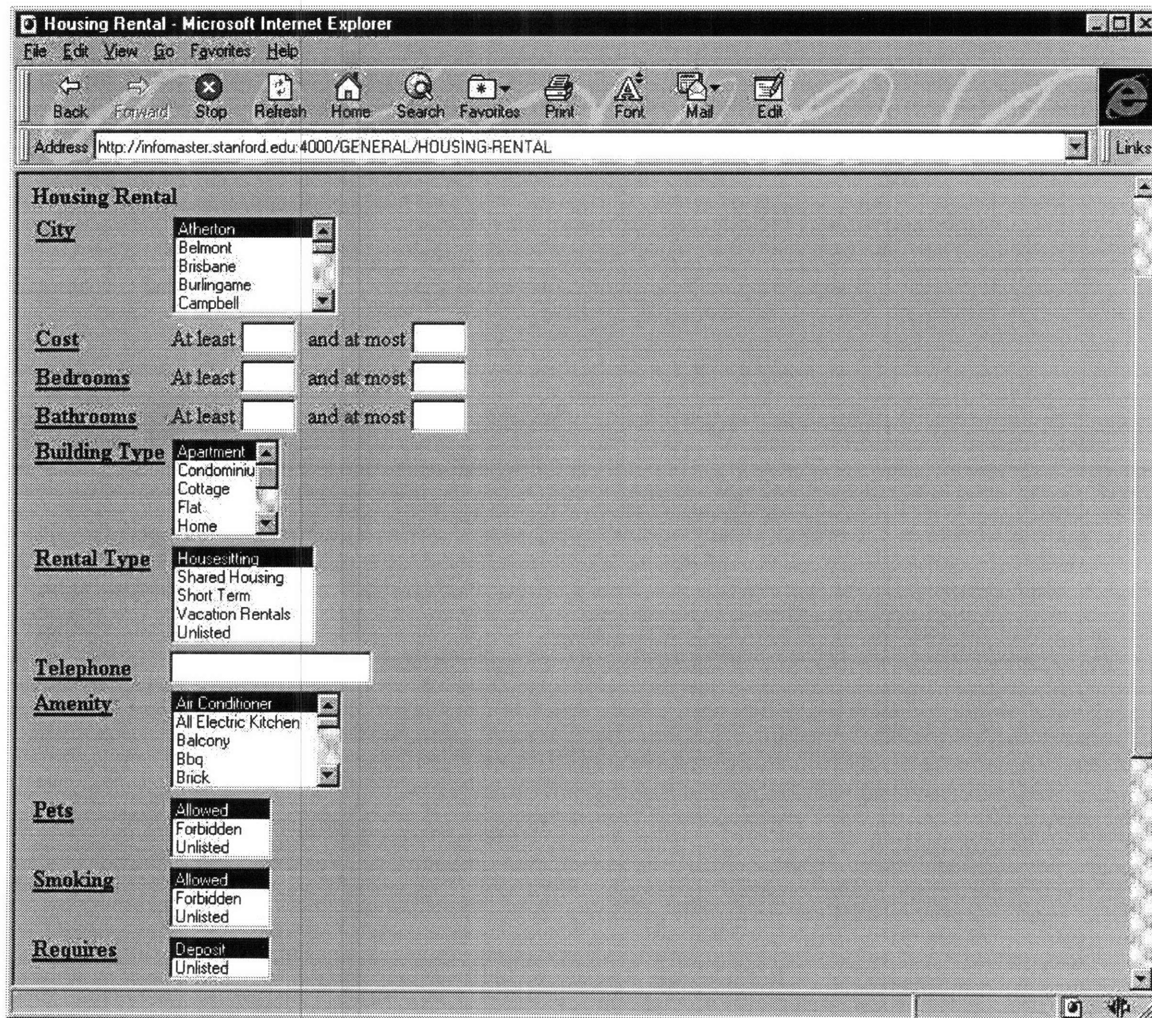
As noted in the Chapter 1, there are many classified-ads style Web sites where people can read posted ads, post ads themselves, or do both. A good example of such a site is the Seattle Times Classifieds [SeaTimes], although there are numerous others. Some of these sites allow people to post new ads directly (through an HTML form usually), while others, usually those tied to newspapers, require the ad to be submitted through conventional means (over the telephone), and then the ad gets automatically posted on the web site. Most of the classified sites also

provide some form of search functionality. Typically this is just AND/OR keyword searching. Some sites, though, allow for richer and more structured searching for ads of interest.

A very impressive site in this regard is InfoMaster, developed at Stanford [InfoMaster][Geddis95], which provides an apartment-finding service for the university community. InfoMaster takes ads for apartments from several Bay Area newspapers, parses them, and stores them in a structured way. Users of InfoMaster can specify, to a high degree of accuracy, the type of apartment they are looking for. An example specification might be, "at least 2 bedrooms, at least 1 bath, a fireplace, and price no more than 1500 dollars per month". Figure 11 below is a screenshot from the InfoMaster UI showing how the user indicates their specifications. InfoMaster will return to the user the listing of the ads (both in the structured and original format) that meet their criteria. Of course, InfoMaster is limited to the types of goods for which a structured representation has been developed, and for which an effective automated way of transforming the original textual ad into the structured format has been developed. While representing ads in a structured way allows InfoMaster to do more accurate searching, there is a limitation. In particular, the structure is fixed, so there may be features of some apartments that are not describable to the system. Ideally, one would like to have an extensible description language, but such languages, like KQML [Finin93] are in reality very difficult to implement.

Figure 11: Infomaster

UI



Kasbah is similar to SeaTimes Classifieds and InfoMaster in that it provides a site where people go to find items of interests. SeaTimes and InfoMaster, though, are primarily oriented towards the buyer – that is, the people that go to their site are usually looking to purchase something. Kasbah, on the other hard, is intended for both buyers and sellers. People wanting to buy and sell do the same thing - create a Kasbah agent to do the task for them.

One difference between these sites and Kasbah is that the former do not support "standing queries", the ability for the user to specify what they are looking for and have the system do that

searching over a prolonged period of time. Kasbah agents, on the other hand, inherently have this ability.

Describing the good in question to the Kasbah agent is very similar to how a person describes to InfoMaster what type of apartment they are seeking. Indeed, the same philosophy is used - having a fixed structured representation of the good (music for Kasbah, apartments for InfoMaster) that is general enough to cover most cases. Most classified sites on the Web today help people find goods they are interested in. While Kasbah does this too, its fundamental purpose is to provide agents which do automated price negotiation on behalf of the user. In other words, Kasbah does not just assist people with finding, but with the actual process of buying and selling.

Price Comparison Agents

The closest systems we have found to Kasbah are those which do automated price comparison shopping. An excellent example of such a system is BargainFinder, developed by Andersen Consulting [BF]. BargainFinder is an agent which comparison shops for CD's. However, it is not a long-lived agent in the sense of Kasbah agents. The user specifies a CD, and then BargainFinder goes and searches about five on-line music stores, looking for the store which has the CD for the best price, and reports its findings back to the user. This service can be very useful to people who want to find the best deal on CD's, but don't want to spend the time and effort to comparison shop themselves.

Like Kasbah, BargainFinder tries to find the best possible deal for the user. BargainFinder's model, though, is that of on-line stores selling to end-users, whereas Kasbah's model is that of end-users selling directly to end-users. Consequently, BargainFinder agents do not have the capability to negotiate - after all, one does not usually haggle over price with the record store. BargainFinder also differs from Kasbah in that it is not a "persistent" agent that exists for an extended period of time. BargainFinder is a "one-shot" agent; that is, when the user explicitly directs it to, it does its comparison shopping task, returns the results to the user, and ceases to

function until instructed to by the user. So it is not very autonomous or pro-active. Kasbah agents, on the other hand, can have a very long life-span (for as long as the user specifies in the "date to sell/buy by" parameter). Once the user has created a Kasbah agent, it begins to negotiate with other agents and will continue to do this until it makes a deal, expires, or is terminated by the user. While the agent is "alive", the user can interact with it, both by changing its control parameters, and through status reports.

The Kasbah agent appears to the end-user to be a real "live" entity, that has its own existence independent of the user. BargainFinder does not have this characteristic. It is fair to say that Kasbah is a much more agent-like than BargainFinder.

The success of BargainFinder is questionable, because several of the on-line music shops used in the comparison shopping have locked it out. The exact reason for this is unknown, but it is likely that these stores do not want their prices to compared to those of other stores. Of course, there is nothing to prevent people from going from store to store - or rather, site to site, doing the comparison shopping themselves, but the idea of an agent that automates this process could be seen as a threat. In the past, stores could get away with charging higher prices than their competitors because they could be fairly assured that most customers would not take the time and effort to do comparison shopping. With BargainFinder making this task so easy, the stores no longer have this assurance. So perhaps the stores have taken the protective measure of locking the BargainFinder agent out.

Kasbah does not have this problem to worry about. Since its model of commerce is a marketplace for end-users, it does not rely on any on-line stores - it is a completely self-contained system.

AdHound

Another system that of interest is AdHound [AdHound]. AdHound is an agent that will search for ads of interest in a single database over an extended period of time. You tell your AdHound agent what things you're interested in, e.g. automobiles with such and such characteristics, and

the agent will search the ad database on a periodic basis and report back to you matches that it finds.

Description of a good's characteristics and the matching against ads in the database is done with keywords. The results of the searches, the frequency of which can be specified by the user (e.g., daily or weekly), are reported back to the user through email. A user can have multiple AdHound agents alive at once - for instance, one agent could be looking for a certain pair of concert tickets, and another agent could be looking for a Harley motorcycle. Once a user has created an agent, it remains alive, continuing to search on a regular basis, until the user explicitly terminates it.

There are many ways in which AdHound is similar to Kasbah. First of all, the AdHound agent has the characteristic of being a persistent, autonomous entity, which exists outside and independently of the user. Once the AdHound agent is created and been assigned its task, it will work on it over time, without requiring further interaction with the user. Like in Kasbah, AdHound users can have multiple agents all working on their behalf. Also, AdHound agents send back periodic status reports to the user indicating what they have found. In terms of differences, AdHound is geared towards people looking to buy things, while in Kasbah buyers and sellers are equally emphasized. Kasbah agents also may been seen as possessing a superset of the functionality of an AdHound agent. Kasbah agents are capable of identifying agents which may be potential customers, i.e. comparing their own item description with that of other agents. This capability may be considered analogous to the AdHound agent's ability to search its ad database to find matches. Once the AdHound agent has found matches, it notifies the user. The Kasbah agent does not stop with finding matches, but engages in the process of negotiating to find the best deal it can. The Kasbah agent has a much bigger job than the AdHound agent.

Other related work

We should briefly mention a couple of other research areas which relate to Kasbah, that do not fall cleanly into the category of software agents.

One area in which a lot of research has been done is agent negotiation [Rosenschein94] [Zlotkin93]. This work tends to be very theoretical, and thus we found it difficult to see how it could be directly applied to building Kasbah agents. This type of work is good for solving problems like the prisoner's dilemma, but we did not found it very useful in terms of building systems that have to interact with real users.

Another area of relevance is agent communication. KQML is perhaps the most notable attempt to build a general purpose agent communication language [Finin93] [Labrou94]. However, it is very complex and difficult to implement, so we chose not to use it and instead went with a simple, custom language that suited the communication needs of our simple agents. Related to agent communication is the study of "speech acts" [Winograd86]. Again, while there are interesting theoretical results to be found in this work, we did not seem a direct application to the kind of systems we were trying to build. Perhaps this is one reason why the trend in agent research over the past several years has been leaning towards building more practical, working systems, rather than conducting theoretical simulations.

Sumary of related work

In this chapter, we have described several systems that have relevance to Kasbah. These systems cross a wide range of overlapping fields – from eager assistant agents to multi-agent systems, to market-based systems to commercial Web sites - which makes sense since Kasbah incorporates ideas from all of them.

In designing and building Kasbah, our goal was not to try to build a "software agent", or "multi-agent system", or "market-based system", or a commercial web site, but to build a service that would prove valuable to the end-user. While there are systems which are similar in certain ways to Kasbah, we have not found a system which has its central feature: the ability to negotiate to find the best deal on behalf of the user. In this regard, we believe that Kasbah is unique. The role of agent as negotiator, haggler, and deal-maker is a new one.

Chapter 4: Architecture and Implementation

In this chapter, we describe the architecture and implementation of a prototype Kasbah system. In particular, we focus on the Kasbah system that was described in the functional overview in Chapter 2 - the one for transacting music. In addition to this system, a couple of other Kasbah systems have been built. One is the precursor to the aforementioned Kasbah. It was an early prototype and was used to conduct the initial experiment performed to assess the viability of the Kasbah concept, which is discussed in the next chapter. The other is a slightly modified Kasbah system that was used to conduct a real-life, one-day agent marketplace experiment involving a couple of hundred participants. This experiment is discussed at length in the next chapter. All of these Kasbah systems share the same basic architecture and a good deal of code. However, they differ certain regards, especially in terms of user interface. Although our discussion here is primarily focused on the "music-based" Kasbah, we point out which components are specific to this particular system, and which are more general-purpose.

Overall Architecture

At a high-level, the Kasbah architecture consists of two primary components: the front-end interface and the back-end engine. These components are briefly described below.

- Front-end: a Web interface that defines how the user interacts with Kasbah. It consists of a set of CGI scripts. The front-end varied extensively between the different Kasbah prototypes.
- Back-end: the marketplace engine is where the Kasbah agents actually "live" and interact with one another. The back-end was initially implemented in Common Lisp; later it was ported to Java for ease-of-programming and interoperability reasons. The same back-end, with minor changes to account for the different kinds of goods being transacted, was used in all three Kasbah prototypes. The Java version of the backend consisted of about 6000 lines of code.

This architecture was intended to be fairly well modularized, so that changes to the front-end would cause at most minimal changes to be made to the back-end, and vice versa. The interface

between the front-end and back-end was designed to be as general-purpose as possible. This decision proved to be a wise one, as the front-end design of our prototypes was constantly undergoing changes, yet these modifications were able to be made without requiring many changes to the back-end.

Physical topology

The physical topology of the Kasbah system is simple. The front-end CGI scripts reside on a Web server. The back-end marketplace engine is located on a high-performance workstation that is capable of running Java. The end-user interacts interacts with Kasbah through a Web browser, e.g. Netscape Navigator, that is running on their local machine. The front-end and back-end servers communicate with each other using a custom protocol over TCP/IP sockets. The front-end Web server and client browser communicate via standard HTTP.

Marketplace engine

The back-end marketplace engine is the "core" of Kasbah. It is implemented as a request-response server: a client (the front-end CGI scripts) sends it a request, the marketplace engine services that request, and then sends back to the client a response containing the results of the request operation. From a high-level functional viewpoint, here are the marketplace services that the back-end provides:

- Create a new user account in the marketplace
- Create a new selling or buying agent for a given user
- List all of a user's agents and their corresponding properties.
- Delete a specified agent for a given user.
- Modify parameters of a specified agent for a given user.
- Get current market data (this feature exists for only one of the Kasbah prototypes; normally this data is only accessible to agents operating within the marketplace)

The front-end sends requests for the above services to the back-end via a simple, string-based protocol that we devised.

In addition to handling requests from the front-end, the back-end is where the agents actually do their negotiating and deal-making. Conceptually, one should think of the back-end as the place where the agents "live", are "running around" talking to one another, "haggling", trying to find the best deal on behalf of the user, all in parallel. In fact, as will be described, the agents are not really operating in parallel, but this is a useful way of thinking about it. The actual implementation of the back-end could , in the future, be changed in such a way -- for instance, becoming more distributed - that they really do run in parallel.

Selling and buying agents

Software agents are long-lived programs that perform some task on behalf of a user [Maes94]. In Kasbah, agents try to buy or sell some good for the best possible price on behalf of the user which created it. To accomplish this task, an agent negotiates with other users' agents in the marketplace, trying to find the best deal subject to a set of user-specified constraints. Once two agents "make a deal" with one another, they notify their respective owners so that they can (physically) consummate the transaction. In Chapter 2, we described the control parameters that the user specifies when creating a new agent. By specifying these parameters, the user defines and constrains the behavior of his or her agent. The overall behavior of the agents is to be assumed by the user to be competent for the task at hand. We will briefly describe the behavior of a selling agent (with buying agents behaving in a symmetrical fashion), and then delve into the technical details of how these behaviors are implemented.

The selling agent is constantly looking for potential buyers. It does this throughout its lifespan, or until it makes a deal. The agent initially starts by offering the good in question to the potential buyers for the user-specified desired price. If the agent finds someone willing to pay that price or higher, then it makes the deal (at the highest price). Otherwise, the agent lowers its asking price and makes a new offer to the potential buyers. By the end of its "lifespan" (determined by the user-specified date to sell by), if the agent has not yet made a deal, it will have lowered its asking

price to the user-specified lowest acceptable price. The rate at which the agent lowers its asking price is dependent upon the user-specified strategy. An "anxious" agent, in a hurry to sell, will lower its price very quickly toward the lowest acceptable price. A "frugal" agent, on the other hand, will initially lower its price slowly, waiting until the very end of the day before lowering its price substantially. A "cool-headed" agent will strike a balance between these extreme strategies."

The description of the above behaviors is highly anthropomorphic; this is how we believe that end-users perceive how their agents work. From observations made during experiments conducted with Kasbah, this was indeed born out. It appeared that end-users had little difficulty in understanding what their agents were doing, although there were some who ascribed more "intelligence" to their agents than was actually present. End-user reaction to Kasbah is described at greater length in the next chapter.

Agents as objects

The back-end marketplace engine is implemented as a continually running Java (originally Common Lisp) program. All of the marketplace and agent activity takes place within this process. We used an object-oriented design approach to building our agents. Each agent is an instance of a Java class. The object-oriented paradigm lends itself very well to modelling agents, since each agent can be thought of as an object with its own state (memory) and methods (behaviors).

Agent Communication

In order for agents to negotiate in the marketplace, they must be able to communicate with one another. Agent communication in Kasbah is based upon a simple request-response model. All communications are strictly agent-to-agent; there is no broadcasting of messages, and an agent cannot eavesdrop on a conversation between two other agents. The flow of a single "conversation" between two agents x and y is as follows.

1. agent x sends request message to agent y
2. agent y processes request message
3. agent y sends response message back to agent x

In our prototype implementations, there were two types of request messages that an agent could send:

- "What is your current asking price?" The agent responds with its current asking price (i.e., the price at which it is willing to buy or sell the good in question).
- "I offer (to sell or buy) the good in question for X dollars. Will you accept?" The agent responds with either a "yes" or "no". Note that this request is considered binding; that is, the agent making the offer is implying that if the response is "yes", it is bound to make a deal at that price. Likewise, the agent responding "yes" has bound itself as well.

Clearly this conversation space is very simple, yet we found it adequate. For more sophisticated agent interactions, a richer language would be needed. The chapter on Future Work expands on this. A great deal of research has been done in the area of agent communications. One question is why we did not choose to use a general-purpose agent communication language like KQML. The answer is complexity. KQML does provide an extensible, robust, language, but for our purpose, it was overkill. Since all of our agents are just Java objects, they can communicate by supporting a predefined set of methods.

One important reason to have an agent communication language is that it allows agents to negotiate with one another without requiring them to have intimate knowledge of how they work. One of our goals in the design of Kasbah was to have it be agent-independent. That is, as long as an agent can speak the common marketplace "language", i.e. can support the appropriate methods, then it can participate in the marketplace. Our architecture thus allows outside parties, or ourselves, in the future, to create new kinds of agents with new behaviors, and have these agents be easily integrated into Kasbah. This extensibility and flexibility is very important if Kasbah, or systems like it, are to be viable in the real world.

Item Descriptions

As discussed in Chapter 2, Kasbah agents must be able to understand what item their owner wishes to buy or sell. To avoid the difficult problem of developing a general-purpose item description language, our implementations of Kasbah constrained the type of goods that they could transact. For two of our prototypes, the type of good that could be bought and sold was hard-wired into the system, so there could never be any ambiguity about what good the user was describing - two items descriptions either matched, or didn't. For the "musical" Kasbah, though, we did face the problem of having to do "soft" or "fuzzy" matching. The structure for describing albums for both buyers and sellers is given below.

- FOR SELLERS: genre (one of rock, pop, classical, etc.); title (user text); artist (user text); condition (one multiple choice); description (user text).
- FOR BUYERS: title (user text); artist (user text); acceptable conditions (multiple choice); keywords (user text)

There are three possible axes on which we want to do comparisons:

1. Compare seller descriptions to see if they are selling the same item
2. Compare a buyer description against a seller description and see if they are a match
3. Compare buyer descriptions to see if they are buying the same item.

How each of these comparisons is done is slightly different, since the semantics of the comparison are different.

Buyer against seller

This is the most common comparison. A buying agent will want to determine if the item the user specified is indeed what the selling agent is offering. Note that it is only buying agents which care about this; sellers don't care who they sell to. This comparison is done by first determining whether the condition of the album being sold meets the requirements specified by the buyer. If this is not so, then the descriptions are deemed not to match. If so, then the title and artist fields of the two descriptions are compared. This string matching routine yields a metric (a number between 0 and 1) which measures the "similarity" between these fields for the two ads. If the

metric is 1.0, then the descriptions are considered a "perfect" match. If the metric is less than some configurable threshold value, then the descriptions are deemed not to match. If the metric is in between, then more comparisons are done. The buyer-specified keywords are used to do a keyword search against the raw text of the seller's description (all of the fields, including "description" and "genre"). This keyword match also yields a metric between 0 and 1, indicating the percentage of buyer-specified keywords which appeared in the seller's description. If this value is greater than some system-configurable threshold, then the descriptions are considered a "soft" match. Otherwise, the descriptions are deemed to NOT match. If a match with a sellers' album description is "soft", the buying agent will ask the user for confirmation that it should indeed be negotiating with this agent.

Seller against seller

This comparison is used to determine whether two selling agents are selling the same album. Note that there could potentially be many different types of comparisons that an agent would want to make. For example, a selling agent might want to know how many other agents are selling an album in the same genre as what it is selling. Or it might want to know how many agents are selling an album by the same artist. For our prototype, we only do a comparison for the same album. This comparison is done by matching the title and artist field (same routine as for buying - selling matching). If the match value are above some configurable threshold, say 0.7, then the descriptions are considered to match. Otherwise, they do not. Note that seller against seller comparisons have no notion of soft or fuzzy matching.

Buyer against buyer

This comparison is used to determine whether two bbuying agents are buying the same album. It is identical to how "seller - seller" comparisons are done. The comparison is based on whether the agents are buying the same album, regardless of the "acceptable conditions" and "keyword" fields - only the title and artist fields are compared.

The comparison and matching techniques described above are ad hoc, and only apply to the particular domain of music albums. That said, they were developed through much trial and error testing, trying to come up with a scheme that works reasonably well most of the time. For example, if I tell my buying agent that I wish to buy "Beethoven's 5th Symphony" (title) by the "Boston Symphony Orchestra" (artist), and some agent is selling an album described as "Betthoven's 5th Symphoni" by the "Doston Symphonee Orchestra", my agent will consider this to be a match. However, it is a soft match, so to be sure that it is actually negotiating to buy the album that I want, my agent will send a communication asking whether it should negotiate with the agent in question. In this example, my answer would be "yes", since the album descriptions are clearly referring to the same recording but have typographic errors. Suppose, though, that I had requested "Beethoven's 9th Symphoni" instead. My agent would still get a match, except that it this case the album descriptions are really referring to different albums. So when my agent asked whether it should negotiate with that agent, I would answer "no". Fundmentally we believe it is important that an agent know its own limitations. Certainly in an ideal world the agents would have AI capabilities and be able to tell, as accurately as a human, whether two item descriptions are referring to the same thing. Until this happens, agents must do the best they can. Sometimes this will mean asking for input from the user. We decided that it is better if the user be queried by its buying agent when it is in doubt as to whether it should be negotiating with some party, rather than having the agent make a deal for something which its owner does not wish to purchase.

Agent Behaviors

Earlier in this chapter we described the end-user's perceived behavior of their selling or buying agent. Let us now examine the actual implemented behavior of the agent. It is obvious that the actual behavior of the agent needs to correspond closely to the user's perception of how it behaves; if not, then the user is sure to be disappointed in its agent's actions or possibly upset (say if it makes a really bad deal). This is a general challenge facing agent researchers - namely, making sure that the agent lives up to the user's expectations.

Finding Interested Parties

One of an agent's core behaviors is to constantly look for agents which are interested parties, i.e. potential buyers or sellers. When a new agent is created, it will first find out who are the agents in the marketplace that it should be negotiating with. Likewise, the already existing agents will take note of the new agent and determine if they should be talking to it. This finding of interested parties can be done in two ways. One way, which was implemented in our first Kasbah prototype, was to have the agent handle all the work. That is, each agent was responsible for querying all of the other agents in the marketplace, asking, what are you buying or selling, and determining who it should talk to (this required the agents to support an additional conversation). We decided that this was inefficient and added a lot of unnecessary complexity to the agent implementation. In the current versions of Kasbah, the marketplace does the work of matching up agents. The marketplace supports a method which agents can call to find out who are other agents interested in (buying) selling what it is trying to sell (buy). An agent can call this method at regular intervals to see if any new agents have been added to the marketplace which it should contact. As noted before, selling agents don't care who they sell to. However, for practical as well as ethical reasons, a Kasbah selling agent will only initiate negotiations with agents who it believes wish to buy what it is selling. However, if a random agent comes along and makes a good offer, the selling agent will accept it, not matter what that agent is really looking to buy. Kasbah buying agents, on the other hand, will only negotiate with agents that are selling what it is interested in, as determined by item description comparisons (described above) as well as user feedback if necessary. This feedback takes the form of asking the user, "should I be negotiating with this agent?", for selling agents whose item description was a soft match to the agent's own. A buying agent will not make a deal with any other agents. A selling agents that believes it is a potential customer is free to make offers, but the buying agent will always return "no", even if the seller's offer meets the buyer's asking price. After all, a buyer isn't going to buy something that they don't want, even if it is a great deal.

Agent's current asking price

Selling and buying agents have an internal variable (or piece of state) called "current ask price". This is always set to the price which the agent is currently willing to sell or buy the good for. As time passes, the agent will adjust this price if it has not made a deal - selling agents will lower it, buying agents will raise it. This price will never be set lower than the user-specified lowest-acceptable price for selling agents, or higher than the user-specified highest-acceptable price for buying agents. When the agent is first created, "current ask price" is set to the desired price. How the agent adjusts its "current ask price" over time is described below. Agents are able to adjust their asking prices over time in order to make deals. Currently these price adjustments are completely deterministic, i.e. they follow some mathematical function of time. One can easily envision future Kasbah agents which adjust their price dynamically according to real-time marketplace conditions - e.g., the number of other agents selling the same good, the number of interested buyers, etc. The three agent strategies described previously ("anxious", "frugal/greedy", and "cool-headed") map directly to price adjustment curves. These curves, as well a description of the user-specified control parameters and how they effect the shape and layout of the curve, is described in Chapter 2.

Passive and Pro-Active Behaviors

Kasbah agents have two behavior modes: passive and pro-active. The passive behavior is what the agent does when other agents make initiatives towards it. The pro-active behavior is what steps the agent takes towards making the best possible deal. We believe it is important for agents to have both passive and pro-active behaviors. This is especially important in long-term marketplaces that include a multitude of agents whose internal behavior and strategies are not publicly known. If an agent has only a passive strategy, i.e. waits for other agents to contact it, and makes no forward attempts to find deals, there is the danger that all other agents in the marketplace will also only have a passive strategy. In this case, no deals will ever be made, because all the agents are waiting for someone else to make contact, which no one ever does. The agent's passive behavior is triggered when it is made an offer by another agent, and works

as follows:

- if the agent is a buying agent, and the contacting agent does not have a perfect or user-validated matching item description, then it will reject the offer.
- otherwise, if the offer is equal to or better than the agent's "current ask price", then it will accept the offer.

The agent's pro-active behavior actively attempts to find the agent X who is willing to pay the most for the good it is selling (if it is a selling agent), or who is willing to sell for the least (if it is a buying agent). The behavior works as follows.

- Based upon the current set of interested parties (potential buyers or sellers), decide which to contact and make an offer. Note that for buying agents, interested parties are those which the agent has done a positive item description match against, possibly confirming with the user that it should be negotiating with some of these agents. For selling agents, interested parties are those that the selling agent (through positive item description matches) believes are trying to buy what it is selling. However, selling agents do not confirm "soft" matches with the user. The agent's strategy is never to talk to a given party twice until it has first contacted all other parties. In this way, it is assured that it doesn't leave out anyone who might make a good deal. Basically, the agent "loops" through all the potential contacts. A single "loop" through the contacts consitutes a "round". The algorithm for deciding which party to contact whenever the pro-active behavior is triggered is this: Consider the contacts that have not been spoken to in the current round. If all have been spoken to, then restart the round. From this set of considered agents, pick one which has never been contacted, or, if all agents under consideration have been contacted, then pick the one whose last known asking or offering price is the best. The idea here 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 to pay a higher (or sell for a lower) price.
- Contact the selected agent (call this agent X). (Assume here that the initiating agent is a selling agent). The agent first asks X what it is currently willing to pay. If X's offer is greater

than the agent's own "current ask price", then make an offer to X at that price. Otherwise (X's offer is lower than the "current ask price"), then make an offer at the "current ask price". The point of the agent first asking X its own asking price is that X might be willing to pay more than what the agent is currently asking. After all, the agent's job is to find the best possible deal. If there's a customer in the marketplace who will pay more than the asking price, then the agent should grab that deal, at least if it wants to be considered competent by its owner.

The agent's pro-active behavior is triggered by the marketplace. Basically, the marketplace acts as a scheduler (described in detail below) which cycles through all of the agents and tells each to "do its thing". Note that in runnings its pro-active behavior is run , an agent will send a message to another agent, thus triggering that agent's passive behavior, i.e. how it responds to offers.

One thing which our agent behaviors assume is that all other agents are honest. That is, if agent Y asks agent Z what its current asking price is, and agent Z responds with, say, 50 dollars, then agent Y may assume that if its makes an offer to Z for 50 dollars, Z will accept it. This assumption of honesty is valid for our prototype Kasbah because we built all the agents that reside in it. Our agents have no provisions for dealing with dishonest agents. They will continue to function but not in what would be considered an "intelligent" manner. If an agent says that its asking price is 60, but rejects all offers at this price, then according to the pro-active behavior described above, our Kasbah agent will continue, futilely, to make offers at 60. What it should probably do is to raise its offer by 1 dollar and try again. Of course, designing agents with these kinds of contingency behaviors which are based on the perceived actions of other agents is much more difficult then assuming that all agents behave in straightforward manner. In a truly open Kasbah marketplace, where outside parties can supply new types of agents, this becomes an important issue. It is conceivable that some agents would be so much more "skilled" at bargaining that they would completely exploit the less developed agents such as our Kasbah agents. In fact, this kind of agent "arms-race", with different parties each trying to build the smartest agent that makes the best deals, is what we hope Kasbah will ultimately encourage. In this way, the state of the art of agent technologies will be pushed forward. Our goal with Kasbah

was not to develop the ultimate agent negotiating strategy (there is certainly a great deal of room for improvement with our current agents), but to demonstrate the viability of a new role for agents.

Marketplace

Like the agents, the marketplace is another Java object in the running server program. The marketplace's role is to house all agent and user information. It keeps track of:

- all the "active" agents: i.e. the agents which have not yet made deals
- all the agents for each user: the marketplace tracks all of their agents, including the ones which have already been terminated or already made a deal (this information is maintained so the user can view a "history" of the agents they have created)
- marketplace data: statistics on deals that have been made, current asking prices for the different types of goods, etc.

As mentioned earlier, agents also rely on the marketplace to provide a "matching" service. By providing this service, the marketplace frees the agents from the burden of having to do this additional work (and the developer of the agent from having to write more code). The "matching" method on the marketplace object may be called by an agent to find the list of all of the other agents in the marketplace which may be interested parties (potential buyers or sellers). This matching is done against the item descriptions of the agents according to the matching rules for the domain of goods that the marketplace supports. For the "musical" Kasbah, these matching rules were described above.

The Scheduler

The marketplace is responsible for "running" the agents. Conceptually, buying and selling agents in the marketplace are constantly talking to each other, going from agent to agent, all at the same time. Because we cannot really run the agents in parallel (the server process is a single thread), the marketplace simulates this by implementing a simple scheduling algorithm. Each agent is allowed exactly one "slice" of execution time per marketplace "cycle". A "cycle" is complete when all the active agents in the marketplace have been run once, and then it begins anew. This

ensures that no agent will have more opportunities to make a deal than any other agent. During its slice, the agent runs its pro-active behavior. In actuality, this is done by having the marketplace call the "run-pro-active-behavior" method which every agent must support. The illusion of the agents running continuously and in parallel is achieved by having their "run-pro-active-behavior" methods called frequently. When the marketplace makes this call, it passes execution control of the main thread to the agent. The agent can then send requests to the marketplace, to other agents, or whatever else its "run-pro-active-behavior" method implements. . For out agents, we limited the amount of work done in this method to contacting a single agent twice (one to ask the agent its current ask price, the other to make an offer). The order in which the agents execute per "cycle" is determined randomly. It is important to note that the "run-pro-active-behavior" method should terminate within a reasonable amount of time (no infinite loops allowed) for fairness. Since we built the agents in the prototype Kasbahs, this was easy to ensure, but in a more open system, where outside parties can submit their own agents, it will be necessary to take more explicit steps to ensure fairness (e.g., going to a multi-threaded model, or having a mechanism that limits how much CPU time an agent can consume during its "slice"). As the marketplace loops through all of the active agents and "runs" them, some will make a deal and remove themselves from the set of active agents. Other agents will expire, i.e. run past the user-specified date to sell by, and be removed.

Note that how the agents are run is independent of the marketplace knowing anything about their implementation. The agents are only required to support a certain set of methods that can be called on them, e.g. "run-pro-active-behavior". What the agents do within these methods is up to them. In addition to running the agents, the marketplace also services the incoming request from the Web servers. The list of requests supported was given above. It was a design criterion, particularly for one of the experiments described earlier, that the requests to the marketplace be processed as quickly as possible, so that the end-user sittting at a Web browser experience minimal delay. To achieve this goal, the marketplace interleaves running the agents with polling the socket to see if a request has arrived. If that is the case, then the marketplace

processes it immediately. This control flow gives priority to processing requests from the Web servers over runnnig agents. For instance, if there is a queue of five pending requests, these will all be processed before a single agent is run.

Front-end interface

The way the end-user accesses Kasbah is through a Web browser. The front-end component consists of a Web server and a set of HTML pages and / or CGI scripts. Using the Web as the interface to Kasbah was chosen for a couple of reasons:

- The Web is ubiqituous, i.e. everywhere. Any machine with a Web browser can access Kasbah
- Easy to build and change the interface.

We will focus discussion in the rest of this section on the "music"-based Kasbah interface, since this is the one which contains the richest set of functionality. All of the Web pages that made up the interface, with the exception of the login-screen, were generated dynamically by CGI scripts. The reason for this is that most pages contained information personalized to the particular user logged in, e.g. their name, the agents that they own, the current status of their agents, etc. This would have been impossible to do without generating the pages "on the fly". All of our CGI scripts were written in C, and consisted altogether of about 4000 lines of code. Whenever the user does something that requires an operation to be done in the backend markeplace server, e.g. create a new agent, the processing script opens a socket connection to the server, sends a request message in the appropriate format, waits for a response, and then closes the socket. The custom protocol we developed for communication between the scripts and marketplace server was string-based, simple and ad hoc. We recommend that future versions of Kasbah which need a more extensible and robust protocol use something like ASN.1, which supports transmission of large and arbitrary data structures, and for which there are parsers which convert form ASN.1 to C data structures, and vice versa. This would be very useful when transmitting complicated requests with many fields.

Creating and modifying agents

One of the critical things that the interface allows the end-user to do is to create and modify selling and buying agents. This is done through HTML forms, screenshots of which were shown in Chapter 2. Basically, the item description and control parameters are all specified through HTML form elements - check boxes, list boxes, etc. When the user has set these as desired, they click the "submit" button on the form. The script that is triggered then parses and collects all of the form values, bundles them into the appropriate request message, and sends it over to the backend server. Some validation of values is done by the processing script; however, the bulk is done by the backend. The backend will return to the script either a result code indicating success, or one which indicates an error occurred. There are several error codes so as to allow the script to determine what went wrong, and return to the user the appropriate error message.

Communicating with the end user

Another critical thing which happens through the interface is agent communication with the user. The different agent communications that we support are described in Chapter 2. The way that they work is as follows. The backend marketplace server supports a few more requests:

1. get_number_of_messages returns the total number of messages that a given user has pending.
2. get_number_of_messages_for_agent returns the number of messages that a given user has pending for a particular agent .
3. get_messages_for_agent returns the actual messages that a given user has pending for a particular agent.
4. response_to_message_for_agent is called by the script to actually send back the user's "reply" to a given message for a particular agent.

The scripts call get_number_of_messages so it can display, at the top of every screen, the number of pending messages that a user has. When the user clicks on the "messages" portion of the menu bar, they see a list of their agents that have pending messages, and the number for

each. This information is gathered by calling get_messages_for_agent. When the user then clicks on a particular agent, they are shown all of the pending messages for that agent. This information comes from calling get_messages_for_agent. Each message is uniquely identified within the scope of a given agent, and is displayed as an HTML form. When the user is ready to respond to that message (the minimum that a user must do is acknowledge having read the message so it will not be displayed any longer), they click the appropriate check boxes, if any, and hit the "submit" button. The processing CGI script parses the response fields, and then sends a response_to_message_for_agent request to the backend, specifying the response parameters (if any), the message ID (this information is stored as a hidden field in the form), and the agent ID (each agent has a unique ID, which is also stored in hidden form fields). Upon receipt of this request, the backend will notify the agent object so it can update its internal messaging data structures accordingly - the message is no longer pending and can be removed. When the script next makes a get_messages_for_agents request, the message responded to will no longer be present.

Backing up the System

As described above, the backend marketplace server is implemented as a single running Java process. This process contains essentially all of the information about the marketplace - the agents, the users, messages, etc., in in-memory data structures. This makes the information highly vulnerable. If the server process were to crash, all the data would be lost, with no means to recover it. Because of this, in our implementations of Kasbah we tried to have some means of backup and recovery. In the original Lisp implementation of the backend server, we leveraged Lisp's ability to dump the currently executing image to a file and then reload it. In the Java implementation, we didn't have this ability. For the one-day Kasbah experiment we performed, though, it was necessary to have some sort of back-up and recovery ability. So we turned to an ad hoc, less than ideal solution. What we did was, for each user, to constantly write out, to a flat

file, enough of the data about their agents such that they could be "rebuilt" if need be. This turned out to be very useful since during the actual experiment the system crashed a couple of times, but we were able to restore it to working order in such short time that no one noticed. Fundamentally, an ad hoc mechanism is not the answer. The Kasbah backend needs to be integrated with a persistent object store so that we can store complex data structures in a very robust and reliable manner. This is, in fact, what is currently being done with the successor system to Kasbah.

Chapter 5: Experimental Results

In this chapter, we discuss two experiments that were performed with prototype Kasbah systems that we built. The primary purpose of these experiments was to assess the overall viability of the Kasbah concept, and to gain specific feedback that could be incorporated into future versions of the system. Some of the issues that we wanted to assess through the experiments were the following:

- How would people react to the concept of Kasbah? Favorable, indifferent, or opposed.
- Would people feel comfortable delegating the task of buying or selling to an autonomous agent? Or would they want to retain control themselves?
- What is the level of control that people want over their agent? Do people want smarter, more autonomous agents, or do they want agents which allow for a greater level of user control.
- Were people satisfied with the job that their agents did? Or would they rather have done the task - find interested parties and negotiate with them – themselves?
- Evaluation of the agent architecture. Do the different agent strategies actually work, e.g. do anxious agent makes deals quicker than frugal agents? Do frugal agents make better deals than anxious agents?
- Quality of the marketplace that is formed. Is it viable? Does it exhibit reasonable characteristics?

We performed two experiments to help answer these questions. The first experiment used the initial Kasbah prototype built, and involved only a small handful of students. Its purpose was mainly to gauge people's reaction to such a system. The second experiment was a much bigger production, involving approximately two hundred participants intensively interacting with Kasbah over a single day. The purpose of this experiment was to see whether Kasbah could form, through the interaction of its many agents, a truly viable marketplace with classic marketplace behaviors (price convergence, relatively stable prices, etc.) We also wanted to see how non-technical users would respond to the concept of instructing to agent to perform the task of buying and selling on their behalf. It should be said up front that the ultimate experiment involving

Kasbah has yet to be done. This experiment is, of course, to place a fully functional Kasbah system on the Web where thousands of people could use it over an extended period of time (several months). Unfortunately it was not possible to perform this experiment within the timeframe of this thesis (it took several months just to build a working Kasbah), and there are still many issues to be addressed (such as those of performance, backup, and scale) before Kasbah could be used on a widespread basis. Currently such efforts are under way at the MIT Media Lab; a new system based on Kasbah, called Bazaar (due in early 1997), provides buying and selling services for a wide range of goods (books, music, Magic trading cards) to the MIT community. Despite the fact that the experiments involving Kasbah described here were not done on a grand scale, and that certain simplifying assumptions were made, we believe that the results were positive enough to support the soundness of the Kasbah concept, namely, the role of agents as buyers, sellers, and negotiators.

Initial experiment

This experiment was performed using the initial Kasbah prototype. The participants were approximately 10 students and 1 faculty from within the author's research group at the Media Lab. The central concept of the experiment was the following:All participants were given an initial allocation of playing cards (such as the kind used for poker or blackjack) and "fake" money. (The cards and money were not allocated physically, but virtually through a custom Web interface.) The goal for each participant was to try to increase the value of their "hand", i.e. the set of cards that they had. The value of a hand was determined according to standard poker rules, e.g. a flush (five consecutive cards) is worth more than two of a kind. To improve the value of their hands, participants bought and sold cards by creating Kasbah buying and selling agents. We did not allow participants to exchange their cards directly, because we wanted to simulate, as much as possible, a real marketplace, which meant using (fake) money as the medium for exchange. By giving participants a fixed amount of money initially, we created an incentive for them to both buy and sell. In order to get the cards needed to improve their hand, they needed to buy cards. But in order to have the money to buy these cards, they needed to sell the cards that don't add

value to their hand. For example, suppose my hand was "King of Spades", "King of Hearts", "three of diamonds", and "four of clovers". I might try to sell the "three of diamonds" and "four of clovers", and with the proceeds from these sales, try to buy a "King of Diamonds" and "King of Clovers", in order to complete the suite of Kings. In order for the participants to have a real-world incentive to improve their hand, they were told that the person with the best hand at the end of the experiment would be given a free gift certificate at the local coffee / ice cream shop. The experiment was run over a two-day period. The domain of playing cards was chosen for simplicity. We didn't want to have to deal with the complexities of item description matching; playing cards could be represented easily and unambiguously.

After the experiment was complete, the participants were all personally interviewed by the author. Throughout the experiment as well, there was frequent interaction with the participants. This enabled the author to gauge their real-time impressions as they interacted with Kasbah. On the whole, the reaction of the participants was positive. Most were entertained by the whole idea of the experiment, and there were some people who dedicated a substantial amount of time and energy towards trying to "win". These people monitored their agents very closely, and adjusted their control parameters frequently. They also engaged in various marketplace tactics to try to "outwit" the other participants. Nearly all the participants indicated that they would be willing to turn over at least a substantial portion of the task of buying and selling low-value items to a software agents. However, this response must be considered biased, given that all of the participants are themselves researchers in the field of autonomous agents. Participants also seemed to generally be satisfied with the level of control they had over their agents. They certainly understood the meaning of the control parameters, and what effect adjusting them would have on the agents' behavior. They also knew that the negotiating strategy of the agents was determinsstic, and they factored this knowledge into their pricing instructions. The fact that the marketplace of agents was very small - at most, there were around 30 agents active at any given time – also made it easy for participants to figure out ways in which to manipulate it. In this sense, we did not really have a truly viable marketplace, because one of the characteristics of a real-world market, such as the stock market, is that it is nearly impossible for any one person to

anticipate what other people in the market are going to do.

Were participants in the experiment satisfied with the job that their agents did for them? My assessment is that the answer leans towards the negative. Many participants were disappointed that the behavior of their agents was so simple and easy for them to understand. Again, one reason for this may be the high level of technical sophistication of the participants, but I think this dissatisfaction is more fundamental. People always want their agents to do more for them, to be smarter. That is only natural. However, when I asked participants what features should be added to the agents to make them better, most of their replies were along the lines of, "have them provide more information to me so I can make a better decision as to how to set my control parameters". Somewhat surprisingly, we did not get feedback that said "have the agents implement more complex behavior". One thing that became very clear to us from this experiment is that people do want to understand the behavior of their agents, at least to a certain level of detail. They do not want an agent whose underlying behavior is so complex and intricate that it cannot be understood, even if that agent does perform its task well. We believe this has directly to do with empowerment. People want to feel that they have control over their agent - for this to happen they must be able to understand what their agent is doing. This is the key to having user-agent "trust". We believe that this criterion, the importance of the user being able to understand and control the behavior of their agents in the desired manner, and thus trust their agent to do the right thing, will be the key factor limiting the complexity of Kasbah agents, more so than any underlying implementation difficulties. Given this, it makes sense that what participants asked was for the agent to provide more useful information, and it also suggests perhaps the correct role for Kasbah agents - not as independent and highly sophisticated deal-makers, but as rather simple assistants to the user, doing the dirty work of gathering the relevant market information and talking to the interested parties, while leaving control over the big decisions (what the asking price should be, how the price should be adjusted over time) to the end-user. Our thinking is that agents should, in the intial versions of Kasbah, be very simple and not too autonomous. However, as users start trusting these agents more as they gain experience

with them, perhaps future generations of agents could become more complex and autonomous, and the user would feel comfortable delegating more "big decisions" to them.

Some of the specific features that participants in the experiment asked for were:

- better control over the agent's pricing curve: the three choices of agent strategy were too limiting for some users.
- Some more common sense smarts: in particular, some users made the observation that the agents would make a deal for a price that met their asking price criteria, despite the fact that there were better deals available!! This shortcoming in the agents was in fact addressed in later versions of Kasbah; it was due to an oversight on the part of the author in writing the agent negotiation code.
- More marketplace information: some users felt that they weren't provided with enough information to make intelligent pricing decisions. For instance, when the user creates a selling agent, and they specify the control parameters "desired price" and "lowest acceptable price", what criteria should they base this decision on? Unless the user has a good sense of the what the going market rates are for the item being sold, then this is tantamount to a guess. Once the agent has been created and begins sending back status reports to the user, these reports will contain market data that helps the user to intelligently adjust the control parameters on the agent (status reports were not implemented for this version of Kasbah). But the user should not have to wait until they created the agent in order to get the market data needed to set the initial pricing instructions. This notion, that the user should have essentially continuous access to the current market conditions (to help them make pricing decisions), was an integral part of the version of Kasbah used in the experiment which we will describe next.

There were many limitations with this experiment in terms of assesing the feasibility of Kasbah in the real world. However, this was not the only goal of this experiment, as has been described - it was more a proof-of-concept. In this sense, we believe it succeeded. The Kasbah concept was demonstrated to work, people successfully bought and sold goods (albeit virtual ones) using

buying and selling agents, without any major problems or confusion. Some of the problems with this experiment:

- unrealistic domain: again, we chose the playing cards and fake money scenario for implementation simplicitly. The problem with treating the experiment as a "game" between the participants is that this is not how people treat buying and selling in the real world. There, with people's material well-being on the line, it is certainly not a game. Also, some participants in the experiment (Brad) spent an inordinate amount of time trying to win the competition, thus almost defeating the purpose of having an agent - to do the most of the hard work so you can worry about other things. In the real world, people using Kasbah are much less likely to keep such a constant watch over their agents and the marketplace.
- Biased participants: as mentioned earlier, the participants all have technical backgrounds and are familiar with agents. This makes them more likely to be supportive of the concept of agents in general, but might also lead them to have higher expectations of what the Kasbah agents should be capable of. On the other hand, they might have lower expectations of what agents can do, compared to normal end-users, because they know about the difficulties of building truly smart agents. Regardless, it is clear that Kasbah needs to be tested by real people (not academics). This is what we did in the next experiment.

Experiment in creating an Agent Marketplace

We now describe another, much larger experiment that we did using Kasbah. It is described in detail in [Chavez97a]. For this experiment the Kasbah back-end server developed by the author was used - with slight modifications - and the front-end Web interface was written by another team of grad students (Daniel Dreilinger, Rob Guttman, and Kathi Blocher).

Experimental Setup

On October 30th, 1996, the Media Lab held a "Digital Life" symposium for approximately 200 guests from industry. The majority of these people were not academics, and could be considered to reflect, in terms of technical proficiency, the vast majority of end-users. None of them had ever

heard about Kasbah before. Each guest was given three physical goods as well as some fake money. We chose to use fake money because of the possibility that something might go wrong (the system might crash) and people would be upset. However, there is no a priori reason why we could not have used real money or digital cash/credit. Doing so would not have fundamentally altered the experiment. Guests were told that they could create selling agents to sell the goods which they owned but did not want, and buying agents to buy the things which they wanted to own. Creating an agent was done essentially as described in Chapter 2. The user specifies the good they want to buy or sell, the desired price (i.e. the initial asking price or offer), the highest/lowest acceptable price (i.e. the final asking price or offer), and the "strategy" that the agent should use to raise or lower its price over time. Two things to note here:

1. There were a limited set of things to buy and sell. This meant that we again avoided the fuzzy item description matching problem.
2. Uusers did not specify a date to sell/buy by - this was hardcoded to 5 pm the day of the experiment.

Guests were able to create selling and buying agents all day long. At the end of the day, they got to keep whatever goods they had accumulated throughout the day. They also had the option of exchanging whatever money they had left over for bottles of wine.

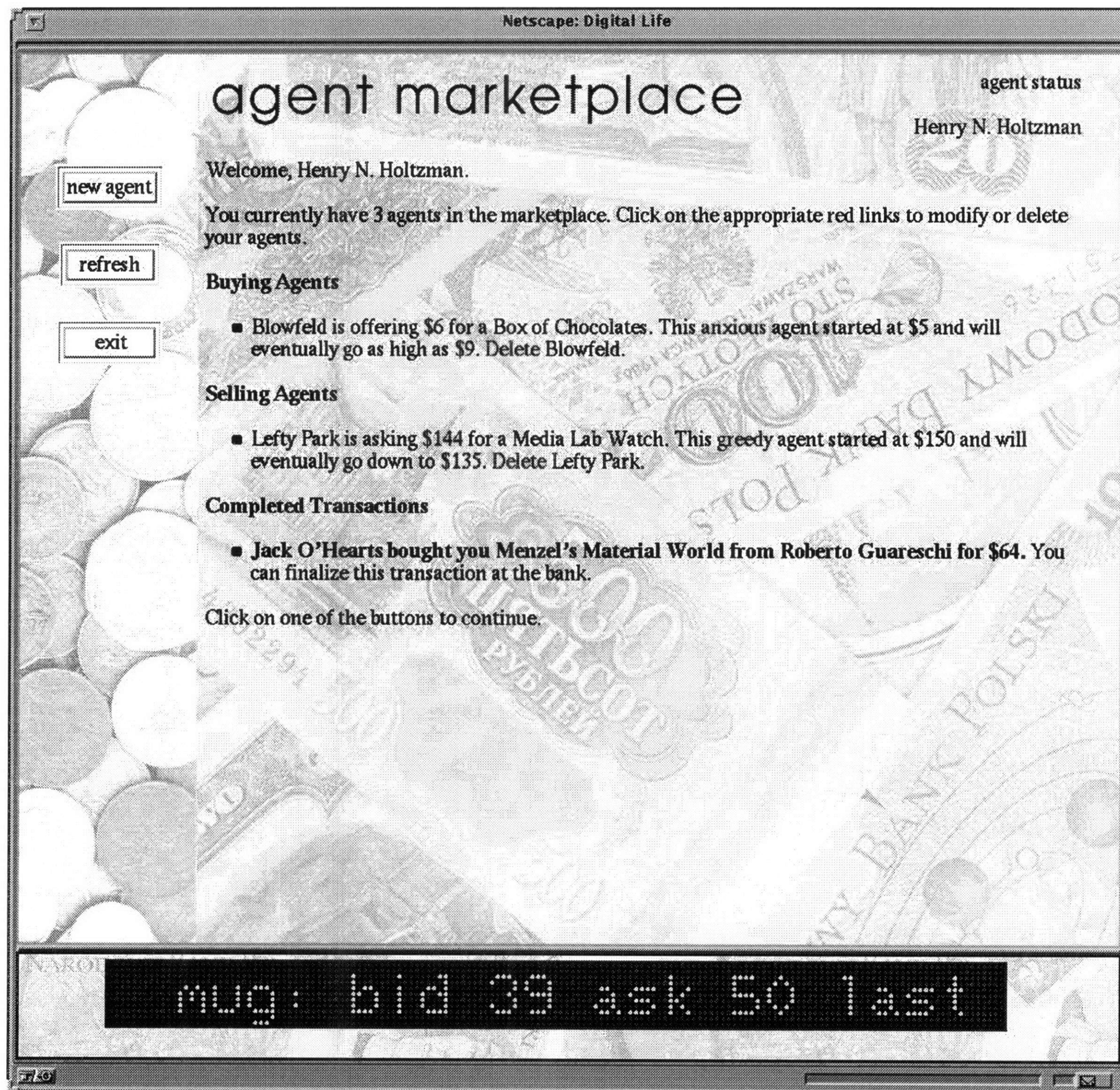
Participants in the experiment were immersed in a rich, interactive environment that involved several pieces:

- interactive kiosks: there were 20 of these. Figure 12 below shows some users interacting with the kiosks. Each consisted of a keyboardless workstation, monitor, mouse, and bar-code reader. Participants would go up to a kiosks and initiate a session by scanning their name tag badge with the bar-code reader. The system recognized the identity of each participant and personalized the session accordingly. This is basically a fancier way of logging into the Kasbah web site as described in Chapter 2. Once logged in, the user could do several things: see a list of active agents (shown in Figure 13 below), see a list of completed transactions, change the pricing instructions for existing agents, or create new

agents. The interface was Web based and entirely driven through pointing-and-clicking with the mouse. The interface was much more streamlined than the standard Kasbah interface shown in Chapter 2, as it was designed to facilitate short interactions, so that all 200 participants could use the 20 kiosks. We observed that the average user session involved creating around three to four agents and took about five minutes.

Figure 12: participants interacting with the kiosks

Figure 13: kiosk login screen



- pager notification: each participant in the experiment was given a personal pager that was used to notify them whenever one of their agents had made a deal. For example, if Joe created an agent named James Bond to sell a lunch pail, and this agent made a deal with one of Pattie's agents, Joe received the message: "Joe, I sold your Media Lab lunch pail to Pattie for $53 - James Bond". Pattie would be paged with a similar message from her agent. The pagers were also used for sending out status reports from agents. These status reports were different than those described in Chapter 2 in that they were event-driven rather than

sent out a regular basis. A status report would be sent from an agent to its owner if it seemed unlikely that the agent would make deal by the end of the day with the current pricing instructions given by the user. For example, if a user created an agent to buy a box of chocolates, and the maximum price the agent would offer was 20 percent lower than the average transaction price for chocolates throughout the day, that agent would page the user with the message: "Tip: you may want to raise my maximum offer if you want to buy those chocolates - James Bond."

- Marketplace data: participants had visual access to a variety of marketplace information which was useful when setting pricing parameters on agents (this is the same information contained in the normal Kasbah status reports described in Chapter 2). A 10 foot high projection display and two large monitors provided visualizations of marketplace statistics, including histograms of the current asking prices of the buying and selling agents for each of oods (there were a total of nine goods used in the experiment), as well as a time series projection showing the trend in transaction prices for each good. Figure 14 below shows a screenshot taken from the time series display. These displays were updated every several minutes.

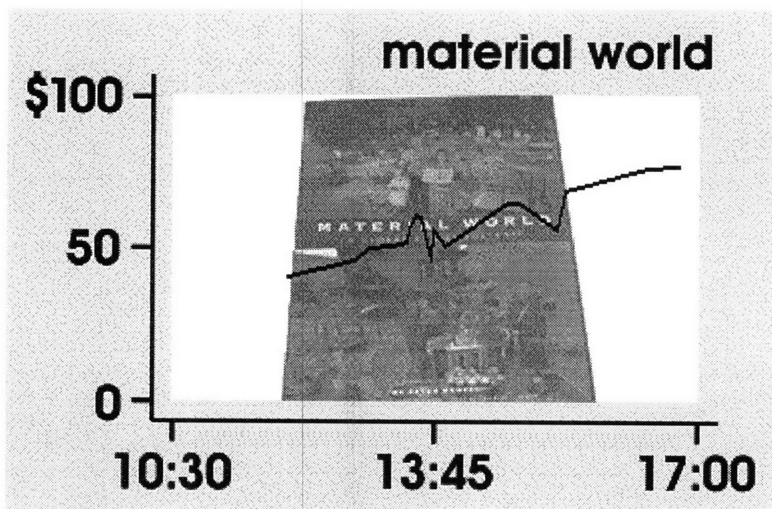


Figure 14: sample time series display

- In addition, there was a scrollling LED ticker tape that displayed up-to-the-minute market information about each of the items being traded, such as the highest bid, lowest asking price, and last transaction amount.
- transaction center: Upon notification of a transaction (either via the pager or through the Web interface), users were told to report to the staffed transaction center to exchange the good for the price that their agents agreed upon. In our experiment, users were bound to the deals that their agents made. The purpose of the transaction center, other than where participants could meet to consummate the transaction, was to handle situations where one of the users showed up but the other failed to for some reason. We didn't want to make participants wait, so in these situations, the transaction center would complete the deal one-sided. When the other participant eventually arrived, the transaction center would carry out the other side of the deal. To support these "asynchronous" transactions, the transaction center had a supply of extra money and goods. The transaction center also served as a "help desk" where participants could go to ask questions, exchange money for wine at the end of the day, make change, etc.

Results from Agent-Marketplace Experiment

First we will discuss quantitative results, then we will provide some qualitative analysis. Altogether there were 171 participants. They created a total of 625 agents during the day, of which 510 agents made deals. This is shown in Figure 15 below. Note that there were nine different types of goods which the agents transacted. Some goods were transacted more than others, as there were different quantities of each type of good, and some goods were more "popular" than others.

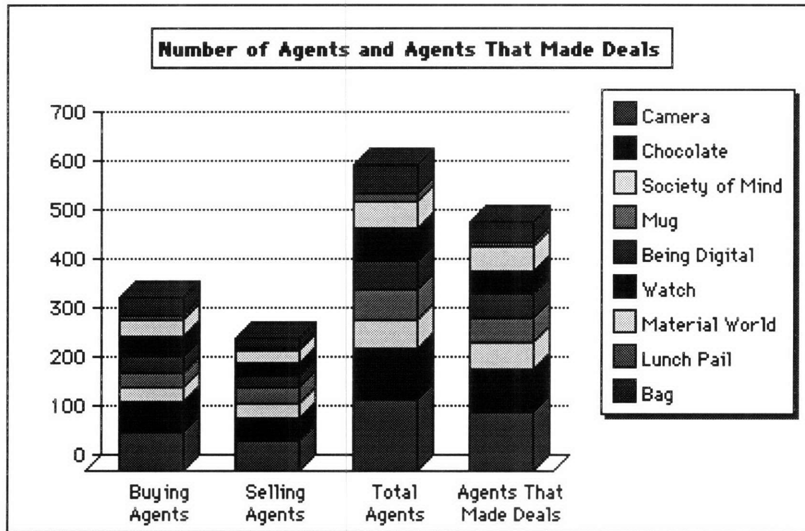


Figure 15: number of agents

Figure 16 shows, as a function of time throughout the day of the experiment, the total number of agents, the number of active agents, and the number of deals that were made. One can see that the number of active agents peaked a little before the middle of the day, and then tailed off towards the end, as the active agents began making deals.

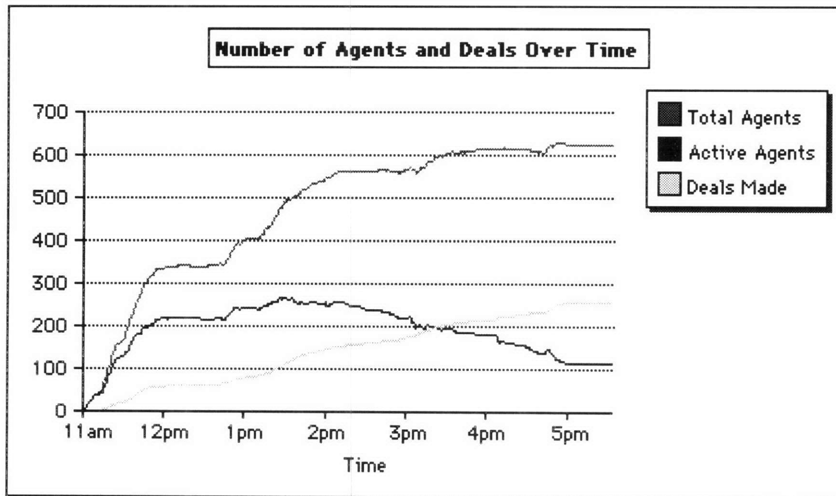


Figure 16: number of agents over time

Figure 17 shows, for each good, the number of agents (buying and selling) for each of the three possible strategies: "frugal/greedy", "anxious", or "cool-headed". For all of the goods, the "cool-headed" strategy was the least popular, while the "frugal/greedy" and "anxious" strategies were

the most common. The strategy defines, along with the other control parameters, the agent's asking price as a function of time, and thus helps determine the likelihood of the agent making a deal. It is interesting that the "cool-headed" strategy was the least popular. Perhaps the sense of urgency in trying to purchase something caused buyers to use the "anxious" strategy, while the desire to earn lots of money from a sale caused sellers to use the "greedy" strategy. But this is just an educated guess. It could conceivably be the other way around too. This kind of analysis was difficult to do without interviewing all 200 participants or having them fill out questionnaires, which we did not do.

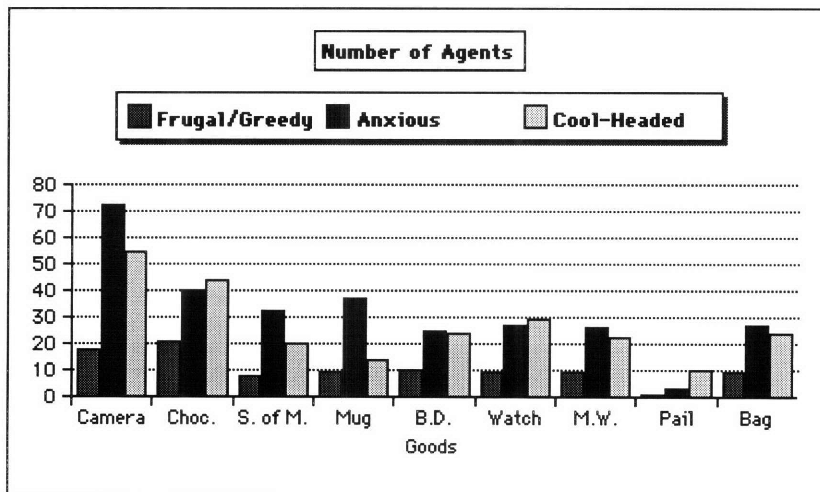


Figure 17: number of agents with each strategy

Figure 18 shows, for each good, the percentage of buying agents that made a deal for each of the three agent strategies. One might expect that "anxious" agents would be more successful in making deals (since they lower their price faster than other agents), and that "frugal" agents would be least successful, but as one can see from the Figure, this is not the case. In fact, for most goods, frugal buying agents actually have a higher success rate of making deals than anxious agents. Upon consideration, though, this result may not be so surprising. Participants were able to change their agents' asking prices, which they did frequently throughout the day. Given two buying agents looking for the same good, one "anxious" and one "frugal", a user could set the pricing parameters of the agents such that the "frugal" one will make a deal but the "anxious" one doesn't - the "anxious" agent could have a much higher initial asking price than the

"frugal" agent. An agent's strategy does not a priori determine the likelihood of an agent making a deal versus any other agent. What matters more is the agent's initial and final asking prices. Of course, given two agents with the same initial and final asking price, the anxious one will always make a deal more quickly than the frugal one. It may in fact make sense that "frugal" agents had a higher success rate in making deals than "anxious" agents. Owners, getting frustrated with the slowness of their frugal agents in making deals - and watching their asking prices drop very slowly (see the pricing curve diagrams) – might drop their asking prices to such an extent that they would make deals.

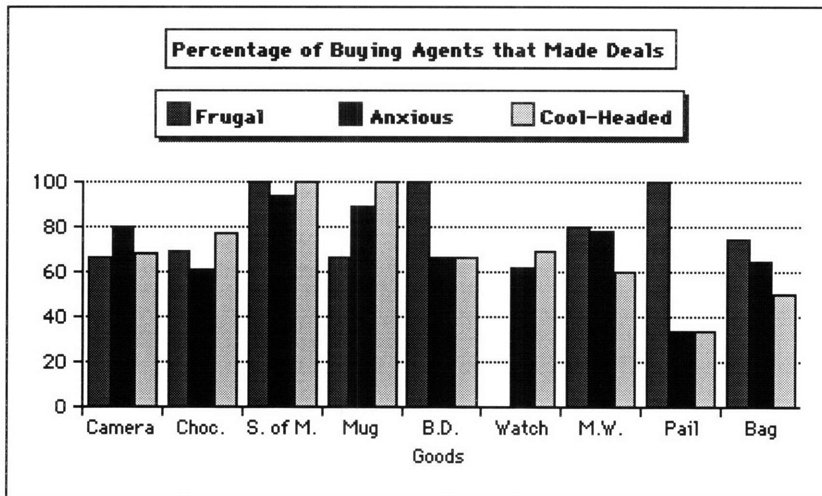


Figure 18: success rate for different strategies

Figure 19 shows, for each good, the average purchase price for buying agents for the three different strategies. One would expect that the frugal agents would have a higher average purchase price than anxious agents. This is in fact what happened. For all of the goods, the anxious agents which made deals did so at a lower average price than frugal agents which made deals. The disparity in this average was not very large for most goods, but it was definitely present. This observation confirms that our notion of agent strategies is a valid one. Frugal agents are more likely to get a better deal than anxious agents. Unfortunately, as discussed above, we could not observe a symmetric tradeoff between getting a better price versus the likelihood of making a deal, as anxious agents did not in general have a higher success rate in making deals than frugal agents.

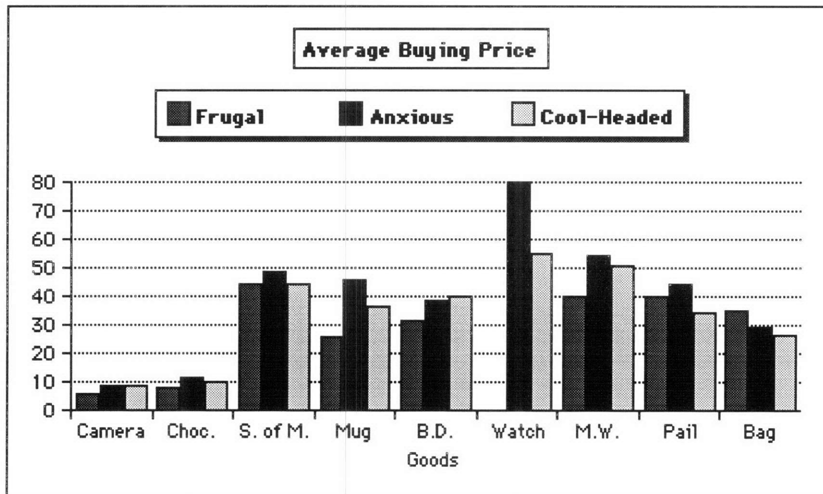


Figure 19: average buying price for each strategy

Figure 20 shows the average buying price (the average price being asked by buying agents) for one the goods in the experiment, a Media Lab watch, over the course of the day. One the whole, the average buying price increased throughout the day. It especially ramped up towards the end of the day. This also happened for the other goods. Our analysis is that this happened because participants knew that they could only purchase bottles of wine with the leftover money that they had at the end of the experiment. Since the fake money had no value in the real world, and many participants did not want wine, they were willing to spend all their money on whatever goods they could buy. Figure 20 also shows one significant exception to the general trend of a rising average buying price. This anomaly was the result of external market forces. One of the participants spread a rumor saying that at the end of the day, every participant would receive a free Media Lab watch. This caused the price of watches to drop precipitously, as many participants went to lower the asking price of their agents buying watches. When the rumor was dispelled, the price of watches quickly went back to their previous level.

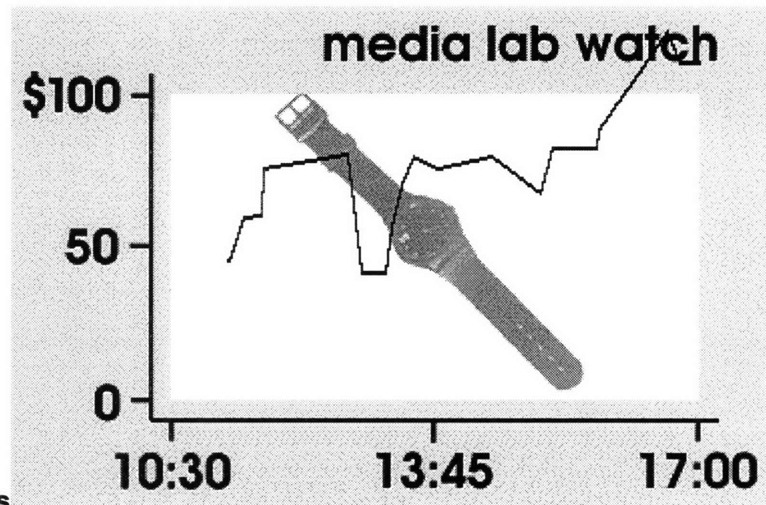


Figure 20: watch time series

Qualitative analysis

While we did not conduct a formal survey of the participants, we did acquire feedback through informal observations and conversations with participants. On the whole, the general impression of the system was very favorable. Participants seemed quite intrigued, some to the point that they skipped sessions which were part of the symposium, so that they could be at the kiosks to track their agents closely. One question of concern going into the experiment was whether black-market trading would occur, and if so, would it distract from the experiment. Would participants just exchange goods between each other, and ignore Kasbah altogether? We intentionally gave no instructions to partcipants prohibiting this kind of trading, to see if it would happen. Black-market trades were observed occuring throughout the day; however, the bulk of the participants used Kasbah for their transactions. This fits with our notion of Kasbah as complementing, rather than competing, with other forms of conducting transactions. A goal of Kasbah is to help reduce transaction costs by having agents do a lot of the work, thus allowing new markets to be formed that were previously prevented by these costs.

Another concern we had was that participants would spend more money than they had. This gets into the whole issue, discussed in Chapter 2, of the extent to which people are legally or morally "bound" to the actions of their agent. If your agent makes a deal to sell a good to someone else,

and that person does not have the means to pay the agreed-upon price, what recourse do you have? In our experiment, because we were using play money, we had the transaction center (the "bank") buy the good from the seller so they would not be upset. In the real world, though, with real money involved, the seller would have no recourse. We >>>other than blacklisting that person for the future did observe some participants trying to buy more than they had money to spend. There were several reasons for this. Some simply did not pay attention to how much money they had and set their agents' offering prices to levels which they could not cover. Others planned on using the proceeds from selling some of their goods to cover their purchases. When these anticipated sales did not occur, they were left short of cash. Still other participants assumed that they would be able to ask for more money from the bank.

Although the experiment affirmed the viability of Kasbah, it did not, like the first experiment definitively confirm that Kasbah will be able to succeed in a real environment. There were also many differences between the Kasbah system used in the Digital Life experiment, and the fully functional one for transacting musical goods described in Chapter 2. The Kasbah used in this experiment did not have to deal with many complex and fundamental issues, such as describing a wide variety of goods, doing "soft" matching, and asking for confirmation feedback from the user. The ability to address these difficult and subtle issues adequately will, in the author's opinion, be key to making Kasbah a truly valuable real-world agent system. The experiments described in this chapter give justification to the hope that Kasbah and systems like it will one day achieve this goal.

Chapter 6: Future Work and Conclusion

In this chapter we discuss future work and research directions for Kasbah, and then conclude.

Future work

As has been noted throughout this thesis, there is much work to be done before Kasbah fulfills its vision of a public Web site where thousands of people go to buy and sell goods of all kinds. We now describe several possible areas of future work for Kasbah. Some of this is already happening with the successor system to Kasbah, called Bazaar.

Item description problem

As we observed, the general problem of having the user describe a wide variety of goods to a software agent is a very difficult one. In this work, we dealt with this problem by limiting the domain of goods which our Kasbah prototypes could transact. For the music-based Kasbah, we developed a structured representation format for musical recordings. For our other prototypes, described in the previous chapter, we avoided the problem altogether by constraining the user to choose goods from a fixed set. Ultimately though this problem must be dealt with in a more general manner. The general problem is this: the most flexible description language for the user is free-form text (which is why newspaper classifieds use it for ads); however, computers' natural language ability is still too primitive to handle this. So, to allow the agents to compare item descriptions, we must have a more structured way, i.e. filling in text fields, of describing goods. But the problem with structured representations is that they are inflexible. For example, if the developer of the structured format provides no way to describe some facet of a good, then the end-user cannot capture that in their item description. This is not much of a problem for simple goods, like CDs, but is one for more complex good, such as furniture, apartments, or cars. The solution we propose is an extensible structured way of describing goods. Basically, various goods would have a default structured representation format that we would devise. However, this

representation can be extended by end-users to capture features which our default format does not express. For example, if a end-user wanted to capture the fact that the car they are selling has a certain style of hubcaps (something which the default format for describing cars does not have), they could add a field called "hubcap type" and insert the desired value - say, "gold-rimmed". There are still some issues with this approach. The big one is that the item description formats for the various goods could become very cluttered with lots of user-added fields, some redundant, each being interpreted slightly differently by very user. One answer is to have a Kasbah administrator go through the item description formats periodically to "clean" them up. This isn't the most elegant solution, but we feel it is better to give the end-user the power to extend the good formats, and then deal with the consequences of that, rather than to fundamentally limit the flexibility of the system.

An alternative approach is to have all goods be described in free text. The agent would do soft matching, i.e. keyword based, and always ask the user whether it should be negotiating with another agent for which there was a soft match. The advantage of this approach is that we do not have to develop an extensible description language and deal with all of its attendant problems. The disadvantage is that the user will have to do more work.

Improved architecture

We attempted to make the architecture of the Kasbah system flexible, i.e. the ability to change the front-end interface without requiring changes to the back-end engine. While this flexibility was useful, it became clear to us in the course of our Digital Life experiment that a new architecture was needed for the back-end. Namely, the Kasbah marketplace server itself that contains all of the agents is too centralized and monolithic. The load on this server, with thousands of agents and users, quickly becomes too much for even the most powerful workstation to handle. Fundamentally our single-process backend engine design does not scale well. It needs to be made more distributed. Agents, or groups of agents, should be able to run on physically separate servers and communicate with each other, not by direct method calls like today, but over some network protocol, e.g. TCP/IP sockets. With such an architecture, a user's agents could run

locally on their machine in the background, much as Yenta agents do. In this architecture, there would still need to be a central index of all active agents so that they know where to find each other. This architecture would also allow for new types of agents to be more easily added to the system, because it would not be necessary to dynamically incorporate their Java class file into the already running marketplace process. All that the agent program would have to do is be able to "talk" the Kasbah "language" over the network. This notion of agents as independent processes, possibly located on separate machines, and communicating over the wire, is similar to the Telescript view of agents [Telescript] [White96], and is closely related to generic distributed architectures such as the industry-standard CORBA or Microsoft's DCOM.

New types of agents

One interesting area of work would be to develop new, more sophisticated types of Kasbah buying and selling agents. In particular, agents that are more responsive to changing market conditions, such as the number of buyers, sellers, average offering price, average asking price, etc. A big challenge would be to give these agents advanced, complex behaviors, while at the same time allowing the user to understand what their agents are doing. As we pointed out, it is critical that users be able to understand, at least a high level, the behavior of their agents if they are to "trust" them and be willing to grant them permission to act on their behalf.

Collaborative agents

Along the lines of developing new types of Kasbah agents, one fascinating area to explore would be agents that collaborate, by sharing the same goal, or working together to achieve common goals. One use of such agents would be when the user wants to make the actions of one agent contingent upon the actions of another agent. For example, the user might want to have agent X buy good A, but only if agent Y first is able to sell good B (perhaps because the proceeds from the sale of good B are going to be used to pay for the purchase of good A). Another example would be agents from different users that collaborate to increase their purchasing power, i.e. form a cartel of buyers or sellers to change market conditions in their collective favor. This opens up

all sorts of interesting possibilities and social implications: would the Kasbah marketplace have the same kind of "problems" - monopolies, oligopolies, cartels, etc. - that affect (usually perceived toin a negative fashion) real-world markets. What means would be used to deal with these problems, if any? Would there be Kasbah "regulator" agents whose job it is to police the market to prevent "illegal" activities from occuring, and ensure that the marketplace does not evolve unfavorable to the "common good"? Having collaborative Kasbah agents would require that their communication language be expanded to allow for a much richer semantics. For more extensive interoperability, it might be useful to leverage existing standards for agents communication, e.g. KQML [Labrou94] [Finin93].

Social consequences

As stated in the functional overview, our current approach for dealing with marketplace fraud, e.g., buyers who back out of deals, or don't have the good they promised to sell, is laissez-faire. We suggested that a blacklisting capability be added to the system which allows users to explicitly specify those users that their agents should not deal with. This functionality should be built and tested through real-world usage. Is it adequate to deal with the problem of fraud? Is fraud a serious problem in real use, better or worse than what the newspaper classifieds experience? Only extensive use of Kasbah system by the public will be able to answer this. Might the "better business bureau" approach be more desirable? This is but one of the many social issues that the existence of Kasbah-like systems raise? If they were to become widespread, what would the legal implications be? How responsible are people for what their agents do. How responsible are the developers of these agents for the actions that they take? The issue of intelligent agents is a touchy; there are those who are strongly philosphically opposed to them on various grounds. What will be the societal ramifications if Kasbah sites proliferate? What would be the role of brokers and middle-men in this world. Would Kasbah agents automate them out of existence? Or would there still be a place for the human touch. We suspect the latter, but time will tell.

End-user testing

Central to all of the aforementioned areas for future work is the need for much more extensive end-user testing. This is in fact under way at the Media Lab. A successor to Kasbah, called Bazaar, will allow members of the MIT community to buy and sell a variety of goods (music, books, and Magic trading cards). The plan is to have the system be available to thousands of end-users over an extended period of time. Results from this experiment should shed much new light on all of the issues we have mentioned.

Conclusion

For this thesis, we developed Kasbah, a Web-based system in which software agents are used in a novel role: negotiating to buy and sell goods on behalf of users. We built several prototype Kasbah systems and in the process addressd a number of issues. The most important of these were how the user describes the item of interest to the agent, how matching is done, the negotiating strategy of the agent, and how the user instructs their agent. We performed two experiments with our Kasbah prototypes. The initial one assessed the feasibility of Kasbah as a general concept, and gathered user feedback and impressions as to what functionality the agents should have. The other experiment involved a much larger number of participants intensively interacting with Kasbah over one day, and proved Kasbah capable of forming a viable marketplace for transacting goods.

Much work remains to be done before Kasbah and systems like it can provide a truly useful service to the end-user masses, by enabling them to buy and sell goods which they would have otherwise not had the time and energy to. We hope that this thesis has made a first, significant step towards realizing this goal.

References

[AdHound] AdOne web site. http://www.adone.com

[BF] BargainFinder web site. http://bf.cstar.ac.com/bf/

[Bradshaw97] Bradshaw, J. Software Agents. MIT Press, Cambridge, MA, Upcoming March 1997.

[Chavez97] Chavez, A., Moukas, A., and Maes, P. Challenger: A Multi-agent System for Distributed Resource Allocation. Proceedings of the First International Conference on Autonomous Agents, Marina del Ray, California, February 1997.

[Chavez97a] Chavez, A., Dreilinger, D., Guttman, R., and Maes, P. A Real-life Experiment in Creating an Agent Marketplace. To appear in Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, April 1997.

[Clearwater96] Clearwater, S. Market-Based Control: A Paradigm for Distributed Resource Allocation, World Scientific Publishing, Singapore, 1996.

[Cypher91] Cypher, A. EAGER: Programming Repetitive Tasks by Example. Proceedings of the CHI 91 Conference, ACM Press, 1991.

[Dent92] Dent, L., Boticario, J., McDermott, J., Mitchell, T., and Zabowski, D. A Personal Learning Apprentice. Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, California, pp. 96-103, 1992: AAAI Press.

[Fido] FIDO: the Shopping Doggie! Web site. http://www.shopfido.com/

[Finin93] Finin, T., Weber, J., Widerhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shaprio, S., and Beck, C. Specification of the KQML agent-communication language. Technical Report EIT TR92-04, Enterprise Integration Technologies, Palo Alto, California, 1993.

[Firefly] Firefly web site. http://www.firefly.com

[Foner95] Foner, L. What's an Agent, Anyway? A Sociological Case Study. Unpublished. 1995.

[Foner96] Foner, L. A Multi-Agent Referral System for Matchmaking. Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, pp. 245-261, April 1996.

[Geddis95]. Geddis, D., Genesereth, M., Keller, A. and Singh, N. Infomaster: A Virtual Information System. Proceedings of CIKM 95 Workshop on Intelligent Information Agents, Baltimore, Maryland, 1995.

[Infomaster] Infomaster web site. http://infomaster.stanford.edu

[Kozierok93] Kozierok, R., and Maes, P. A learning interface agent for scheduling meetings. Proceedings of the ACM SIGCHI International Workshop on Intelligent User Interfaces, Orlando, Florida, pp. 81-88, 1993: ACM Press.

[Kuokka95] Kuokka, D., and Harada, L. Matchmaking for Information Agents. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 95, 1995.

[Lashkari94] Lashkari, Y., Metral, M., and Maes, P. Collaborative Interface Agents. Proceedings of the AAAI 94 Conference, Seattle, Washington, August 1994.

[Labrou94] Labrou, Y., and Finin, T. A semantics approach for KQML – a general purpose communication language for software agents. Proceedings of CIKM 94, New York, 1994: ACM Press.

[Maes93] Maes, P., and Kozierok, R. Learning interface agents. Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington, DC, pp. 459-465, 1993: AAAI Press.

[Maes94] Maes, P. Agents that Reduce Work and Information Overload. Communications of the ACM, Vol. 37, No. 7, pp. 31-40, July 1994.

[Maes95] Maes, P. Intelligent Software. Scientific American, Vol. 273, No. 3, pp. 84-86, September 1995.

[Malone88] Malone, T., Fikes, R., Grant, K., and Howard, M. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments. In: The Ecology of Computation, Ed. Huberman, B.A. Elservier, Holland, 1988.

[Resnick94] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., Riedl, J. GroupLens: An Open Architecture for the Collaborative Filtering of Netnews. Proceedings of the CSCW 1994 conference, October 1994.

[Rosenschein94] Rosenschein, J., and Zlotkin, G. Rules of encounter: designing coventions for automated negotiation among computers. Cambridge, MA, 1994: MIT Press.

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

[Shardanand94] Shardanand, U., and Maes, P. Ringo: A social information filtering system for recommending music. Internal Report, MIT Media Laboratory, May 1994.

[Shardanand95] Shardanand, U., and Maes, P. Social Information Filtering: Algorithms for Automating Word of Mouth. Proceedings of CHI 95 Conference, Denver, Colorado, 1995.

[Telescript] Telescript Technology: The Foundation for the Electronic Marketplace. http://www.genmagic.com/Telescript/Whitepapers/wp1/whitepaper-1.html.

[White96] White, J. Telescript Technology: Mobile Agents. General Magic White Paper. 1996.

[Winograd86] Winograd, T., and Flores, F. Understanding computers and cognition: A new foundation for design. Addison Wesley, Reading, MA, 1986.

[Zlotkin93] Zlotkin, G., and Rosenschein, J. A domain theory for task oriented negotiations. Proceedings of the Thirteenth Join Conference on Artificial Intelligence, Chambery, France, pp. 416-422, 1993: Morgan Kaufmann.