Market Maker: an Agent-Mediated Marketplace Infrastructure

By

David Yi Wang

B.S. Electrical Engineering and Computer Science (1995)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 1999 in Partial Fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

at the
Massachusetts Institute of Technology

June 1999

© Copyright 1999 David Yi Wang. 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

May 26, 1999

Certified by __________________________ Patricia Maes
Professor of Media, Arts, and Sciences
Thesis Supervisor

Accepted by ____________________________ Arthur C. Smith
Chairman, Department Committee on Graduate Theses

WITHDRAWN FROM
MITLIBRARIES
Market Maker: an Agent-Mediated Marketplace Infrastructure

By

David Yi Wang

B.S. Electrical Engineering and Computer Science (1995)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 1999 in Partial Fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Market Maker is a tool for creating agent based electronic marketplaces. Market maker supports automated product identification, buyer and seller qualification, and negotiations by software agents to facilitate transactions. The concept of agent based marketplace was first demonstrated through Kasbah, an electronic marketplace developed in the Media Lab’s Software Agents group in 1996. Market Maker extends the Kasbah concepts by providing a framework for creating marketplaces on demand.

Thesis Supervisor: Patricia Maes
Title: Professor of Media, Arts, and Sciences
Table of Content

1 Introduction ........................................................................................................................................... 5
2 Background........................................................................................................................................... 7

3 Market Maker Overview ................................................................................................................... 9
   3.1 Dynamic Marketplace Creation ......................................................................................................... 9
   3.2 Full HTML Access .................................................................................................................................. 10
   3.3 Modularity ......................................................................................................................................... 11
   3.4 Data persistence ..................................................................................................................................... 11
   3.5 Portability ........................................................................................................................................... 12

4 Components Analysis ........................................................................................................................... 13
   4.1 User Interface Module ......................................................................................................................... 14
   4.2 Client Handler ....................................................................................................................................... 20
   4.3 Buffers ............................................................................................................................................... 22
   4.4 Software Agents .................................................................................................................................... 23
   4.5 Market Host and Agent Cluster ........................................................................................................... 27
   4.6 Data Persistence Interface and Database .......................................................................................... 30

5 Conclusions and Future Work .............................................................................................................. 32

6 References ........................................................................................................................................... 34

7 Appendix ........................................................................................................................................... 35
Acknowledgements

Market Maker is part of the Agent Mediated Electronic Commerce initiative at the Media Lab’s Software Agents group. Market Maker is a continuation of the electronic marketplace experiment, started as Kasbah in 1996. The concept of agent based transactions, and the HTML user interface design are based on the Kasbah project. The author would like to thank the Kasbah design and development team, for demonstrating and proving the concept of agent based transaction marketplace.

The author would like to thank Professor Patricia Maes for her guidance, and fellow graduate students Giorgos Zacharia, Alexandros Moukas, and Robert Guttman for their expertise and assistance. Without them this project would not have been possible.

A number of M.I.T. undergraduate students participated in the project development: James Chen, Jim Matysczak, Terrence Poon, Christopher Roy, Scott Schneider, and Kenneth Walker. The author would like to thank these students for their valuable contributions.
1 Introduction

In 1996 the Software Agents group developed an agent based consumer-to-consumer marketplace named Kasbah [1]. MIT students are able to buy and sell used items, such as books and music CDs, on the marketplace. Prior to Kasbah, students used newsgroups, mailing lists, and bulletin board postings, in advertising items both wanted and for sale. Prospective buyers and sellers contact each other through email or phone, to find out more details about the item in questions. Parties sometime engage in informal bargaining to determine the final transaction price.

Kasbah provides a centralized electronic marketplace accessible to all through web browsers. Registered users are able to browse items posted, and to obtain various marketplace information. A seller may post items for sale by creating software agents in either the Books or Music category, the two types of products supported in the marketplace. The seller can specify the description of the book, price range, agent negotiation behaviors, and buyer qualification requirements such as location and reputation ratings. Similarly, a buyer may post items wanted by creating software agents with corresponding types of information. Once created, software agents enter the marketplace, actively seek out prospective buyer/seller agents with matching item descriptions. Once the appropriate buyer and seller agents identify each other, negotiation is initiated. Agents negotiate with each other by sending and receiving offer messages, with item and price information. Upon a successful conclusion of a negotiation, buyers and sellers receive notification email from their agents, indicating the final price and contact information. The exchange of payment and product takes place outside of the Kasbah marketplace.

As an agent based electronic marketplace, Kasbah has a number of advantages over traditional consumer-to-consumer marketplaces. With agents as active presence, users no longer need to be involved in the constant tasks of searching for items, and in negotiating with all interested parties. Software agents are able to perform these tasks efficiently and autonomously.

In a Media Lab symposium in 1996, attendees were introduced to the Kasbah marketplace in a full day experiment. Each participant was given three objects, and certain virtual currency. The goal for each participant was to buy and sell in the marketplace, collecting wanted goods while doing so or generating enough profit by the end of the day to be converted into prize items [2]. This experiment first demonstrated the extent of Kasbah’s capability as an agent-mediated marketplace.
The Kasbah system implementation is limited as an infrastructure for creating new marketplaces. The knowledge of an agent is built into the system, such that an agent under the category of books is programmed differently than an agent under the music category. This implementation was designed to give agents inherent category information, so that they are able to behave differently based on the types of transaction categories they reside in. This implementation also presents a challenge when a new category of goods needs to be created, or when an existing transaction category requires modification. The addition of a new category requires the implementation of a new agent category, with corresponding category specific information. In addition, marketplace support structures need to be modified such that new agent types are recognized and incorporated into the system. These limitations make it difficult to use Kasbah for new marketplace construction purposes.

In order to facilitate the extension of agent-mediated transaction concepts to a wider range of applications, the Market Maker project was proposed. Market Maker is a tool for creating electronic marketplaces like Kasbah. Market Maker has been used to create several different marketplaces for goods (Books, Music, Electronics, Computer Software), as well as services (Translation Service and Financial Services). The reminder of this document is organized as follows. Section 2 describes current state of electronic marketplaces. Section 3 discusses the important characteristics and design issues of Market Maker. Section 4 analyzes Market Maker component structures. Section 5 provides the conclusion notes.
2 Background

A number of electronic marketplaces have been developed and are in commercial use as of 1999. Financial markets (stocks, bonds, futures, and commodities) are among the first to use electronic networks to facilitate transactions. NASDAQ, one of the largest stock markets, conducts transaction operations electronically [3]. Unlike NYSE (The New York Stock Exchange), there is no trading floor for human traders to gather about. All buying and selling offers are posted electronically on NASDAQ’s proprietary network by market participants and market makers. Transactions are completed either through automated systems or telephone communications.

Non-financial markets are now also migrating toward electronic transaction environments, mostly over the Internet. These markets can be categorized into three major areas, based on their participants: business-to-business (B-to-B), business-to-consumer (B-to-C), and Consumer-to-Consumer (C-to-C). Traditional markets for the three above areas are human interaction based. Participants are involved at all stages of the buying and selling process. The recent rise in Internet’s popularity created the opportunity for improving marketplace efficiency, by migrating some aspects of human interaction based transactions toward electronic based transactions.

In the business-to-business area, companies are forming online procurement operations, such as General Electric’s Trading Process Network (TPN) [4]. In the business-to-consumer area, companies such as Priceline.com are modifying traditional retailing operations for online environments [5]. In consumer-to-consumer areas, eBay has pioneered the online auction marketplace for collectable and second-hand goods [6].

The standard Consumer Buying Behavior framework identifies the following six stages of a transaction:

1. Need Identification
2. Product Brokering
3. Merchant Brokering
4. Negotiation
5. Payment and Delivery
6. Service and Evaluation

As of 1999, commercial electronic marketplaces are mostly concentrated on stages 1, 2, 3 and 4. Comparison shopping engines such as BargainFinder are able to perform cross-merchant comparisons.
based on price [7]. Marketplaces such as eBay facilitate price negotiations through auctions. Collaborative
filtering techniques are used by companies such as Firefly to assist consumers in identifying products they
would be interested in [8].

The Market Maker project addresses aspects of stages 2, 3, and 4. In contrast to current
commercial electronic marketplaces, it does not use an auction model as the negotiation framework.
Instead software agents negotiate with each other through sending buying and selling offer messages. In an
auction environment, the current high bidder is under the obligation of purchasing the item, unless a higher
bid is made. In Market Maker, agents are able to control the duration of buying and selling offers. Each
offer has a limited validity period, and offers can only be accepted before such period expires. Agents can
therefore effectively manage their outstanding offers and obligations under Market Maker.

The introduction of electronic marketplaces also brings forth the question of their impact on
traditional transaction methods. Most participants view them as a more efficient medium for conducting
transactions, due to the high speed and low cost of information distribution. In addition, with information
supplied electronically, software programs are able to easily sort and analyze such data, facilitating large
scale price comparisons and rapid negotiations. It has been speculated that the electronic marketplaces will
intensify pricing competition thereby benefiting buyers at the expense of sellers. It is worth noting that
while many companies participating in electronic marketplaces advertise low pricing, other factors such as
service and reliability are also valued by buyers when conducting negotiations. The electronic
marketplaces of the future may increase the trend of complex negotiations, where participants engage in
integrative negotiation over various aspects of a transaction, pricing only being one of the considerations
[9].
3 Market Maker Overview

Market Maker is designed to incorporate Kasbah’s agent based transaction concepts, and to provide a flexible infrastructure for creating Kasbah-like electronic marketplaces. Market Maker is designed with the following goals in mind:

- Dynamic marketplace creation
- Full HTML access
- Modularity
- Data persistence
- Portability

These goals are described in more detail below.

3.1 Dynamic Marketplace Creation

Transactions in Market Maker are conducted by software agents representing buyers and sellers. Agents have full knowledge of the particular categories that they deal with at run time, and are able to process category specific information intelligently. Category specific information for agents is not built into the implementation. Instead, a generic category of adaptable agents is implemented. When created, each agent becomes specialized transaction agent for a particular category. Category specific information necessary to enable this adaptation resides in a set of description tables, and is read by agents during the process of specialization.

The main advantage of using dynamic adaptable agent design is that new marketplace categories can be defined at run time, facilitating the creation of new transaction types on demand. Existing transaction categories can be improved and modified through the same channel, without disrupting ongoing agent negotiation activities.

Category specific knowledge includes category item description attributes, attributes checking and processing requirements, item matching and evaluation criteria. Each agent stores in its memory a copy of the above content for relevant categories. During subsequent interactions, agents are able to negotiate transactions by referencing the local category information.

As marketplace modifications and improvements take place, agents created at various marketplace stages may have in memory different versions of transaction category information. This presents an
obstacle to agent communications, as agents no longer have uniform understanding and interpretation
capabilities. One solution is to mandate uniform information version control over all agents, such that
when a modification to a global transaction category occurs, all agents are required to perform version
upgrade. While this solution is simple to implement, the uniform global version requirement represents a
substantial constraint to users. Whenever a marketplace update occurs, users would need to revise any
affected agents, removing outdated information fields and making new additions.

3.2 Full HTML Access

Market Maker is fully accessible through HTML browsers. Users typically perform the following
tasks while using the marketplace

- Registration
- Browsing
- Agent creation
- Agent administration
- User profile administration

Registration requires users to supply basic profile information such as name, email, and location.
Upon completion of the HTML form, users are notified via email regarding their registration status, and are
given instructions for signing onto the marketplace. Information collected in the user profile is used for
negotiation considerations, such as location constraints. Once a transaction negotiation is successful,
contact information is supplied to the buyer and seller.

Browsing and searching functions are accessed by users entering queries and clicking on relevant
HTML links. Information necessary to process such user requests is embedded in HTML queries, and is
parsed by the marketplace client handler module. Similar to web search engines such as Yahoo, users are
able to define item searches at various levels, and to obtain a list of item descriptions as the result.

Agent creation is the main method of conducting transactions in Market Maker. A seller with an
item available, may create a seller’s agent at the appropriate transaction categories. The seller then
provides a description of the item to the agent, negotiation preferences, and buyer qualifications. The data
is sent to the marketplace through HTML post operations. The buyer goes through a similar process to
create a buyer’s agent. This agent creation process is broken down into several dynamically generated
HTML form pages, where data entered on prior pages are embedded as variable settings.
Agent administration features are available to users on their home pages. Once logged onto the marketplace, a user is presented with a list of agents he or she created, sorted into three status settings: active, inactive, and completed. The list contains a description of each agent, buying or selling status, item description, current price, and activity information. Each agent description contains embedded HTML links that will access the agent administration features.

The user profile administration function is accessible through the navigation menu presented on top of the marketplace web pages. Users are able to modify profile information, as well as password settings. Other marketplace related features are present on the same navigation menu, including introduction information to Market Maker, and general marketplace statistics.

3.3 Modularity

The Market Maker implementation follows a modular design, and is comprised of the following function modules: user interface, agent functions, marketplace infrastructure, data persistence interface, and data storage. The modular structure of the implementation provides a simple framework for making future improvements and substitutions among different marketplace components.

At a higher abstraction level, the data object structure that Market Maker operates on can also be viewed as modular. Data components can be modified independently of each other, without compromising the overall data integrity. Under such data structure, different versions of agents and transaction category definitions can coexist and interact. Implementations of future marketplace improvements are not limited by the legacy data effects.

3.4 Data persistence

All data components in Market Maker are persistently stored inside a database module. The database used is a standard RDBM, with no explicit object persistence support. To ensure the persistence, consistency, and integrity of all object information both in memory and in the database, the marketplace utilizes a set of data interface modules that control all data object access to the database.

Several Object Oriented databases (OODB) are available for implicit data persistence support. These databases are specialized applications, with small installed base compared to standard RDBM.
Because currently no industry standard specification exists for OODB, they are not suitable for ensuring portability among all marketplace components.

Data persistence is critical to the usefulness of the marketplace. Regular programming object components are specific to the particular runtime environment they are created in. Without the database module, user-created agents would only exist in the marketplace as long as the marketplace run time environment is not interrupted. Stopping and starting the marketplace would be an interruption of the environment, and would result in the loss of all previous data objects in memory.

With data persistence support, the marketplace is able to completely resume its prior object collection and continue operations across different run-time sessions.

### 3.5 Portability

Market Maker’s modular components are connected to each other through a network communications interface, implemented using TCP/IP protocols. With network message passing, modular components can be distributed over a number of computers, reducing peak processing capacity required at any one particular computer.

The marketplace user interface is HTML based, programming modules are implemented using Java, and the data modules are implemented in SQL. All are standard languages widely available on a variety of software and hardware platforms. As a result the Market Maker project is portable over a wide range of current computing environments.
4 Components Analysis

This section will examine in detail the modular components within Market Maker, including specific implementation decisions, trade-offs made among various aspects of the features and functions, and experience and insight gained from the implementation.

Exhibit A: Components Overview

As shown in the above exhibit, Market Maker's individual modules are connected to each other through network message passing. Individual modules can be substituted and modified, provided the message passing specifications are satisfied. Module functions are described in detail in the following sections.
4.1 User Interface Module

Exhibit B: User’s Perspective of the Market Maker interface

From a user’s perspective as shown in the above exhibit, Market Maker consists of a set of HTML pages customized with the user’s information and queries. Two navigation menus are present at the left and top right of subsequent pages after user login. At the Market Maker home page, users may proceed with login, registration, or obtaining help information. Once a user enters the marketplace, a customized homepage with user’s agents information is presented, with two navigation menus available on this page. The menu options are as listed in the above exhibit.
Market Maker presents to the user a set of HTML pages as outputs, and accepts as inputs from the user HTML GET and POST operations. From the viewpoint of the buyers and sellers, the marketplace is comprised of a number of web pages. Main navigation functions can be accessed through the two sets of menus on web pages.

The first navigation menu is visible on the left side of the web pages. Access to the home page, new agent creation, item browse and search, market info, and logout are presented here. The second navigation menu is visible on the top right corner of the web pages. Access to marketplace help information, user profile, better business bureau, and feedback forms are presented here.

**Exhibit C: Navigation Displays**

As the user chooses one of the above menu options, an HTML request is generated to the marketplace, signaling the user’s navigation intentions. While these menu options look identical to different users, they contain embedded information regarding the user identity, and a time stamped authentication ticket. This provides information to the marketplace, such that it is able to customize output information for each user request. For example, a user request for the home page would be processed with the user ID, such that only information on agents created by this user would be presented.
The only static HTML pages in use by Market Maker are the initial user registration screens, and the login screen. Given that it is necessary for the user interface to keep track of user identity across many HTML requests, all other web pages are dynamically generated by the marketplace with built-in identification data.

This implementation does not require the use of session tracking capabilities for HTML handlers. HTML session tracking can be done through the use of cookies, which are tagging information supplied to user web browsers. Use of session tracking also requires the storage of authentication information on client computers, otherwise the marketplace is susceptible to identity spoofing by malicious users.

The implementation of the Market Maker User Interface consists of several components. The User Interface program module is implemented in Java Servlets. Servlet programs under maker.servlets package are used as HTML request handlers, responding to incoming user requests, parsing input information from links and forms, communicating with the marketplace about user requests, and responding to the user with HTML result pages. The marketplace home page, along with other static HTML pages and assorted images are served by a standard web server.

The maker.servlets package consists of over 20 individual Java Servlet programs. Each program handles a specific HTML page request from the user, and generates the corresponding result page as output. Since Market Maker does not require session tracking support, Servlet programs that require input information from each other can only receive it indirectly, through user HTML inputs. Each Servlet program packages prior input data needed for the next program, and embeds the data into HTML forms as hidden variables. Subsequent Servlet programs obtain access to the expected inputs by re-parsing hidden variable information from HTML forms.
Exhibit D: Example of connected Servlet programs communicating via HTML forms

When the user selects the Create New Agent option, screen 1 is first presented, mainly an HTML form with a list of current transaction categories, and buying/selling actions as user input options. This screen is generated by Servlet program 1, with user identification and authentication information as form hidden variables. Servlet program 2 takes user input, namely the action type and transaction category selection, and generates a category specific information page for the user to supply an item description, as shown on screen 2. Servlet program 3 in the series takes item description information from the user as HTML form variable values. Because program 3 does not have direct access to input data from screen 1, it requires the screen 2 program to supply it indirectly. The format of this indirect input data to program 3 is:

```html
<input type=hidden name="i" value='AkRXxCTAhCejJMiMsDSE3d'>
<input type=hidden name="n" value="Dave">  
<input type=hidden name="actiontype" value="selling">  
<input type=hidden name="categorynumber" value="1">  
<input type=hidden name="categoryname" value="Books">  
```

Five hidden variables are defined inside screen 2’s HTML form. The variable “actiontype” indicates selling, and the variable “categorynumber” indicates ID number for the transaction category selected by the user on screen 1.
Exhibit E: Example of connected Servlet programs communicating via HTML forms

(continued)

The variable data are parsed and re-packaged by all subsequent Servlet programs in the same series. The final program in this series, newagentform3, requires the variable values in order to package new agent creation information for marketplace host programs.

Communications between Servlet programs and the marketplace host occurs at three main points during the new agent creation process. Program 1 queries the marketplace host for a list of available transaction categories to be presented. Program 2 queries the host for a description presentation of the particular category as selected by the user on screen 1. Market Maker is designed to incorporate dynamic transaction category information, therefore at implementation time program 2 does not have the knowledge of the actual category presentation information. It initiates the information request with a category ID to the host, and returns the result to the user in the form of screen 2. Program 4 is the final screen for user inputs during the agent creation process. Once all inputs are complete, Servlet program 5, the last program in the series, packages information from user inputs at screen 1 to 4, and transmits them to the host, signaling the user request for creating a new agent.

As shown by the User’s prospective diagram on Market Maker interface, agent creation is among one of several options available to the user. Other options, such as user browse and search, are processed
in a similar fashion by a series of Servlet programs, with inter-connection among the programs handled indirectly through HTML form variables.

The result of browse and search queries are derived from the marketplace data, and can not be generated independently from Servlet programs. For these data results, Servlet programs initiate queries to the market host with the appropriate user input, and present the results back to the user, formatted as HTML pages.

Other user options are handled through a combination of dynamic HTML and static page information. Header information and the navigation menu options are always dynamically generated, due to the necessity of embedded information to be included. The market place help screen content has mostly static information, and is stored in the system as text files. When a user initiates a request for help information, and the Servlet help program generates a combined HTML page by incorporating text file information with customized headers.
4.2 Client Handler

Client requests are first handled by Servlets, then passed on to marketplace host programs. Client Handler, contained in ClientHandler.java, processes Servlet connections received through TCP/IP. Communications take the form of network message passing through Java supplied network pipelines and socket layers.

Client Handler is a multi-threaded, asynchronous program. Each time it receives a Servlet program request for connection, the handler spawns a separate Java thread with an instance of itself. This implementation has the benefit of providing scalability. Users are able to initiate marketplace requests in parallel, and simultaneous usage of the marketplace is only limited by the capacity of host computing environments.

Exhibit F: Multiple Instances of Client Handler Handling Parallel User Requests
The Client Handler program (CL) is implemented with a set of method calls, each corresponds to a Servlet program in the User Interface module that is capable of generating requests to the CL. A call to the Handler module begins with a connection to the Host program main network socket. Information regarding IP address and port number of this socket is shared among the UI and other modules. Once the dispatcher unit detects a request for connection, it creates a new Java thread, with an instance of Client Handler module, and an individual socket connection to this UI request, and initiates the thread run.

Client Handler within each thread expects header information from Servlet programs to initiate the request. Servlet programs supply the identification information and specify request types. Individual methods within CL are designed to further process requests beyond header information. If the client request is for a list of transaction categories available, as generated by Servlet program 1 identified in Section 3.1, then a corresponding CL method returns the list of category names and IDs. If it is Servlet program 2 requesting individual category description data, then its corresponding CL method first parses incoming data to obtain the category ID variable value, then generates a new request to the market host module for the category description data. Once the response from the host is obtained, category description information is then packaged and returned to the Servlet program.

Client Handler is designed to be an asynchronous program. Different instances of CL do not require explicit coordination among them for shared resources. Instead, a number of buffers are used in order to allocate market resources necessary for handling client requests. These buffers coordinate actions between the asynchronous CL and the synchronous market host cycle program.

These buffers represent an implementation complexity, but they are necessary for the performance as perceived by users of the marketplace. Asynchronous processing allows CL methods to instantly respond to user requests, without the need to wait for system resources to free up or request queues to clear. Under this implementation, users can directly benefit from the fast response time of Servlet programs.
4.3 Buffers

Buffers in Market Maker are used as interfaces among various modules. They serve two main implementation purposes:

- Buffers are used to preserve the correctness of interactions between synchronous modules and asynchronous ones. Modules such as Client Handler are able to deposit data and requests to buffers at any time. Periodically, synchronous modules such as the Market Host retrieve information from these buffers.

- Buffers are used to improve module performance. Due to the artifacts of today's network and storage systems, it is often more efficient to perform batch tasks, than to process individual tasks separately.

Exhibit G: Buffer Configurations

[Diagram showing process flow with Process 1, Process 2, Process 3, Asynchronous Deposits, Queued data and tasks, and Batch Withdraws connected to Host]
4.4 Software Agents

Software Agents are the focus of the Market Maker marketplace. Instances of software agents are created by users for transaction purposes. Agents are capable of identifying and qualifying prospective buyers and sellers. They search actively for appropriate transaction arrangements in the marketplace, by examining transaction information supplied by the users. Once one or more possible transaction partners are found, agents will initiate negotiations through generation of buying and selling offers. Agents are also capable of responding to offers made by others.

Exhibit H: Agent Capabilities

Agents are first created through a set of user specifications, consisting of:

- Category identification
- Item descriptions
- Negotiation strategies
- Qualification requirements

Category identification and item descriptions are the basis to inform the agent on what is being transacted. A sample user input is as follows:

Category: Translation Services
Action: Buying
Origin Language: Greek
Target Language: English
Level of Expertise: Medium
Type of Document: Business
Number of Words: 1,000
Immediacy of Service: 1 week
Negotiation strategies inform the agent on how price negotiation is to be conducted. A sample input is:

```
Active Period: 7 days
Starting Price: $50.00
Ending Price: $100.00
Pricing Function: Linear
```

Users may also specify how prospective buyers and sellers are to be qualified by the agent, based on a set of user profile criteria. A sample input is:

```
Location: Within MIT
Minimum Reputation: Good
```

The above information forms the User Specification module within an agent. At the process of agent creation, the above data are sent to the Market Host, which in turn generates an instance of a new agent with the user as the agent owner, with the input data as the agent’s initial knowledge base. As discussed in Section 3, Market Maker implements an adaptable generic software agent, without any built-in knowledge of particular transaction categories. This facilitates the creation of new transaction categories in Market Maker. However, a generic agent does not have any basis to interpret the input data on item description, which is necessary for agents to be able to compare different item descriptions and identify appropriate potential transactions. Therefore, category specific knowledge needs to be supplied to the agents at run time.

The process of a generic agent obtaining specialization in a particular transaction category can be viewed as several steps. First the user selects one of the categories in the transaction list. The category ID is supplied to the Market Host, which in turn returns the category specification to the new agent process. The agent proceeds to build the category specific knowledge module, containing the following information:

- Description fields
- Conversion functions
- Validity requirements
- Comparison functions
- Formatting functions
- Annotation List
The category specifications provide a set of operation guidelines for the agent to follow. Given a list of description fields, the agent is able to prompt the user for appropriate information on item descriptions. The validity functions ensure that the agent only accept correct input from the user for item descriptions. Conversion functions specify the nature of the data type for input fields, so that the agent is able to perform appropriate data conversions. Formatting and annotation information assist the agent in presenting relevant information in correct and user understandable formats. Comparison functions are referenced when the agent needs to identify compatible items for transactions, while filtering marketplace item descriptions listed by other agents.

Once an agent is created, it enters the marketplace. First the agent registers itself with the Agent Directory, an agent equivalent of the Yellow Pages. The agent is then positioned inside the Agent Buffer for addition to the marketplace transaction process. At the start of a market cycle, the Market Host accepts all pending agents in the buffer into the host. When each agent enters the appropriate transaction category, a category-wide broadcast announcement is made to all existing agents regarding the new arrival.

Agents in the marketplace are constantly attempting to expand their transaction partners list. A buyer agent and a seller agent may include each other on the partner list and agree to negotiate only after the following criteria are met. First the agent assesses the item description of the other agent, using category specific information and user specification data. If the item description appears to match, then the agent proceeds to qualify the prospective buyer/seller profile. Conditions such as location and reputation rating are then checked for user specifications. Provided the above conditions are met, then the agent adds the reference to the other agent, considering it as a prospect for negotiations.

At this point the agent generates a notification message to its user, indicating it has found a possible negotiation partner. The user is asked to confirm the decision made by the agent. This serves as an assurance of user control over agents in the marketplace. With the confirmation as a prerequisite to agent negotiations, users are guaranteed that their agents would not unilaterally start negotiating for an incorrect item.
Exhibit I: Agent Message Processing

As shown in Exhibit I, agents interact with other agents and with users through message passing mechanisms. Offer messages from other agents will prompt a response message in return, in the form of a counter offer, offer acceptance, or offer decline. An acceptance message will trigger both agents to contact Market Host, notifying the marketplace of a successful conclusion to the negotiation. Agents may also request information from each other for transaction qualification purposes, such as item description and user profile.

Users are able to interact with agents they own through messages generated by the User Interface. The owner of the agent may perform re-configurations regarding negotiation behaviors and buyer/seller qualifications, obtain agent status reports, and examine the list of potential transaction partners an agent has identified. Users may approve or disapprove of each potential transaction in the list, as a form of control over agent negotiations. Once a potential transaction partner is approved, the agent assumes further negotiation authority under the price range and negotiation strategy given. Provided the agent is successful in the deal making, its owner is under the obligation to follow through on the transaction based on the terms the agent has committed to.
4.5 Market Host and Agent Cluster

The Market Host module provides the infrastructure support for agents and User Interface modules. All agent objects reside in Agent Cluster, and are accessed by the Market Host as they are invoked during market cycles. Market cycle refers to the repeating process in which agents conduct transactions.

Exhibit J: Market Host Cycle

At the start of the Market Host, initialization steps are performed for network modules and market objects. Data integrity check is performed as objects are loaded into memory. Market Host then proceeds...
to restore last saved host states and agent states, in order to maintain a continuous marketplace progression over multiple runtime environments.

Market Host provides the following set of agent transaction support:

**Message Passing:** Agents receive incoming messages and produce outgoing messages at each market cycle. Outgoing messages are sorted by the Market Host based on destination agent information. When an agent becomes active, its first task is to receive new messages from Market Host. Upon completion of the agent processing, it may generate one or more messages to other agents, which are stored and sorted by the marketplace.

**Resource Allocation:** Market Host administers the resource allocation among agents, and has the authority to activate and deactivate individual agent activities. Market Host is designed to ensure that all agents will have access to the marketplace resources during each market cycle, and have equal opportunities to conduct transactions.

**Status Tracking:** Market Host performs agent status checks at the beginning of each market cycle. Agents that are no longer active, or are disabled by the user are removed from the active participants list, and are no longer included in further transaction processing.

**Transaction Completion:** Market Host processes completed transaction notices from agents once a conclusion is reached. The host updates agent status, sends notification to agent owners, and records the transaction in marketplace log.

At the start of a market cycle, Market Host incorporates new agents and users into the marketplace. While users are able to create agents asynchronously of market progress, agents that are created during a market cycle are stored in the agent buffer until the start of next available market cycle. Similar process applies to new user data. Under the current implementation, a market cycle is 5 seconds long. Agents are delayed for at most for 5 seconds before becoming fully active in the marketplace.

Market Host keeps a set of directories for objects, including agents, user profiles, and item description information. These directories provide the primary indexing structure for the marketplace objects, and are referenced when the objects are needed. The directory structure is designed to optimize objects access speed. In addition, the directory can serve a virtual reference translation, mapping in memory objects to persistent object storage images.
Exhibit K: Directory Structures and Reference Conversions

Provided agents reference to each other and information objects through directory reference, it becomes possible to store actual object instances outside of memory, into storage files or database records. The process of swapping objects in and out of memory occurs transparently to agents, and is managed by the directory structure and data persistence module. Without this implementation, the maximum agent population in Market Host would be limited by the size of memory in the marketplace run time environment.

Market Host supports users through a set of profile administration functions and access control procedures. A user may obtain authentication from the marketplace by providing the username and password information. Once the user information is verified, Market Host provides an authentication ticket to the user, through the User Interface module. This ticket carries identity, access, and login session time information, and is embedded into all web pages accessed by the user. Market Host is responsible for verifying the validity of such user authentication tickets, and to provide appropriate market access.
4.6 Data Persistence Interface and Database

Persistent data objects in Market Maker are stored on a RDBM database at all time. This includes user profiles, item descriptions, agent profiles and states, and transaction records. Objects are assigned with persistent status if the information and states they contain need to be preserved across multiple run time environments. Non-persistent objects in programs are discarded when the run time environment ends, usually occurs when a program is completed or interrupted. If agent objects are not persistent, then all information user created through agents would be lost once the marketplace operation is interrupted. It would then be necessary for users to re-create agents again, and would be an unacceptable burden to most users.

One implementation option is to apply a database centered design, where persistent information is stored on the database, and all operations are done through database queries. This design is popular with commercial electronic marketplace applications, because of its relative simplicity in implementation. However the design is less suited to a dynamic information environment such as Market Maker would require. In an agent based marketplace, agents constantly modify internal states as they progress through transaction negotiation process, to reflect new market information, current negotiation offers and obligations, and negotiation partners status. A traditional database centered design would not provide the sufficient dynamic object support.

Market Maker follows a complete object based design, implementing object persistence through a custom persistent data interface module. This interface supports transparent relocation of agent objects and other data objects between the memory and the database. While an agent object is in memory, data interface module updates the stored agent image on the database, as the agent internal states and knowledge base change. Should an unexpected interruption occur to Market Maker, objects can be restored and operations resumed, without any negative impacts to agent behaviors and negotiations progress.

Interfacing between persistent data module and database is done through JDBC/ODBC bridge connections over TCP/IP. A commercial driver is used to translation JDBC requests into ODBC, such that Market Maker is able to issue queries and obtain data from the RDBM database.
Exhibit L: Persistent Object Storage and Retrieval

As shown in the above exhibit, a specialized persistence interface controls external object references for all persistent objects in the marketplace. Both the database object image and object memory instance, if any, are synchronized to each other. Objects such as agents operate in memory while they are active. As object memory instance data and states change, the corresponding object image in the database is updated by the interface module. The persistence process is transparent to external object references.
5 Conclusions and Future Work

As a general market structure, Market Maker provides the tool to expand software agent base electronic commerce environments. Market Maker has the flexibility to accommodate transactions over many categories of goods and services, as well as providing a framework for dynamic transaction category configurations. Market Maker can be used for a variety of marketplaces, such as consumer-to-consumer, consumer-to-business, and business-to-business environments.

Agent interactions over service categories such as translations present several new challenges. Agents need to be able to negotiate with each other based on seller’s knowledge, background, experience, and expertise, and matching them with buyer’s needs and requirements. In addition, issues such as scheduling and reference checking may arise. Several factors are necessary to facilitate such agent matching. Information regarding seller’s knowledge and experience needs to be categorized and stored. Some aspects of such information are subjective and do not allow easy translation into electronic representations, and a flexible matching and searching algorithm may be needed. Future expansions of the marketplace functionality will be better able to address these above issues.

Software agents currently operate based a set of user defined negotiation behaviors, in terms of time limit, start and end price, as well as price/time function. Future agents may incorporate complex price negotiation functions while operating in integrative negotiation environments. The concept of price differentiation based on individual user preferences can be extended beyond product features. Agents can incorporate reputation, location, and a host of other factors in deciding the final price. And such price is specific to a single possible transaction, making it possible to differentiate based on buyer and seller profiles as well.

The competitive nature of many marketplaces implies that participants operating on simple decision criteria and behavior patterns may be at a disadvantage. Simple agents operate on the assumptions that there is equal information uncertainty among agents. However such assumptions may not hold if some agents are able to collect, analyze, and infer confidential preference information from simple agents. Agents using integrative negotiation strategies are also at risk, because such agents may disclose confidential user preferences in the hope of achieving optimum allocation results for all parties. Disadvantageous situations may arise when such “trusting” agents operate in a competitive environment.
Specifically competing agents are able to manipulate “trusting” agents based on information obtained from their preference sharing behavior and therefore could extract more profits from them [10].

Given that each agent represents the interests of a particular user, and not the negotiation community as a whole, it follows that in a competitive environment agents may try to outmaneuver other agents in an attempt to further their users’ interests. Agents can model each other’s behavior in order to understand opponent reactions, and to find the optimum course of action based on anticipated competitor responses. Alternatively, such competitive situations can be avoided by imposing agent behavior rules over the marketplace, with all agents operating on equal level.

Agents in Market Maker follow the offer-response protocol in negotiating with each other. Other models of negotiation may include auctions and reverse auctions. Bartering can be incorporated into the above negotiation models such that in addition to offering a price based on currency, the agents may negotiate with each other by offering an exchange of a set of goods and services. Such bartering exchanges occur frequently at the smallest scale of transactions, such as goods swapping among friends, as well as in large scale exchanges, where countries conduct trades without currency.
6 References


### Appendix

Summary of Market Maker directories, files and programs:

#### Directories:
1. servlet: User Interface Servlet programs
2. wwwroot: web server files
3. setup: setup information
4. host: marketplace program files

#### Programs and Files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agentmessages.java</td>
<td>Agent messages screen</td>
</tr>
<tr>
<td>Agentmessagesconfirmed.java</td>
<td></td>
</tr>
<tr>
<td>Browse.java</td>
<td>Marketplace browsing</td>
</tr>
<tr>
<td>Browseform1.java</td>
<td></td>
</tr>
<tr>
<td>Browseform2.java</td>
<td></td>
</tr>
<tr>
<td>Completedagent.java</td>
<td>Completed agent information</td>
</tr>
<tr>
<td>Emailpassword.java</td>
<td>Password reminder</td>
</tr>
<tr>
<td>Feedback.java</td>
<td>Marketplace feedback</td>
</tr>
<tr>
<td>Home.java</td>
<td>Homepage</td>
</tr>
<tr>
<td>Infoheader.java</td>
<td>Utility</td>
</tr>
<tr>
<td>Kasbahinfo.java</td>
<td>Help information</td>
</tr>
<tr>
<td>Login.java</td>
<td>Login</td>
</tr>
<tr>
<td>Logout.java</td>
<td>Logout</td>
</tr>
<tr>
<td>Marketinfo.java</td>
<td>Help Information</td>
</tr>
<tr>
<td>Newagent.java</td>
<td>New agent processing</td>
</tr>
<tr>
<td>Newagentform1.java</td>
<td></td>
</tr>
<tr>
<td>Newagentform2.java</td>
<td></td>
</tr>
<tr>
<td>Newagentform3.java</td>
<td></td>
</tr>
<tr>
<td>Newagentform4.java</td>
<td></td>
</tr>
<tr>
<td>Newreputation.java</td>
<td></td>
</tr>
<tr>
<td>Newuser.java</td>
<td>New user registration</td>
</tr>
<tr>
<td>Profile.java</td>
<td>User profiles</td>
</tr>
<tr>
<td>Profileupdate.java</td>
<td></td>
</tr>
<tr>
<td>Reconfigagent.java</td>
<td>Agent reconfiguration</td>
</tr>
<tr>
<td>Reconfigagentconfirmed.java</td>
<td></td>
</tr>
<tr>
<td>Registration.java</td>
<td>Registration</td>
</tr>
<tr>
<td>Removeagent.java</td>
<td>Agent removal</td>
</tr>
<tr>
<td>Removeagentconfirmed.java</td>
<td></td>
</tr>
<tr>
<td>Reputation.java</td>
<td>Reputation information</td>
</tr>
<tr>
<td>Search.java</td>
<td>Marketplace search function</td>
</tr>
<tr>
<td>Searchform1.java</td>
<td></td>
</tr>
<tr>
<td>Searchform2.java</td>
<td></td>
</tr>
<tr>
<td>Searchform3.java</td>
<td></td>
</tr>
<tr>
<td>Uilibrary.java</td>
<td>Utility</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Agent.java</td>
<td>Agent definitions</td>
</tr>
<tr>
<td>AgentBuffer.java</td>
<td>Agent buffer and directory service</td>
</tr>
<tr>
<td>AgentDirectory.java</td>
<td></td>
</tr>
<tr>
<td>ClientHandler.java</td>
<td>Client Handler</td>
</tr>
<tr>
<td>Datacoder.java</td>
<td>Utility</td>
</tr>
<tr>
<td>DbAgent.java</td>
<td></td>
</tr>
<tr>
<td>DbCat.java</td>
<td>Data Interface</td>
</tr>
<tr>
<td>DbHandler.java</td>
<td></td>
</tr>
<tr>
<td>DbUser.java</td>
<td></td>
</tr>
<tr>
<td>Host.java</td>
<td>Market Host</td>
</tr>
<tr>
<td>Market.java</td>
<td></td>
</tr>
<tr>
<td>ItemDescription.java</td>
<td>Item definitions</td>
</tr>
<tr>
<td>MailHandler.java</td>
<td></td>
</tr>
<tr>
<td>MarketInfoBuffer.java</td>
<td></td>
</tr>
<tr>
<td>MarketLog.java</td>
<td>Utility</td>
</tr>
<tr>
<td>MarketMessage.java</td>
<td></td>
</tr>
<tr>
<td>MarketMessage2.java</td>
<td>Message definitions</td>
</tr>
<tr>
<td>MarketParam.java</td>
<td></td>
</tr>
<tr>
<td>Status.java</td>
<td></td>
</tr>
<tr>
<td>Utilibrary.java</td>
<td>Utility</td>
</tr>
<tr>
<td>Userinfo.java</td>
<td>User Profile</td>
</tr>
</tbody>
</table>