Toward a Virtual Marketplace: Architectures and Strategies

Maksim B. Tsvetovatyy and Maria Gini

Department of Computer Science, University of Minnesota
4-192 EE/CSci Building
200 Union Street S.E., Minneapolis, MN 55455

tsvetova@cs.umn.edu, gini@cs.umn.edu

Abstract

In recent years, many researchers as well as commercial companies have attempted to create intelligent agent-based markets or retail outlets. So far, these systems have fallen short of changing the way commerce is done over the Internet. An incomplete implementation of the market metaphor, lack of automated purchasing, and lack of agent cooperation algorithms are, in our opinion, the major reasons of this shortfall.

In this research, we attempt to address these problems by designing an open marketplace architecture that includes all elements required for simulating a real market (*i.e.*, communications, goods storage and transfer, banking, administration and policing, etc.). We also address the issues of automated purchasing and agent cooperation by devising strategies and algorithms for them.

This paper also reports findings that resulted from implementing and conducting experiments with a free-market agent architecture (MAGMA). MAGMA is an extensible architecture that provides all services essential to agent-based commercial activities. These services are available through an open-standard messaging API, which allows use of a heterogeneous set of agents independently of platform and language.

1 Introduction

Development of the Internet has spurred a number of attempts to create a virtual marketplace where agents, both human-controlled and equipped with intelligent algorithms, participate in trading of physical and electronic goods, as well as stocks and other investments.

The purpose of this research is to propose an architecture for such a marketplace, including required infrastructure, communications, security and business aspects.

In our research, we focused on the following topics:

- What infrastructure elements are needed to serve an agent-based marketplace, and how they can be implemented?

- How a purchase-for-use marketplace can be transformed into an investment marketplace, and how can automated decision algorithms be designed to serve such a market?
- What are good strategies for creating alliances of such agents?

To answer these questions, we implemented a prototype of an agent marketplace architecture, called MAGMA (Minnesota AGent Marketplace Architecture), and completed a series of experiments with it, determining its strengths and weaknesses, as well as directions for future development.

Currently there are several intelligent agent-based online shopping services. The most prominent are Bargain Finder, developed by Andersen Consulting, FireFly, from FireFly, Inc, and ShopBot, from the University of Washington. Curiously, Bargain Finder and FireFly only include shopping for music CDs and tapes.

Bargain Finder [5] (see `http://bf.cstar.ac.com`) is an agent that searches several online music stores for lowest prices on CDs and cassettes. Unfortunately, even though it has been described as a revolutionary system, it is not much more than a database search engine. Shopping is limited to retailers that subscribe and pay for the customer referrals. Also, customers have to know and spell correctly the name of the artist and album, so the system is more oriented toward a user that knows what he wants and not a casual browser.

FireFly (see `http://www.agents-inc.com/`) is an agent-based mail order outlet. The system attempts to establish the user's preferences in music styles and artists through surveys and ratings, and then offers the user a list of CDs that conform to these preferences. This system worked very well after a few minutes of training (entering and rating CDs), returning a complete set of Miles Davis albums after we entered a reference to John Coltrane (both jazz artists). This system is a fine collection browser, even though it is still limited to one retailer and does not have any way of comparison-shopping.

ShopBot [4] is a domain independent comparison shopping agent that explores home pages of several vendors on the World Wide Web and learns how to shop.

All these systems are interesting shopping experiences, but they do not go all the way in providing a virtual free market metaphor. The system that most closely resembles our approach to designing a virtual marketplace is Kasbah [3]. Kasbah allows users to create agents that buy and sell goods on behalf of the user. The main difference between Kasbah and MAGMA is that MAGMA is designed to be a distributed system written on top of an open messaging API. In this paper, we discuss our attempt to create a agent-based virtual marketplace, as well as describe our working prototype of MAGMA.

2 Architecture for Agent-Based Marketplace

An agent-based marketplace needs to exhibit many properties attributed to physical marketplaces [1]. There has to be a banking system, some communication infrastructure, systems that enable agents to transport and store goods securely, as well as administrative and police systems.

2.1 Banking

In order for an agent-based marketplace to become anything more than a toy, it has to be able to communicate with existing banking and financial services. Thus, the design of

a banking system, as well as agent-to-bank communications has to be built on an open standard, allowing traditional banks to integrate a marketplace interface into their legacy systems.

Another crucial element in providing financial services over agent-based systems is security. Consequently, the communication protocols for all communications, especially agent-to-bank and bank-to-bank, need to include a layer of encryption, as well as other safeguards.

One of the methods to make monetary transactions secure is to use wrappers that make funds inaccessible to anyone except the intended recipient. Another way is to coordinate every transaction with all parties to be sure that no fraudulent activity is occurring.

2.2 Communication Infrastructure

It is important to have an efficient and robust communication infrastructure built into the core of the system. For example, such a system should not rely on a central hub to route the messages, but instead have a mesh of redundant hubs, interconnected with each other. This system would be somewhat similar to the Internet email system, and, in fact, could even use the existing facilities.

A communication system has to be built on an open standard to enable developers to produce platform-independent systems that plug into the marketplace architecture.

Even though the communication system is independent of agent architectures, there are some design limitations and communication protocols that must be followed:

- Agents must be able to access global blackboards that contain offers to buy and sell from other agents.
- Agents must use a common language for all outbound communications. Minimal language must include posting and responding to offers, negotiating and executing transactions. If agents work as a part of the team with managers and other agents, they must be able to communicate with members of the team.

A sample algorithm that can be used in an automated agent is shown in Figure 1.

2.3 Transfer and Storage of Goods

An important aspect in an electronic market is the representation and handling of physical goods. While these goods can be easily represented by software objects, these objects have to exhibit some of the qualities of their physical counterparts.

For example, these objects have to be copy-protected to ensure that an object cannot be in more than one place at the same time. To prevent theft, objects may be encoded by the owner, making them accessible only to agents authorized by the owner.

Agents should also be able to arrange for physical shipment of items purchased or provide a "raincheck" that will enable the buyer to have the item shipped at a later time. Electronic items, such as software, results of database queries, or books from an online library, can be downloaded directly to the buyer over one of the existing protocols (such as FTP or HTTP).

Existence of the rainchecks can provide an interesting development similar to futures trading in commodity markets. An agent can profit by buying a raincheck when the price is low and selling it to another agent or shipping the item later, when the price has risen.

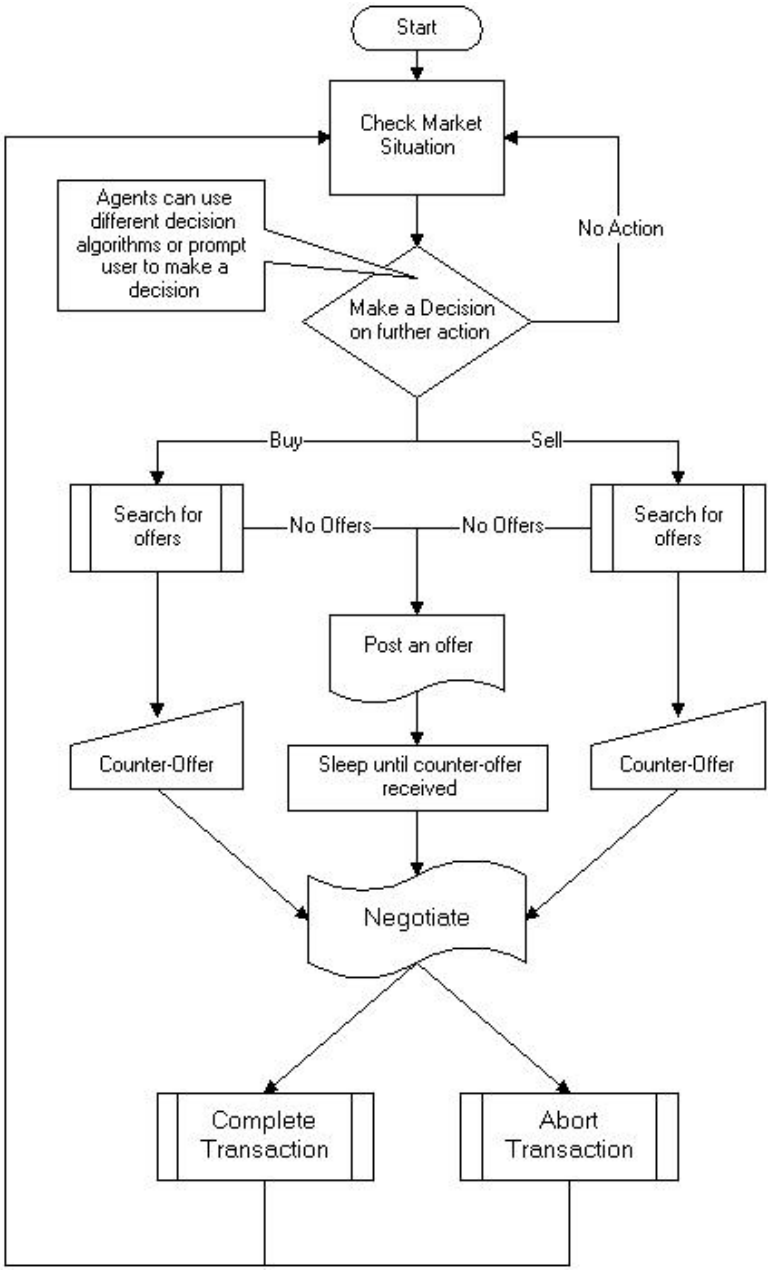

Figure 1: A sample agent algorithm

2.4 Administration and Policing

In order for the system to be successful as a business, there must be a central administration that monitors the activity of agents and makes sure that there are no illegal transactions. Other functions of the administrator would be to collect sales taxes and commissions, and maintain credit and service ratings for agents. Most likely, the administration of such a system will have to be only partially automated, with a human operator controlling the main aspects of a system's operation.

3 From Buying for Use to Buying for Investment

As soon as the notion of a raincheck, or delayed shipment, is introduced, we can move from a marketplace metaphor to an investment market metaphor. Instead of trading physical goods, agents may buy and sell promissory notes, or futures, that secure such shipment. So, instead of buying goods for consumption, agents may buy futures solely for resale at a later date.

If such a market develops, it would be appropriate to automate the decision process that controls buying and selling of futures. In the next section we will discuss several strategies for such decision algorithms that we have investigated. We will implement them in the next version of our prototype.

3.1 Decision Algorithms

The simplest form of trading strategy is *Fixed-Term Investment* [2]. Agents using this strategy purchase stocks at the lowest possible price at an arbitrary time, wait for the amount of time set by the operator and sell at the highest possible price. This strategy is not guaranteed to bring any profit, but is sometimes used in real stock markets for investment in stable stocks (such as AT&T or IBM) when the economy is growing.

Another simple form of trading strategy would be to *Follow Another Agent* (*i.e*, hopefully, using a strategy other than following). Following can be either "dumb", when the agent follows only one trader no matter what happens, or "smart", when the agent follows agents that make the most profit (profit or loss can be disclosed at the end of each day). Followers can constitute a large force in the market and have to be seriously considered by other traders.

A third strategy, *Monotonic Reactor* [2], is a more adaptive and dynamic strategy that is likely to bring profits in short-term investments. Agents using this strategy wait until a stock exhibits a monotonic increase, and then make a purchasing decision. If the stock in trading exhibits some form of sinusoidal fluctuations over time, this strategy will bring very good results. However, functions that represent price fluctuations in the market are far more complex than simple sinusoidal functions, so this algorithm will not be able to predict long-term trends in stocks.

A more complex strategy, *PMax with Adaptive Extensions*, will attempt to predict trends by estimating the effects of supply and demand on prices. PMax is an adaptive algorithm that uses a neural network and a system of automatically adjusted thresholds to predict price fluctuations over the short run and make purchasing decisions accordingly. In its prediction, PMax considers not only the current market situation (supply and demand for the stock, and its volatility) but the effects of its own actions and actions of its followers on the market. PMax also calculates the probability of external events altering the market situation, and adjusts its thresholds for more cautious or more aggressive trading.

One of PMax's advantages is that most of its decision-making is adjusted automatically when a change occurs. Although PMax has a learning curve during which it will make very bad decisions, it is aware of this and restrains itself until a significant percentage of decisions are successful.

One of PMax's biggest disadvantages is that, if given authority over high-volume trading, it will actually create price fluctuations instead of merely responding to them. For example, if such a PMax agent decides that the price of the stock will fall, and sells all of

its holdings, while triggering its followers to do the same, the volume might be big enough to create a downturn in the price.

3.2 Multiagent Systems

When multiple agents operate in the same market, there are interesting possibilities for them forming firms and alliances [6], [7]. We briefly examine some of the different organizations that multiagents systems can take.

3.2.1 Top-Down Managed Firm

This arrangement consists of many trading agents subservient to one or more management agents. The system can include more than one level of management, with different levels responsible for different types of decisions. All money available to traders is distributed centrally (the agent receives money from its direct manager, which, in turn, is allocated money from its manager, and on to the highest executive).

Managers allocate money to traders according to their projections and trading history, and have to approve buying or selling decisions. If a manager decides to use a different strategy on a certain stock, it can reallocate agents. If a trader has shown its ability over time, it can be allowed to trade more freely without reporting every transaction to the manager. The advantages of this approach are that the manager is aware of everything that is going on and has enough information to make "big picture" decisions. This scheme also facilitates use of a tree-like vertical structure with high administrators making general policy decisions and lower level managers and traders making more and more detailed decisions.

However, the system requires increasingly complex manager programs, deeply rooted in economic theories. Even though there is a niche for predicting trends in the market and determining their cause-and-effect relationship, it is an extremely complex problem in itself. Also, managers need to be aware of the way lower agents work and are architecture-dependent, thus making this the least flexible of the organization structures.

3.2.2 Competitive alliance

This system employs independent trading agents. Agents are free to make their own decisions, can share information or join or leave the alliance at their discretion. This arrangement uses a blackboard to publicize information to member agents. When need arises (for example, for sale or loan negotiation), agents can open private communication channels secured by encryption to exchange information directly.

Each agent receives its money from the human owners through a link between physical and virtual banking systems. If an agent needs extra money for a transaction but does not have any available, it can post a loan request to the blackboard. Agents that have available money can enter loan negotiations – depending on their general policies.

Also, each agent posts its profit/loss report to the blackboard. This report serves as the agent's "credit history" that other agents can use to grant or refuse loans or adjust interest rates.

This system is completely architecture-independent, and can employ agents written in different languages, and running on different systems. The system could even include

human traders working side-by-side with electronic agents or providing them with advice via the blackboard.

3.2.3 Cooperative Alliance

This system is technically similar to the competitive alliance, but here the alliance of agents is one team whose goal is to bring profit to the whole team, not to each individual agent, as in the competitive alliance. Agents are free to make their own trading decisions, choose their trading strategies and share information and funds. Each agent is free to keep tabs on other agents but is not required to.

We can also introduce credit banks, that could either be non-profit "Credit Union" agents that pay interest to lenders and charge the same rate from borrowers, or for-profit "Bank" agents that keep a portion of interest to receive profit and cover operating costs. The system needs no central administration and will ultimately be self-regulating. If a trading agent has accumulated enough capital, it can also provide banking services to other agents.

Although it is not necessary, the system might use an "adviser" agent (which might be electronic or human) providing help in viewing the 'big picture' of the market and trends.

3.2.4 Hybrid

The forms of administration and alliances described above can coexist in a hybrid setting, which could ultimately provide the most flexible trading environment.

4 Architecture of the MAGMA System

The MAGMA (Minnesota AGent Marketplace Architecture) system is a prototype of the virtual free market. MAGMA is written in Allegro Common Lisp with CLOS and CLIM X-Windows libraries.

Currently, the system simulates a network-based marketplace by running multiple processes inside one Lisp image (using Allegro's multiprocessing libraries). The only communication permitted among agents and between agents and market infrastructure elements is via messages that are similar to messages typically sent over the network [8].

Even though the current incarnation of MAGMA is proprietary, all parts of the system are designed for an open-standard system. When the networked version (NetMAGMA) is implemented, any agent that conforms to the messaging API can join the trading. Thus, in the current design of MAGMA there is a heavy emphasis on security, as well as creating redundant infrastructure and robust messaging API.

4.1 MAGMA Subsystems

MAGMA consists of several functionally independent subsystems, divided into general-purpose agents and infrastructure subsystems. The infrastructure subsystems facilitate transactions such as messaging, secure money and object transfer, bidding, commissions and sales taxes, as well as a primitive form of advertising. These systems include banks, offer-boards, cash-register and messaging facilities.

4.1.1 Widgets

Widget is a top-level class that includes all objects traded in MAGMA. The Widget class includes a data structure with slots for widget name, description, and pointer to its contents. Currently the contents are represented by a string, but later versions will include specially coded URLs to pages where a buyer's agent can secure shipping of a physical object or download electronic information.

Although there is no separate facility to handle widget transfers, it is extremely important that widgets behave like objects in the real world. Widgets cannot be owned by more than one agent, cannot be in more than one place at the same time, and cannot be sold twice. Although selling electronic information would involve making multiple copies of the same file, for tracking purposes it is better to create single-use widget wrappers.

To ensure the "physical qualities" of widgets, we have created widget wrappers with owner keys. A simple algorithm produces a unique and random 5-digit number, which is the only way to access the contents of the widget. When a widget is sold, the new owner receives the access key as well as the widget, and immediately changes the key (so if a rogue agent wants to sell a widget twice, its key would be invalidated).

Even though this is a fairly secure protocol, there is still a several millisecond-long window of opportunity for fraud, if the seller agent can change the key while the widget is in transfer or send an invalid key.

4.1.2 Messaging Facilities

Messaging facilities provide all communications between agents and with other subsystems. This completely isolates the agents from the top-level Lisp environment.

Each agent is assigned a mailbox and entered into a directory. When an agent or an infrastructure subsystem needs to send a message to another agent, the message is passed to the messaging system that takes care of delivering it.

Mailboxes are password-protected, and since passwords are stored in a process-specific environment, they are inaccessible to anyone except the agent that they are assigned to. All passwords are unique 8-digit randomly selected numbers.

In future versions we will develop redundant directories with parallelized searching. This will provide faster and reliable message delivery under any system load.

4.1.3 Banks

A bank consists of a table of accounts and a set of routines that produce secure money transmissions.

Accounts are password-protected and include a checking account balance and a credit line. In this prototype, the credit line is only used by the register, but in future versions there will be algorithms determining whether to grant or deny credit to an agent.

Money is transmitted in a secure wrapper, which we call a *check*. In fact, even though the mode of transmission is similar to sending a check, all information is cross-checked between accounts, which makes a check forgery-proof and single-use.

Each check is outfitted with a single-use unique key generated by a special algorithm. This key, as well as other information on the check, is stored at the bank until the check is deposited. If any of these fields do not match, the deposit fails. If the deposit succeeds, this information is removed, making future use of the key impossible.

Currently, MAGMA includes one centralized bank, but a simple addition to the messaging API will allow not only multiple banks, but also an open-standard bank system. This will allow third-party developers to design links from existing "real-life" bank computer systems to virtual market banks.

4.1.4 Offer Boards

Offer boards are places where both buyer and seller agents can post public offers. These offers are later retrieved by agents that are interested in responding to them.

An offer board consists of offer tables and search facilities. The current prototype has a flat table architecture, with entries indexed by their names. Each cell of the table contains a list of similar items, sorted by price.

The table can be searched using the name of the item, and returns the lowest offers for this item (the number of offers has to be supplied by the user).

In future versions, this flat structure will be changed to a tree or graph structure, with a graphical browser interface.

4.1.5 Register

The register is used as an intermediate storage during transfer of a widget to ensure that proper sales tax and commission is collected. It also serves as a security monitor to avoid fraudulent transactions where the seller sends an unusable widget (*i.e.*, widget with an invalid key) or the buyer has insufficient funds to pay for the widget.

Widgets are stored in a table, indexed by a randomly generated location marker. After receiving the widget from the seller, the register adds sales tax and commission and forwards a check to the seller, along with a location marker for widget storage. This location marker and total cost, including tax and commissions, are wrapped in a message and sent to the buyer. The buyer writes out a check and sends it to the register, which forwards the widget to the buyer.

Registers have to be centralized and owned by the company that runs MAGMA. This is the only way security of transactions can be ensured. To reduce system load, it might be beneficial to use more than one register (either by redundancy or by splitting them into "departments"). This can be done by incorporating the address or name of the register into location marker (i.e., instead of using 70949 as a marker, use 70949@register5.electronics or something similar).

5 Architecture of Agents in the MAGMA System

In MAGMA, each agent is represented by two independent processes, running simultaneously, and a CLOS data structure. The two processes are a *frame process* and a *main agent process*. The CLOS data structure keeps track of the agent's inventory and bank balances, as well as housekeeping data for running both processes.

The *frame process* is a CLIM frame. All user interaction is handled by this process, so the main agent process can process messages and do other tasks without waiting for the user to finish entering data or make a purchasing decision. The only way the frame process can communicate with the main agent process is by sending messages. The main process can control the frame process by sending messages to its input streams or by displaying

interface objects that cause certain pieces of code to be executed when the user invokes them.

The frame process also has a separate stack-group (a coroutine) that displays the status of the main process. To honor the tradition started by Netscape^{TM}, it is a simple animation that advances one frame every time the main process fetches messages.

The *main agent process* has several components. The main component is a fetch-decode-dispatch loop. Its purpose is to fetch messages from the mailbox, separate contents from message wrappers and use a hierarchical dispatch system to call the appropriate action. In the future, it might be appropriate to integrate public-key cryptography into the messaging system, so the loop will also include a decryption step.

Currently, agents recognize the following messages:

Commands

Sell	(COM SELL arguments)	Sells a widget from inventory
Buy	(COM BUY arguments)	Tries to buy a widget specified by the arguments

Requests

Money	(MREQ recipient amount)	Sends the amount of money to recipient
Widget	(WREQ recipient widget-id)	Sends a widget to recipient (Commands and requests can be only issued by the frame process or an authorized agent. Others will be ignored)

Transfers

Money	(MON check)	Receives and deposits money
Widget	(WID widget)	Receives a widget
Key	(KEY key)	Uses a key to take a widget from the register

Offers

Counter-offers	(OFFER COUNTER)	Makes a counter offer
Accept	(OFFER ACCEPT)	Accepts an offer
Reject	(OFFER REJECT)	Rejects an offer

In this version of MAGMA, agents can only react to accept offers. Currently we are working on implementing negotiation strategies to enable agents to negotiate for lower prices.

5.1 Example

To illustrate the operation of the system, we will use a simple market situation with one seller and one buyer. This is sufficient to demonstrate the processes and exchanges that happen while keeping the size of the log manageable.

For this run, we used a widget taken from the "Canonical list of computer viruses" from rec.humor USENET group: "MCI VIRUS - Encourages you to send it to your friends

and family."

1. The system is initialized with 2 agents. Each receives $1000.
   Frame processes are started and windows are opened.
   The user presses the "Run" buttons to start the main processes.

   ```
   |Agent1059|
   Frame Event: Button pressed - Run
   Agent1059: Starting up...Ready
   |Agent1088|
   Frame Event: Button pressed - Run
   Agent1088: Starting up...Ready
   ```

   Now that all processes are initialized and ready, we can proceed.

2. Adding a widget to the inventory

   ```
   |Agent1059|
   Frame Event: Button pressed - New Widget
   Frame Event: Display form
   Frame Event: User input in form
   :NAME "MCI VIRUS"
   :CONTENTS
   "Encourages you to send it to your friends and family."
   ```

   A widget object is created with this data and sent from the frame process to the agent process in a message:

   ```
   From: |Agent1059| To: |Agent1059|
   Message: (WID
              #S(WIDGET
        :NAME "MCI VIRUS"
        :CONTENTS
        "Encourages you to send it to your friends and family."
        :KEY
        50344)
     50344)

   Agent1059:Received a widget "MCI VIRUS"
   ```

   As soon as the agent process receives the wrapped widget, it adds the widget to its inventory and displays a button corresponding to the widget in the inventory window. When the button is pressed, it initiates the sale of the widget. This is shown in Figure 2.

3. Offering a widget for sale.

   The user presses the inventory button for MCI VIRUS and initiates a form display:

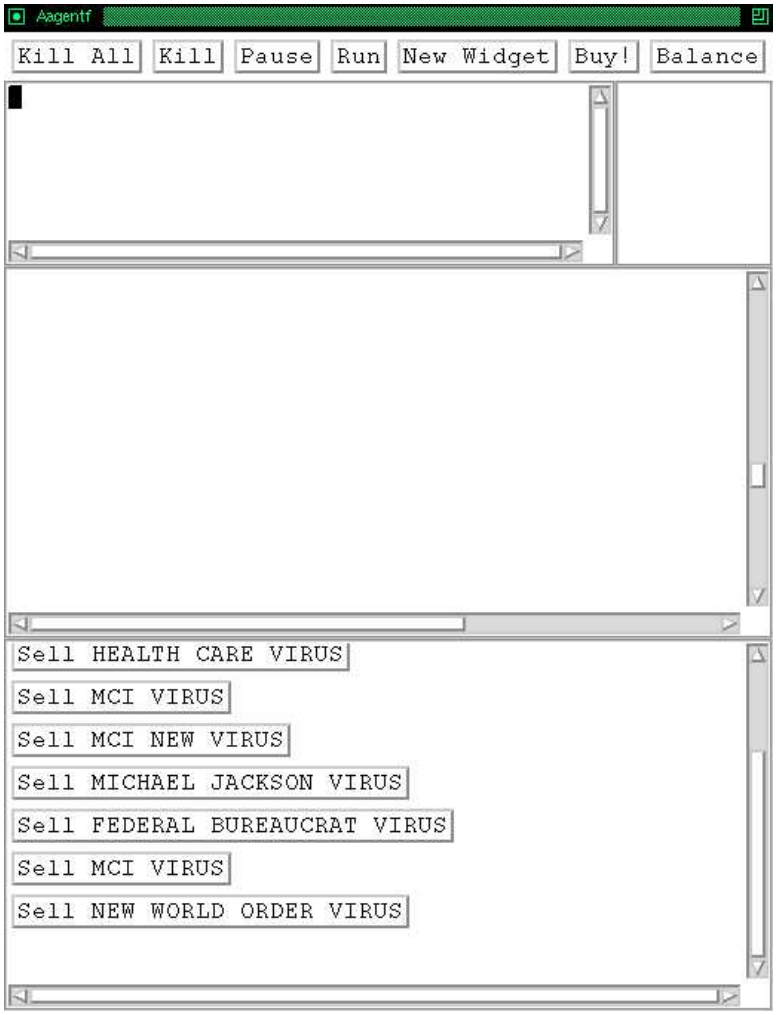

Figure 2: Seller agent with inventory

```
|Agent1059|
Frame Event: Button pressed - Sell MCI VIRUS
Frame Event: Display form
Frame Event: User input in form
:DESCRIPTION "MCI Virus description"
:PRICE 10.0
```

After the information is entered, the interface process forms a command message and sends it to the agent process:

```
From: |Agent1059| To: |Agent1059|
Message: (COM
  SELL
          "MCI VIRUS"
  "MCI Virus description"
  10.0)
```

The agent process makes an offer and sends it to the offer-board:

```
Offer from Agent1059, #S(OFFER :WIDGET "MCI VIRUS"
       :WIDGET-DESCRIPTION
       "MCI Virus description"
       :PRICE 10.0 :TYPE T
       :ACTION INIT :ID 63692)
```

Now the user of a different agent would like to purchase the widget.

The user presses the "Buy!" button and enters the category in which to search for offers, as well as the number of offers he is willing to consider:

```
|Agent1088|
Frame Event: Button pressed - Buy!
Frame Event: Display form
Frame Event: User input in form
:CATEGORY "MCI VIRUS"
:NUMBER 5
```

The frame process calls the offer-board search and receives a list of five (or whatever the value of :NUMBER is) offers, sorted by price (lowest bids come first):

```
Searching for offers...
```

Only one offer is found, and a button is displayed with this offer's description and price. If there were more offers, there would be as many buttons as offers, as shown on Figure 3:

```
Frame Event: Display button
```

When the button is pressed, the agent receives a command to follow up on the offer and buy the widget:

```
|Agent1088|:
buying MCI Virus description for $10.00 from |Agent1059|

From: |Agent1088| To: |Agent1088|
Message: (COM
  BUY
  #S(OFFER
     :WIDGET "MCI VIRUS"
     :WIDGET-DESCRIPTION "MCI Virus description"
     :PRICE 10.0
     :TYPE T
     :ACTION INIT
     :ID 63692)
  |Agent1059|
   10.0)
```

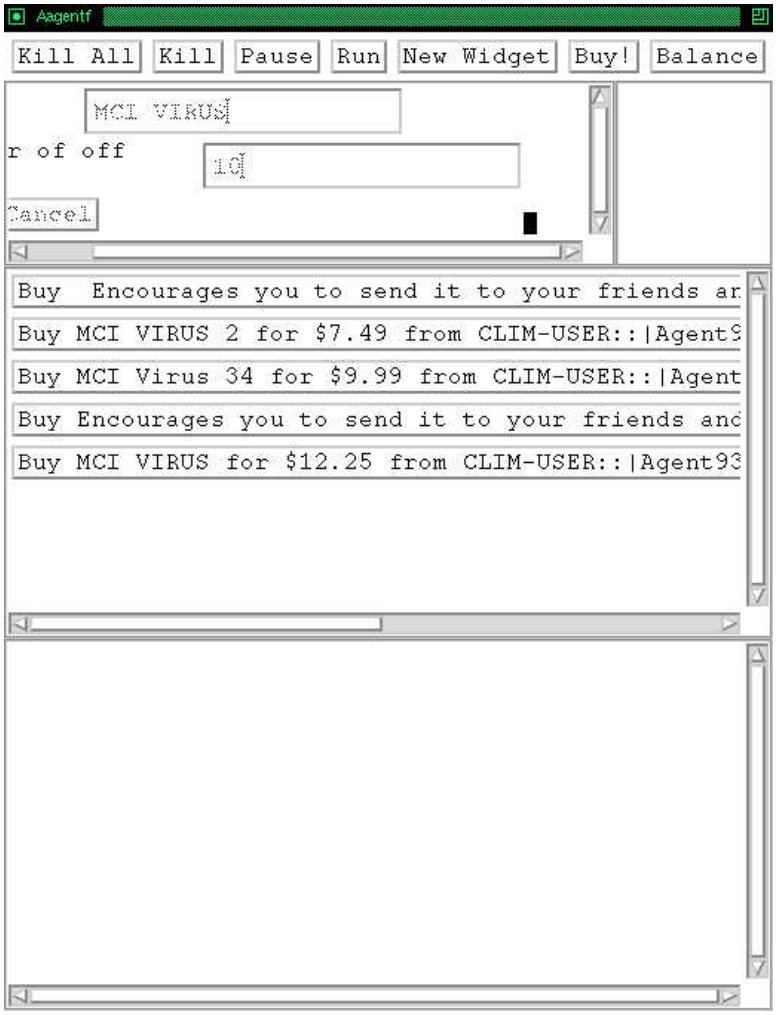

Figure 3: Buyer agent after receiving multiple bids

The agent process sends an accept offer to the seller agent:

```
From: |Agent1088| To: |Agent1059|
Message: (OFFER
  CLIM:ACCEPT
  #S(OFFER
     :WIDGET "MCI VIRUS"
     :WIDGET-DESCRIPTION "MCI Virus description"
     :PRICE 10.0
     :TYPE T
     :ACTION INIT
     :ID 63692)
  10.0)
```

The seller takes the widget out of its inventory and sends it to the register:

```
Sent to register by |Agent1059| and stored at 73693:
```

```
#S(WIDGET :NAME "MCI VIRUS"
  :CONTENTS
  "Encourages you to send it to your friends and family."
  :KEY 50344)
```

Immediately, the register writes a check and sends it to the seller, together with a key for the widget retrieval. The seller deposits the check upon receipt and sends the key to the buyer:

```
Bank: Check written from REGISTER to |Agent1059| for 10.00
Bank: Check deposited by |Agent1059|

From: |Agent1059| To: |Agent1088|
Message: (KEY
  73693
  10.0)
```

The buyer receives the message and sends a check and the widget key to the register. The register responds by sending the widget to the buyer and depositing the check.

```
Bank: Check written from |Agent1088| to REGISTER for 10.00

Taken from register by |Agent1088| from 73693

Bank: Check deposited by REGISTER
```

After the widget is received, it is added to the agent's inventory and the cycle is ready to start again:

```
Agent1088:Received a widget "MCI VIRUS"
```

6 Directions for Future Work

At this time, many problems associated with using MAGMA in the real world remain unsolved. Some of them will be addressed in the second version of MAGMA currently under development. The system will be written in Java and C++ and built as an open API, allowing third-party developers to build their own agents or interface with legacy systems in any available language that supports linking of C libraries and socket communication.

- **Networking.** The first and foremost problem that needs to be solved is network communications for agents. In MAGMA 2.0 we will use sockets to make connections between the agents running within a Web browser and messaging hubs running on the server.

- **Security.** Even though current version of MAGMA incorporates simple encryption and authentication, any commercial implementation must include more and stronger security features. MAGMA 2.0 will include RSA public-key encryption in the low-level communications layer, ensuring that no messages are passed unencrypted. The most important messages such as money transfers will be double-encrypted and will include checksums and authentication codes to ensure integrity.

- **Interface to Existing Systems** has to be designed in a way that will not require major changes in the legacy software used by financial institutions and shipping companies. Due to a large installed base of SQL-compatible databases in the financial and other business sectors, as well availability of products that interface SQL engines with older mainframe-based system, a logical choice for such interface would be a translator between the agent messaging language used in MAGMA and an industry standard query language such as SQL. Even though MAGMA 2.0 is not likely to include SQL interface, it will be possible to use agent API routines in conjunction with an existing SQL-compatible product. When JDBC (Java DataBase Connectivity) API is released, the connection will become even more natural.

- **Intelligent Negotiation and Decision Algorithms** are extremely important if the system is to gain acceptance from major retail companies. If they are effective, companies can conduct their Internet sales automatically or with a minimal human input, allowing them to lower costs to justify investment in a new system. In MAGMA 2.0 we will implement and test several strategies and algorithms, including some of the ones described earlier in the paper.

- **Business Prospects.** Even though MAGMA is an academic project, we will attempt to implement the second version using commercial-grade techniques and test it in the real world. Some of the smaller-scale applications of MAGMA might include service like used book, CD-ROM or computer parts exchange, or a listing service such as resume bank or a personal ad system.

  If these tests prove successful, and after problems found during these tests are remedied, MAGMA will be ready to be released into the second stage of testing with small business applications.

References

[1] Y. Amihud. *Bidding and Auctioning for Procurement and Allocation.* New York University Press, New York, 1978.

[2] J. Case. *Economics and the Competititive Process - Studies in Game Theory and Mathematical Economics.* New York University Press, New York, 1979.

[3] Anthony Chavez and Pattie Maes. Kasbah: and agent marketplace for buying and selling goods. In *Proc. PAAM96*, page to appear, 1996.

[4] Robert Doorenbos, Oren Etzioni, and Daniel Weld. A scalable comparison-shopping agent for the World Wide Web. Technical Report UW-CSE-96-01-03, University of Washington, 1996.

[5] B. Krulwich. Bargain finder agent prototype. Technical report, Anderson Consulting, `http://bf.cstar.ac.com/bf/`, 1995.

[6] Jeffrey Rosenschein and Gilad Zlotkin. Designing conventions for automated negotiation. *AI Magazine*, pages 29–46, Fall 1994.

[7] T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *First International Conference on Multiagent Systems (ICMAS-95)*, pages 328–335, San Francisco, 1995.

[8] Ying Sun and Daniel Weld. Automating bargaining agents (preliminary results). Technical Report UW-CSE-95-01-04, University of Washington, 1995.