Heterogeneous Agent Development: A Multi-Agent System for Testing Stock Trading Algorithms

Stephen Russell
Information Systems Department
University of Maryland Baltimore County
Baltimore, MD
stephen.russell@umbc.edu

Victoria Yoon
Information Systems Department
University of Maryland Baltimore County
Baltimore, MD
yoon@umbc.edu

ABSTRACT

Intelligent agents have often been used as a method for simulating an active equity market environment. While agents have been used extensively in trading and market simulations, agents have not been used in a system that only evaluates trading algorithms. To accomplish the simulations, agents are developed in a single or proprietary programming language. The use of agents developed in Microsoft's .Net framework and CLR provides flexibility, scalability, compatibility, and interoperability beyond traditional agent development environments. This paper presents a multi-agent system developed using native JAVA, VB.Net, C# and PHP, all in the .Net environment. The system will demonstrate its abilities by comparing two equity trading algorithms.

KEYWORDS: Multi-agent systems, Simulation, Architecture, JADE, Microsoft .Net

INTRODUCTION

Intelligent agents, learning algorithms and genetic programming have been applied extensively to the problem of simulating and predicting stock market trading. Many researchers have developed algorithms and methods that attempt to identify patterns or predict movements in the market. These algorithms are frequently tested in simulated environments. As a result, a significant amount of research has been conducted to create realistic simulations of the actual stock market. Relative to intelligent agents, prior work has emphasized the use of agents as a method to support trading functions or market environment simulation, not to solely support the evaluation of trading policies or rules.

Historically the intelligent agents used in financial systems of this nature, are developed using a single programming language such as JAVA. This work examines the development of agents using multiple programming languages natively. The developed architecture employs intelligent agents to retrieve and manage stock data and high volume stocks as defined by a selected source, which is itself an agent. Agents are used in this research to enable scalability and flexibility in the architecture's design. In this context, the proposed system employs agents to deliver the classical benefits of agent systems. By using agents in this manner, the system has enhanced modularity and capability in stock selection. The innovation in this work is twofold. One, a multi-agent system will be developed to evaluate stock trading algorithms. Two, the system will use 4 different programming languages in a single development environment to implement the multi-agent system.

The objective of this research is to determine whether heterogeneous programming languages can be effectively used in a single development environment to create a multi-agent system. In this case, the expectation is that agents can be developed using tools that best fit the goal and behaviors of the individual agent. The research evaluates a system architecture in which various equity trading algorithms can be tested; deliberately applying intelligent agents to non-simulation functions. As an initial study, Bill William's Awesome Oscillator (AO) algorithm was compared to a time-based trade as the basis for validating the system.

BACKGROUND

The evolution of agents in financial applications has expanded rapidly in the past 10 years. These applications have included monitoring applications, simulations and most recently, decision support. Most of the emphasis in agent research in stock market applications has been in simulating market environments. These simulations model both traders and the methods they employ. In most simulated markets, traders fall into two basic type: those who follow the fundamentals suggested by the CAPM model and those who follow technical trading rules such as buy if the price is above its 50 day moving average and

sell if it is below (Howard, 1999). Many financial organizations make significant profits by developing and trading using these rules.

There has been a recent trend in rule-based trading that supports a belief that volume precedes price. In other terms, a stock's momentum is an indicator of its future trend. This supposition has lead to a variety of studies and approaches that expand and build on this theory. Bill Williams has developed several models and built a business exploiting these purported market opportunities (Williams, 2003). One method created by Williams, uses an oscillator to determine a stock's momentum. The output of the oscillator creates an indicator (signal) of a stock's momentum and subsequently a stock's future value. The oscillator used by Williams was originally developed by Tom Joseph to try to track Elliot wave. Williams adopted and modified the algorithm and subsequently named it Williams Awesome Oscillator (AO). Williams' AO utilizes moving averages combined with trading rules, based on pre-defined periods or bars. A description of the AO algorithm can be seen in Appendix A.

As the number of rule-based trading approaches increase, so does the importance of evaluating these algorithms. A multi-agent system (MAS) approach is a natural fit for evaluating stock algorithms because the tasks required to complete this process are diverse and many of the activities are concurrent. Research applying agent technologies to financial applications has been extensive. However none of these efforts have focused solely on evaluating trading algorithms/rules. Prior agent research has either combined the applications of these rules with simulations or applied a suite of rules to decision support. Additionally, the MAS research prototypes and systems developed in earlier work were developed in a single programming environment such as JAVA, or where the interoperability of the agents was outside the development environment, using agent communication languages (ACLs). Our approach extends the current methods by using Microsoft's .Net framework to develop a multi-agent system for trading algorithm evaluation where agents, developed using heterogeneous programming languages natively, are integrated into a single system.

PRIOR RESEARCH

Previous agent research in stock trading emphasizes trading simulations/environments or simply proving a particular algorithm or approach is better than others. Approaches using agent architectures have employed the agents to simulate trading interactions, facilitate communication between components, or provide decision support (Luo & Lui 2002; Davis et al, 2002; Ajenstat, 2004; Mistry, 2003). For example, Mistry's work describes a simulation framework that employed intelligent agents as entities in a market, such as traders, market makers, and research analysts. The goal of Mistry's research was to develop a market simulation framework that could be employed by economists to simulate market activity. Unlike the system developed in this research, Mistry's work is designed to simulate market chaos and not test a specific approach or algorithm. Much of the prior research in trading systems emphasizes intelligent agents as a framework for simulating the market environment, as compared to the basis for a system to isolate and test multiple trading algorithms.

Agents have been utilized as software architecture in other research, notably in Genesereth and Ketchpel's (1994) work. This research has touted intelligent agents as a useful software engineering architecture that can easily address the difficulties created by the intrinsically heterogeneous nature of software programs. The research in this paper builds on Genesereth and Ketchpel's theory by demonstrating and application of agents in stock selection and information retrieval for defined simulation functions. In addition to proposing the use of agents in the context of a system whose sole function is to evaluate stock algorithms, which is novel, our research extends Genesereth and Ketchpel's work by demonstrating that the development of agent systems using the Microsoft .Net environment provides significant interoperability (at a development level) and enhancements in programming agent functionality. The use of .Net provides the ability to integrate multiple programming languages into a single development environment; allowing heterogeneous agents and functionality to be integrated into a homogeneous development platform. .Net has been shown to be a highly effective multi-language platform allowing the combination heterogeneous networks of workstations and multiprocessors as a unique metacomputing system for implementing concurrent objects (Nebro et al., 2002).

Our research does not seek to answer the debate over the benefits of .Net versus Java or other languages. Nor does it seek to evaluate the performance of .Net languages compared to others. The purpose of this research is to evaluate the viability of .Net to provide an environment that is consistent with the objectives of agent development without rigidly defining the development platform. A non-comprehensive, but lengthy list of .Net interoperable languages can be seen at http://www.dotnetpowered.com/languages.aspx. This site lists over 30 interoperable languages including JAVA, PHP, LISP, Perl, Python, and Fortran. The .Net platform delivers a significant benefit to MAS development by delivering a language agnostic platform. Most of the existing agent implementations require that users implement their technology using JAVA, other proprietary technologies, or a single programming language. The benefits of the .Net common language runtime (CLR) and the common intermediate language (CIL) have been detailed in several research and industry forums (Bres et al. 2004; Syme, 2001; Nebro et al., 2002), but the implications of this environment for MAS development has not been explored.

The remainder of this paper will discuss two main objectives. The first objective is to develop a multi-agent architecture for testing trading algorithms using Microsoft .Net framework. This architecture implements heterogeneous multi-agent software in a single homogeneous platform. The second objective is to develop the prototype system and test by determining whether the AO will perform better than a time-based algorithm using the agent-based architecture.

AGENT-BASED ARCHITECTURE

To restrict the scope of the research, the agents are limited to a single system function: stock sample selection. Stock selection comprises 3 activities: stock symbol source location, stock selection, and symbol data collection. The current agent architecture implements very little learning capabilities. However it would be relatively simple to enhance the selection processing or add ontologies for the various selection sources within these components. Because the objective of this research was to demonstrate the feasibility of language agnostic agent development, the agent components were deliberately kept simple and the functionality limited to well-studied goals and behaviors. Figure 1 describes the agent architecture implemented in the system. The inclusion of the agent architecture provides proactive and reactive capabilities for distributed data cleansing and collection applications that may be necessary for more complicated algorithm testing.

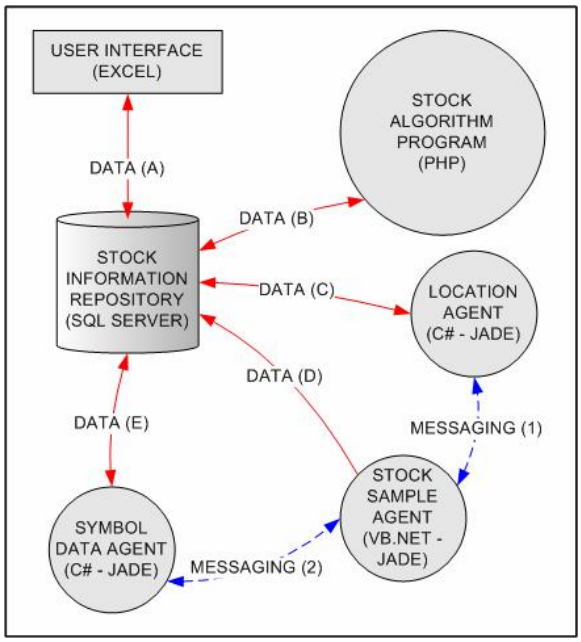


Figure 1. Agent architecture

The system utilizes the agents to identify and retrieve stock ticker information. This gives the architecture the flexibility to add intelligence regarding which stocks and from where sample stocks are selected. Notifications flow between the agents to announce activity status. These notification messages are extensible markup language (XML) based and trigger activities of the other agents. The agents communicate each other using the system Agent Communication Language (ACL), which is a combination of Foundation for Intelligent Physical Agents (FIPA) ACL (Foundation for Intelligent Physical Agents, 2003) and XML. The details of the system ACL are beyond the scope of this paper.

The location agent is responsible for identifying sources where sample stock symbols could be obtained. The location agent performs the function of identifying and locating high volume stock symbols to be used for the algorithm testing. Again, to limit the scope of this research, the input for this agent was given specifically as the Investors Business Daily (IBD) site. The implementation of an agent in this role allows greater "intelligence" in selecting the stock samples as part of future work.

The stock sample agent is responsible for retrieving the sample symbols, parsing the retrieved file, and adding the information to the database. The stock sample agent manages the retrieval and parsing of the stock symbols as given by the location agent. This agent waits for an ACL message (1) from the location agent that includes the whereabouts of the symbol data. This location instruction directs the stock sample agent to fetch and parse the content for high volume symbols. The stock sample agent provides the capability for smart parsing of stock data and is the repository for various symbol data ontologies.

The symbol data agent retrieves the stock symbol data file, parses the file and adds the information to the database. The symbol data agent collects open, close, high, low and current price at a particular bar interval for each symbol. This agent receives ACL messages (2) from the stock sample agent to know what symbol data to collect. For this experiment the symbol data agent sent an e-mail notifying a remote user which symbols were needed, and then monitored an FTP site until the files containing this data, were uploaded to this site. Once the files were found, the agent downloaded the files, parsed them and inserted the information in the system database. The agent's acquisition methods can easily be modified to handle collection from a variety of sources as required, without modification of any of the other system or agent components.

The stock algorithm is a "black box" object that handles the algorithm implementation and stock trading activities. The stock algorithm program was not designed with a multi-agent architecture because prior research has implemented many different agent-based simulations and this research did not have that emphasis. While the stock algorithm program is a monolithic object-oriented program in this architecture, it is possible to extend this component to include agents for the functions it implements as well. Further detail on the implementation of this component can be found in the implementation section of the paper.

Data in the architecture is stored in the stock information repository. The data model for the repository can be seen in Appendix B. Users of the system enter data (A) through an Excel interface directly into the database; future work would expand this interface and should include an evaluation or analysis agent in this role. The data entered by the user sets up the run/evaluation parameters. We will describe this in greater detail in the experimentation section. The stock algorithm uses this data (B) as parameters for each run and returns the results to the database. The location agent retrieves information from the database regarding where to obtain sample stock symbols (C); future implementation would include locations that the agent discovered. The sample stock agent stores the stock list data (D) it has collected from the symbol location before notifying the symbol data agent, who retrieves the list a fetches the symbol data, storing its findings in the repository.

AGENT ARCHITECTURE IMPLEMENTATION

The agents in the architecture are implemented using a .Net port of the JAVA Agent Development (JADE) framework to make the system capable of interacting with other FIPA compliant interfaces and agents. The.Net framework allows future extensibility to web services, as well as demonstrates the potential for distributed processing applications in a business environment where Microsoft Windows PCs are common. .Net also allows the system to take advantage of the .Net framework abstractions for compatibility with multiple database platforms, XML, regular expressions and internet-based methods such as web services, email, and http. Additionally the robust library of functions and managed code features of .Net allowed the development to emphasize programming the implementation of the JADE framework agents.

The .Net port of JADE was compiled using the .Net J# into a dynamic link library (dll). The location and symbol data agents' implementations were written in C# using the JADE dll. To add a third programming language to the project, the stock sample agent was written in native VB.Net and compiled into a dll. The VB.Net implementation also used the JADE dll. 1300 lines of code were written using native PHP to create the stock algorithm program as shown in figure 1. The stock algorithm program is comprised of several objects: algorithm objects (AO and time-based), a run profile object -- responsible processing the algorithms, and a results object -- responsible for managing the algorithm results. The choice of PHP was made to further emphasize the heterogeneous nature of system. The PHP code was compiled into a .Net compatible dll using Phlanger (Benda & Matoušek, 2005). Each component was coded in its native programming language using the visual studio development environment, integrated into a single .Net solution, and compiled as a single system.

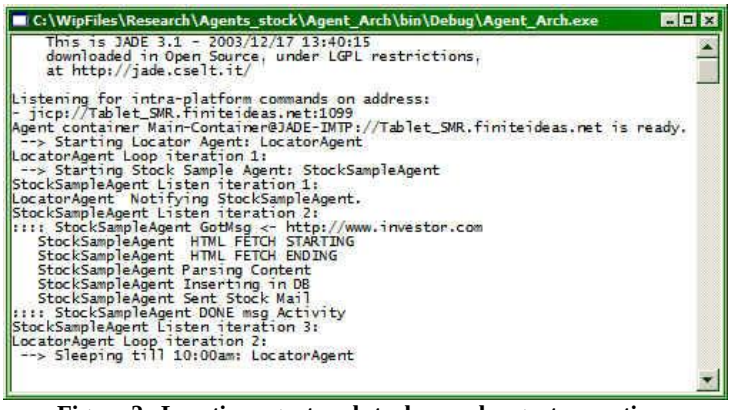


Figure 2. Location agent and stock sample agent operation

AGENT OPERATION

During normal operations all of the agents ran inside a single JADE container, on a single computer. However, the JADE framework and .Net are cross platform and the agents could be allocated across several computer systems, distributing the processing load. The execution of the agents is parallel by definition, as JADE intrinsically provides this capability.

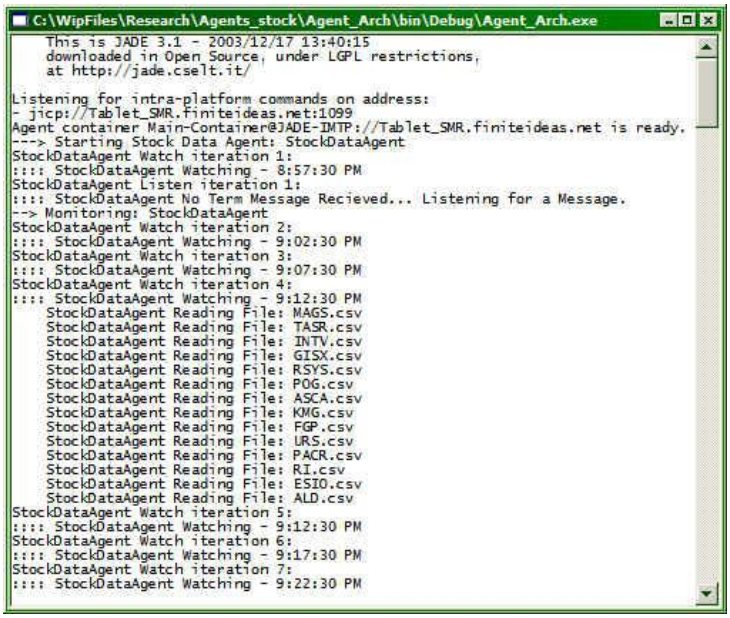


Figure 3. Symbol data agent operation

Figure 2 above, shows the operation of the location agent and the stock sample agent. Examining the operation, the agents start in parallel, with the stock sample agent listening for a message from the location agent. The location agent sends the stock sample agent a message with the location of the stock pool. Once the stock sample agent receives this message it begins to download the page, parsing the results, and extracting the stock samples for insertion into the database. Once it has completed that task, it sends an email to a predefined address notifying a user to retrieve the pricing bar data. It then returns to a listening state to await another request. The location agent runs on a schedule and wakes up to run at 10:00am.

Figure 3 shows the operation of the symbol data agent. The operation of this agent is separated here for discussion purposes only. In the experiment, the symbol data agent ran in the same container as the other agents and its output was shown in the same window. The symbol data agent polls an FTP site waiting for stock price data files to be uploaded by the user notified by the stock sample agent. This agent polls the location every 5 minutes until files are seen, or a termination message is received. Once the files are seen they are parsed and inserted into the database where the stock algorithm program can utilize them.

AO VERSUS TIME-BASED TRADE EXPERIMENT

Each day Investors Business Daily (IBD) posts high volume stocks for a given period. The system used the stock sample agent to fetch IBD's list of high-volume stocks at 10:00am, after being notified by the location agent. IBD posts five upward trending (increasing in price) stocks and five downward trending stocks on its home page (www.investors.com). The selection of samples from IBD was chosen because it provided securities that have been "pre-identified" as volume movers.

The stock sample agent was instructed to download and parse the data, off the market open time of 9:00am, to eliminate market manipulations from the previous day. Data was collected for 1-2 days, every other week, for a total of 5 days. This sampling method was employed to get a broader range of samples over time, increasing the exposure to normal market entropy.

The system was designed to perform the following operations to compare the AO versus the time-based algorithms. These functions were coded in the PHP stock algorithm program.

1. Use high volume stocks identified by IBD as the selected trading pool
2. Make a single trade on these stocks (buy long or sell short). *X* number of "round-trip" trades per day
3. Use the AO algorithm and a time-based trade as "what" transaction to make "when"
4. Each trade is conducted for each algorithm: AO and time-based
5. Allocate a pool of money to each algorithm and allow the algorithm to go into debt
6. Use the same amount of money per trade for each algorithm

Example Time-Based Run

1. Make entry (long or short) at 10:00am
    a. Only take long position for stocks that are positive (price moving up)
    b. Only take short position on stocks that are negative (price moving down)
2. If stop loss percentage is reached at any point during the day then position is exited at market rate, at that time
3. If position is still held at 3:00pm, exit

Example AO Run

1. No trades made until after Oscillator Algorithm generates signal to make transaction
    a. Only take long positions for stocks that are positive (price moving up)
    b. Only take short positions on stocks that are negative (price moving down)
2. Use AO signal to enter and exit trades – use 1 minute bars
3. If stop loss percentage is reached at any point then position is exited, at market rate, at that time
4. If position is still held at 3:00pm, exit

A user enters the following data through the Excel interface as parameters for the runs. For both of the AO and time-based algorithms, runs were made with the trading day limited between 10:00am and 3:00pm. No commission (0%) was applied, with a 5% stop loss. Each trade was limited to $1,000.00 with 1 round-trip trade per day. Symbol, bardate, and pricedirection are all set by the high volume IBD trading pool.

Descriptive Statistics

	N	Range	Minimum	Maximum	Sum	Mean	Std.	Variance	Skewness		Kurtosis	
	Statistic	Statistic	Statistic	Statistic	Statistic	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Std. Error
AORev	52	54.81	-40.00	14.81	-36.25	-.6971	8.84129	78.168	-1.952	.330	7.029	.650
TimeRev	52	377.690	-129.150	248.540	12.244	.23546	67.349158	4535.909	2.413	.330	7.442	.650
Valid N (listwise)	52											

Paired Samples Correlations

		N	Correlation	Sig.
Pair 1	AORev & TimeRev	52	.281	.044

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	97.5% Confidence Interval of the Difference Lower	Upper			
Pair 1	AORev - TimeRev	-.932562	65.419584	9.072064	-21.8846	20.019476	-.103	51	.919

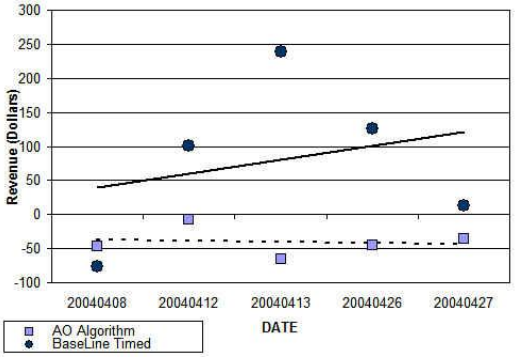
Table 1, statistics for total revenue & paired t-test

DATA ANALYSIS

The results of the runs were summarized using Microsoft Access. Basic descriptive statistics were used to determine which algorithm performed better. Additional analysis was conducted using SPSS to examine the resulting distributions and apply paired t-tests for hypothesis testing.

The analysis of the MAS was done by evaluating how well it performed the collection functions and how well the system isolated and controlled the testing variables. The data from the processing runs was used to evaluate whether the system was successful in satisfying these requirements. If the system was successful, the data from both samples would be from similar distributions and have similar characteristics, exclusive of how the stock trade performed.

A .025 level of significance (97.5% confidence interval) was used to determine the validity of the system and compare the differences between the two algorithms. This increased the type II error in this experiment. However the type II error can be reduced in future experiments, as additional samples are gathered using the validated system. To evaluate the overall performance of the architecture, a t-test was used to examine whether the system kept variables constant between the two algorithm samples. The results of the data analysis can be seen in table 1, above.



Run Data

Date	AO	Timed
20040408	-46.33	-76.53
20040412	38.93	177.43
20040413	-58.14	138.25
20040426	20.73	-113.80
20040427	8.55	-113.10
TOTAL	**-36.25**	**12.24**

Figure 5 Daily trade revenue

RESULTS

Given the results obtained by the system, it is clear that the heterogeneous code was integrated into a single multi agent system and the MAS architecture performed as it was intended. Figure 5 shows the results of stock evaluation runs. The AO algorithm did not outperform the time-based algorithm. The time-based algorithm was able to take advantage of volume momentum while the AO algorithm was subject to price fluctuations during the day. On some days the AO outperformed the time-based algorithm and on others it did not. Despite this, the data trends and statistical tests confirm that there is a difference between the two samples and that the time algorithm performed better overall compared to the AO algorithm. While the results appear to indicate better performance from the time algorithm, additional data/runs are necessary to make a definitive conclusion.

CONCLUSIONS AND FUTURE DIRECTION

In this paper a MAS for testing trading algorithms is proposed and investigated. The system was evaluated by examining whether it effectively implemented agents, developed using heterogeneous programming languages, controlled trading variables, and provided a platform for the evaluation of algorithms' performance. The contributions of this research can be divided into two main parts. The first part is demonstrating that .Net can be an effective tool for developing heterogeneous multi-agent systems using their native programming languages. The second part is the development of a multi-agent system that could be used to effectively isolate and test trading algorithms, controlling the trader's variables while minimizing the impact of external variables such as market prices and economic conditions.

The validation of these objectives is purely empirical. However if this activity was not successful, it would not have been possible to perform the experiment comparing the AO and the time-based trading approaches. While the agent functionality is limited in this implementation, this research clearly demonstrates that heterogeneous agents can be developed and implemented in their native language using Microsoft's .Net framework. In this system, agents were implemented in a single environment using 4 programming languages: JAVA, C#, PHP, and VB. The implication of this research on intelligent agent development represents a significant shift in the integration of diverse agent architectures. Further the native features of the .Net platform (such as tight integration with XML, HTTP, and web services) provide additional benefits. In the extreme, our approach demonstrates agents can be developed in the language that provides the most appropriate capabilities and integrated in a common platform.

The system was able to determine which algorithm performed better: AO or time-based. The time-based algorithm had higher yields compared to the AO. This may be caused by the fact that the tests used volume trending stocks and the oscillating algorithm was not allowed to oscillate, therefore losing any potential performance advantage. The use of judgment sampling may have affected the outcome of the algorithm assessment. This is because the sample was heavily dependent on a single source of secondary data. By adding additional learning capabilities to the intelligent agent components of the system the impact of judgment sampling can effectively be neutralized. The current collection methods of data used for processing also did not facilitate real time trading. The timing of the sample collection may introduce or increase errors caused by some of the uncontrolled variables such as economic conditions. Again, the agent architecture could be enhanced to address both of these issues.

Significant work can be done to expand the intelligence and scope of the agent components in the system. For example, in the current implementation the location agent follows a single instruction. A future enhancement would make the location agent smarter and more active so that it actually searches for sources of sample stocks. Additionally, because many stock and financial algorithms require extensive processing, the system should be tested using the agents not only for retrieval but also for distributed processing. Using agents in the algorithm processing component of the system may create a suitable platform for distributed processing. It would also be interesting to see how the architecture integrated with other agent-based simulation platforms.

REFERENCES

1. Ajenstat, J. (2004). Virtual Decision Maker for Stock Market Trading as a Network of Cooperating Autonomous Intelligent Agents. *Proceedings of the 37th Hawaii International Conference on System Sciences* - 2004
2. Alemany, M. B., (2004). Jade over .NET [Online]. Available: http://casal.upc.es/%7Emarti23/JadeOverNET.html [20 June 2004]
3. Bauer. R. J. (1999) *Technical Market Indicators: Analysis and Performance*, John Wiley & Sons
4. Benda, J., and Matoušek, T. (2005). Phalanger the PHP Language Compiler for .NET Framework [Online]. Available: http://www.php-compiler.net/ [2005, March 26]
5. Bengtsson, M. and Ekman, M. *Adaptive Rule Based Trading. A Test of the Efficient Market Hypothesis* [Online]. Chalmers University of Technology, Available: http://www.ce.chalmers.se/staff/mekman/MasterThesisEconomics.pdf [2004, April 20].
6. Bres, Y., Serpett, B., and Serrano, M. (2004). Bigloo.NET: Compiling Scheme to .NET CLR. *Journal of Object Technology*, 3, No. 9, October 2004, Special issue: .NET Technologies 2004 workshop, pp. 71–94, <http://www.jot.fm/issues/issue 2004 10/article4>
7. Colby, R. W. and Meyers, T. A. (1988) *The Encyclopedia of Technical Market Indicators*, McGraw-Hill Trade
8. Conway, M. R. (2002) *Professional Stock Trading: System Design and Automation*, Acme Trader
9. Davis, D. N., Luo Y., and Lui, K. (2002). Combining KADS with ZEUS to Develop a Multi-Agent E-Commerce
10. Application. *Journal of E-Commerce Research* 2002
11. Finin, T. and Wiederhold, G. (1991). An Overview of KQML: A Knowledge Query and Manipulation Language, available through the Stanford University Computer Science Department
12. Foundation for Intelligent Physical Agents, (2003). Welcome to the Foundation for Intelligent Physical Agents [Online]. Available: http://www.fipa.org/ [2004 June 10]
13. Genesereth, M. and Ketchpel, S., (1994). *Software Agents*, Communications of the ACM, 37, No. 7, pages 48-53, (1994)
14. Howard, M. (1999). "The Evolution of Trading Rules in an Artificial Stock Market," *Computing in Economics and Finance* 1999 712, Society for Computational Economics.
15. Joseph, T. (2004). eSignal Central: The Learning Center [Online]. Available: http://www.esignalcentral.com/learning/likepro/tjoseph/default.asp [2004, April 20]
16. Kaufman, P. J. (1998) *Trading Systems and Methods*, John Wiley & Sons; 3rd edition
17. Labrou, Y., Finin, T., and Peng, Y. (1999). The Interoperability Problem: Bringing together Mobile Agents and Agent Communication Languages. *Proceedings of the Hawaii International Conference On System Sciences*, January 5-8, 1999, Maui, Hawaii
18. Link, M. (2003) *High Probability Trading*, McGraw-Hill Trade
19. Luo, Y. and Lui, K. (2002). A Multi-Agent Decision Support System for Stock Trading. *IEEE Network*, January/February 2002
20. Mistry, A. (2003) *Studying Financial Market Behavior with an Agent-Based Simulation* [Online] Cornell University. Avaliable: http://www.cs.cornell.edu/boom/2003sp/ProjectArch/Agent-Based%20MarSim/ [2004, April 20].
21. Nebro, A.J., Alba, E., Luna, F. and Troya, J.M. (2002). *.NET as a Platform for Implementing Concurrent Objects*. Monien and Feldman, EURO-PAR 2002, pp. 125–130
22. Pasternak, M. (2004). Technical Analysis Education [Online]. Available: http://www.streetauthority.com/terms/onbalancevolume.asp [2004, April 16]
23. Pretcher, R. (2001). *Elliott Wave Principle: Key to Market Behavior*, John Wiley & Sons; 10th edition, pp. 34

24. Syme, D. (2001). ILX: Extending the .NET Common IL for Functional Language Interoperability. *Proceedings of Babel'01*, 2001.
25. Taylor, S. (2004) Technical Analysis 101 - Rate Of Change (ROC) [Online] Available: http://www.investopedia.com/articles/technical/100801.asp [2004, April 30].
26. Telecom Italia Lab (2004 – Copyright). Jade - Java Agent DEvelopment Framework [Online]. Available: http://jade.cselt.it/ [2004, June 18]
27. Williams, B. M. (2003). ProfitUnity Trading Group [Online]. Available: http://www.profitunity.com/ [2004, April 16]

APPENDIX A - WILLIAMS AWESOME OSCILLATOR

AO makes trades based on market momentum

- The basis for this is that stock in motion tend to stay in motion, in the direction that they are going
- Conceptually momentum indicates the price trend

A Bar contains open close high and low for a given period (1 minute to one day)

- 5-bar moving average of midpoints subtracted from 34-bar moving average of midpoints = market momentum
- Midpoints are defined by $\frac{H - L}{2}$ where H ≡ high of period (bar) and L ≡ low of period (bar)

General Oscillator: n period simple moving average is: $\frac{\sum_{i=1}^{n} P_{t-i+1}}{n} = Average(n,t)$

Where *n* is the number of periods, P is price at time (*t*)

AO = Average (5,t) – Average (34,t)

- If AO is positive and momentum is positive enter or hold, if momentum is negative exit or hold
- If AO is negative and momentum is negative enter or hold, if momentum is positive exit or hold
- If AO is 0 hold

APPENDIX B – STOCK INFORMATION RESPOSITORY DATA MODEL