An Agent Task Force for Stock Trading

Rui Pedro Barbosa and Orlando Belo

Abstract. In this article the authors present the simulated trading results of a system consisting of 60 intelligent agents, each being responsible for day trading a stock listed on the NYSE or the NASDAQ stock exchange. These agents were implemented according to an architecture that was previously applied to currency trading with interesting results. The performance of the stock trading agents, once integrated in a diversified investment system, showed similar promise. The trading simulation was done using out-of-sample price data for the period between February of 2006 and October of 2010. Throughout this period, the system's performance compared favorably with that of the buy-and-hold strategy, both in terms of return and maximum drawdown. These results indicate that agent technology might be of use for this particular practical application, a conclusion that should interest the investment industry.

Keywords: Intelligent Agent, Stock Trading, Financial Data Mining.

1 Introduction

Financial trading seems like the perfect setting for applying agent technology [1][2]. Most investment analysis, be it technical or fundamental, is nothing more than number crushing, data mining and pattern matching, and these are all tasks at which software agents should be able to excel. This assumption was partially vindicated by the research deriving from the Penn-Lehman Automated Trading Project [3]. This project showed interesting results in what regards to automated stock trading, but for the most part the agents created for this project's trading competitions were more like simple trading bots rather than actual intelligent agents. We believe that, in order to be truly "intelligent", a trading agent must meet at least the following requirements: it should be able to decide when to buy, short sell or close open trades of a given financial instrument; it should be able to perform money and risk management; it should be able to keep learning over time, even as it trades; it should be capable of adapting to changes in market conditions; and finally, it should be smart enough to stop trading when the market becomes less predictable, and to resume trading once the conditions improve. An intelligent agent [4] with these capabilities should be well prepared to trade autonomously and unassisted for an indefinite period of time, hence completely replacing a human trader. In a previous article [5], we proposed an agent architecture that might allow for the creation of trading agents that should more or less meet the

Rui Pedro Barbosa · Orlando Belo
Department of Informatics, University of Minho, Portugal
e-mail: {rui.barbosa,obelo}@di.uminho.pt

Y. Demazeau et al. (Eds.): Adv. on Prac. Appl. of Agents and Mult. Sys., AISC 88, pp. 287–297.
springerlink.com

aforementioned requirements. This architecture was applied in the development of a multi-agent currency trading system [6] that showed promising results in a trading simulation encompassing a period of around 2 years. In this article we will describe the use of this same architecture (with a few improvements) in the development of a diversified trading system consisting of 60 stock trading agents, each of which will be trained and configured to autonomously day trade a stock listed on the NYSE or the NASDAQ. Before presenting the actual details of the agent implementation, we will start by describing the proposed agent architecture. The article will end with the reporting of the results obtained by the agent-based trading system in an out-of-sample period comprising around 5 years' worth of data. The system's performance will be compared with that of a simple buy-and-hold strategy, both in terms of return and maximum drawdown. This strategy will be used as the benchmark for our results.

2 The Trading Agent Architecture

As previously mentioned, there are several requirements that a software trading agent needs to meet before it can actually be considered "intelligent". If these requirements are fulfilled, the agent might be able to trade successfully in the long run. The architecture we proposed in the past (figure 1) [5], should allow for the development of this type of agents. It consists of three modules: 1) the prediction module, responsible for forecasting the direction of a stock's price throughout the next trading day (this module is implemented using an ensemble of data mining models); 2) the empirical knowledge module, responsible for deciding how much to invest in each trade (this module is implemented using a case-based reasoning system); and 3) the domain knowledge module, responsible for making the final trading decisions (this module is implemented using an expert system).

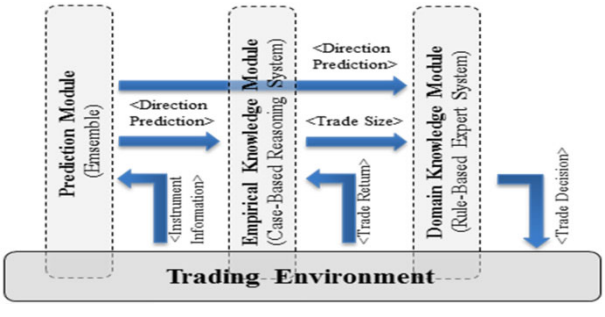

Fig. 1 The trading agent architecture.

The prediction module is responsible for deciding if a stock should be bought or short sold, based on the predictions of the data mining models in the ensemble. Each model tries to predict if the stock price will increase or decrease throughout the next trading day (from open to close). Once this is done, the models' predictions are aggregated into a single forecast, which the agent utilizes to pick the direction of the trade: if this forecast dictates that the stock price will increase, then the module's decision is to buy the stock when the market opens, and sell it when it closes; if, on the

other hand, it dictates that the price will decrease, the module's decision is to short sell the stock at the open and cover at the close. The algorithm behind the decisions of the prediction module is shown in figure 2. Before each forecast, this component splits the available data into two datasets: the test set, with the most recent n instances, and the training set, with all the rest. The training set is used to retrain each data mining model, after which the retrained models make direction forecasts for the instances in the test set, and trades are simulated accordingly. The results of this trading simulation are subsequently utilized to calculate the profit factors of each model:

$$Overall\ PF = \frac{\sum returns\ of\ profitable\ trades}{\sum |returns\ of\ unprofitable\ trades|} - 1 \quad (1)$$

$$Long\ PF = \frac{\sum returns\ of\ profitable\ long\ trades}{\sum |returns\ of\ unprofitable\ long\ trades|} - 1 \quad (2)$$

$$Short\ PF = \frac{\sum returns\ of\ profitable\ short\ trades}{\sum |returns\ of\ unprofitable\ short\ trades|} - 1 \quad (3)$$

The overall profit factors are utilized to decide if a retrained model should become a part of the ensemble, or if the version prior to retraining should be kept, by choosing the one with the highest profitability. The long and short profit factors are used as the model's vote weights: if the model predicts a price increase, the weight of its vote is its long profit factor, and if it predicts a price decrease, the weight of its vote is its short profit factor. A negative weight is replaced with zero, meaning that the model's prediction is ignored when the models' votes are aggregated to calculate the forecast of the ensemble. The votes' aggregation is accomplished by adding the weights of the votes of the models that predict a price increase, and subtracting the weights of the votes of the models that predict a price decrease. If the resulting value is greater than zero, the final ensemble prediction is that the price will increase, and therefore the stock should be bought; if it is lower than zero, the ensemble prediction is that the price will decrease, hence the stock should be short sold; finally, if it is exactly zero, the ensemble does not make a prediction, and the agent does not trade.

While the prediction module allows the agent to automatically decide when to buy or short sell a financial instrument, that is not sufficient to make it completely autonomous. Besides being able to decide on the direction of each trade (i.e., if it should go long or short), it also needs to be capable of deciding how much to invest (i.e., the size of each trade). More specifically, it should be able to decrease the investment amount or even stop trading when the perceived risk is bigger. That is the purpose of the empirical knowledge module. For each potential trade, it can set the investment amount to three different sizes: if the trade is expected to be profitable, a standard, user-defined trade size is used; if there are doubts regarding the profitability of the trade, half the standard trade size is used; finally, if the trade is expected to be unprofitable, the size is set to zero, which means that the agent will not make the trade. The empirical knowledge module gets its name from the fact that it uses information from previous trades to decide the amount to invest in new trades. The module's main component is a case-based reasoning system. In this system, each case corresponds to a trade that was previously

executed by the agent, and contains the following information: the direction predicted by the prediction module, the direction predicted by each model in the prediction module's ensemble, and the return of the trade. The empirical knowledge module tries to capitalize on the bigger profitability associated with certain combinations of models' predictions. For example, our empirical studies show that trades carried out when all the data mining models make the same prediction, i.e., all predict a price increase or all predict a price decrease, are consistently more profitable than those performed when the predictions are mixed. The algorithm governing the empirical knowledge module's decisions is shown in figure 3.

```
Split the available instances in two datasets: the test set with the most
recent n instances, and the training set with all the rest.

For each model in the ensemble:
  - For each instance in the test set, make the model predict its class, and
    use the prediction to simulate a trade.
  - Use the simulation results to calculate the model's overall profit factor
    (equation 1), long profit factor (equation 2) and short profit factor
    (equation 3).
  - Retrain the model with the training set.
  - For each instance in the test set, make the retrained model predict its
    class, and use the prediction to simulate a trade.
  - Use the simulation results to calculate the retrained model's overall
    profit factor (equation 1), long profit factor (equation 2) and short
    profit factor (equation 3).
  - If the overall profit factor of the retrained model is greater than or
    equal to the overall profit factor of the model before retraining: replace
    the model in the ensemble with the retrained model.
  - Else: discard the retrained model.
  - Use the model in the ensemble to make a prediction for the target period.
  - If it predicts a price increase: set the weight of its vote to its long
    profit factor.
  - If it predicts a price decrease: set the weight of its vote to its short
    profit factor.
  - If the weight is a negative number: set it to zero.

Add the weights of the votes of all the models that predict a price increase,
and subtract from this value the weights of the votes of all the models that
predict a price decrease.

If the resulting value is greater than zero: predict a price increase in the
target period.

Else if it is lower than zero: predict a price decrease in the target period.

Else: do not make a prediction for the target period.

Once the target period ends, create a new instance and add it to the
available data.
```

Fig. 2 Pseudo-code of the algorithm used by the prediction module to make the price direction forecast for a given target period.

Both the prediction and the empirical knowledge modules were devised in way that allows the agents to keep learning while they trade. However, there will always be some expert knowledge that the agents will not be able to pick up from practice. The domain knowledge module was created to overcome this problem. As its name implies, its main responsibility is to use domain-specific knowledge to make trading decisions. This module consists of a rule-based expert system in which trading experts can insert rules to regulate the trading activity of the agents. These rules can be related to many different aspects of trading; for example, they can define low liquidity periods

during which the agents should not trade, or they can be used to make the agents close open trades when a certain profit or loss is reached. In the proposed agent architecture, the domain knowledge module is responsible for making the final trading decisions, by taking into account the prediction module's recommendations to buy or short sell the financial instrument, the empirical knowledge module's trade size suggestions, and its own expert rules.

Considering the modules' algorithms, and looking at the agent architecture as a whole, we believe that a trading agent that is based on this architecture should be able to:

- Keep learning new trading patterns as time goes by, as a result of the periodic retraining with new data of the models in the ensemble. This process is essential to the agent's autonomy, because it enables it to update its prediction mechanism without external assistance.
- Adapt to changes in market conditions, as a result of the continuous reweighting of the models' votes according to their simulated profitability: those that have been more profitable in the recent past increase in weight, while those that have been less profitable decrease in weight.
- Know when to stay out of the market. All the modules in the agent's architecture can help in this decision: the prediction module will prevent it from trading if the recent past profitability of all the models in the ensemble is negative (because all the vote weights will be set to zero); the empirical knowledge module stops a trade by setting its size to zero, whenever the cases in the database show that similar trades in the past were unprofitable; finally, the domain knowledge module can prevent trades from being open according to the expert's rules (which could, for example, stop the agent from trading based on time restrictions, instrument price levels, or market volatility).

```
Get the ensemble's and the models' price direction forecasts for the target
period from the prediction module.
Retrieve from the database all the cases with the same combination of
predictions.
While the number of retrieved cases is lower than a user-defined minimum:
   remove the last model's prediction from the search and retrieve the cases
   again.
Calculate the overall profit factor of the retrieved cases (equation 1).
If the profit factor is greater than or equal to a user-defined threshold:
   make the trade size equal to the standard size.
Else if it is lower than another user-defined threshold:
   make the trade size equal to zero.
Else:
   make the trade size equal to half the standard size.
Once the trade is closed, inserted a new case in the database.
```

Fig. 3 Pseudo-code of the algorithm used by the empirical knowledge module to make the trade size decision for a given target period.

Obviously, there is no guarantee that the described agent architecture will be able to produce a successful agent for every stock and every trading timeframe combination. Nevertheless, if some of the agents created do possess a statistical trading edge, it is possible that their success could make up for the trading losses of the least competent agents in a diversified multi-agent trading system.

3 The Intelligent Stock Trading Agents

Using the referred architecture, we set out to create 60 stock trading agents. Each agent was configured to day trade a single stock (i.e., to open a position when the market opens every day, and close it at the end of the trading session) according to the following methodology:

- 11 data mining models were placed in the agent's ensemble. These models were randomly selected by an automatic process which trained several hundred models with random parameters and attributes, and then picked the 11 best, according to the performance shown with a very small set of test data. Many different types of classification and regression models were considered, among which artificial neural networks, the naïve Bayes classifier, decision trees and rule learners. As for the attributes used, they were all price and time-based, and included technical analysis indicators like the RSI, the Williams %R and moving averages. The models were trained and tested with the Weka API [7].
- profit factors were calculated using the last 50 instances, hence the models' vote weights for each trade were based on the profitability shown in the previous two and a half months of trading;
- a trade's size was set to zero if similar trades retrieved from the database had a profit factor lower than zero; if the profit factor was between 0 and 1, the trade size was set to half the standard size;
- the following rules were inserted in the agent's expert system: do not trade around Christmas Day or Good Friday (to prevent it from trading in low liquidity days); do not trade if the stock's price is below $10 (because the trading costs can become prohibitively expensive if the stock price is too low, assuming a fixed commission per share,); close a trade if it reaches a profit equal to 2/3 of the average price range in the last 5 days (this is a take-profit rule that instructs the agent to close an open trade once it reaches a reasonable profit). The expert rules were handled by the JBoss Drools engine [8].

The 60 stocks traded by the agents were selected according to two loosely defined requirements: they had to have a high beta, and the market cap of the corresponding companies had to be relatively big. For each of these stocks, we collected as much historical price data as we could find, up to January of 2006. This data was converted into instances, which were utilized to train the agents. Once trained, each agent simulated trades for the out-of-sample period between February of 2006 and October of 2010 (i.e., a total of 1,194 test instances per agent). The cumulative return of the 5 most profitable and the 5 least profitable agents throughout this period is shown in figure 4. As expected, not all the agents achieved an acceptable performance. Overall, the 60 agents made a total of 35,453 trades, 53.8% of which were profitable. These results reflect a commission of $0.01 per stock traded, calculated assuming a standard trade size of $20,000. As a side note, we should point out that the percentage of profitable trades to be expected from a completely random strategy should be, on average, well below 50%, due to these commissions alone.

Since the objective of our work was to create a system with the potential to be profitable in the long run, it was clear we needed to implement some sort of investment diversification. In order to accomplish this, we integrated the 60 intelligent agents into a diversified trading system, and made them share the monetary resources.

The system's main advantage is that it eliminates much of the trading risk associated with the individual agents, because the losses of the worst agents are compensated by the gains of the best ones. The accumulated return achieved with this system in the test period is shown in figure 5 and summarized in table 1. Looking at the difference between the system's gross and net returns, we can see that more than a third of its profit was wasted with trading commissions, which goes to show just how important these expenses are in this type of simulation.

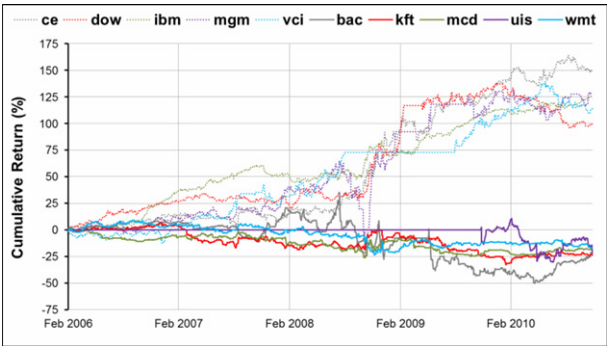

Fig. 4 Individual net cumulative returns of the 5 most profitable and the 5 least profitable stock trading agents in the test period.

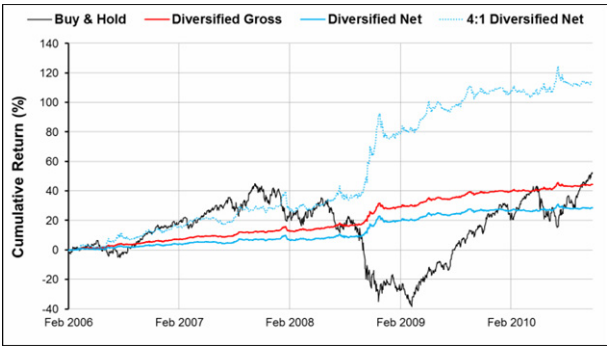

Fig. 5 Gross and net cumulative returns of the diversified trading system.

When compared with the buy-and-hold investment strategy (which implies simply buying equal amounts of the 60 stocks at the beginning of the simulation period, and holding them till the end), the system was not as profitable: it obtained a net return of 28.7%, versus a 51.8% return for the buy-and-hold strategy (calculated using stock prices adjusted for dividends and splits). Nevertheless, if we look at the chart in figure 5, we can see that the system holds a significant advantage over the buy-and-hold strategy, which is the much smaller volatility of its return curve. This advantage is evident in the difference between the two strategies' maximum drawdown (which measures the maximum accumulated loss during the simulation period): the buy-and-hold strategy had an unacceptable 57.5% maximum drawdown, while the system's was just 4.2%. What this means, in practical terms, is that the system's strategy was

much safer in the test period. The lower risk implies that it should be better suited for using leverage (i.e., trading on margin with borrowed funds). If, for example, the system was configured to trade with 4:1 leverage (which would simply require quadrupling the agents' standard trade sizes), its net return in the simulation period would increase to 114.6%, with a maximum drawdown of 16.6%. This would be an excellent performance by any standards, and would soundly beat that of the buy-and-hold strategy. Our results hint that the development of a completely autonomous agent-based trading system with the potential to achieve an acceptable performance might be an actual possibility. This corroborates our belief that agent technology can be of use for this type of practical application.

Table 1 Comparison between the cumulative return of the diversified trading system and the buy-and-hold investment strategy.

Strategy		*Return (%)*	*Max Drawdown (%)*	*Ratio*
Buy & Hold		51.8	57.5	0.9
Diversified	Gross	44.5	3.9	11.5
	Net	**28.7**	**4.2**	**6.9**
4:1 Diversified	Gross	178.0	15.5	11.5
	Net	**114.6**	**16.6**	**6.9**

The performance of the described investment system attests the usefulness of the agent architecture we proposed. Nevertheless, there is always the possibility that a different architecture could yield better returns. In particular, it is possible that agents built using a subset of the architecture modules described in the previous section might perform better. In order to test this hypothesis, we implemented the 60 agents utilizing the following subsets: first, the agents were built with just the prediction module; next, we implemented them using the combination between the prediction module and the empirical knowledge module; finally, we combined the prediction module with the domain knowledge module. We then tested the diversified trading system using these three types of agents. Its accumulated return and maximum drawdown, for each simulation run, is summarized in table 2. We can verify that, when the simplest architecture was used, the system was barely profitable, having achieved a return of just 11.0%, with a maximum drawdown of 18.2%. This makes it clear that the agents' forecasting mechanisms can be much improved, be it with better data mining models or better training attributes. Adding the empirical knowledge module to the architecture considerably improved both metrics. An even bigger improvement was obtained when the prediction module was combined with the domain knowledge module. Finally, the use of all three modules resulted in the performance with the best risk-reward ratio, which confirms that all the modules in the proposed architecture were making an important contribution to the system's overall performance.

We should point out that, even though the system that was presented consists of a decentralized group of intelligent autonomous agents, it might not be considered a true multi-agent system due to the lack of inter-agent communication. As is, the agents' only interaction is with other market participants (software and human) through the stock broker. Unlike the previously mentioned multi-agent Forex trading system [6], there is no obvious reason for having the stock trading agents communicate with each other while trading. Nevertheless, there is a particular situation in which the agents

might need to report their decisions to another agent. If the base currency of their trading account is different from that in which their stocks are denominated, then it would make sense to hedge the currency exposure whenever a stock trade is opened (because buying/short selling a stock will generate a long/short exposure to the corresponding currency). For this reason, we introduced a new agent in the system, named hedging agent, which is responsible for receiving the investment decisions from the stock trading agents, and making the appropriate currency trades in the Forex market to hedge the currency exposures. This interaction is depicted in the sequence diagram in figure 6 (for simplicity's sake, only one trading agent is shown).

Table 2 Net cumulative return of the diversified investment strategy when the agents are built using different architectures.

Architecture	*Return (%)*	*Max Drawdown (%)*	*Ratio*
Prediction Module	11.0	18.2	0.6
Prediction & Empirical Modules	17.9	8.5	2.1
Prediction & Domain Modules	29.6	11.0	2.7
All Modules	28.7	4.2	6.9

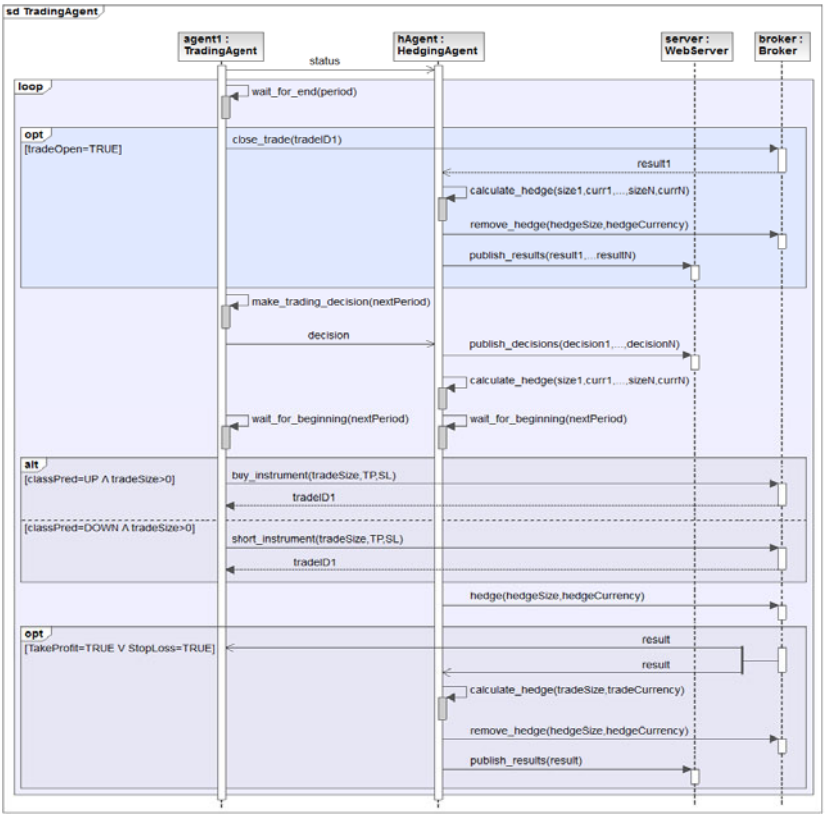

Fig. 6 Sequence diagram describing the interaction between the stock trading agents, the hedging agent and the stock broker.

A typical disclaimer when presenting trading results is that "past performance does not guarantee future returns". Put another way, we cannot be sure that the described system will be able to trade successfully in the future (by successfully we mean profitably with low risk) just because it did so in the past. In order to keep testing it going forward, we made the hedging agent responsible for posting the trading decisions of the 60 agents in a public website, every day before market open. This website has been available at http://ruibarbosa.eu/iquant/iquant_all.html since 2009, allowing its visitors to follow the agents' activity live, and to witness the forward-testing of the trading system. This autonomous system will be kept running continuously over time, so that its performance can be analyzed over the long run.

4 Conclusions and Future Work

The research described in this article leads us to conclude that it may be possible to create an autonomous agent-based trading system that can achieve acceptable results, and that the proposed agent architecture might be a viable option to build this type of systems. Both conclusions should be of interest to the investment industry.

Despite the compelling trading simulation results we obtained, there is still a lot to improve in our stock trading system. First of all, it is likely that using better attributes to train the agents will increase their accuracy, which in turn might make them more profitable. Adding more agents to the system should also improve its performance (in terms of risk-adjusted return), as this would increase the investment diversification, especially if the new agents were configured to trade other types of financial instruments, such as commodity futures or currency pairs. For a real-life trading system, we might also consider a strategy to get rid of the worst agents, possibly replacing them with better ones. We did not do this in our testing, in order to avoid survivorship bias, but it is clear that this measure should vastly improve the system over time. While the system's simulation results do not guarantee that it will be successful in the future, we do believe that they were at least good enough to justify further research on this topic. As is, the system's performance is empirical proof of the usefulness and potential of agent technology for this particular practical application.

References

[1] Vytelingum, P., Dash, R.K., He, M., Sykulski, A., Jennings, N.R.: Trading strategies for markets: A design framework and its application. In: La Poutré, H., Sadeh, N.M., Janson, S. (eds.) AMEC 2005 and TADA 2005. LNCS (LNAI), vol. 3937, pp. 171–186. Springer, Heidelberg (2006)

[2] Wang, H., Mylopoulos, J., Liao, S.: Intelligent agents and financial risk monitoring systems. Communications of the ACM - Robots: Intelligence, Versatility, Adaptivity 45(3), 83–88 (2002)

[3] Kearns, M., Ortiz, L.: The Penn-Lehman Automated Trading Project. IEEE Intelligent Systems 18(6), 22–31 (2003)

[4] Jennings, N., Wooldridge, M.: Applications of Intelligent Agents. In: Agent Technology: Foundations, Applications and Markets, pp. 3–28 (1998)
[5] Barbosa, R., Belo, O.: A Step-By-Step Implementation of a Hybrid USD/JPY Trading Agent. International Journal of Agent Technologies and Systems 1(2), 19–35 (2009)
[6] Barbosa, R.P., Belo, O.: Multi-Agent Forex Trading System. In: Hãkansson, A., Hartung, R., Nguyen, N.T. (eds.) Agent and Multi-agent Technology for Internet and Enterprise Systems. Studies in Computational Intelligence, vol. 289, pp. 91–118. Springer, Heidelberg (2010)
[7] Weka Data Mining API, http://www.cs.waikato.ac.nz/~ml/weka/
[8] JBoss Drools Rule Engine, http://www.jboss.org/drools