An Agent Based Architecture for Highly Competitive Electronic Markets

José Alberto R. P. Sardinha
sardinha@inf.puc-rio.br

Ruy L. Milidiú
milidiu@inf.puc-rio.br

Carlos J. P. de Lucena
lucena@inf.puc-rio.br

Patrick M. Paranhos
paranhospatrick@yahoo.com

Pedro M. Cunha
cunhapedro@yahoo.com

PUC-RioInf.MCC37/04 October, 2004

Abstract: In this paper we present a multi-agent architecture for trading in electronic markets with asynchronous and related auctions. This architecture enables the development of a multi-agent system for a highly competitive environment, where all participants are competing for a limited number of goods. We define intelligent agent roles that tackle sub problems of trading, and present a solution for combining these results in a distributed environment. The agents' typical tasks are price prediction, bid planning, good allocations, negotiation, among others. We use the Trading Agent Competition (TAC) environment as a case study to illustrate the suitability of our approach. We also present *LearnAgents*, a multi-agent system based on our architecture that achieved the third place in the 2004 TAC Classic competition.

Keywords: Software Agents, Electronic Commerce, Trading Systems, Artificial Intelligence, Machine Learning.

Resumo: Este artigo apresenta uma arquitetura multi-agente especializada em comércio para mercados eletrônicos com leilões relacionados e assíncronos. Essa arquitetura permite o desenvolvimento de um sistema multi-agente para ambientes altamente competitivos, onde todos os participantes competem por um número limitado de bens. Nós definimos papéis de agentes que resolvem subproblemas do comércio, e apresentamos uma técnica para combinação dos resultados dos agentes. Os principais papéis são predição de preços, planejamento de lances, alocação de bens, negociação, entre outros. Utilizamos o ambiente do *Trading Agent Competition* (TAC) como estudo de caso para ilustrar a nossa estratégia. O *LearnAgents* é um sistema multi-agente baseado em nossa arquitetura que conseguiu o terceiro lugar na competição do TAC Classic de 2004.

Palavras-Chave: Agentes de Software, Comércio Eletrônico, Sistema para Comércio, Inteligência Artificial, Aprendizado Mecanizado.

1. Introduction

The Internet is an environment that enables the development of efficient e-commerce solutions. The global economy relies on these solutions to achieve a cost-effective policy for trading, and this is clearly shown by the number of applications and web sites for the B2B (business-to-business) and B2C (business-to-consumer) markets. Auctions are being used in many procurement solutions because they support efficient policies and valuable trade transactions.

However, many automatic trading applications still deal with goods as independent entities. This premise is rarely true, because humans and companies are normally interested in buying many items at a time. In some cases, goods can have a high interdependency, such as a TV and DVD. What is the point of buying a DVD without a TV? Time restrictions also apply in these markets, because customers would like to buy all these related goods at the same time.

Multi-Agent Systems [1] [2] is a technology that has been recently used in many simulators and intelligent systems to help humans perform several time-consuming tasks. The system's goal is achieved when software agents [3] react to events, execute strategies, interact, and participate in organizations. Applications for e-commerce, information retrieval, and business intelligence are using this technology to build distributed systems over the Internet.

Electronic markets are an example of complex and open environments where agents have to deal with many unknown situations. Multi-agent systems are probably the most suitable technology for dealing with decision making strategies in such environments, because they permit an easy combination of various artificial intelligence techniques in distributed environments. Consequently, these agent based systems are highly adaptable and easy to scale up.

Trading systems [4] [5] [6] are using the agent technology to build fully automated solutions that provide valuable trade transactions. We present in this paper an agent based architecture for trading in highly competitive markets. This architecture is called *LearnAgents* and was used to build a trading system that participated in the 2004 TAC Classic Tournament. The system achieved the third place in the competition.

2. The Trading Agent Competition

The Trading Agent Competition (TAC) is designed to promote and encourage high quality research into the trading agent problem. There are two scenarios in this annual competition: TAC Classic [7] - a "travel agent" scenario based on complex procurement on multiple simultaneous auctions; and, TAC SCM [8] - a PC manufacturer scenario based on sourcing of components, manufacturing of PC's and sales to customers.

In the TAC Classic, each agent has the goal of assembling travel packages from TACtown to Tampa during a 5-day period. The agent acts on behalf of eight clients, and the objective

of the travel agent is to maximize the total satisfaction of its clients with the minimum expenditure. Travel packages consist of items, such as round-trip flight, a hotel reservation, and tickets to some entertainment events.

There are clear interdependencies in the TAC Classic scenario, since a traveler needs a hotel for every night between arrival and departure of the flight, and can attend entertainment events only during that interval. The three types of goods (flights, hotels, entertainment) are purchased in separate markets with different rules. In the competition, several games are played during each round in order to evaluate each agent's average performance and to smooth the variations in client preferences.

We decided to use the TAC Classic domain to test our architecture because it resembles a competitive electronic market, and can be used as a benchmark for evaluating the performance of our trading system.

3. The *LearnAgents* Architecture

The *LearnAgents* architecture is depicted in Figure 1. We define intelligent agent roles that tackle sub problems of trading, such as price prediction, bid planning, goods allocation, and negotiation. This architecture is modeled as a multi agent system instead of a single agent, because it facilitated the development of modular entities that are asynchronous, distributed, and reusable.

The Corporate Knowledge Base is a shared memory blackboard [2], where every agent can read and write information in a distributed environment. The agents in the system use the knowledge base to combine the sub problems of trading. Therefore, agents can use information posted in the knowledge base by other agents to build their own solutions, and share these results through the same knowledge base.

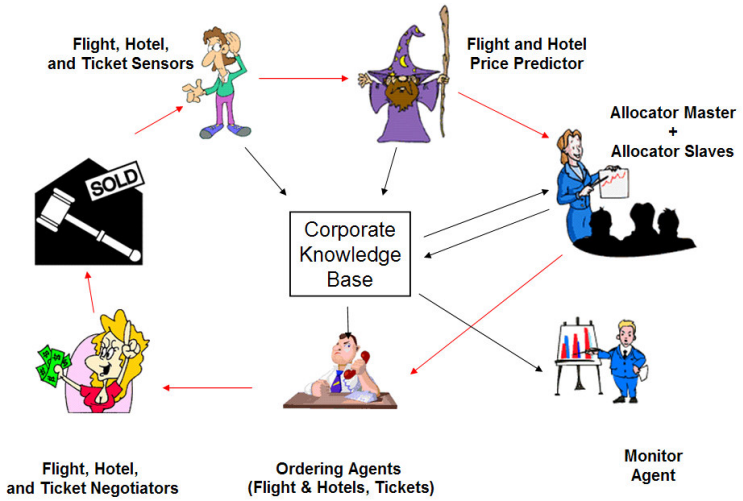

Figure 1: The *LearnAgents* Architecture

The Hotel Sensor Agent, Flight Sensor Agent and Ticket Sensor Agent are responsible for the market sensory and knowledge building of the system. These agents collect price information from auctions and store them in the knowledge base. The sensor agents also receive all the events from the environment, and are responsible for notifying other agents in the system with these events. The Price Predictor Agent is also responsible for knowledge building. This agent predicts hotel and flight auction prices for the knowledge base. The price predictor agent uses the price history in the knowledge to forecast auction quotes.

The agent roles responsible for bid planning are the Allocator Master Agent and the Allocator Slave Agents. The slave agents calculate different scenarios based on the prices in the knowledge base. These scenarios define travel good types and quantities, and characterize a list of different strategies for the system. The Allocator Master Agent is responsible for combining all the scenarios generated from the other slave agents. These scenarios are then stored in the knowledge base.
The Ordering Agent is responsible for good allocation, and it decides the quantity of the required travel goods based on the scenarios in the knowledge base. This decision is based on a scenario with a high profit and a low risk.

The Hotel Negotiator Agent, Flight Negotiator Agent, Ticket Negotiator Agent are responsible for negotiation travel goods in the auctions based on the decision from the Ordering Agent. The negotiators have the goal to buy all the requested goods with the minimum expenditure.

The Monitor Agent is responsible for saving data from the environment and evaluating the performance of the multi agent system. An agent framework [9] was used to implement the trading system with all the agent roles described above. Every agent in the system is an asynchronous and distributed entity, and the current version has 48 agents running in 4 different CPUs (2 Windows and 2 Linux).

3.1 The *LearnAgents* in the TAC Classic Environment

The first agents to act in the TAC environment are the sensors. They update the knowledge base with auction prices every time a “quote update” event is received. The Hotel Sensor is also responsible for retrieving the client information of the eight clients and storing it in the knowledge base. Client information specify a single preferred arrival and departure date, a premium value for upgrading to the better hotel, and entertainment premium values.

Tickets for flights are sold in single seller auctions, and there is an independent auction for each day and direction (in or out). The flight quotes are updated every 10 seconds by a random walk, but the Flight Sensor only receives these quotes every 30 seconds. The Price Predictor receives a message from the Flight Sensor when this event occurs, and stores the new predictions for the flight auctions in the knowledge base. Flight auctions close at the end of the game.

Hotel auctions are Standard English ascending multi-unit, except that they close at randomly determined times. A randomly chosen hotel auction closes at minute one of the

game. Consequently, the agents cannot tell in advance which hotel auction will close at which time. The other hotel auctions close randomly at each minute thereafter. Price quotes are only generated once per minute, and the Hotel Sensor updates the knowledge base with these quotes. The Price Predictor receives a message from the Hotel Sensor when this event occurs, and stores the new predictions for the hotel auctions in the knowledge base. The price quote is the ask price, calculated as the 16th highest price among all bid units. The 16 highest price buy units will be matched when a hotel auction closes and the agents will pay the ask price for the hotel rooms.

The entertainment ticket auctions are standard continuous double auctions such as a stock market. These auctions close at the end of the game, and agents may submit bids with buy or sell points. Price quotes are sent every 10 seconds in response to new bids.

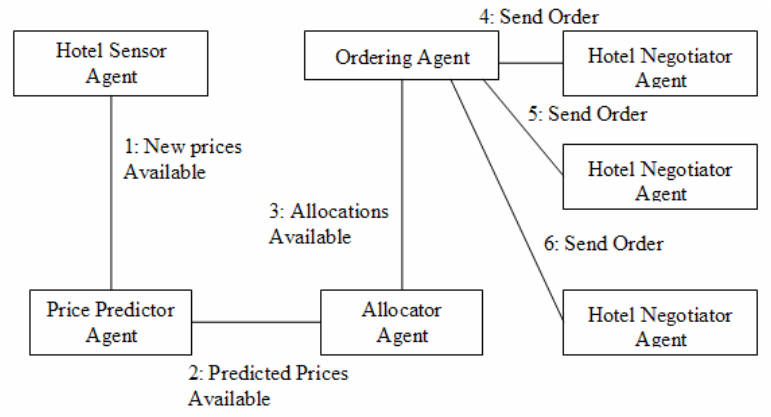

Figure 2: The collaboration between the agents

We present a collaboration diagram of the *LearnAgents* in figure 2. The Hotel Sensor receives every minute the new quotes for the hotel auctions, and sends a message to the Price Predictor informing that the quotes have been updated in the knowledge base. The Price Predictor then calculates the predicted prices of the quotes. The Allocator Master only receives a message from the Price Predictor after these predicted quotes are available in the knowledge base.

The Allocator Agents calculate different scenarios based on the available quotes in the knowledge base. These scenarios are travel good allocations (flights, hotels and tickets) that are calculated using an integer programming solver called XPRESS [10]. The Allocator Agents also calculate ticket allocation and maximum bid values of goods.

The Ordering Agent receives a message from the Allocator Master after the allocations are calculated. The agent then decides the quantity of the required travel goods and the maximum price of the bids based on these allocations. The negotiators receive a message from the Ordering Agent with the quantities of goods to buy. These agents start negotiating the goods in the auctions.

3.2 Price Predictor Agent

The Price Predictor Agents uses a moving average [11] technique to forecast hotel auction ask prices, and a maximum likelihood [12] strategy to predict a hidden random variable x (from a uniform distribution on [-10,30]) chosen for each flight auction. The initial price of each flight auction is chosen from a uniform distribution on [250,400]. The process used to update flight prices is a random walk, and is perturbed every 10 seconds by a value drawn uniformly from:

$[-10, x(t)]$ if $x(t) > 0$,
$[x(t), 10]$ if $x(t) < 0$,
$[-10, 10]$ if $x(t) = 0$,
where $x(t) = 10 + (t / 9{:}00)*(x - 10)$

Although the flight auctions update the prices every 10 seconds, the Flight Sensor Agent only receives a flight quote update event every 30 seconds (where 3 updates have occurred). The random variable x is predicted using the maximum likelihood:

$$\text{argmax} \prod P\,(y_t + y_{t-10} + y_{t-20} = s \mid x)$$

The y_t is the flight price increment at time t, and s is the diference between the actual price quote and the last price quote (30 seconds before). We use the expected value *E(quote)* of the process described above to calculate the predicted price quote. Figure 3 presents the prediction of the ask price of the inflight auction of day 3 in the game 6091 (2004 TAC Classic Finals).

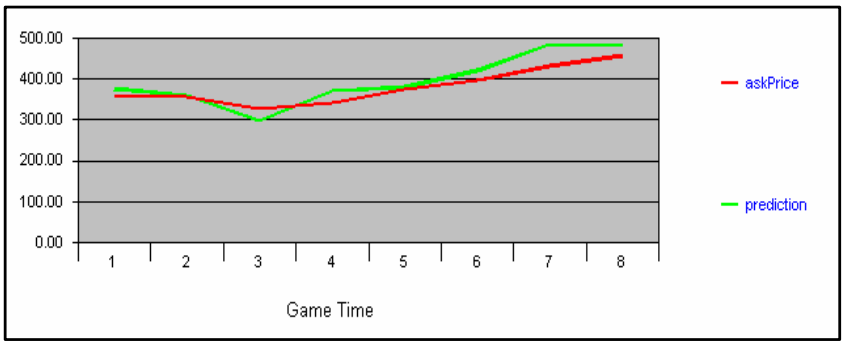

Figure 3: Graph of the Prediction of the Flight Auction

3.3 Allocator Agents

The Allocator Agent uses an integer programming model to build different scenarios. The model receives the clients' preferences, the current prices, the predicted prices, and the goods that are currently owned. The solver is set in motion not only in search of the optimal allocation, but as many best allocations as the solver can optimize in 25 seconds. Each

allocation defines the number of goods per client that must be purchased in each auction and respective score. The objective of the solver is to maximize the following equation:

$$\sum_{client=1}^{8} Utility(client) - \sum_{client=1}^{8} Cost(client) - OtherCosts - HomogeneityFunc$$

A client's utility, measured in dollars, from a feasible travel package and a feasible entertainment package is given by the following: *Utility(client) = 1000 - travel_penalty + hotel_bonus + fun_bonus*. A travel package is feasible if it contains rooms for every night between the arrival and departure dates. The *travel_penalty* is calculated by *100*(|AA - PA| + |AD - PD|)*, where *AA* is the actual arrival date, *PA* is the preferred arrival date, *AD* is the actual departure date, and *PD* is the preferred departure date.

The *hotel_bonus* is calculated by *GHI * HP*, where *GHI* is the good hotel indicator (assumes value 1 if rooms are in good hotel, or assumes value 0 if rooms are in bad hotel), and *HP* is the premium value for upgrading to the better hotel. The *fun_bonus* is calculated by *AWI*AW + API*AP + MUI * MU*, where *AWI*, *API*, and *MUI* are ticket indicators (value {0,1}) for each event type (Alligator wrestling, Amusement park, Museum), and *AW*, *AP* and *MU* are premium values for each type of entertainment event. An entertainment package is feasible if none of the tickets are for events on the same day, and all of the tickets coincide with nights the client is in town.

The *Cost(client)* is the sum of the expenditure per client in the in flight, out flight, hotel, and entertainment ticket auctions. The *OtherCosts* are the sum of the expenditure of all unused goods, transaction losses, etc. The *HomogeneityFunc* is a function that includes an extra penalty in the score, and is expressed by:

$$\sum_{day=1}^{4} Penalty(hotelType, day).ExtraRoom(hotelType, day)$$

The *Penalty(hotelType,day)* is an extra penalty given for every exceeding *hotelType* (0 = Bad Hotel, 1 = Good Hotel) and *day* (1 to 4) in case of *hotel(hotelType,day)* > *maxHotel(hotelType,day)*. The *hotel(hotelType,day)* is the number of hotel goods allocated, and *maxHotel(hotelType,day)* is a parameter that defines the maximum number of hotel goods that are not penalized. Consequenlty:

ExtraRoom(hotelType,day) = 0, if *hotel(hotelType,day)* < *maxHotel(hotelType,day)*
or
ExtraRoom(hotelType,day) = *hotel(hotelType,day) - maxHotel(hotelType,day)*,
if *hotel(hotelType,day)* > *maxHotel(hotelType,day)*

The *HomogeneityFunc* models the risk of accumulating too many clients in a given day and a given hotel type. This function penalizes allocations that decide to buy a great number of hotel rooms of a same type and in a same day.

The Allocator Agent also calculates the maximum bid value of each available good. Let *G*(g)* denote the score of the optimal allocation assuming we hold good *g*. To calculate the maximum value of a bid for good *g*, we run the solver with the price $p'_g = \infty$ for good *g*. This optimal allocation yields a score of *G*(g)'* . The maximum bid is calculated by *G*(g) - G*(g)'*.

3.4 Ordering Agent

The optimal allocation and sub-optimal allocations are calculated once per minute by the Allocator Master Agent. The Ordering Agent uses a heuristic based on these allocations for defining the goods that must be purchased. For every client *client*, an allocation contains the following information: an actual arrival date *AA*, an actual departure date *AD*, a hotel type *hotelType*, and entertainment packages *e(day)*.

We denote the optimal allocation as *G** and the sub- allocations as *G(n)*, where *n* is an index for the computed sub-allocations and assumes values from 1…*N*. The Ordering Agent sends purchase orders for negotiators based on the following rules:

- Send "in flight" order on day *AA* for *client* in *G**, only if :
  - $a > p.N$, where *a* is the number of occurrences an "in flight" *AA* for *client* is found in *G(n)*, and *p* is a percentile.
  - The predicted price indicates the ask price is rising.
- Send "out flight" order on day *DD* for *client* in *G**, only if :
  - $b > p.N$, where *b* is the number of occurrences an "out flight" *DD* for *client* is found in *G(n)*, and *p* is a percentile.
  - The predicted price indicates the ask price is rising.
- Send *hotelType* order on *day* for *client* in *G**
- Send entertainment type *e(day)* for *client* in *G**

This heuristic uses *G** and *G(n)* to define the goods that must be purchased considering the uncertainty of the market. If a flight good in *G** is not present in at least *p* percent of the allocations *G(n)*, then the decision is delayed for the next minute when the Allocator Master Agent re-calculates the allocations.

3.5 Negotiator Agents

The Hotel Negotiator Agent uses a minimax procedure [13] and reinforcement learning technique [12] to calculate bids in the hotel auctions. The minimax procedure is a search technique for a two player game that is used to find optimal bids. In this game there are two players: MAX and MIN. The Hotel Negotiator Agent is modeled as the MAX player and the market's response as the MIN player. A depth-first search tree is generated, where the current game position is the root. The final game position is evaluated from MAX's point of view, and the inner node values of the tree are filled bottom-up with the evaluated values.

The nodes that belong to the MAX player receive the maximum value of the children. The nodes for the MIN player will select the minimum value of the children. The minimax procedure is also combined with a pruning technique called Alpha-Beta [13].

The states of the minimax search tree consist of the following information: (i) *AskPrice*: the current ask price; (ii) *LastAskPrice*: the ask price of the previuous minute; (iii) *deltaAskPrice = AskPrice – LastAskPrice*; (iv) *Gama*: a constant; (v) *Bid*: the current bid; (vi) *LastBid*: the bid sent in the last minute; (vii) *t*: time of the game.

The Hotel Negotiator Agent calculates the curent bid (MAX player decision) using the following alternatives:

(i) *Bid = LastBid + 0.5*Gama*deltaAskPrice*;
(ii) *Bid = LastBid + 2*Gama*deltaAskPrice*;
(iii) *Bid = LastBid + 5*Gama*deltaAskPrice*.

The MIN player is modeled as the market response:

(i) *AskPrice = AskPrice + 0.5*Gama*deltaAskPrice*;
(ii) *AskPrice = AskPrice + 2*Gama*deltaAskPrice*;
(iii) *AskPrice = AskPrice + 5*Gama*deltaAskPrice*.

The Hotel Negotiator Agent uses an evaluation function in the minimax procedure, because it is unfeasible to explore all the states when a bid is calculated. This evaluation function is used to produce scores on intermediate states. This evaluation function uses a single-layer perceptron with only one artificial neuron. The perceptron receives a state of the minimax search tree and scores the state between 0 and 1. The states with scores close to 1 are considered good states and the goal of the agent is to adapt the weights of the perceptron to encounter final states that represent optimal bids.

Our problem is how find a configuration of weights in the perceptron that yield optimal bid decisions. Instead of using a supervised learning [12] strategy where the perceptron is presented with a learning set, we use a perceptron rule [12] combined with a reinforcement learning strategy. The agent has to keep a log of all the states traversed in an auction in order to use this strategy. The perceptron rule is used to adapt the traversed states when the auction closes: $w_i=w_i+\eta.(y(t)–d(t)).x_i$, where w_i is the i-th weight of the perceptron; x_i is the i-th data input of the state in the minimax search tree; $y(t)$ is the actual result of the evaluation function; $d(t)$ is the expected result of the evaluation function that is calculated with a reinforcement learning formula; and η is the learning rate.

The expected value of $d(t)$ is first calculated on the last state saved in the log (representing the last bid sent to an auction before it closes). The agent only accomplishes a successful goal if the bid sent acquires all the requested goods from the Ordering Agent and the difference between the bid and the askprice is not superior to a constant *k*. This successful goal yields a reward of $d(t) = 1$. In all other cases the reward of $d(t) = 0$.

The following "successful" traversed states (a bottom-up direction) use the reinforcement learning formula: $d(t-\Delta t) = d(t) + \beta.\ (reward(t) - d(t))$, where $d(t)$ is the expected output of the decision point at time t, β is the reinforcement learning rate, $reward(t) = 1 – (bid(t) –$

askprice(t)) / *askprice(t)* is the reward obtained in the decision point at time *t*. The "unsuccessful" traversed states use the formula: $d(t-\Delta t) = d(t) + \beta.d(t)$.

Total number of requested hotel goods from the Ordering Agent	Total number of hotel goods acquired by Hotel Negotiators	Percentage (%)
1597	**1515**	**94,87**

Table 1: Performance of the Hotel Negotiator Agent in the 2004 TAC Classic Final

The Ordering Agent sends buying orders every minute to the 8 Hotel Negotiator Agents (one for each auction). These buying orders consist of a number of requested goods and the high price (maximum bid value that must be sent). Table 1 presents the performance of the Hotel Negotiators in the 2004 TAC Classic Finals. The Ordering Agent requested 1597 hotel goods in the 35 games played, and 94,87% of these goods were among the 16th highest bid units.

Figure 4: The bids sent to an Auction by the Hotel Negotiator Agent

Figure 4 depicts a graph of the bids sent to the Good Hotel auction of day 1 in game 6089 (2004 TAC Finals). The bids sent are always close to the ask price in order to buy the requested goods with the minimum expenditure, but must not exceed the maximum bid (called high price) defined by the Allocator Agents.

The Flight Negotiators simply send bids with quantities and prices that match the current ask price of the flight auction. The Ticket Negotiators buy ticket goods in the market sending bids with prices that match the sell bid point, and sell goods with prices that are equals to the maximum bid (calculated by the Allocator Agent) with an extra increment.

3.6 Monitor Agent

The Monitor Agent has a very important role in system, because it monitors the individual performance of the Price Predictor Agent, Allocator Agents, Ordering Agent, Negotiator Agents, and the performance of the multi agent system. This agent collects important information in the knowledge base and stores them in a relational database. We depict in figure 5 the interface of the Monitor Agent that permits an online evaluation of the system.

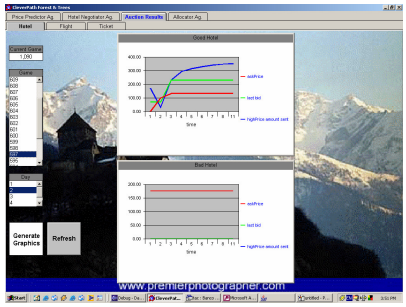

Figure 5: Monitor Agent Interface

4. Final Comments

In this paper we describe a multi-agent system architecture called *LearnAgents* for trading in electronic markets with asynchronous and related auctions. The architecture has a design based on the integration of intelligent agents. These agents combine decision making strategies for trading problems and are executed in a distributed environment. Consequently, this agent based system is highly adaptable and easy to scale up.

Every agent in the system is responsible for solving a specific sub problem of trading. The Sensors Agents collect information from the market. The Predictor Agent forecasts price quotes of the auctions. The Allocator Agents calculate optimal and sub optimal allocations with an integer programming model. The Ordering Agent uses heuristics to extract relevant information from the allocations, and makes decisions about buying and selling goods. The Negotiators trade in the market, following the Ordering Agent instructions to maximize the system profit.

Scores for competition TAC 2004 Finals (game 6084 - 6118)

Competition started at 2004-07-22 13:00:00 and ended at 2004-07-22 19:57:00.
Final Scores.

Position	Agent	Avg Score	Games Played	Zero Games
1	whitebear04	4122.11	35	0
2	Walverine	3848.97	35	0
3	LearnAgents	3736.62	35	0
4	SICS02	3708.24	35	0
5	NNN	3665.97	35	0
6	UMTac-04	3281.43	35	0
7	Agent@CSE	3262.51	35	2
8	RoxyBot	2015.02	35	0

Scores last updated after game 6118 on server tac1.sics.se version 1.0 beta 10

Figure 6: Final Results of the 2004 TAC Classic Competition

The solution also presents a design with an easy modularity, understandability, reusability and interoperability. These features are important for the evolution of the system when modifications are needed to adapt to the market changes.

Finally, we present in figure 6 the final results of the *LearnAgents* in the 2004 TAC Classic tournament. The tournament had 14 participants from universities and research institutes from around the world that includes Brazil, China, France, Israel, Japan, Macau, Netherlands, Sweden, and USA. The *LearnAgents* finished the tournament in the third place, and in figure 7 we present the statistics of the 35 games played in the final round. We can observe in this highly competitive environment that the *LearnAgents* only obtained positive scores. This fact proves the robustness of our system in such environments. We can also observer in figure 6 that the *LearnAgents* needs a 10.32% improvement to reach *Whitebear04* (first place). We believe this score improvement can be obtained with our flexible architecture.

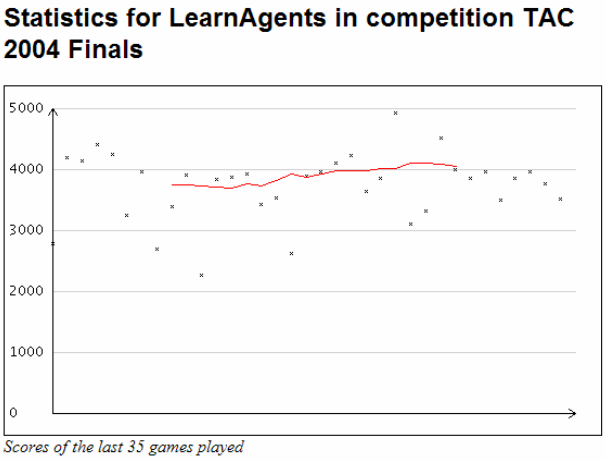

Figure 7: Statistics of the *LearnAgents*

References

[1] Wooldridge, M. *Intelligent Agents*. In: Weiss, G. Multiagent systems: a modern approach to distributed artificial intelligence. The MIT Press, Second printing, 2000.
[2] Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Pub Co, 1999.
[3] Garcia, A., Lucena, C. J. and Cowan, D. *Agents in Object-Oriented Software Engineering*. Software: Practice and Experience, Elsevier, May 2004, pp. 1-32.
[4] Stone, P., Liittman, M. L., Singh, S. and Kearns, M. (2001). *ATTac-2000: An adaptive autonomous bidding agent*. Journal of Artificial Intelligence Research, 15, pp. 189-206.
[5] Vetsikas, I. A. and Selman, B. *A principled study of the design tradeoffs for autonomous trading agents*. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, pages 473-480, July 2003, Melbourne, Australia.

[6] Cheng, S-F, Leung, E., Lochner, K. M., O'Malley, K., Reeves, D. M., Schvartzman, L. J. and Wellman, M. P. *Walverine: A Walrasian Trading Agent*. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, pages 465-472, July 2003, Melbourne, Australia.
[7] TAC Team: Wellman, M. P., Wurman, P. R., O'Malley, K., Bangera, R., Lin, S., Reeves, D. and Walsh, W. E. *Designing the Market Game for a Trading Agent Competition*. IEEE Internet Computing, pp. 43-51, March/April 2001 (Vol. 5, No. 2).
[8] Arunachalam, R. and Sadeh, N. *The 2003 Supply Chain Management Trading Agent Competition*. Trading Agent Design and Analysis workshop at The Third International Joint Conference on Autonomous Agents & Multi Agent Systems. July 2004, New York, USA.
[9] Sardinha, J. A. R. P., Ribeiro, P. C., Lucena, C. J. P., and Milidiú, R. L. *An Object-Oriented Framework for Building Software Agents*. Journal of Object Technology. January - February 2003, Vol. 2, No. 1.
[10] Dash Optimization. http://www.dashoptimization.com
[11] Bowerman, B. L. and O'Connell, R. T. *Forecasting and Time Series: An Applied Approach*. Thomson Learning, 3rd edition, 1993. ASIN: 0534932517.
[12] Mitchell, T. M. *Machine Learning*. McGraw-Hill, 1997. ISBN 0070428077.
[13] Russell, S. and Norvig, M. *Artificial Intelligence*. Prentice Hall, 1995. ISBN 0-13-103805-2.