# A Methodology for Building Trading Agents in 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

PUC-RioInf.MCC36/04 October, 2004

**Abstract:** In this paper we present a methodology that guides the development of a multi-agent system for trading in electronic markets. Our approach uses a strategy of decomposing a complex trading problem into sub-problems, and defines agent roles that tackle these sub-problems. We also present an incremental development process for building asynchronous and distributed agents, which enable effortlessly the combination of intelligent agent strategies. This creates a highly adaptable system for unknown situations that is easy to scale up. We use the Trading Agent Competition (TAC) environment as a case study to illustrate the suitability of our approach.

**Keywords:** Software Agents, Electronic Commerce, Trading Systems, Artificial Intelligence, Software Engineering.

**Resumo:** Este artigo apresenta uma metodologia para guiar o desenvolvimento de um sistema multi agente especializado em comércio para mercados eletrônicos. Nossa proposta utiliza a estratégia de decompor um problema complexo em subproblemas, e define papeis de agentes que atacam esses subproblemas. Apresentamos também um processo incremental de desenvolvimento para a criação de agentes assíncronos e distribuídos, que permitem a combinação de estratégias de inteligência sem muito esforço. Isso permite criar sistemas com alta adaptabilidade para situações desconhecidas e que são fáceis de escalar. Utilizamos o ambiente do *Trading Agent Competition* (TAC) como estudo de caso para ilustrar a nossa abordagem.

**Palavras-Chave:** Agentes de Software, Comércio Eletrônico, Sistema para Comércio, Inteligência Artificial, Engenharia de Software.

## 1. Introduction

The Internet is an environment that enables the development of efficient e-commerce solutions. E-commerce is extremely important in today's global economy, because goods are being traded in a more cost-effective manner. This is clearly shown by the number of applications and web sites for the B2B (business-to-business) and B2C (business-to-consumer) markets.

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, because agent modeling permits an easy combination of various artificial intelligence techniques in distributed environments. These agent based systems are highly adaptable and easy to scale up.

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.

We present in this paper a methodology for modeling, developing, testing and evaluating trading systems. The methodology is composed of the following key elements: the definition of the system's goal and the decomposition process in sub-goals; a generic architecture for trading that guides the definition of agent roles in the modeling process; an incremental development phase with simulation and testing.

The methodology must create a multi agent system that integrates artificial intelligence techniques, permits the addition of new agents effortlessly, guides the testing of different techniques, and enables the distribution of agents when computational resources are limited. The main goal is to leverage the performance of the system.

The *LearnAgents* [4], a trading agent based system, is presented in section 2 as a case study of our methodology. Our system participated in the 2004 Trading Agent Competition [5] (TAC) and achieved the third place in the TAC Classic tournament [6]. The methodology with the generic architecture is described in section 3. The final comments are presented in section 4.

## 2. The *LearnAgents* and TAC

The Trading Agent Competition (TAC) is designed to promote and encourage high quality research into the trading agent problem. There are two scenarios in the annual competition: TAC Classic [6] - a "travel agent" scenario based on complex procurement on multiple simultaneous auctions; and, TAC SCM [7] - 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 model the *LearnAgents* as a multi agent system instead of a single agent, because we needed a system with modular entities that are asynchronous, distributed, reusable, and easy to interoperate. We define agent roles that tackle sub problems of trading, such as price prediction, bid planning, goods allocation, bidding, among others. The system's goal is to acquire travel packages for clients with as much profit as possible. This profit is defined as the sum of the utilities of the eight clients in the TAC game, minus the costs of acquiring the travel goods in the auctions.

## 3. The Methodology for Building Trading Systems

The methodology uses software engineering principles and techniques, and creates a modular design with an easy understandability, reusability, and interoperability. Our main goal is to build trading systems that are easy to include, modify, test and evaluate intelligent agents in distributed environments. We use an incremental development strategy that permits a step by step evaluation of the system. This strategy minimizes development errors and guarantees the system is improving its performance. The entire methodology is depicted in figure 1.

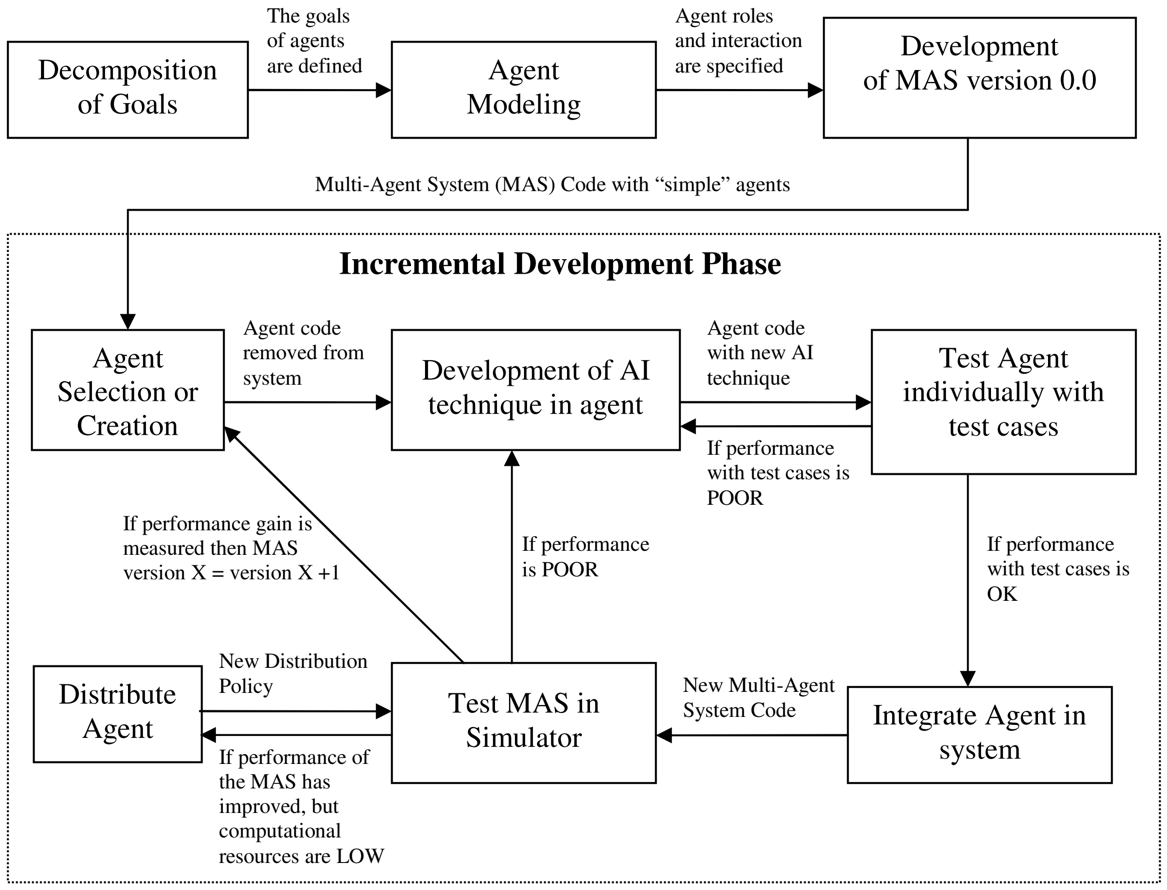

Decomposition of Goals
The goals of agents are defined
Agent Modeling
Agent roles and interaction are specified
Development of MAS version 0.0
Multi-Agent System (MAS) Code with "simple" agents
Incremental Development Phase
Agent Selection or Creation
Agent code removed from system
Development of AI technique in agent
Agent code with new AI technique
Test Agent individually with test cases
If performance with test cases is POOR
If performance gain is measured then MAS version X = version X +1
If performance is POOR
If performance with test cases is OK
Distribute Agent
New Distribution Policy
Test MAS in Simulator
New Multi-Agent System Code
Integrate Agent in system
If performance of the MAS has improved, but computational resources are LOW

Figure 1: The Methodology for Building Trading Agents

## 3.1 Decomposition of Goals

The decomposition of goals is an analysis phase that divides the main goal of the system in many sub goals. These sub goals are associated to agents in the modeling process, and create a system based on specialized agents. Therefore, every agent tackles a sub problem of the trading scenario. This process helps to reduce the complexity of the problem. However, two key elements are important in this decomposition process: (1) We must always use this decomposition process with the main purpose of reducing the complexity of the development stage; (2) the agent's service/functionality must produce solutions that are easy to combine.

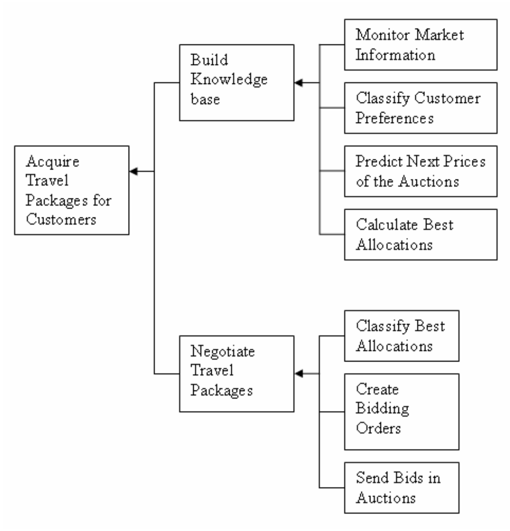

Acquire Travel Packages for Customers
Build Knowledge base
Negotiate Travel Packages
Monitor Market Information
Classify Customer Preferences
Predict Next Prices of the Auctions
Calculate Best Allocations
Classify Best Allocations
Create Bidding Orders
Send Bids in Auctions

Figure 2 - The Goal Diagram of the *LearnAgents*

The goal of the *LearnAgents* is to acquire travel packages for clients with as much profit as possible. This profit is defined as the sum of the utilities of the eight clients in the TAC game, minus the costs of acquiring the travel goods in the auctions. This goal is presented in figure 2, and is decomposed in two other sub goals: build a knowledge base for decision making; and negotiate travel packages based on this knowledge base.

The knowledge base is built with the current prices of the auctions (sub goal: Monitor Market Information), the clients preferences - expressed as a utility table for each travel package (sub goal: Classify Customer Preferences), the future prices of the auctions (sub goal: Predict Next Prices of the Auctions), and a list of different scenarios based on the prices and client's preferences (sub goal: Calculate Best Allocations).

The negotiation of travel goods depends on the following actions: selection of a scenario with a high profit and a low risk (sub goal: Classify Best Allocations), definition of the number and type of travel goods to be bought based on the selected scenario (sub goal: Create Bidding Orders), and the price definition of the bids for the auctions (sub goal: Send Bids in Auctions).

## 3.2 Agent Modeling

The agent modeling phase specifies the agent types and their relationship. This process defines the system structural base, and identifies the entities that will carry on the goals elicited in the prior phase. We present in figure 3 a generic architecture for trading systems. The architecture separates the agent roles in three categories: Sensory and Knowledge Building – Agents responsible for collecting information of the market and making inferences such as prediction and classification; Planning - Agents that use market information to define plans; and, Effectors and Negotiators – Agents that use the plans and define bid strategies in the market.

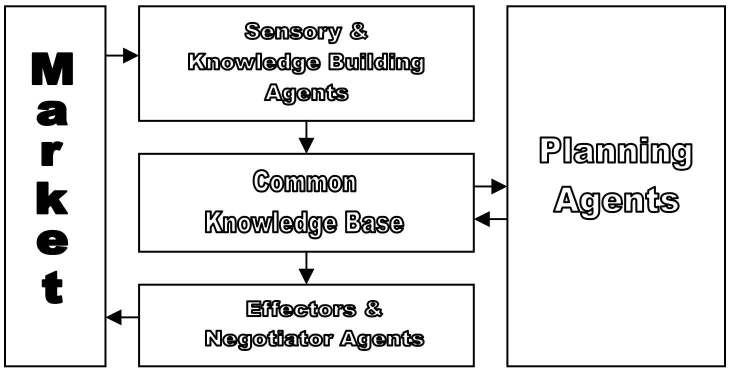

Market
Sensory & Knowledge Building Agents
Common Knowledge Base
Effectors & Negotiator Agents
Planning Agents

Figure 3 - A Generic Architecture for Trading Systems

The Sensory and Knowledge Building Agents in the *LearnAgents* are: **Hotel Sensor Agent, Flight Sensor Agent and Ticket Sensor Agent** – These agents collect price information from auctions and store them in the knowledge base. The sensor agents receive all the events from the environment, and filter these events to other agents in the system. **Price Predictor Agent** – This agent predicts hotel and flight auction prices for the knowledge base. The price predictor agent uses the price history in the knowledge base (collected by the sensor agents) to forecast future ask prices (ask price - calculated as the 16th highest price among all bid units in the auction).

The Planning Agents are: **Allocator Master Agent, Allocator Slave Agents** – The Allocator Master Agent and Allocator Slave Agents calculate different scenarios based on the prices in the knowledge base. These scenarios define travel good types and quantities, and define a list of different strategies for the system. The Allocator Master Agents is responsible for combining all the scenarios generated from the other Allocator Agents and storing them in the knowledge base. **Ordering Agent** – This agent decides the quantity of the required travel goods and the maximum price of the bids based on the scenarios in the knowledge base. This decision is based on a scenario with a high profit and a low risk.

The Effectors and Negotiator Agents are: **Hotel Negotiator Agent, Flight Negotiator Agent, Ticket Negotiator Agent** – These agents negotiate the 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 common knowledge base is an important element in a trading system with asynchronous and distributed agents. This repository permits an easy combination of solutions.

Modeling languages such as MAS-ML [8], Gaia [9], AUML [10] are used in this phase for identifying agent types, environment objects, system events, behavioral descriptions and interaction protocols. We used the Anote [11] to model the *LearnAgents*.

## 3.3 Development Phase of Agents

Instead of building the multi agent system with all the complex agents in a single stage, we propose an incremental development that minimizes code errors and builds a system that is easy to maintain and evolve. The first version of the multi agent system is composed of simple agents with quite any intelligence. This first step is important to test the communication of the agents and the interaction with the environment. The following phases of the process will select agents that will use modules of intelligence in order to improve the performance of the system.

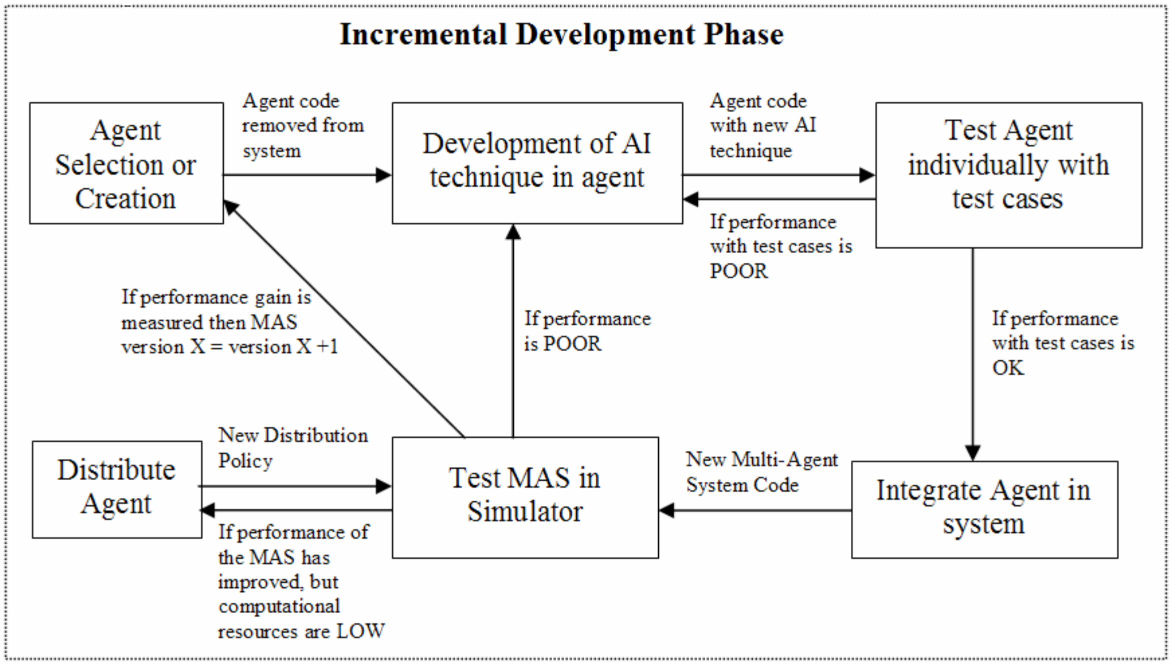

Incremental Development Phase
Agent Selection or Creation
Agent code removed from system
Development of AI technique in agent
Agent code with new AI technique
Test Agent individually with test cases
If performance with test cases is POOR
If performance gain is measured then MAS version X = version X +1
If performance is POOR
If performance with test cases is OK
Distribute Agent
New Distribution Policy
Test MAS in Simulator
New Multi-Agent System Code
Integrate Agent in system
If performance of the MAS has improved, but computational resources are LOW

Figure 4: The Incremental Development Phase

The incremental development starts with the selection of an agent that will receive an intelligent module. This module is built with the goal of improving the performance of the agent and the multi agent system. We propose design patterns [12] [13] to implement these modules. We then test the agent individually, before re-integrating it back in the system. This test is built by the system engineer, and is normally small software modules that generate test cases. Code errors of the intelligent module are found at this point.

The re-integration of the agent is done after the test cases confirm that the intelligent module works correctly. These tests cases do not guarantee any performance improvement of the system. Therefore, a performance test is done with the multi agent system and the new intelligent agent. A simulator of the real trading environment is required for this phase, and we only give a new version to the multi agent system if a performance gain is measured.

However, there are cases when a performance gain is measured but the new intelligent agent starts to use too much computational resources such as CPU and memory. This slows down the system, and the responses of the system to the market are delayed. Consequently, we propose the distribution of this new agent to another CPU in the network. There are two important requirements for this distribution policy: (1) An agent framework with a good communication infrastructure that enables an easy distribution of agents; (2) A knowledge base placed in a distributed blackboard [2] that can be accessed by any agent. In the *LearnAgents*, we used the MAS Framework [14] and the distributed blackboard is implemented as one of the features of the framework.

| ***LearnAgents* version** | **Agent Selected / Intelligent Module** | **Average Score** | **Games Played** |
|---|---|---|---|
| 0.0 | --- | -1855 | 100 |
| 1.0 | Allocator / Solver – Integer Programming Model, Ordering Agent / Heuristic | 1372 | 100 |
| 2.0 | Hotel Negotiator / Minimax, N.N. and Reinforcement Learning | 1752 | 100 |
| 3.0 | Price Predictor (hotel) / Moving Average | 2224 | 100 |
| 4.0 | Allocator (Ticket) /Solver - Integer Programming Model, Ordering Agent / Heuristic | 2856 | 100 |
| 5.0 | Allocator / Solver – Integer Programming Model (second version) | 3370 | 100 |
| 6.0 | Price Predictor (flight) / Maximum Likelihood | 3705 | 100 |

Table 1: The Evolution of the MAS and the Performance Gain

The first version of the *LearnAgents* was built with only reactive agents [1]. Our main goal at this point was to test the communication between the agents and the execution of the system in the environment. Although we were not expecting a good average score, a benchmark was executed to measure the evolution of the performance gain as shown in table 1. The benchmark used in table 1 is calculated against 7 simulated competitors called *dummy agents* [6]. The system plays 100 games in the benchmark against the *dummies* in order to evaluate the average performance and to smooth the variations in client preferences. The simulator of the real competition can be downloaded in the TAC web site[5].

The first selected agent was the Allocator Agent (version 1.0 in table 1). The agent calculates different scenarios based on the prices in the knowledge base. These scenarios are travel good allocations (only flights and hotels) that are calculated using an integer programming solver called XPRESS [15]. 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. After testing the new Allocator Agent with the test cases, we also modified the Ordering Agent with a new heuristic. The Ordering Agent had to decide the quantity of the required travel goods and the maximum price of the bids based on the allocations produced by the Allocator Agent. The multi agent system obtained a score of 1372 in the benchmark against 7 *dummies*.

The Negotiator Agent was the next selected agent for adding of an intelligent module. This module uses a min-max procedure [16] and an evaluation function calculated by a neural network [17] and a reinforcement learning [17] technique. The Negotiator Agent's goal is to send optimal bids to the auctions with the minimum expenditure. Therefore, the agent adapts the weights of the neural network and searches for an evaluation function that produces these optimal bids. The benchmark of the version 2.0 was 1752.

The next selected agent to include the intelligent module was the Price Predictor Agent. This agent uses a price history in the knowledge base to forecast ask prices. The technique implemented in the intelligent module is called moving average [18]. This agent is important for the system, because it helped to deal with uncertainty of the auction prices. Consequently, the Allocator Agent could now calculate scenarios based on these predicted prices. This version 3.0 now scored 2224 in the benchmark.

We then decided to improve the Allocator Agent in the version 4.0 of the system. The flight and hotel goods were the only allocations calculated by the integer programming solver. We extended the model and included the entertainment tickets, and modified the Ordering Agent heuristic to decide the quantity of tickets to buy based on these new allocations. We executed the benchmark and achieved a score of 2856. This agent improved the overall score, but the system started to present delays in some responses to the market. This was clear evidence that CPU resources were low, so we created an Allocator Slave to solve the problem. This new agent also used an integer programming solver that calculated the tickets allocation, and was now executing in another CPU.

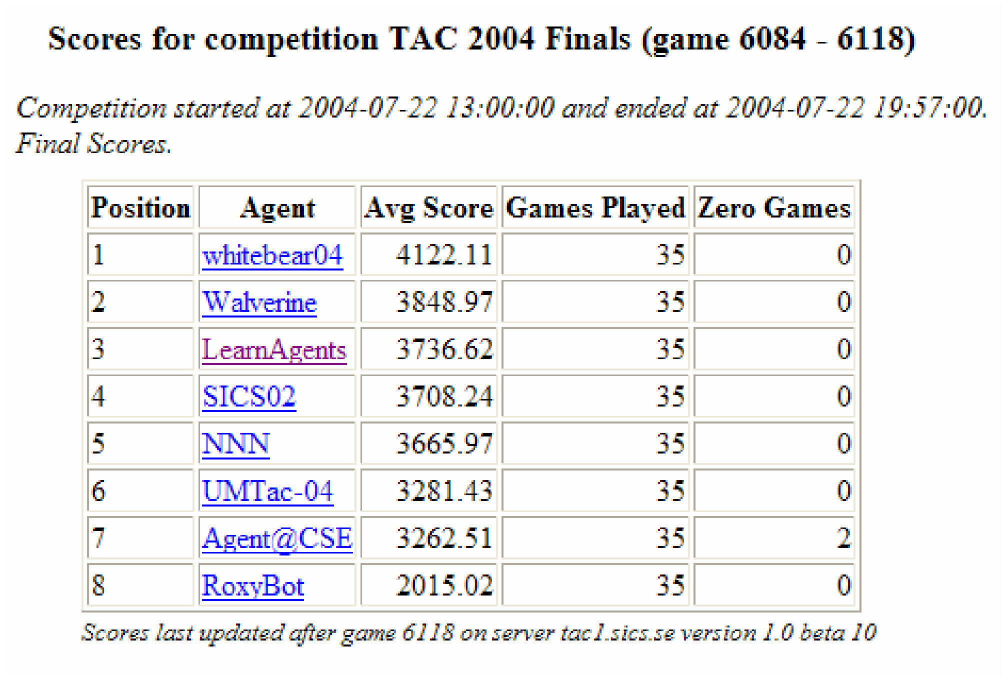
The Allocator Agent was again selected for improvements in version 5.0. The integer programming model was modified to include some parameters for tuning purposes. The travel good allocations are now calculated based on these parameters. We can select parameters that produce allocations with more hotel rooms from Shoreline Shanties [6] (the bad hotel) then the Tampa Towers [6] (the good hotel) in this version. The benchmark with the best parameter configuration produced a score of 3370.

Our last modification before the 2004 TAC Classic Tournament was in the Price Predictor Agent. This agent was only predicting hotel auction prices and we now wanted the prediction of flight auction prices. The flight auction prices are modified according to a stochastic function. The process uses a random walk that starts between \$250 and \$400 and is perturbed every 10 seconds. We included an intelligent module based on a technique called maximum likelihood [17] to predict a parameter of the stochastic function that was not revealed directly to the agents. The predicted parameter is then used to define the expected value of the flight auction prices. This benchmark achieved a score of 3705 in this 6.0 version.

## 4. Final Comments

The methodology creates a disciplined process for modeling, developing, testing and evaluating trading systems. This process creates a multi agent system that integrates artificial intelligence techniques, permits the addition of new agents effortlessly, guides the testing of different techniques, and enables the distribution of agents when computational resources are limited.

The incremental development was extremely important for the development process of the *LearnAgents*. If all the techniques were added at once, we would never be able to evaluate the individual performance gain of each technique. This step-by-step process also prevents the development of agents that deteriorate the performance, because we only include techniques that present improvements. The process presents an easy method to add new agents with different techniques, and minimizes error development in concurrent and distributed agents.



**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 5 - The Scores for the Finals of the TAC 2004

The figure 5 presents the final results of the *LearnAgents* in the 2004 TAC Classic tournament. The tournament had 14 participants from universities, research institutes and technology companies from around the world (USA, Brazil, France, Sweden, Netherlands, Israel, Macau, China, Japan, etc.). The *LearnAgents* finished the tournament in the third place, and in figure 6 we present the statistics of the 35 games played in the final round.

An interesting observation is the similarity between the best score in the simulation process (3705) and the score in the final round of the competition. Although dummy agents use very naïve strategies, our system presents a strategy that is able to adapt to different competitors in an efficient manner. We also present in figure 6 a graph with the statistics of the 35 games played, and we can observe in this highly competitive environment that the *LearnAgents* only obtained positive scores. This fact proves the robustness of the system in such environment.

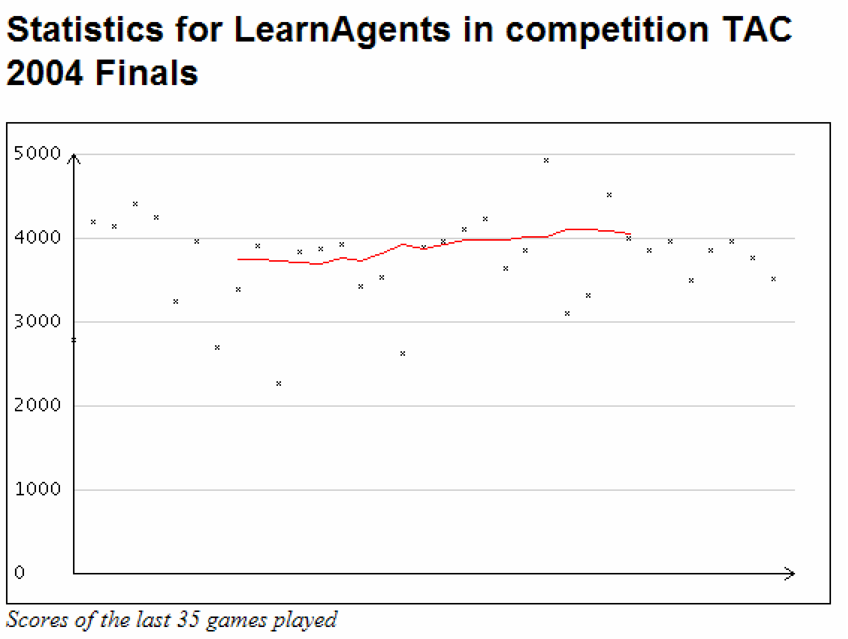


Figure 6 - The Statistics of the *LearnAgents* in the final round

## References

[1] M. Wooldridge. ***Intelligent Agents***. In: G. Weiss. Multiagent systems: a modern approach to distributed artificial intelligence. The MIT Press, Second printing, 2000.
[2] J. Ferber. ***Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence***. Addison-Wesley Pub Co, 1999.
[3] Alessandro Garcia, Carlos J. P. Lucena, D. Cowan. ***Agents in Object-Oriented Software Engineering***. Software: Practice and Experience, Elsevier, May 2004, pp. 1-32.
[4] José Alberto R. P. Sardinha, Ruy L. Milidiú, Carlos J. P. Lucena, Patrick M. Paranhos, Pedro M. Cunha. ***LearnAgents - A multi-agent system for the TAC Classic***. Poster Session at The Third International Joint Conference on Autonomous Agents & Multi Agent Systems (Trading Agent Competition), July 2004, New York, USA.
[5] Trading Agent Competition (TAC), *http://www.sics.se/tac*
[6] TAC Team: Michael P. Wellman, Peter R. Wurman, Kevin O'Malley, Roshan Bangera, Shou-de Lin, Daniel Reeves, William E. Walsh. ***Designing the Market Game for a Trading Agent Competition***. IEEE Internet Computing, pp. 43-51, March/April 2001 (Vol. 5, No. 2).
[7] Raghu Arunachalam, Norman Sadeh. ***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.

[8] Viviane T. da Silva, Ricardo Choren, Carlos J. P. Lucena. ***A UML Based Approach for Modeling and Implementing Multi-Agent Systems*.** The Third International Joint Conference on Autonomous Agents & Multi Agent Systems, pp. 914, July 2004, New York, USA.
[9] M. Wooldridge, N. Jennings, D. Kinny. ***The Gaia Methodology for Agent-Oriented Analysis and Design***. Journal of Autonomous Agents and Multi-Agent Systems 3 (3) 285-312. Kluwer Academic Publishers, 2000.
[10] James Odell, H. Van Dyke Parunak, Bernhard Bauer. ***Extending UML for Agents***. Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, 2000.
[11] Ricardo Choren. ***Uma Linguagem de Modelagem para Sistemas Baseados em Agentes***. Ph.D. Thesis. Computer Science Department, PUC-Rio. December 2002.
[12] Alessandro F. Garcia, Uirá Kulesza, José Alberto R. P. Sardinha, Ruy L. Milidiú, Carlos J. P. Lucena. ***The Learning Aspect Pattern***. The 11th Conference on Pattern Languages of Programs (PLoP2004), September 8 - 12, 2004, Allterton Park, Monticello, Illinios, USA.
[13] José Alberto R. P. Sardinha, Alessandro F. Garcia, Ruy L. Milidiú, Carlos J. P. Lucena. ***The Agent Learning Pattern***. Fourth Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP'04. August, 2004, Fortaleza, Brazil.
[14] José Alberto R. P. Sardinha, Paula C. Ribeiro, Carlos J. P. Lucena, Ruy L. Milidiú. ***An Object-Oriented Framework for Building Software Agents***. Journal of Object Technology. January - February 2003, Vol. 2, No. 1.
[15] Dash Optimization. *http://www.dashoptimization.com*
[16] Stuart Russell, Meter Norvig. ***Artificial Intelligence***. Prentice Hall, 1995. ISBN 0-13-103805-2.
[17] Tom M. Mitchell. ***Machine Learning***. McGraw-Hill, 1997. ISBN 0070428077.
[18] Bowerman, B. L.; O'Connell, R. T. ***Forecasting and Time Series: An Applied Approach.*** Thomson Learning, 3rd edition, 1993. ASIN: 0534932517.