11

Development and Evolution of the Tiancalli Project

Macías Galindo Daniel, Vilariño Ayala Darnes and López y López Fabiola
Benemerita Universidad Autonoma de Puebla, Puebla, Mexico

1. Introduction

Every year since 2003, a group of universities and research centers compete against each other to prove mechanisms for supply chain situations, in a simulation known as The Trading Agent Competition: Supply Chain Management Game (TAC SCM). Several common situations such as customer and supplier care, and storage management are considered as important but should always be well-balanced in order to obtain the highest amount amongst other five agents in competence. With the purpose of participating on the TAC SCM, the Tiancalli Project was created since 2004. Some experiments were conducted in order to develop the first agent –Tiancalli 2005- which participated in the current SCM competition, and since then, evolutions and changes over the structure of the agent have been done to create even better agents for fast decision-making.

In this paper, we would like to present the course of this research, detailing it as much as possible. Thus a comparison between the three main versions of the Tiancalli agent, from 2005 to 2007 will be presented, in order to prove if the current agent *plays* better than the previous ones. In order to demonstrate the improvements obtained with every agent, an experiment with 100 games -25 for each version- has been designed and presented on the latter pages of this chapter. An analysis for each of the three agents is performed, by measuring the results obtained from each simulation.

It is a fact that Tiancalli 2005 is more *reactive* than the 2007 version, which is presented as an *intelligent* agent. However, this fact will be discussed and even proved on this paper. This analysis will also serve to suggest new implementations for the new version of the agent, which will be participating on the 2009 game –due to the 2008 competition has already taken place.

The metrics proposed and generated and the strategies and algorithms developed to make Tiancalli an agent capable of winning a higher amount of orders and a null generation of late deliveries are exposed. These both points are considered the most important on developing a successful SCM player agent.

2. The trading agent competition and the Tiancalli project

Supply chain is the part of management which is "…concerned with planning and coordinating bidding, production, sourcing and procurement activities across the multiple organizations involved in the delivery of one or more products (Arunachalam & Sadeh,

Source: Supply Chain, The Way to Flat Organisation, Book edited by: Yanfang Huo and Fu Jia,
ISBN 978-953-7619-35-0, pp. 404, December 2008, I-Tech, Vienna, Austria

2004)". With the purpose of analyzing the behaviour of dynamic markets based on supply chains, the Trading Agent Research Group has developed the TAC SCM platform. On this simulation there are three main entities: Customers, Suppliers and Agents, the latter also own an assembly factory and a bank account.

Supply chains are the group of processes and activities required in order to transform raw materials to final products. On TAC SCM, a software agent should be developed by a participant institute. This agent should buy components, assemble them to create and sell computers. These activities are performed on a changing environment, against five other agents. The strategies implemented on each agent will allow it to gain customer orders and supplier orders. Finally, this will help the agent to get the highest amount through 220-days simulations.

The use of agent technology is encouraged because of the following features:

- Agents must be enabled to adapt themselves to their environment, and sometimes, with their actions, modify it.
- Agents must be proactive; this means that depending on the see-through capacity of the agent, they must be able to modify "intelligently" their behaviour before an expected situation arrives.
- Sometimes, agents must act as a "reaction" coming from the changes on the environment.
- Finally, the agents must be able to cooperate and collaborate to offer accurate answers, in other words, to work as a society.

2.1 The trading agent competition: supply chain management game description

The simulation consists on developing an autonomous agent capable of performing common activities on a supply chain process. The activities performed are, in general:

- To attend a group of customers interested on acquiring a product, for this game it will be computers. The agent must propose them an affordable price that should be more interesting than the offers made by the rivals.
- To request a group of suppliers to obtain the raw materials to assemble a computer. For this game, the components to produce a PC are: processor, motherboard, memory and hard disk. The agent should be able to request the materials and offer an interesting price to the producers –better than prices offered by the rivals-, in order to acquire components.
- To organize and plan the production and delivery of computers on time to avoid penalties for late deliveries or for storing too many components or computers.
- All the entities of the game are limited in resources: the suppliers have a limited production, and the agent owns a factory that can produce less than 500 computers per day. The participants are then competing against them to dominate the market.
- The market is not a static environment, so the agent must be able to adapt to the current conditions, for example: unavailability of components, preference over certain type of computer, and other conditions which may be changed by the other agents on the competition.

The TAC SCM scenario is formed by six agents capable of producing PCs, a group of customers and eight different suppliers, two for each type of component. The simulation longs 55 minutes that represent 220 TAC days when the agents have to make important decisions about their tasks, in order to obtain the highest incomes and become the winner.

The agent starts without money in its bank account. The common competition is to go from negative amount to positive during the whole game. Then, it is important to obtain components, sell them and avoid the penalties that can be applied for both late deliveries and component storage.
The components available are listed as follows: (a) Processors of 2 or 5 GHz, supplied by Pintel or IMD –so the processors are the only element that can be considered as four different; (b) Motherboard for Pintel or IMD processors, supplied by Basus and Macrostar; (c) Memory of 1 or 2 GB, supplied by MEC and Queenmax; and (d) Hard disks of 300 or 500 GB, supplied by Mintor and Watergate. The total combination gives 16 types of computers and 10 different components to combine.
This is a brief competition of the TAC SCM game. For more details about it, please check (Arunachalam et al, 2003).
As it is mentioned, many universities and research centres have tried to approach the problem from different perspectives. Some of the most remarked techniques include fuzzy logic, stochastic methods, regression trees and prediction techniques, all of them applied on Multi-Agent and Sub-Agent systems.
With the purpose of creating an agent that could participate on the TAC SCM tournament, and following a philosophy of gradual improvements by knowing the basics of the competition, the Project TIANCALLI was created on 2004. The project consisted on presenting an agent for the following SCM games, by a process of reading, comprehension and implementation of several learning techniques. Several experiments which allowed understanding the behaviour of the simulated markets were run before obtaining the first agent, Tiancalli 2005. There were other implementations on the following years -2006 and 2007-, and we postponed the presentation of the 2008 agent because of lack of time, for 2009. With each year, the agent presented many improvements, especially on areas such as:

- The relationship with the customers.
- The relationship with the suppliers.
- The use of the full capacity of the factory during the whole simulation.
- The decisions taken in order to obtain more incomes than expenses.

In this chapter, the full notes of each of the performed Tiancalli agents are presented. It is organized as follows: On section 3, the nine non-so-called-Tiancalli agents are presented as a background of the first analysis made about the TAC SCM game. On section 4, the "Tiancalli 2005" agent and its features are discussed. Then the sections 5 and 6 present the background and the final implementations for the agents "Tiancalli06" and "Tiancalli07", where a subagent perspective was approached. On section 7, a mathematical comparison between the obtained results for these three agents is presented and discussed. Finally, section 8 and the conclusions present the issues not corrected on the previous versions of the agent in order to obtain the principles to construct the Tiancalli09 agent.

3. The agents before Tiancalli

In order to know how to approach to the SCM game, several experiments were conducted to know how the platform functioned. These agents served to know which strategy could work better with the found limitations of our first design, this is why they are mainly divided on two groups:

- Those agents which purchase components after the orders from the customers arrive, which are known as the "Loco_Avorazado" agents –which can be translated as "Mad

and voracious". They were implemented by considering the features of the voracious algorithms. They can be easily described as agents who want to "have customers first, and then find how to satisfy them".

- Those agents which purchase components before the orders from the customers arrive. They are better known as the "Previsor" agents –which can be translated as "Foreseer". Their philosophy is based on the idea of "selling what we already have".

Both approaches served to know how the platform behaved. These agents used essentialy the same commands but allowed to understand how much does a supplier delayed on delivering the items to the agent, and how to control the storage and the assembly factory of the agent.

3.1 Loco_Avorazado agents

A. Loco_Avorazado agent. Two versions of these agents were constructed. The first version had implemented the following behaviour:

- The agent selects its customers by taking the due date of each request. This date must be of eight days after the current TAC day.
- The agent offers to customers at 95% of the customer suggested price.
- The agent requests its components after the offers to the customers have been made.
- The agent chooses its suppliers at random.

The results obtained for this agent are shown in Figure 1. Some observations were made and are described next.

- The agent takes 10% of total requests sent in a TAC day.
- The factory of the agent is used in a 24%.
- The customer chooses this agent in 96% of the offers made by it.
- The agent delivers an average of 80% of the total orders obtained.
- The incomes for this agent are between $4 and $8.2 M, and it always obtains the last place.

B. Loco_Avorazado_2 agent. The second version, named Loco_Avorazado_2, was implemented long after the first version. It implemented several features of the Previsor agents –which will be described next- and it served to discard or continue implementing these agents. Its behaviour can be described on the following quotes:

- Considering the delivery problem of the previous version, this agent chooses customers by taking the delivery date of the request of over 10 days after the current date.
- The agent offers to the customers a static price of 95% of their suggested price.
- The agent also makes requests of components in the same day that the offers for customers are sent.
- The agent chooses suppliers at random. It means that the preference concept is not implemented here.

After implementing the agent and obtaining results, some other facts obtained of the performance of this agent are discussed next.

- Loco_Avorazado_2 takes almost the 25% of total requests made daily by customers, and gains all these requests in a static price environment.
- The agent uses over 60% of factory duty cycles.
- The customers prefer the agent offers in 96% of the cases.
- The agent has incomes from $4 to $37 M and can reach first place in most of the simulations.

3.2 Previsor agents

A. Previsor agent. The agents that belong to the Previsor Series implement the strategy of "first buy, and then sell what was bought". The first version of this agent works as follows:

- The method for handling customer requests chooses them by defining a function value that considers the suggested price and the quantity of computers that the customer wants to purchase. This function is established on (1):

$$f = \frac{reserve\,\mathrm{Pr}\,ice + (quantity * 100)}{2} \tag{1}$$

Where:

- *reservePrice* is the price suggested by the customer (top price)
- *quantity* is the amount of computers requested.

All those request that exceed a *f* value of 1500 are considered to be attended by the agent.

- The assembly of computers is made when the offers are sent by the agent to the customer.
- The agent obtains its inventory a priori based on static daily amounts. Then it has to optimally assign the inventory and the free duty cycles of the factory. When the inventory and the free cycles are calculated for the current day, an optimization problem is organized and intended to be solved. This problem is formulated on (2).

$$\begin{aligned} &Maximize \;\; n_1x_1 + n_2x_2 + n_3x_3 + ... + n_jx_j \\ &adjusted\ to: \quad k_1x_1 + k_2x_2 + k_3x_3 + ... + k_jx_j \le 2000 \\ &\qquad\qquad c_1^1x_1 + c_2^1x_2 + c_3^1x_3 + ... + c_j^1x_j \le 100 \\ &\qquad\qquad c_1^2x_1 + c_2^2x_2 + c_3^2x_3 + ... + c_j^2x_j \le 100 \\ &\qquad\qquad ... \\ &\qquad\qquad c_1^ix_1 + c_2^ix_2 + c_3^ix_3 + ... + c_j^ix_j \le 100 \end{aligned} \tag{2}$$

Where:

- x_j can be binary {0,1}
- *j* is the amount of requests that satisfy the conditions to be considered
- *n* is the product of the quantity and the unit price of the request
- *k* is the product of the quantity and duty cycles
- c_j^i is the quantity of component *i* used in request *j*

This problem is best known in literature as the Multidimensional Knapsack Problem. It can be solved by using different types of algorithms. For this strategy, we solved the problem using a Monte Carlo Algorithm. This algorithm randomly selects the requests that could be satisfied with agent limitations.

With this mechanism as the essential core of this and the following Previsor agents, this version achieved the following results:

- The agent has a customer acceptance rate of 96%, mainly because the agent offered the 95% of the price suggested by the customer.
- Monte Carlo Algorithm is a selection method that is little effective for so many request in a very few seconds. Some days, the algorithm never returns a solution, so the agent cannot take any request. Then, the agent loses a lot of money because of the a priori purchasing system.

- There is an important stock of components and assembled computers at the end of the round, basically because the objective is to always have items on the store. Then a mechanism for managing the inventory is required
- The Previsor agent always makes 6th on any round.

B. Previsor_2 agent. This agent is an improvement of the Previsor agent. It was intended to correct the absence of response by using the Monte Carlo algorithm, by implementing a selective algorithm. Its functionalities are described next.

- The agent chooses its customers by using the same function than Previsor as the first filter to choose customers.
- The agent obtains its inventory by purchasing a constant amount of each component on each TAC day.
- Its customer requests selection works with a selective algorithm instead of the Monte Carlo algorithm of Previsor agent. In this case the agent orders upwards the requests by considering the f value of the first filter, and decides which ones could be satisfied with both the inventory and the free duty cycles limitations.
- The orders for assembling computers are made once the offers are sent to the customer.
- The results of one simulation with this agent are shown in Figure 3. Some remarks on this agent are shown next.
- The customer accepts 99% of the offers made by the agent. So, a feared trouble about the inventory is partially corrected on a static environment –because the Dummy agents do not change their prices.
- The factory of the agent has a remaining inventory –but quite lower than the Previsor agent stock.
- The incomes for this agent are from under zero to $14 M, and sometimes it gets from 3rd to last place. This lack of consistence in the results is not enough to win to a Dummy agent.

C. Previsor_3 agent. This agent became the first winner agent of this project, working over static environments with Dummy agents. Its features are enlisted next.

- The Previsor_3 agent uses the same function to handle customer orders. However, it implements a new component purchase systems that defines certain quotes for purchasing. These quotes are established as follows:
  - A Normal Quote is applied when there are between 500 and 2000 components on stock.
  - An Extended Quote is applied when there are less than 500 components in stock.
  - A Minimal Quote is applied when there are more than 2000 components in stock.

These quotes are applied for each TAC day, excepting the last 10 days when the agent applies only Minimal Quotes.

There is also an important rule about the quotes. The quotes for purchasing processors are always half the quotes for purchasing motherboards, memories and hard disks, because there are four types of processors and only the two types of the other three components. The objective of this rule is to have less stock at the end of the round.

- Computer production is made in the same way than every previous Previsor agent.
- The agent limits its components purchase. This is made in two ways: the agent offers computers until three days before the game ends. Also, the agent considers all the customer requests before the game ends. This is done in order to empty the inventory.
- The results of Previsor_3 agent in a game are shown in Figure 4. Other results are discussed next.

- The agent offers prices in two ways. They could be static at 90.7% or non-static in a random range between 75 and 90.7%. The agent always wins the customer orders, in a static pricing environment.
- There is a remaining inventory of components and computers, but lower than on Previsor_2 agent.
- The agent gets incomes from $29 to $40 M in the rounds. This performance makes that agent to get the first place in all simulations competing against Dummy agents.
- All orders received by the agent are delivered on time; there are very few penalties which do not interfere with the final result in all tests.

D. Previsor_4 agent. This was the latest Previsor implementation, which allowed starting with the development of the first "Tiancalli 2005" agent. Its features are presented next:

- Its behavior is similar to Previsor_3 agent except that this agent has methods to sell the assembled computers in stock to other customers; it means, in case that a customer does not accept the agent offer, the assigned computers for this sale are reassigned to other requests, and so until they are out of the stock. This was implemented after developing a prototype of Vendedor_Inmediato agent, a hybrid that will be discussed next. • The agent also offers non-static prices, depending on the behaviours of the other agents in the current game. As this was the latest agent of the platform research phase, it was intended to give this agent the chance to change the prices given to the customer. This was done by considering the acceptance and reject of the offers sent by the agent, and the price was moved from 75 to 95% of the customer suggested price.
- The results of the performance of the Previsor_4/4x agent are shown in Figure 6. Some other interesting facts of this agent are shown below.
- This is the first agent that implements a system of non-static prices which is not random –as on the hybrid agents shown next.
- Almost all deliveries are on time. There are very few penalties which do not affect the final result in all tests.
- There is an inventory remaining of components and computers. It is the smallest of any other implemented agent at this point of the work.
- Previsor_4 gets incomes from $19 to $30 M in test rounds against Dummy agents.
- This is the first agent that was tested to compete against the other ones implemented in this work, and always reaches first place.

3.3 Other agents

Some other hybrid agents were created, in order to test other possibilities for analyzing the platform. They are described below.

A. Vendedor_Inmediato agent. This agent implements the same function for the suggested price and computers quantity that Previsor series. Also it is a testing agent for computer inventory handling, which leads to the development of Previsor_4/4x agent (described previously).

This agent purchases the components a priori. When the components are delivered, it orders the factory a constant number of computers that must be assembled. Then the restrictions for this agent are not the components nor duty cycles, but the assembled computers currently in disposal.

This agent could have been the best agent in the phase, but due to the lack of time to develop it, it was left as a reference intend for the agents developed on the following years.

It was a good step to avoid the dependence on the factory cycles, but the main situation was that a higher penalty is applied for computers than for components. So in a real competition, to risk this situation is very difficult.

B. Previsor_Avorazado agent. It was a combination of both Previsor and Loco_Avorazado strategies. It takes the early purchase system implemented in Previsor agent and its filters to consider requests and assembling, and the ability to search for more incomes with the due date mechanism of the Loco_Avorazado agent.

- The agent chooses customers limited by components in store and duty cycles. It also chooses extra customers that their due date is over 10 days, both of them must previously satisfy function value.
- The offers made by the agent have static price about 90.7% of the suggested price for customers –this value was considered from previous agents.
- As the agent chooses the selected requests, an extra amount of no more than 10% of the total production is sent to the "long-term production". This means that the agent should ask for these extra components and try to assemble these computers on the following five days at least. However, as the factory is mostly used at full capacity, it was not clearly defined how to obtain this daily 10 percent required for extra orders. It was mostly obtained from rejected offers.

The performance of Previsor_Avorazado agent in a game is shown in Figure 9. Some other observations are made below.

- The agent gets all the orders from the customers, because of its static prices of 90.7%, which can not be improved by a Dummy agent.
- The factory uses over 60% of its daily duty cycles.
- Some orders which should wait for components are sent before the components are received. Then some orders are delivered late or never delivered, or the buy ratio is smaller.
- There are more outcomes than incomes. However, in some simulations, the agent gets first place and in the worst case, the 4th place.

4. The Tiancalli 2005 agent

The presented background is quite consistent to provide tools to develop the first version of the "Tiancalli 2005" agent. As it was explained before, the agent is based on the implementation of Previsor_4x agent, but it was modified to consider the market price changes which occur through the competition.

As a first effort to get a well-defined structure of the agent, it was partitioned on three submodules, which persisted over the design of the following versions of the agent. These modules are explained next.

A. Customer Selection System (CSS). The submodule included all the behaviors to choose and give a price to an expected customer. To choose a customer, the agent has two established filters. On the entire incoming request, first the agent has to consider that the customer must offer a higher amount than the production cost. This cost is calculated with the function seen on (1) where, as previously seen, *reservePrice* is the customer suggested price, and *quantity* is the amount of computers requested by the customer. The f parameter is a numerical value which was determined through several calculations as the average cost for any computer – not exactly the real one, however. Thus, if the request exceeded an f value of 1500, it is considered as winnable, which means that the agent may get an income for delivering this

order in time. Then the request passes to the second filter. All the approved requests are ordered from top to down considering the *reservePrice* offered.

The second filter is applied once the Inventory System has determined the current inventory. Then, the entire ordered requests are subject to the availability of components in stock. If there is enough stock to satisfy the complete order, either by:

- using already-assembled computers -and if that decision could not affect another previously made orders; or,
- using the stock components –which will require duty cycles from the factory.

Those requests that pass the second filter are then considered as *offers*, and the next step is to define the price for the customer. However, to use already-assembled computers, the agent must have considered this order *delayed*, in other words, that the customer is no more expecting the requested computers. In most of the cases, the agent only offers computers that still should be assembled.

As the offered price has to be at least equal or less to the customer suggested price, this price can be calculated in two different ways:

a) Mirror Pricing. When the *reservePrice* is higher than the base price defined by the agent during the simulation, Tiancalli uses the Mirror Pricing Strategy. It consists on applying the function (3) to the *reservePrice* defined previously, as shown.

$$off\ \mathrm{Pr}ice = \left(2 - \frac{reserve\,\mathrm{Pr}ice}{base\,\mathrm{Pr}ice}\right) * reserve\,\mathrm{Pr}ice \tag{3}$$

The basePrice applied is the static price of a computer by considering the initial price of the components required to assemble it. Commonly, the obtained *offPrice* will give a lower price than the customer suggested price. Otherwise, then the next strategy is used.

b) Factor Discount. In the special cases where the *reservePrice* is smaller than the base price, the Mirror Pricing strategy could give a higher price than the customer price. Then another strategy should be applied. A factor discount for each type of computer –which is stored in the Inventory System- is taken and applied to the price obtained with the previous strategy. It is certain that always the price will be lower than the original customer price, however if it fails, the Inventory System can apply another discount –this will be discussed on next. The function used then in this strategy is represented on (4):

$$offPrice = suggestedPrice * discountFactor \tag{4}$$

This discount factor is modified for each computer type at the end of each TAC day, using the following algorithm:

```
For each computer i[1...16] and this i computer discountFactor
df[0...1]
              Calculate Factor = orders received / offers sent
      If Factor value goes:
                  under 30%: lower df by 5%
            between 30 and 90%: lower df by 1%
            over 90%: wait for 5 days with the same df value; if the
            condition is maintained, increase df by 1%.
Next i
```

The orders received come from the offers made on the previous TAC day –so the offers must be saved on the agent IS. If just a few orders –or none- have been received with this discount factor, it means the offered price is too high, so it must be highly decreased. If most of the offers become orders, a trigger is thrown in order to expect good results with this price and then, if this condition is maintained on the following four days, the discount factor is increased.

There is an additional strategy that is normally applied after the TAC day 200. It is applied to decrease the amount of components in stock. However, this strategy can be applied if the agent has not gained any order on five straight TAC days. With this strategy, the agent fixes the discount factor under 50%, as an emergency method to gain customers.

B. Supplier Purchase System (SPS). The platform had a bug which allowed agents to make big purchases at the zero day of the competition –the configuration day- so as it was unknown to us once the competition started, a very primitive strategy was implemented, in order to get components for five non-consecutive TAC days. After the "zero day", and through the first ten TAC days, Tiancalli tries to purchase components for making an initial stock. So the agent does not offer computers until TAC day 11. Then the purchase of components becomes somehow difficult and limited because of the limited production due to the zero day bug.

To calculate an accurate price for the supplier, a similar idea to the customer pricing was implemented. In this case, instead of a percentage, a binary parameter was used –if the request became an offer from the supplier or not. On the first case, a trigger to wait if the price is accepted on the following three TAC days is thrown; then the offered price may go down a percent unit. On the latter case, the factor for the component must be increased 2% and expect the next day result.

As it is shown, the agent only makes one request per day for each component. This was procured in order to maintain certain organization with the requests; however it is insufficient to reach the half of the game with many components in stock to keep production levels high.

The SPS communicates with the IS as it holds the component prices and the pricing factor committed previously. Depending on the level of stock maintained for each component, the agent may request a different amount of components, which is explained next, as the quota system was explained before in the paper:

- A Normal Quote requires 100 components.
- An Extended Quote requires 250 components.
- A Minimal Quote requires 10 components –this quote is commonly used to poll the market.

The proposed quota must be modified as each TAC day passes, for example: a Normal Quote applies for 500 to 2000 components on the early competition days. However, the quote is reduced on 10 to 100 as the simulation comes close to the end –approx. on day 200.

C. Inventory System (IS). The IS works more as a database than as a system, because it stores most of the factors, prices and numerical considerations applied on the performance of the agent. This system offers limits for prices –for example, no discount factor for computers may go under 40%, however it is a possible situation for any game.

The IS is modified by the methods applied within both CSS and SPS, considering the customer acceptance ratio –the amount of orders against the one of offers sent by the agent-, and the supplier acceptance value –if it was accepted or rejected.

First the Tiancalli agent was able to choose a discount factor of just one value for all the computers and another for all the components. It was necessary to implement an array of sixteen values for each computer and ten for each component. These arrays -which represent discount factor for each component and computer-, were used to give the agent an individual perspective of each market element during the simulation.

D. Issues related to the performance of Tiancalli 2005. The Tiancalli 2005 agent had a desirable performance, by achieving the mid-table during most of the competition and reaching the Quarter Finals. The main issues found with this agent are enlisted next:

- The final implementation of the agent was created during the competition, so many modules and methods which were programmed on the agent do the same.
- The methods for modifying the discount factors are very restrictive and are commonly modified during a competition. The agent goes from 95% to 40% and resets to 95% in no more than twenty days, so the prices do not remain static in most of the time.
- The agent does not take automatically any learning from each game, as the values are reset on each simulation, and if some learning is acquired, it is applied by the programmer and not the agent itself.
- The simulation was not completely understood, and several facts, such as the factory usage, the punctual deliveries and the disposition of components, did not work properly, delivering several penalties to the final results of the agent in each game.

These are some of the issues that are expected to be corrected on the following version for 2006. However, for a better comprehension and review of the previous agents and the performance of Tiancalli 2005, we encourage you to read (Macías et al, 2005).

5. Development of the agent Tiancalli06

Tiancalli06 is a software agent that develops real-time techniques such as case-event learning. This allows the agent to improve to the Tiancalli 2005 agent on the following tasks:

- It has an improved mechanism for managing the component purchase, in order to become the preferred customer to the suppliers. The agent offers a better price based on the current situation of the game, in order to dispose of components during the whole game.
- The price selector for customers is improved by considering several other factors such as the current day and acceptance in the market for each computer.
- The factory is better organized and the agent intends to keep the lowest stock possible to satisfy orders on the following days.

This agent is organized on a better architecture which has the same three modules –defined from now on as sub-systems- of the previous agent, but well defined and separated. Then, if a modification is required, the programmer may direct easily to the required class and update it without affecting the normal performance of the agent. There are other classes included on the architecture of the agent that keep the game information inside the agent, and allow displaying the relevant information of a game on output files. The architecture is displayed on Figure 1.

A. The Customer Selection System version 2(CSSv2). For the analysis of the customer requests, the agent considers and makes the following activities:

- At the beginning of the game, the agent applies an initial discount factor for each type of computer –this discount factor works as in the previous version.

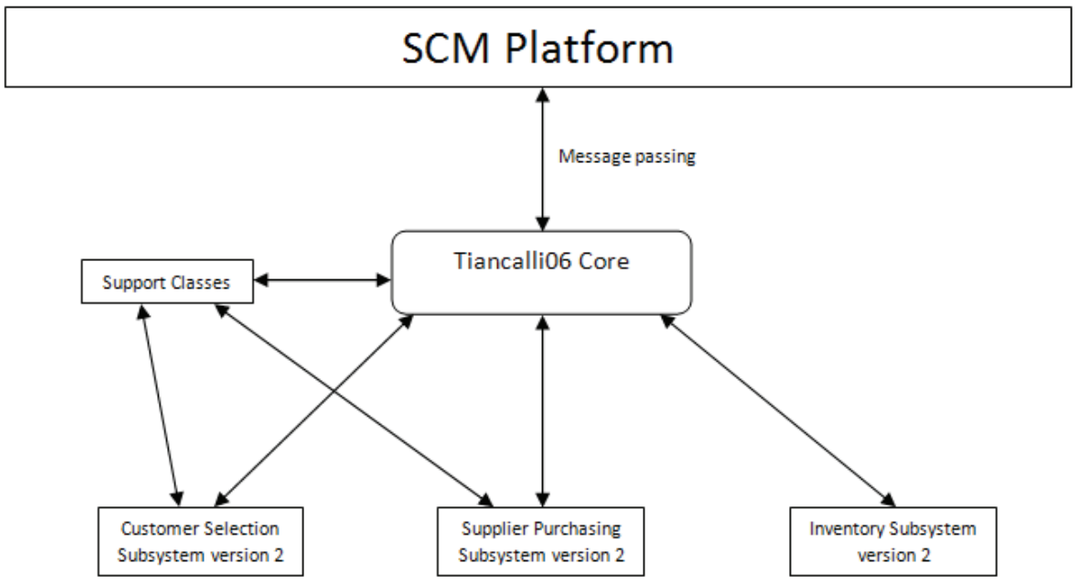

Fig. 1. The architecture of the agent Tiancalli06.

- From the TAC day 11 and until day 216, the agent analyzes the customer requests –the final day was decided because this day is the latest when the agent could delivery an order because of the process that will be explained later.
- The analysis of requests is done considering the available inventory for the next TAC day and if the request may produce higher incomes. The latter is determined by comparing the offered price with a calculated average price, which is described on function 5.

$$csp \times dsf_{pcType} \geq mp_{pcType} \tag{5}$$

Where *csp* is the price that the customer is willing to pay; *dsf* is the discount that the agent offers to the customer for this PC type (*pcType* goes from 1 to 16); and *mp* is the lowest price for a computer determined by the agent. This value is calculated by adding the average price of the four components that form any computer. Hence the first filter considered for the previous agent was changed for this process on this version. The entire group of requests that satisfy this requirement will pass to the following filter.

- The second filter works as on the previous version, any request for there is either enough both components and duty cycles, or enough free computers in stock, will be considered as offers. The determined price for these offers will be these which were calculated on the first filter –then, to save time the agent saves the price when the first filter is passed so when the offer is sent, it is sent with the stored price.
- The discount factor can never be more than 100% -in fact, it is never over 95% for most of the games-, or less than 42%. This is done in order to offer an optimal price to the customer that guarantees an even little income.
- The mechanism for changing the discount factor was slightly changed from the previous version. The calculation is the same; however more situations to move the prices are different, with this a better movement of the prices is allowed, and the agent

may even try with high prices for the customer. Then the function to determine the discount movement is shown next.

```
factor = (orders from the current day) / (offers from the previous
day)
If Factor is:
- Zero, then to lower the discount by 10%
- under 20%, then to lower the discount by 2%
- between 20 and 50%, then lower the discount by 1%
- between 75 and 90%, then keep the discount
- over 90%, then increase the discount by 1%.
```

- The complete routine to deliver a PC to a customer consists on three TAC days, as following:

DAY d: The agent receives the request and sends an offer. Then the agent plans the factory usage for the following day –if required, only if the computers are not still assembled. This behaviour is approximated to the one implemented on the Previsor agents; however the computers are assembled before the agent receives any response from the customer.
DAY d+1: The agent receives the orders from the customers; if not, the rejected computers will be, on the following day, disposed for other requests. If the orders are received, the agent plans the delivery of the computers for the following day –because the computers are being assembled on the current day.
DAY d+2: The request is satisfied, and the agent expects the incomes on the following days –for the agreed date.
With this new structure of the customer attention, the agent is pretended to deliver on time all of its orders. It is expected that the agent increases its acceptance amongst the other agents, by evaluating simpler parameters for giving prices to the requests received.
B. Supplier Purchase System version 2 (SPSv2). For the interaction with the suppliers, the second version of this agent implemented the following activities:

- An improved purchase for the zero day. With this, the agent gets approximately the 35% of the total purchase of the game on the first day. The agent requests the components on determined days which are considered as crucial. The days are 9, 25, 50 and 100.
- The pricing for the suppliers is proposed on the same way that the previous version of the agent, with a specific discount factor for each type of component –not so for each seller. Each TAC day, the agent requests no more than 100 components to each supplier.
- The price is influenced by the acceptance or rejection of the price of the previous days, and each day that the price is accepted, the system receives and stores this price and calculates an average with the previous accepted prices, in order to propose the computer base price that is used on the CSSv2 system. Then the discount factor movement is determined with the following pseudo code:

```
If request is accepted
  then decrease discountFactor by 1%
Else
  increase discountFactor by 1.5%
```

- The system pretends to offer an initial purchase volume of approximately 70,000 computers per game –this was a calculation employed to determine how many computers an agent can assemble using the total factory cycles and components.
- Obviously this quantity can be affected by the market preference –maybe one type of computer is more purchased than the other-, so the system must be able to determine when a computer is preferred, and intend to offer more of these computers. Then, the system fixes at the start of a game, an initial quote of 70,000 components. When this quantity is almost to be reached and there are more TAC days following, this quote is modified as follows:

```
for id = 1 to 16, do:
  if (initialQuote (componentid) * 0.9) • soldComponents(id))
    then initialQuote(componentid) is increased by 500 components
```

- It must be noticed that the count that matters is the quantity of components sold and not those purchased. This feature is intended to decrease the amount of stored components at the end of the game.

C. Inventory System version 2 (ISv2). On the new version of the IS, a more precise calculation for stored components and computers is obtained daily. Before deciding if most components must be purchased, the real quantity available of components available in stock is determined with the function (6).

$$realInventoryForNextDay = componentAcquiredYesterday - componentSoldYesterday \quad (6)$$

This formula marks a significant difference against the agent of 2005, because it only considered raw components that were purchased and not those which are still waiting to be assembled. Also, this version considers computers in stock that have not been sold. The agent has on disposal such components –for they need duty cycles to become computers- and computers to be offered to the clients. However, this system is centralized and offers the inventory and the calculations to perform the operations of the other two subsystems.

D. The performance of Tiancalli06. The main issue that Tiancalli06 deals with is the price estimation. The changing conditions of the market and the changes that several agents try to impose on the competition are the most difficult challenges for a very sensitive mechanism of pricing. However, the performance of the agent was acceptable, especially on the Seeding phase of the competition.

The agent achieved a constant acquisition of orders through most of the games; however the agent had difficulties in most of the games to get orders due to this pricing mechanism. Anyway, the most important achievement of the agent was the avoidance of late deliveries. The agent deliveries all of its orders in time, and just one game had penalties because of a connection issue at the end of the game. Finally, the storing prices were reduced because of the new quotes mechanism for purchasing components that were described on the SPSv2.

For second year, the agent lasted until Quarter Finals and achieved a final 18th place. This year, the competition presented more agents and the improved versions of the previous participants.

Although the advances were significant, the next version of the agent should offer better pricing systems. Some of the other agents implemented fuzzy logic or prediction heuristics to determine the prices. Then the next effort for Tiancalli07 should include an intelligent mechanism for pricing. The results obtained with the agent can be consulted on (Macías et

al, 2006, 2) and an interesting comparison about the results of both agents Tiancalli 2005 and Tiancalli06 can be found on (Macías et al, 2006, 1).

6. Development of the Tiancalli07 agent

In the effort of improving the performance on the activities that the agent does on the competition, the agent Tiancalli07 was developed. This agent features new prediction techniques to give prices to both customers and suppliers. The obtained prices are stored in external files, which are updated during the current game, and used for the upcoming games. Since the beginning of the participation of the Tiancalli agents, any of the developed agents took experiences from the previous games and stored the information automatically, so it is the first agent which can be considered as "evolutionary". As it is the last agent developed by the same team project, it should offer a brand new and detailed architecture, conformed by three subagents –as defined on (He et al, 2006)- with specific functionalities, and two extra modules for information control. The general architecture is presented on the following figure.

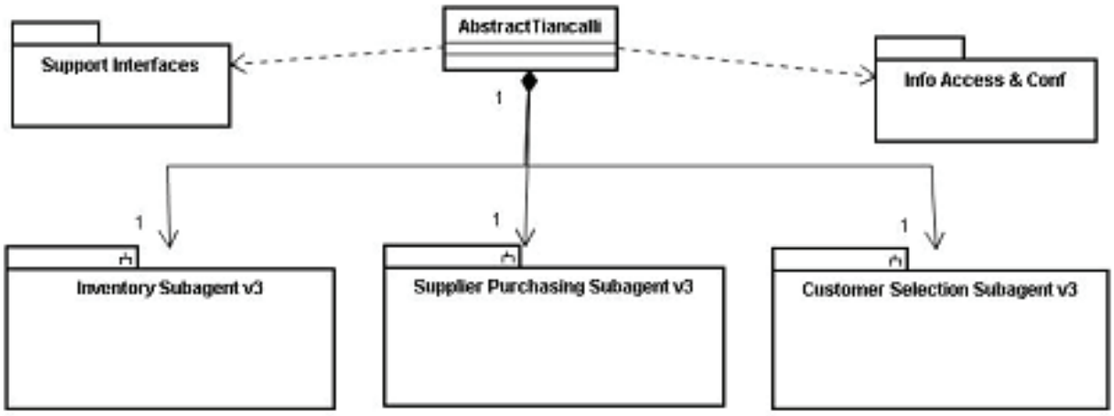

Fig. 2. Architecture of the Tiancalli07 agent.

A. Customer Selection Subagent version 3 (CSSv3). The general problem of attending the customers is subdivided in several areas. This allows the recognition of the main activities in order to correct any significant behavior. Some of these areas are described with their current developed activities:

- Customer selection. This is the least modified area of the general problem. It uses both filters applied since the CSSv2 on Tiancalli06. Just as a reminder, the first filter checks all those customer requests for which their customer suggested price –multiplied by the current discount ratio- satisfy the base price for assembling the current computer type. The second filter checks the availability of both components and duty cycles, or free computers that remain in stock. Those orders that satisfy both filters are considered as reasonable and the subagent then intends to give them a good price.
- Customer pricing. This are, on the contrary, is the most modified of the agent. A regression tree has been designed to have a better statistic of pricing movement. The tree is controlled by three main variables, which are described as follows:
  - Requested quantity of computers. It is required to order each type of computer, from top to down, as all the requests have arrived to the agent during the past

days. For the classification, fuzzy values of maximum, high, medium and low are applied to determine this variable.

- Price ratio. This is determined by considering three prices: pmin which is the minimal price that offered the agent on the previous day and became order; pens that is the base price to assemble the requested type of computer; and psug that is the suggested price by the customer. However only three cases are considered – because the others are illogical:

$$p_{sug} > p_{ens} > p_{\min} \rightarrow case\,1$$
$$p_{ens} > p_{sug} > p_{\min} \rightarrow case\,2$$
$$p_{ens} > p_{\min} > p_{sug} \rightarrow case\,3$$

- Range of days. It considers the day when the computer is requested. The whole competition is divided then in subintervals of ten days, which give a total of 22 different ranges.

Once the concepts for pricing are presented, the structure of the trees is suggested. Sixteen trees, which represent each type of computer, were built to represent the required structure. Each tree has four branches for the requested quantity of computers, then three sub-branches for price ratio, and finally 22 leaves for each sub-branch. This leaves a total of 243 leaves, which include each a range of values –maximum and minimum- that is used to determine the customer price. The leaves also include the acceptance ratio of each leaf during the current game.

The acceptance ratio modifies the range of values as following:

- If the prices are accepted, the tendency is to close the intervals in order to find a convergence on the prices. If the optimal is found, the optimal should tend to increase its value in order to search for more incomes.
- If the prices are rejected, the tendency is to open the intervals, in order to reduce the minimal price and improve the prices against the other agents in the competition.

The trees are updated with each TAC game and stored on external files. These files are loaded at the zero day.

B. Supplier Purchasing Subagent version 3 (SPSv3). The subagent is in charge of performing the following activities:

- Price calculations. In order to calculate the base price that is considered on the first filter of the CSSv3, this subagent must determine a price for each component. This is done by applying the following pseudo code:

```
Variables:
Input: p as the offered Price, q as the quantity offered
Intermediate: qPurchased as the previously purchased amount of
components, avgP as the previous average price, temp as a temporary
variable.

AT THE BEGINNING OF THE SIMULATION:
qPurchased ←q, avgP ← p
EVERY DAY WHEN THE AGENT RECEIVES A SUPPLIER OFFER:
temp ← qPurchased + q
```

$$avgp \leftarrow \frac{qPurchased \times p}{temp} + \frac{q \times p}{temp}$$

```
qPurchased ←temp
```

This calculation generates a base price for each component, which once it is added to the remaining components, can be applied for the base price of each computer. This price is better approached than the previous versions, because the price is determined by the amount of computers which have been already acquired.

- Component purchasing. The subagent recognizes two ways to purchase components: one at the beginning of the game with the zero day purchase, and the other during the competition. The first purchase is intended to get an initial stock for the first orders; the latter purchase is to maintain this stock. As the zero day purchase was corrected by the game developers –you can still buy components but the prices are higher and consider a general statistic for all the requester agents- the amount of components acquired on these days has been reduced; hence the strategy must be improved to acquire the items during the whole competition.

The pricing system implements a single list for each component –ten lists in total- with 22 spaces. These spaces should be filled with the maximum and minimum ranges of discount factor during each 10 TAC days. As it can be seen the structure is similar but simpler than the trees implemented for the customer pricing. The ranges are modified by following these rules:

- If the supplier accepts the discount, the ranges are closed, tending to find an optimal. Once an optimal value is found, decrease this value to find a minimum value.
- If the supplier rejects the price, the ranges are opened by increasing the maximum price, in order to improve the prices and make them competitive.

C. Inventory Subagent version 3 (ISv3). The ISv3 is in charge of organizing production and delivery of computers to the customer. Its main goals are the following:

- Avoid the loss of orders by production delays, and reduce at most the delayed deliveries, if possible nullify them.
- Bring an accurate use of the factory and stock of components and computers, by delivering real statistics of the current situation to the other subagents. Sometimes this statistic must foresee the availability of components that will arrive on the following days.

D. Support packages. The whole operational environment of the Tiancalli07 agent includes two support packages that will be described next.

- Information Access and Configuration Package (IA&CP). This package is in charge of handling the most important data about the current simulation, in order to bring these data fast to any requiring entity. Also several methods for reading and writing external files are included to save the knowledge generated during the simulation. Finally, a configuration file is included, which commands the whole system the files that will be used to generate the initial trees and lists for the current simulation. The configuration file and the files for the storage of the structures are overwritten or substituted at the end of a simulation.
- Support Interfaces Package (SIP). The current package is only used for developing purposes. It implements classes for graphic interfaces to manage the information generated during a game. The programmers can so recognize the behavior of the current simulation with more detail than with the sole Agentware interface –the

original SCM interface included with the downloadable test agent. The developed interfaces are to display information such as: (a) maximal and minimal customer acceptance prices; (b) amount of requests against amount of orders, and (c) demand representation for the customers.

E. Implementation of the knowledge repositories. The files for configuring the agent are stored on a subfolder named "playbooks". The first read configuration file is named "init.tcf" and includes, mainly, the names of the files that include both the trees –p####.cus, where # is a serial number- and the lists –s####.sup. The serial number is updated once a simulation is finished.

The general structure of a customer tree file is the following:

```
p0003    //file name
16    //number of trees included
4    //possible values for the first parameter of the tree
3    // possible values for the second parameter of the tree
22    // possible values for the third parameter of the tree
a0a 90000 100000 10  19
a0b 90000 100000 15  16
a0c 90000 100000 24  30
…
```

The first five lines of the file are explained in the structure itself. The following lines are the ones that represent the tree structure as following: the first letter (a) could have three other values (b, c or d) and represents the demand of the current computer type –where a is *maximum* and d is *low*, as explained before-; then a number (0) can take other two values (1 or 2) and represents the price relationship; and the last letter (a) represents the segment of time on the competition where the computer is requested –it may have 22 values from a to v, where a is the segment from day 1 to day 10 and v the segment from day 211 to 220.

The following two quantities -90,000 and 100,000- represent the percentage applied for the discount factor. They are not decimal values because of the expense to store a *double* value, and the capacity to represent exact quantities which can not be represented with a wide double variable. This is explained as follows:

Considering the minimum amount that can be applied to the percentage discount so that you can discount one unit of money, the obtained number is 0.00012. So if for example, the range is between 90,000 and 90,012, the price may vary in just one unit, maybe the price will be 1500 or 1501 with this range. The value of 0.00012 is considered as the less significant number in the price calculation. This is why an integer –formed with the double value and multiplied by one hundred thousand- is required to store this factor.

The last couple of numbers represent the ratio of orders against the offers proposed. So in the first element of the tree –a0a- it can be observed that this leaf has generated 10 orders of 19 offers to the customer. Then the efficiency of the leaf can be also obtained.

This structure of the file and the tree allows the system to obtain easily the leaves, so the time required to seek the leaf is minimum. But the structure to store the prices for suppliers is not so different. The list is stored on a file named "s###.sup". Here an example is presented:

```
s0003
1
16
22
```

```
aa 74450 74453 11  13
ab 69550 72864 8  10
```

The file stores the name of it, which is s0003. It will propose the construction of only one list of 16 x 22 elements (the sixteen components divided by supplier and the 22 ranges of TAC days, as explained previously. The list is represented now with two elements –aa, ab- which represent, respectively, the component type and the range of days.

F. The performance of the Tiancalli07 agent.

The results obtained with this agent were compared with the previous two agents by separate on (Macías et al, 2007). However, the results in the competition were not as good as expected: the agent achieved again the 17th place. The comments about this result and several remarks on what issues should be corrected on the next version of the agent are discussed on the conclusions of this paper.

7. Experiment

In the previous lines, all the Tiancalli agents that have been developed until now have been described with detail. As it has been discussed, the intention of building an intelligent agent that can participate on the TAC SCM game has been reached with Tiancalli07. How the intended mechanisms worked in order to obtain this intelligent behavior, is one of the goals of the current paper. To prove the thesis that Tiancalli07 works in a more intelligent way than the two previous versions of reactive agents, the following experiment has been developed.

Over a local computer with the SCM platform, the three agents have been installed and configured to play one against the others on fifty games. The other three agents participating are *dummy* agents, in order to not alter the basic results –it has been tested that when many agents are running on a local platform, the messages are not well received by the agents.

The average results are presented on Table 1.

The following lines are a discussion of the obtained results on the experiment.

- The agent that receives more money from its customers is Tiancalli 2005, closely followed by Tiancalli07. However, Tiancalli 2005 is the agent which expenses more money to assemble its computers. This can be explained as follows: the agent of 2007 has improved its mechanisms to purchase components with lower prices, and sell computers to higher prices. This difference can be explained by consulting the parameter "Ratio S/P" that is intended as the amount of incomes divided against the amount of expenses, or in single words, the benefit that the agent receives for each $1 that it invests. Curiously, this is the only important parameter where Tiancalli06 is over Tiancalli 2005.
- The agent Tiancalli07 remains more time on the competition with a positive balance. This is also reflected on the final result of the competition, where the same agent also takes the lead.
- The amount for storing components and computers is the highest on the agent of 2007. It may be explained by considering that the agent risks more the prices to the customers, and many more computers remain in the current inventory. Also a huge problem for this agent is that when purchasing components, the market tends to offer more components of one type instead of another –for example, there are more processors than motherboards. The absence of one type of component will lead to have a big inventory during days –the days that will take to the agent to get the absent items-, and so the storage costs will be higher.

Agent	Tiancalli 2005	Tiancalli06	Tiancalli07
Revenue	91,535,200.85	33,321,284.62	85,679,710.08
Interest	142,610.15	82,597.46	342,191.54
Material	78,282,031.77	25,593,433.46	56,140,179.23
Storage	1,181,238.23	647,656.08	1,841,481.00
Penalty	288,055.54	0.00	0.00
Penalty%	0.4%	0	0
Ratio S/P	115%	127%	148%
Result	11,926,485	7,162,793	28,040,241
Orders	3,976	1,833	5,122
Utilization	73%	24%	56%
Delayed	27	0	0
Missed	8	0	0
Del. Performance	99%	100%	100%

Table 1. Results of the experiment of fifty simulations between Tiancalli and *dummy* agents.

- The only agent that receives penalties for late or missed deliveries is the 2005 agent. As it is shown, the amount of penalties represents less than 1% of the total outcomes that the agent makes during the game. It is not an important quantity, however, while it is kept as low or null as possible.
- The difference on the final result of an average game of Tiancalli07 is twice and a bit more of the result of Tiancalli 2005. The mechanisms implemented on Tiancalli06 allow it to obtain better prices than the agent of 2005; however the amount of components acquired is not enough to produce computers.
- The agent of 2007 obtains more orders, but uses the factory less than Tiancalli 2005. This is possible by consulting the formula that Tiancalli 2005 uses to determine the potential customers; it tends to accept customers which require a lot of computers –but also with a high amount of money to be paid. Then the agent will require assembling more computers. The case with Tiancalli07 is that the agent takes all the possible requests which may generate an income, so the quantity requested is not so important; then the agent may satisfy more orders without caring the requested quantity of computers.
- It may be discussed that "Tiancalli06 was an evolution over Tiancalli 2005" due to the poor obtained results. However, an important argument to defend this hypothesis is the ratio of sells against purchases, where it is clearly noticeable that Tiancalli06 obtains more money per invested unit than the agent from 2005. The mechanisms for purchasing components were not substantially different on Tiancalli06, but it is possible that maybe installing SPSv3 on Tiancalli06 the results obtained would be better.
- On the experiment, it is important to notice that in most of the simulations, Tiancalli07 obtained the first place; however, just in four simulations, the agent descended to second or even fourth place. This argument can be justified with the slow learning curve that the agent performs for new prices and conditions of the market. Tiancalli 2005 obtained only one second, two fourth places and fifth place in the simulations, and Tiancalli06 always obtained sixth place.

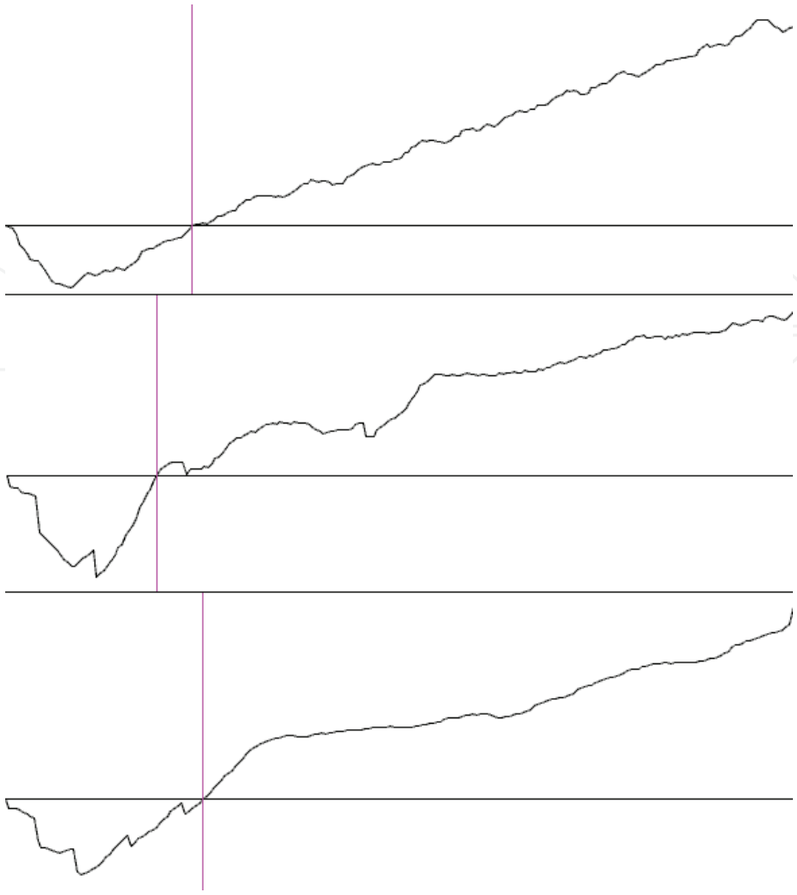

Fig. 3. The balance of the bank account of Tiancalli 2005 (top graph), Tiancalli06 (middle graph) and Tiancalli07 (lower graph). The vertical lines show the moment when the agent has a positive balance.

- On Figure 3, an example of a simulation is shown, and it displays the balance during the simulation time. The vertical line shows the moment when the agent passes the negative balance, and surprisingly, the agent that achieves this first is Tiancalli06, on day 42. Both Tiancalli 2005 and Tiancalli07 obtain the positive balance over 50 days of competition. However, in the graph can be seen that the agent from 2007 has a more constant line than the other agents, and Tiancalli06 has the most erratic line, due to the lack of components during most of the competition.

8. Conclusions and future work

This paper presents the deepest analysis about the construction of the Tiancalli agents since 2005. It intends to describe all the effort that has been developed and both the evolution and experience acquired during the three years participating on this competition, the TAC SCM. It is remarkable that the agent concepts have evolved from Tiancalli 2005 to Tiancalli07. Certainly, the results achieved with the latest version of the agent are far from the previous results; however during the last competition these results could not be reflected on the reached place of the competition. Several considerations and facts that are planned for the following versions of the Tiancalli agent are the following:

- The proposed learning curve for the learning structures –but specially the tree- is too slow, and many more simulations must be done in order to obtain an agent with enough knowledge. If new conditions are added and the structure changes, it only affects to the leaves that participated on the price proposal, the others remain static. So, in order to promote the learning capabilities of the agent, several modifications about the knowledge update must be conducted.
- There is a lot of work done on the pricing system for customers; however the prices for suppliers are the most common issue and the penalties for storing too many useless components –referring to "useless" because they can not be used to assemble any computer due to absence of another component- are still high. There must be an improvement on the IS structure that intends to get the missing components as soon as possible in order to avoid important penalties.
- There must be an incentive to produce more computers on the factory. It is used the half of the total capacity.
- Finally, these improvements should be noticed on the competition, reaching even better results.
- There is still a lot of work to do about the TAC SCM. It is expected that this work serves to promote and invite researchers and universities to participate on the competition. To review more information and results of the Trading Agent Competition and all the contests, it is suggested to check the website: http://www.sics.se/tac

9. References

Arunachalam, R., & Sadeh, N., (2004). The supply chain trading agent competition. *Electronic Commerce Research and Applications 4, 2005*. pp. 63–81. Revised and extended version of paper at *AAMAS-04 Workshop on Trading Agent Design and Analysis*, New York, 2004.

Arunachalam, R., et al (2003), Design of the Supply Chain Trading Competition, *IJCAI-03 Workshop on "Trading Agent Design and Analysis"*, Mexico, August 2003.

He, M., et al, (2006). Designing a successful trading agent for supply chain management. In: *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, pp. 1159–1166, Hakodate, Japan, 2006.

Macías, D., et al (2005). Tiancalli: An Agent for the Supply Chain Management Game 2005. *La Computación en Puebla en el Siglo XXI*, pp. 11-16, ISBN, Puebla, México, November 2005. BUAP FCC, Puebla.

Macías, D., et al (2006,1) Statistic Analysis for the Tiancalli Agents on TAC SCM 2005 and 2006. *In Proceedings of the 15th International Conference on Computing (CIC'2006)*, pp. 161-166, Mexico City, Mexico, November 2006, ISBN, IEEE Mexico,

Macías, D., et al (2006,2). Tiancalli06: An Agent for the Supply Chain Management Game 2006, *Proceedings of the International Conference on Computational Intelligence for Modelling, Control & Automation -CIMCA 2006*, published on CD-ROM, ISBN, NSW, Australia, November 2006., IEEE, Sydney, Australia.

Macías, D., et al (2007) Desarrollo de Agentes Inteligentes para la TAC SCM 2007 (Development of Intelligent Agents for the TAC SCM 2007), *Actas de la XII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA-TTIA 2007)*, pp. 81-90, ISBN, Salamanca, Spain, November 2007, University of Salamanca, Spain.

Supply Chain the Way to Flat Organisation

Edited by Julio Ponce and Adem Karahoca

ISBN 978-953-7619-35-0
Hard cover, 436 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

With the ever-increasing levels of volatility in demand and more and more turbulent market conditions, there is a growing acceptance that individual businesses can no longer compete as stand-alone entities but rather as supply chains. Supply chain management (SCM) has been both an emergent field of practice and an academic domain to help firms satisfy customer needs more responsively with improved quality, reduction cost and higher flexibility. This book discusses some of the latest development and findings addressing a number of key areas of aspect of supply chain management, including the application and development ICT and the RFID technique in SCM, SCM modeling and control, and number of emerging trends and issues.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen