TAC Report
Agent “Tacy”
Trading Agent Competition

By Eric Neher and Pascal Büttiker

Course “Agent Systems”
Teacher: Marie Persson

Version 1.0
March 2013

Table of Content

1 Introduction

The aim of this report is to describe the design and implementation strategies of our Agent "Tacy", which is able to compete in the Trading Agent Competition (TAC) environment.

2 Purpose of Agent System

"The Trading Agent Competition (TAC) is an international forum designed to promote and encourage high quality research into the trading agent problem." [1]

This report, as well as the implementation of Tacy itself was created in the context of the course "Agent Systems" at the Blekinge Institute of Technology. The aim of this assignment was to develop a strategy and implementation with which an agent is capable to compete with other agents in a dynamic environment.

3 Process

At the beginning of the course we tried to understand what an Agent System is and how agents work. To do so we read the book "An Introduction to Multi Agent Systems" from Wiley Wooldridge. Once we understood what agents are and what advantages, disadvantages, problems and solutions come with them we started to get familiar with the TAC Classic Game from SICS. We read the TAC Classis Game description and while reading we began to understand more and more what agents are good for. The TAC Classic as perfect example of how agents try to achieve their design objectives.

In a further step we set up the environment (IDE, TAC Server etc.) and started to analyze the dummy agent. While analyzing we developed strategies of how to improve the dummy agent. We wrote our strategy thoughts down in this paper and started implementing some parts. As we tried to implement our ideas we figured that we are not that happy how the provided AgentWare was set up (just integer values instead of a clean Object-Design using Enums and Classes) which is why we first improved and refactored some parts of the AgentWare implementation itself before we really began to implement our strategies.

4 Design Strategy

4.1 Introduction

Starting to think about strategies of an agent in the TAC Classic an interesting thought came to our minds. We figured that there are two main strategy categories, which are worth some thoughts.

The first is the more obvious kind of strategy:

- Optimization of the own utility

The second category is a quite different approach:

- Reduce the utility of the other agents

Since the second category brings more risks and most of the famous TAC winners followed a strategy of the first category we decided also to stick to an optimization approach. However, a combination of the two strategies might be worth thinking about.

First of all, we had to separate between happenings we can predict and ones we can't. For that reason we created the following list:

Predictable	Not predictable
Flight price tendencies	Initial distribution of event tickets
	Hotel auctions closing
	Entertainment prices*

* Entertainment Tickets are traded between the agents, which is why a free market development, which depends on supply and demand, has to be expected.

4.2 Flights

4.2.1 Introduction

Based on our analysis of the TAC Classic description it seems that the flight auctions are the least complex ones, due to the facts that they are one-sided an there are unlimited tickets to buy. Furthermore, the decision which tickets should be bought is pretty straightforward, since the clients provide us with preferred arrival (PA) and departure (PD) dates. However, it is interesting to ask which the best time to buy a certain ticket is. Flight prices start between $250 and $400 and change every 10 seconds by a value based on a certain calculation to a minimum of $150 or a maximum of $800.

The flight prices tend to get more expensive over time, which means that buying flight tickets early seems legit. However, a ticket that is cheap and fulfills our need to match with a PA or PD of a client is not necessarily useful. We have to consider the following dependency with the hotel rooms:

- Flights without the corresponding hotel rooms are useless
- Hotel rooms without the corresponding flight are useless

Since we cannot sell a flight once we bought it, we have to make sure to not waste money on useless flights or hotels. It is worth mentioning though, that a bid on a flight can be withdrawn if we consider a certain flight as not valuable anymore.

4.2.2 Pricing / Bidding

Tickets prices tend to rise over time, but there is still a little chance in the beginning of the game for prices to decline. In order to take advantage of the described facts above, we place flight bids at the beginning of the game beneath the ask price with an offset of $15. We analyzed that price falls of $15 is not that uncommon, which is why we choose that offset. So this means our bids won't match at first and become pending bids. If a pending bid of ours didn't match after 3 minutes we will withdraw it and replace it by a bid that will match immediately. This is because we figured that after 3 minutes into the game waiting for flight prices drops is too risky. Furthermore, if the price quotes of an auction raise more than $15 we will withdraw the pending bid and buy the flight immediately even if the 3 minutes mark haven't been passed. This way we minimized the risk of overpaying for flights.

We are aware that in flight auctions there isn't much money saving potential. Nevertheless, we try to optimize our score wherever we can.

4.3 Hotels

4.3.1 Introduction

Hotel auctions are one-sided as well. There are just two hotels selling 16 rooms for each hotel and night. So there are 8 auctions.

Two hotels with each 16 rooms per night equal a total of 32 rooms per night. Assuming that in a game instance there will be around 10 agents each with 8 clients would equals a total of 80 clients. Of course we can't tell how long each client will stay but assuming that a client stays an average period of 3 days in TACCity would mean that there is an overall demand for 240 rooms. 32 rooms are available for each night and the game period contains 4 nights which means there are 128 rooms available in total. Therefore, regarding this scenario, there would be 112 nights on which clients wound not have a room.

If this is a realistic scenario, the competition would be pretty big and we would have to make sure to get all the rooms we need. One of the worst things that could happen is that a package becomes unfeasible because we were not able to buy a hotel room for a night during a client's trip. The consequences would be 0 utility for this client.

We consider this as a high risk and take the following measures to minimalize it:

- Rather invest more money in hotels than taking the risk of not-feasible packages
- Monitor HQW of auctions

4.3.2 Worst-Case scenario

Should the above measures be fruitless and we are not able to buy hotels for a certain day we take the following measures to avoid the worst case of 0 utility points:

- Shorten the stay of client and buy another flight
  - Advantage: package is feasible again and useless entertainment tickets could be sold
  - Disadvantage: travel penalty and costs for new flight

Of course we have to decide if these measures are profitable. For that we compare the new hypothetical utility of the client (measurements would have been taken and the package is feasible again) with the total cost for rescuing the package.

Client example:

PA:	Day 1
PD:	Day 5
HP:	150
AW:	0
AP:	100
MU:	200

We assume we have the flight tickets according to PA and PD, no ticket AP and a ticket MU on day 4. Unfortunately all rooms for day 4 are already sold and therefore we just have rooms on day 1, 2 and 3. All these rooms are in SS. So let's calculate the hypothetical utility if we rescue the package and shorten the trip for 1 day.

1000 – 0 – 100 + 0 + 0 + 0 + 0 = 900

We now have to subtract the cost for the new outfight on day 4. We assume the maximum cost of $800 (resulting in an utility of 100), which is still better than 0. Furthermore, we might get rid of the useless MU ticket and receive some income, or might use it for another client.
We conclude that it is worth trying to rescue a package. Nevertheless, it is it a complex optimization problem, where much more research could be done (if we had the time to do so).

4.3.3 Hotel types

We primary focused our attention to the strategy of deciding which type of hotel we should buy. We assume better hotel rooms (TT) to be more expensive. In order to determine if it makes sense for a client to invest in TT rooms we calculate the total hypothetical costs for the client staying in TT or SS. We then subtract the total SS cost from the total TT cost. This difference represents the additional costs for the client to stay in the TT hotel during his trip.

Since the premium hotel value (HP) is just granted once for the whole trip, we then compare the HP with the additional costs. The Premium Value has to be at least $30 higher than the additional costs. If so we the client agent will request TT hotel rooms, if not SS. Furthermore, we figured that it generally makes more sense investing in TT rooms if a client trip is short rather than long. That is why we added a precondition to our calculation. If the client needs more than 2 rooms for his trip to get feasible, he will automatically skip the calculations and request cheap hotel rooms.

4.3.4 Pricing / Bidding

The fact that hotel auctions close randomly makes the attempt to predict the closure of them pointless. The hotel auctions are the most critical in TAC Classic, as a single missing Hotel room can render a whole package unfeasible.

Therefore, in hotel auctions we try to get our rooms at any cost – as long as the cost are not higher than loosing the whole package. We increase our bid by [current_ask_price + 20$] when our HQW drops beyond what we need.

To avoid the worst case – that is an unfeasible package – we consider shortening the trip. When a hotel room auction for a certain day has closed, and it is clear that we miss a certain amount of hotel rooms, we can reorganize the client preferences and shorten the trip. This normally means that we have to buy an additional flight, and probably sell some already owned entertainment tickets.

4.4 Entertainment

4.4.1 Introduction

Entertainment auctions are the only double auctions in the game which means that agents can act as sellers and buyers. At the beginning of the game we will receive a total of 12 tickets. The possible allocations of the tickets are the following ones:

Day 1	Day 2	Day 3	Day 4
4	4		
2	2		
4	4		
2		2	
4	4		
		2	2
4	4		
	2		2
4		4	
2	2		
4		4	
2		2	
4		4	
		2	2
4		4	
	2		2

Day 1	Day 2	Day 3	Day 4
		4	4
2	2		
		4	4
2		2	
		4	4
		2	2
		4	4
	2		2
	4		4
2	2		
	4		4
2		2	
	4		4
		2	2
	4		4
	2		2

4.4.2 Premium value

Entertainment events contain no risk of making a package unfeasible. They just represent a chance to increase a client's utility. Since the premium values for the entertainment events is a final value between $0 and $200 it makes no sense to sell entertainment tickets for more than $200 and obviously there is also no point of buying event tickets which are more expensive than $200. Or more precisely:

If a client's premium event value is not higher than any current ask price for the corresponding event there is no point of buying this ticket. In order to determine which tickets are worth buying every client generates a list consisting out of all entertainment auctions and their current value for the client. The current value is calculated by subtracting the current askprice from the corresponding premium value of the client. The list is then sorted by the values. A client afterwards goes through every entry in the list (starting at the auction with the highest value) and checks if the day of the auction is a day on which the client still misses an enter-tainment event. This dynamic solution gives us the opportunity of flexibility within the dynam-ic entertainment auction market.

4.4.3 Pricing / Bidding

Since Entertainment auctions are double sided auctions we want to make use of the pending bids once more. Such a pending bid could get matched by another agent who just tries to get rid of tickets and places his bids with a relatively low price.

On each pulse, we analyze which tickets we need and which can be sold. We send sell bids for all tickets we don’t need, generally starting at a high price (hoping they are important to someone). Off course, it is better to sell the tickets at a low price than not at all. Therefore, (depending on the current quote ask price) we adjust our prices in the later game time, until the tickets are sold.

5 Implementation

5.1 Introduction

As we analyzed the code of the DummyAgent we figured that we need to refactor some parts of the TAC Agent ware and implement the given logic in a more object oriented way. During our design process we came to the conclusion that we need to assign our items to the clients ourselves in order to be able to prioritize goods. For example an entertainment event may be more important than another because the client who ordered the first item has already a feasible travel package. This meant that we needed to develop some sort of distribution strategy additional to our bidding strategies.
In the following paragraphs we will describe how we abstracted the different roles, which are needed to achieve the design objectives. In this section we refer the words "items" and "goods" as the needed artifacts such as flights, hotel rooms and entertainment tickets.

5.2 Services

We use the ServiceLocator Pattern to work with our services, which are responsible for different tasks. The ServiceLocator manages all our primary service instances and provides an easy way to access them from anywhere in our software. All services are implemented as Singletons.

The following list shows the services and their responsibility.

• **ClientManager**	Manages all the ClientAgents
• **TradeMaster**	Manages the item stock, requests, bids
• **AuctionInformationManager**	Manages the quotes and their history
• **AuctionRiskManager**	Using the quote history, provides risk analysis

The next points discuss the purpose of each service in detail.

5.3 Clients – ClientManager

We defined that every one of our 8 client is an individual agent with its own desires, believes and intentions. The "ClientManager" manages all the clients and is responsible to delegate the pulse event down to each **ClientAgent**. The pulse event indicates a chance for a **ClientAgent** that he may take action.

5.3.1 ClientAgents

Represents an individual client (client package) and contains the logic for requesting items such as Flights, Hotel and Events. A ClientAgent uses the following classes:

- **ClientPreferences**
  Represents a client's preferences. Over time, they may differ from the original client preferences, depending on what is possible to achive.
- **ClientPackage**
  Represents a client's package and contains information which items are currently assigned to the client.

This separation gives us some advantages as well as creating some new problems. It reduces the problem in most parts to a single package and its optimization. However, we need also to be aware that we don't lose the big picture. On the one hand we are looking for a local optimum (optimization of a client's package). On the other hand we need to achieve a global optimum as well (distribution of the items; make as much utility as possible). In order to achieve these two optimums we introduced the TradeMaster.

The TradeMaster service is an important bridge between the individual ClientAgents. Each Client knows, based on its preferences and the corresponding strategy, what it wants and for what price and requests the preferred goods from the "TradeMaster". The TradeMaster is then able of trying to achieve the global optimum as it knows what each client wants.

5.4 Bidding – TradeMaster

The TradeMaster is responsible for synchronizing the bids, keeping track of the overall possession of items (Stock management) and triggering the distribution strategy for assigning the possessed items to the clients. Incoming Transactions and Bid-Updates are redirected to the TradeMaster.

The TradeMaster has 3 distinct data structures:

• **Item Stock**	Keep track of all owned items
• **Avaiable Item Stock**	Keep track of all items, which are not assigned to a ClientAgent
• **ItemRequests**	Keep track of all ItemRequests of the ClientAgents

The lifecycle of the TradeMaster works as follows:

1. All ClientAgents are pulsed to consider placing item requests on the TradeMaster
2. The TradeMaster allocates and assigns available items to ClientAgents
3. Distribute the possessed items among the clients based on the "ClientPackageAllocationStrategy"
4. Compare the remaining requests to the available items
5. Create bids for the missing items based on the gathered requests (quantity and price)

5.4.1 ClientPackageAllocationStrategy

Off course, the TAC Server himself does the final package assignment, but we need to make assumptions about the current situation and the future as well. Therefore, the ClientAgents need to know which items they own, and which they miss.

ClientAgents can place item requests on the TradeMaster but that does not guarantee that the item can be fetched as a matter of fact. It may even come to the case that different Client Agents want the same item. The ClientPackageAllocationStrategy is then responsible to decide which ClientAgent gets the item, based upon the hypothetical profit.

5.5 Auctions – AuctionInformationManager

The Purpose of the “AuctionInformationManager” is to keep track of the quote development for each auction. It provides a quote history for each auction by storing every quote during the whole TAC Game. This allows our agent to analyze bid rates of auctions and determine their price growth.

5.6 Auction risk analysis – AuctionRiskManager

The AuctionRiskManager is highly coupled with the AuctionInformationManager, as the risk analysis is based upon the current Quotes, their history and their development over time.

A high frequency of bids and therefore a high bid rate is a common indicator of a risky auction. Another indicator is the price growth over time. An auction which had a price growth of 200% in just one minute for example can be considered as a highly risky auction. These two indicators could be very useful for hotel auctions. Let’s assume we have a client which is currently bidding for a TT hotel room. If there would be a bid rate of 1 new bid every 2 seconds the ClientAgent could consider this auction as too risky and change its intention towards buying a SS hotel room instead.

5.7 Communication / Event handling

The overall communication of Tacy is based upon callbacks from the TAC Agent ware. The incoming events are delegated to the responsible services. We have further developed the callback approach to work with our ClientAgents. We generally use “pulse” events to tell the different subsystems that they **may** take action. We do not control our subsystems (for example ClientAgents) absolutely by invoking specific methods but let them a certain room for their own responsibility.

On each pulse event, a ClientAgent analyzes his desires and intensions, comparing them with the environment and takes appropriate actions. This allows the ClientAgents to express their wishes by placing “ItemRequests” to the TradeMaster. They also provide a “*int* ***want****(Auction itemType)”* method, so other subsystem, for instance the TradeMaster, can ask the ClientAgent if it wants a certain item and if, how many of it.

7 Conclusion

The Trading Agent Competition Classic was a very interesting challenge with which we never found ourselves confronted before. The game gave us a basic but very solid idea and understanding about the concept of agent systems. We have spent a lot of time abstracting the system and dividing the system into subsystems in order to be able to solve the problem bit by bit. The greater challenge, however, was to come up with strategies to make our agent competitive. The fact of not knowing other agents strategies makes the own design process very difficult. Nevertheless, we made rich experiences during our process of developing and implementing tactics to achieve a global optimum. We therefore conclude the TAC Classic assignment as a successful experience.

8 Bibliography

[1] Trading Agent Competition
http://tac.sics.se/page.php?id=1
02.16.2013

[2] Wiley Wooldridge, An Introduction to Multi Agent Systems

9 Appendix 1 - Source Code

Due to the fact that we have almost created 20 classes and made a lot of changes in the agent ware (consisting now of 21 classes) we provide the source code in a separate zip archive file.