Bidding Algorithm Comparison for Double Auctions

Cong Chen  
Yiling Chen  
Mark Cohen  
Edward J. Glantz  
Rashaad E.T. Jones

School of Information Sciences and Technology  
The Pennsylvania State University  
University Park, PA
ABSTRACT ................................................................................................................................................. 4
Keywords ......................................................................................................................................................... 4
1. INTRODUCTION ........................................................................................................................................ 4
2. AGENT STRATEGY DESIGN ................................................................................................................... 4
   2.1 Assumptions and Notation Background .......................................................................................... 4
   2.1.1 Auction Protocol ....................................................................................................................... 4
   2.1.2 Agent Utility Model .................................................................................................................. 5
   2.2 Zero Intelligence model (ZI) ........................................................................................................... 7
   2.3 Zero Intelligence Plus model (ZIP) ................................................................................................ 7
   2.4 Rational Agent model (RA) ............................................................................................................ 8
3. EXPERIMENTAL DESIGN ..................................................................................................................... 8
   Experiments and Evaluation .................................................................................................................. 8
System Design .............................................................................................................................................. Error! Bookmark not defined.
References ....................................................................................................................................................... 13
Attachment 1 - Process Model .................................................................................................................... 14
   Process Model for the Facilitator: ....................................................................................................... 14
   Process Model for buyer agents: ......................................................................................................... 14
   Process Model for seller agents: ......................................................................................................... 15
Knowledge model ....................................................................................................................................... 15
   Market Knowledge .............................................................................................................................. 15
   Agent Knowledge (buyer and seller) .................................................................................................... 16
Agent-architecture design ............................................................................................................................ 16
   The Facilitator Agent: ......................................................................................................................... 17
   The Buyer Agents: .............................................................................................................................. 17
   The Seller Agents ................................................................................................................................ 17
Diagram to illustrate Agent Architecture and Knowledge Model: ......................................................... 18
Choice of Tools ........................................................................................................................................... 19
   Prerequisites of choosing the tool ....................................................................................................... 19
   Drawbacks of the tools not chosen ..................................................................................................... 20
CIAgent Framework .................................................................................................................................. 20
Implementation Report: ............................................................................................................................... 21
Description of the Java Source Code Added ............................................................................................. 21
   Communication Protocol: .................................................................................................................. 21
   System Classes ..................................................................................................................................... 21
   Communication Classes ....................................................................................................................... 22
   Agent Classes ....................................................................................................................................... 23
Compiling and Running the Application .................................................................................................... 24
Javadoc ......................................................................................................................................................... 25
Appendix 2 – ZIP and RA Pseudo Code ................................................................................................. 25
   ZIP (Zero Intelligence Plus) Agents: .................................................................................................. 25
   if agent is a seller ............................................................................................................................... 25
   RA (Rational Agents): ....................................................................................................................... 26
ATTACHMENT 3 – JAVA SOURCE CODE ............................................................................................... 28
Description of the Java Source Code ......................................................................................................... 29
   System Classes ..................................................................................................................................... 29
   Main ....................................................................................................................................................... 29
   Communication Classes ....................................................................................................................... 30
ABSTRACT
This project evaluates the performance of three different bidding agent strategies in a double auction environment. The chosen bidding strategies are the Zero Intelligence model, Zero Intelligence Plus model, and the Rational Agent model. These range from a purely reactive trading model (Zero Intelligence) to a model that tries to find the pattern of its auction counterpart (Rational Agent). The goal of this project is to compare the performance of these three strategic bidding models and in doing so assess the value of intelligence in double auctions.

Keywords
Multi-agent system, double auction, bidding and bargaining agents

1. INTRODUCTION
In a double auction (DA), sellers and buyers meet and submit bids (for buyers) and asks (for sellers) to specify the price at which they are willing to buy and sell items. A transaction occurs when the best bid and the best ask prices meet or cross. The continuous double auction (CDA) is a special kind of DA where bids and asks can be submitted, and trades occur, anytime the market is open. In our experiment setting, to simplify the implementation without affecting the comparison results, the auction agents can submit bids/asks when market is open and periodically the facilitator agent (market) will announce transaction results to all the auction agents.

The double auction, which is the principal market institution for real-world trading of equities and derivatives, is a mature mechanism that has been frequently investigated. Much of the knowledge about DA markets is due to an accumulated body of experimental research dating back to early 1960s [1-7]. DA markets are impressively robust in the sense that almost all of the experimental research on the DA markets, whatever the experimental settings are, showed that the resulted allocations were nearly 100% efficient. Efficient allocation is indicated by prices and quantities converging to the competitive equilibrium.

Intrigued by the robustness of DAs in general, our group was motivated to implement a multi-agent system for a double auction. For one thing, the allocation efficiency property of DAs provides a potential baseline for comparison. Also, it is possible to create a controlled environment by intentionally fixing some variables in a DA. In doing so, we can quantitatively measure agent performance with minimal noise.

In the next section we review the design of the various agent strategies involved. In Section 3, we describe the experimental design, and in Section 4 we describe the experiment results. These results are then evaluated in Section 5, the conclusion.

2. AGENT STRATEGY DESIGN
In this section we briefly describe the three bidding algorithms: Zero Intelligence model (ZI), Zero Intelligence Plus model (ZIP), and the Rational Agent model (RA).

2.1 Assumptions and Notation Background

2.1.1 Auction Protocol
In our experiments, the DA has 6 buyers and 6 sellers. There is only one kind of items for trading in this market. For each buyer, we fix the number of items the buyer wants to buy, and specify the buyer’s reservation price for
each item (how much the item worth for the buyer). Similarly, for each seller, we fix the number of items the seller is willing to sell and specify the seller’s reservation price for each item (how much the item cost for the seller). These values are unchanged for the market from period to period, no matter what kind of agents we use as buyers or sellers. By doing so, we create fixed demand and supply for the DA market. The theoretical total profits for buyers and total profits for sellers under efficient allocation can be calculated as our evaluation baseline. All our experiments are conducted in this controlled market.

2.1.2 Agent Utility Model

Each experiment consists of 5 trading periods, each of which represents a double auction. It starts when the facilitator announces that the auction has started. It ends when there are no new bids or asks for a while and the facilitator announces the auction is terminated. Initially, each agent is given a list of 6 reservation prices for the units to be traded. By doing this, each buyer is provided an individual demand curve, and each seller is provided an individual supply curve. By fixing the reservation price from trading period to trading period, and from experiment to experiment, the market demand and supply is fixed.

The details of the reservation price for 6 buyers are displayed in Table 1. The reservation price for buyers are arranged in a decreasing order, representing that buyer’s value decrease (i.e. the buyer is willing to pay less for each additional unit).

<table>
<thead>
<tr>
<th>Buyer #1</th>
<th>Unit 1</th>
<th>Unit 2</th>
<th>Unit 3</th>
<th>Unit 4</th>
<th>Unit 5</th>
<th>Unit 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buyer #2</td>
<td>$140</td>
<td>$130</td>
<td>$120</td>
<td>$110</td>
<td>$80</td>
<td>$60</td>
</tr>
<tr>
<td>Buyer #3</td>
<td>$100</td>
<td>$80</td>
<td>$60</td>
<td>$40</td>
<td>$20</td>
<td>$10</td>
</tr>
<tr>
<td>Buyer #4</td>
<td>$155</td>
<td>$130</td>
<td>$120</td>
<td>$100</td>
<td>$80</td>
<td>$60</td>
</tr>
<tr>
<td>Buyer #5</td>
<td>$150</td>
<td>$130</td>
<td>$110</td>
<td>$100</td>
<td>$90</td>
<td>$70</td>
</tr>
<tr>
<td>Buyer #6</td>
<td>$110</td>
<td>$100</td>
<td>$80</td>
<td>$60</td>
<td>$40</td>
<td>$20</td>
</tr>
</tbody>
</table>

The details of the reservation price for 6 sellers are displayed in Table 2. The reservation price for sellers are arranged in increasing order to represent that seller’s costs increase (i.e. the seller wants to sell each additional unit for a higher cost).

<table>
<thead>
<tr>
<th>Seller #1</th>
<th>Unit 1</th>
<th>Unit 2</th>
<th>Unit 3</th>
<th>Unit 4</th>
<th>Unit 5</th>
<th>Unit 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seller #2</td>
<td>$20</td>
<td>$30</td>
<td>$40</td>
<td>$60</td>
<td>$80</td>
<td>$100</td>
</tr>
<tr>
<td>Seller #3</td>
<td>$20</td>
<td>$40</td>
<td>$60</td>
<td>$80</td>
<td>$100</td>
<td>$120</td>
</tr>
<tr>
<td>Seller #3</td>
<td>$50</td>
<td>$60</td>
<td>$80</td>
<td>$100</td>
<td>$120</td>
<td>$140</td>
</tr>
</tbody>
</table>
The aggregation of individual demand forms the market demand, and the aggregation of individual supply forms the market supply. Based on the individual supply and demand specified by the above reservation prices, the market demand and supply curves are shown in Figure 1. At a price equal to $155, buyer #4 would like to buy 1 unit of the product. Since all other buyer reservation prices are lower than $155, the total market demand quantity at $155 is 1 unit. At a price of $150, buyer #4 would like to buy one unit (his value for the unit is higher than $150), and both buyer #2 and #5 would like to buy 1 unit each. Hence, the total market demand quantity at $150 is 3 units. In this way, the rest of the market demand curve can be derived. Similarly, the market supply curve is formed using the reservation price of sellers. At a price of $5, seller #4 is the only agent who would like to sell a unit. The total market supply at $5 is 1 unit. At price equal to $10, seller #4 would like to sell 1 unit, and seller #5 would like to sell 1 unit. Thus, the total market supply at $10 is 2 units.

According to the competitive market theory, the competitive equilibrium price for this market is $80 and the total traded quantity for each trading period between 20 and 25. Previous experimental research on the DA market has shown that DA markets will converge to a competitive equilibrium. Thus, when competitive equilibrium is achieved, the theoretical profits that each buyer (and buyer group) can get in each trading period are shown in Table 3. For example, buyer #1 will buy 4 or 5 units at an equilibrium price of $80. This buyer will earn $60 ($140-$80) for the first unit, $50 ($130-$80) for the second unit, $40 ($120-$80) for the third unit,
and $30 ($110-$80) for the fourth unit. If the fifth unit is purchased at a price of $80, the profit for the unit is $0. So, the theoretical profit for buyer #1 is $180 ($60+$50+$40+$30).

### Table 3 Theoretical Profits of Buyers for Each Trading Period

<table>
<thead>
<tr>
<th></th>
<th>Buyer #1</th>
<th>Buyer #2</th>
<th>Buyer #3</th>
<th>Buyer #4</th>
<th>Buyer #5</th>
<th>Buyer #6</th>
<th>All Buyers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>$180</td>
<td>$190</td>
<td>$20</td>
<td>$185</td>
<td>$180</td>
<td>$50</td>
<td>$805</td>
</tr>
</tbody>
</table>

When competitive equilibrium is achieved, theoretical profits that each seller and the seller group can get in each trading period are shown in Table 4. For example, seller #1 will sell 4 or 5 units at equilibrium price $80. He will earn $60 ($80-$20) for the first unit, $50 ($80-$30) for the second unit, $40 ($80-$40) for the third unit, and $20 ($80-$60) for the fourth unit. If he sells the fifth unit at price $80, his profit for the unit is $0. So, the theoretical profit for seller #1 is $170 ($60+$50+$40+$20).

### Table 4 Theoretical Profits of Sellers for Each Trading Period

<table>
<thead>
<tr>
<th></th>
<th>Seller #1</th>
<th>Seller #2</th>
<th>Seller #3</th>
<th>Seller #4</th>
<th>Seller #5</th>
<th>Seller #6</th>
<th>All Seller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>$170</td>
<td>$120</td>
<td>$50</td>
<td>$185</td>
<td>$220</td>
<td>$60</td>
<td>$805</td>
</tr>
</tbody>
</table>

Next, in the following sections we briefly describe the three kinds of bidding agent models used in this experiment: Zero Intelligence model (ZI), Zero Intelligence Plus model (ZIP), and the Rational Agent model (RA).

### 2.2 Zero Intelligence model (ZI)

ZI agents [8], as their name indicates, use an extremely simple strategy: they randomly draw bid or ask prices from a uniform distribution over a price interval, regardless of the market status. ZI agents only guarantee that they will not submit a bid or ask price which will result in negative profit. Other than this, their behavior is purely random.

### 2.3 Zero Intelligence Plus model (ZIP)

ZIP agents [9] improve ZI agents by incorporating an elementary learning algorithm that enables agents to adjust random bids or ask prices according to the transaction history. As ZI agents, initially ZIP agents randomly choose positive surplus values as their bids or ask prices. Observing market activities, they make small adjustments to their bids or ask prices. For failed trades (the highest bid is lower than the lowest ask price), ZIP agent buyers set their target bid price as slightly higher than the highest existing bid, while ZIP agent sellers set their target ask price as slightly lower than the lowest existing ask. If trades are taking place, ZIP agent buyers target at a price slightly lower than the lowest existing ask, while ZIP agent sellers target at a price slightly higher than the highest existing bid. Given the target price, agents do not jump straightly to that value, but moves toward it at a rate determined by the learning rule.
2.4 Rational Agent model (RA)

For experience, our group developed a “rational agent model” for strategic bidding instead of using one by Gjerstad and Dickhaut [10]. We suggest that agents that determine probability by using data about previous bids, ask prices, and trades, can determine the probability of a bid or ask being accepted at any given price. Using the probabilities, agents can then calculate the expected utility of any bid or ask price, and select the bid or ask price that maximizes their expected utility. The interpolation and calculation of RA is relatively complex compared to ZI and ZIP, but it can guarantee to perform at least as good as ZIP in the worst case when it cannot identify its competitor.

The RA works by bidding 1 (or asking 100) until a certain number of rounds have passed, and then it tries to find the pattern of its auction counterpart. If it realizes that its counterpart is the decreasing the ask (or increasing the bid), then it stays at the same bid (or ask). If the RA agent does not notice a pattern, then it will increase its bid (or decrease its ask) by $1. As the facilitator is designed to only accept higher bids or lower asks, if the RA does not notice a pattern, it falls back to behaving like a ZIP.

To summarize these three strategies, the ZI agents do not learn from past experience, the ZIP agents have very simple learning rules, and the RA agents incorporate anticipatory behavior. With these three agents, we will be able to assess the importance of intelligence in bidding agents in a DA by comparing their performance in our controlled DA markets.

3. EXPERIMENTAL DESIGN

A total of nine experiments will be conducted with one group of homogenous buyers competing against a different group of homogenous sellers. For example, in one experiment, buyers using the ZI strategy may compete against sellers also using the ZI strategy, while in a different experiment sellers using ZIP strategy might compete against RA buyers. The complete list of the different experiments that will be used is illustrated in Table 5. Each experiment consists of 5 trading periods.

**Experiments and Evaluation**

We will measure the performance of agents as a group, that is, we will measure the profit of all buyer agents and the profit of all seller agents in each experiment. The evaluation of their performance will be conducted in two directions. First, we want to evaluate how well agents perform in general. This will be achieved by comparing agents’ actual group profits with the theoretical profits when efficient allocation is reached in our DA market settings. Second, we are interested in determining how well these agent strategies perform against each other. We are going to compare actual group profits for different agent models. For example, we will compare the ZI buyers’ actual profit in experiment 2 (which is against ZIP sellers) with the ZIP buyers’ actual profit in experiment 4 (which is against ZI sellers).

<table>
<thead>
<tr>
<th>Experiment</th>
<th>ZI Sellers</th>
<th>ZIP Sellers</th>
<th>RA Sellers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>ZI Buyers</td>
<td>Experiment 2</td>
<td>Experiment 3</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>ZIP Buyers</td>
<td>Experiment 5</td>
<td>Experiment 6</td>
</tr>
<tr>
<td>Experiment 7</td>
<td>RA Buyers</td>
<td>Experiment 8</td>
<td>Experiment 9</td>
</tr>
</tbody>
</table>

To summarize, the first direction of our evaluation process enables assessment of the general performance of each kind of bidding agents in a DA. Only if the bidding strategies have acceptable performance, is it meaningful to compare which one performs better. The second direction of our evaluation process explores the impact of the ability to learn on an agent’s performance in a DA.
4. EXPERIMENTAL RESULTS

We summarize our experimental results in Table 6. Table 6 presents profits of buyer groups and seller groups in each experiment. We also listed theoretical buyer group profits and seller group profits for reference. As we have indicated, the auction parameters we chosen enable both buyer groups and seller groups each earn $805 in one auction at competitive equilibrium. An experiment consists of 5 auctions. Hence, the theoretical profit for both buyer groups and seller groups is $4025.

<table>
<thead>
<tr>
<th>Buyers-Sellers</th>
<th>Profit for Buyers</th>
<th>Profit for Sellers</th>
<th>Theoretical Profit for Buyers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZI-ZI</td>
<td>4644</td>
<td>3396</td>
<td>4025</td>
</tr>
<tr>
<td>ZI-ZIP</td>
<td>2440</td>
<td>5560</td>
<td>4025</td>
</tr>
<tr>
<td>ZI-RA</td>
<td>1782</td>
<td>6268</td>
<td>4025</td>
</tr>
<tr>
<td>ZIP-ZI</td>
<td>6045</td>
<td>2005</td>
<td>4025</td>
</tr>
<tr>
<td>ZIP-ZIP</td>
<td>4165</td>
<td>3885</td>
<td>4025</td>
</tr>
<tr>
<td>ZIP-RA</td>
<td>1106</td>
<td>6858</td>
<td>4025</td>
</tr>
<tr>
<td>RA-ZI</td>
<td>6546</td>
<td>464</td>
<td>4025</td>
</tr>
<tr>
<td>RA-ZIP</td>
<td>6661</td>
<td>569</td>
<td>4025</td>
</tr>
<tr>
<td>RA-RA</td>
<td>3102</td>
<td>4932</td>
<td>4025</td>
</tr>
</tbody>
</table>

Comparing actual profits with theoretical profits, we can see that when buyer agents and seller agents are of the same type, e.g. ZI buyers vs. ZI sellers, profit of buyer groups and profit of seller groups are of similar amount of theoretical buyer group profits and seller group profits. But when buyer agents and seller agents are of different type, there usually is one type of the agents that are good at exploring profit, and earn significantly extra amount of profit than theoretical profit. But the total surplus of the market, which is the total profits of buyers and sellers, is always very close to its theoretical value. Thus, we believe that performance of software agents is acceptable in general. We will go to investigate which kind of agents is “smarter”.

From Table 5, we can roughly say that ZIP agents perform better than ZI agents, and RA agents perform better than both ZI and ZIP agents. When competing with ZI agents, ZIP agents react to current market situation by slightly adjusting their profit margins, hence tending to keep a relatively high profit margin and relatively steady transaction prices. ZIP buyers can explore ZI seller’s “mistakes” of selling at a very low price. ZIP sellers can explore ZI buyer’s “mistakes” of buying at a very high price. The learning algorithm of ZIP is simple, but it is very effective. RA agents have a special bidding (asking) strategy, when they detect that their competitors are ZIP. They will try to keep very high profit margin and trigger ZIP buyers to increase bids or ZIP sellers to decrease asks continuously. In this way, they can earn extra profit. But if they can not tell whether their competitors are ZIP agents, themselves will fall back to be ZIP agents. RA’s learning algorithm works very well when compete with agents of different type. Even if RA makes the wrong judgments, such as treating ZIP agents as Non-ZIP agents or treat ZI agents as ZIP agents, performance of RA agents are still very good. We can see that RA agents earn the highest actual profits when they compete with ZIP agents or ZI agents. However, RA agent has a weakness. When they RA buyers compete with RA sellers, they tend to make wrong judgments in both sides. They will treat each other as ZIP agents and keep very high profit margins which decrease very slow. The result is that negotiation time is long. Our RA buyers vs. RA sellers experiment took us several hours, with time of other experiments various from several minutes to half an hour. From the perspective of negotiation time, RA agents are not very efficient.

We also record the market transaction prices. From the plot of these price sequences, which agent perform better in the market is clearer. We select several plots to illustrate as below.
The transaction prices for ZI buyers vs. ZI sellers are basically randomly fluctuate around the equilibrium price, $80. It is the expected result of the random behavior of ZI agents.

Transaction prices for ZIP buyers vs. ZIP sellers nicely converge to the equilibrium price. After it convergence, transaction price almost stick to the equilibrium price. It shows that the simple learning ability of ZIP agents can drive the market to efficient allocation.

From the transaction prices for ZI buyers vs. ZIP sellers, we can see that ZIP sellers successfully ZI buyer’s profits. At most of the time, the transaction price is higher than the equilibrium price, $80, which means that ZIP sellers sell items at a more profitable price.
The competition of RA buyers vs. ZIP sellers shows the strength of RA agents. The transaction price is lower than the equilibrium price most of the time and is never higher than it. It means that RA buyers buy items at more profitable prices.

5. CONCLUSION
In this project we evaluated the performance of three different bidding agent strategies (Zero Intelligence model, Zero Intelligence Plus model, and the Rational Agent model) in a double auction environment. The ZI model was purely reactive, the ZIP model exhibited simple learning ability by react to market current conditions, while the RA model includes a naïve reasoning ability to guess what other agents are, in addition to the learning ability of ZIP model.

As we expected we found that learning is crucial for agent’s performance in double auction markets. Purely reactive agents, ZI, perform satisfactorily only when both buyers and sellers are ZI agents. ZIP agents outperform ZI agents because they can learn from environment and adjust their behavior. RA agents, who can inference type of other agents, outperform ZIP agents in our market settings. Thus, how to make agents to be truly “intelligent” through learning is a very important issue.

For future work we are suggest
References

Attachment 1 - Process Model

The following process model has been developed to illustrate the processes and logic that will be provided in our simulated DA:

A Facilitator agent that manages the marketplace, several Buyer agents and Seller agents interact within that marketplace. Facilitator (market) is the matchmaker between the buyers and sellers. All agents must register with the Facilitator agent before they can interact with other agents. Sellers submit asks to the Facilitator, Buyers submit bids to the facilitator, if there is a transaction that can be performed (there is a bid higher than an ask), the Facilitator will perform and announce the transaction.

The logic can be transformed to the following process models:

**Process Model for the Facilitator:**

```java
while (market is open) {
    process bids and asks Messages from buyers and sellers;
    //test if there is a match between the bids and the asks
    if (no match) then
        announce the highest bid and the lowest ask;
        wait for new or updated bids or asks;
    }else  
    {perform transaction;
    send messages to buyers and sellers to announce transaction price;
    wait for bids or asks for next item;
    }
}
```

**Process Model for buyer agents:**

```java
while (market is open) {
    processTimePop();//wakeup periodically to submit a bid;
    keep track of market activities
    //only applicable for agents with learning capabilities, who use market history in
deciding bidding prices
}
processTimePop() {
    pick up the first item in the wishing list;
    decide a bidding price for it;
    if (current bidding price > highest bidding price in the market) {
        construct a Message to be sent;
        send Message to the Facilitator; //submitting the //bid
Process Model for seller agents:

While (market is open)
{
    wakeup periodically to submit an ask;
    keep track of market activities
    //only applicable for agents with learning capabilities, who use market history in deciding asking prices
}

processTimePop()
{
    pick up the first item in the selling list;
    decide an asking price for it;
    if (current asking price < lowest asking price in the market)
    {
        construct a Message to be sent;
        send Message to the Facilitator; //submitting the //ask
    }
}

These processes are only presented as pseudo code to illustrate the logic. Details about these processes and additional processes are needed to further illustrate the steps involved in the three different agent strategies.

Knowledge model

The domain knowledge, as described below, includes market knowledge, agent knowledge (buyer and seller) and system knowledge.

Market Knowledge

Our experiments used the following DA market rules [16]:

1. The “NYSE” spread-improvement rules are in effect, requiring that new bids be priced higher than the current best bid (and the equivalent for asks). This is in accordance with other DA studies. This is believed to facilitate convergence to equilibrium.

2. All orders were for a single unit only. For simplicity (and conformance with prior DA studies), player may have only one open order.

3. Orders remain open after submission until either they are traded or the trading period expires.

4. Orders may not be withdrawn once submitted (subject to the NYSE rule), but may be modified.

5. Trades occur when the best bid matches or crosses best ask price. If they cross (i.e., bid price exceeds the ask price), the trade price was the price of the order submitted first.

6. For each trading period, agents start with a fresh supply of cash. Each agent is given a list of 8-14 limit prices for the units to be traded, arranged in order from most to least valuable (i.e., the buyer values
decreased and the seller costs increased). Approximately half of the units are tradable for positive surplus at equilibrium. The limit prices are generated from a base set of three linear schedules in which each successive unit increased in cost or decreased in value at a constant rate. These rates varied in the three schedules; however, the total theoretical surplus was designed to be about the same in each. Total theoretical surplus is designed to evenly split between buyers and sellers.

Agent Knowledge (buyer and seller)

ZI agents (no learning)
- Randomly draw bid or ask prices (regardless of the market status)
- Will not submit a bid or ask price which will result in negative profit (choose positive surplus values)

ZIP agents (simple learning)
- Randomly draw bid or ask prices
- Incorporate an elementary learning algorithm (adjust random bids or ask prices according to the transaction history) for failed (highest bid is lower than the lowest ask price) trades:
  - ZIP agent buyers set target bid price slightly higher than highest existing bid
  - Target price slightly lower than the lowest existing ask if trades are taking place
  - ZIP agent sellers set target ask price slightly lower than lowest existing ask.
  - Target price slightly higher than the highest existing bid if trades are taking place
  - Agents do not jump directly to a given target price, but incrementally approach at a rate determined by learning.
- Will not submit a bid or ask price which will result in negative profit (choose positive surplus values)

RA agents (reorganization of competitor’s behavior)

Maintains state information about the environment
Adaptive
“Game Theoretic Approach”
•Initial conservative strategy
•Strategy is determined by finding a bid/ask pattern

System Knowledge
System information will include a record of participating agents, roles (seller or buyer), a log of bid-ask, whether a trade takes place, and if so the price, amount and time of transaction. For every experiment running, two csv files are generated with a Profit Table recording profit for every agent and a Transaction History Table with the auction round number and every transaction price for transactions that took place.

Agent-architecture design
Three types of agents are proposed for the design of the multi-agent DA simulation: the facilitator, several buyers, and several agents.
The Facilitator Agent:

The facilitator agent will be responsible for conducting the market place according to the trading rules of a DA. All agent communication must take place via the facilitator. The processes it will be responsible for include:

- Collecting current bids and asks
- Checking for compatibility between these bids and asks
- Conducting a transaction if a compatible bid/ask is found
- Announcing the highest bid and the lowest ask to all buyers and sellers if no compatibility exists between the current bids and asks
- Determining when the market is open or closed for business

In addition, the facilitator will be responsible for the following knowledge:

- Trading rules
- Current bids and asks
- Status of the market (open/close)

The Buyer Agents:

The buyer agents will be responsible for purchasing items from the sellers using the current bidding rules and an internal bidding strategy. Although there will be different buyer agents that implement different bidding strategies, all buyers will be responsible for the following processes:

- Present a current bid
- Update its bid based on its current knowledge
- Update the agent’s current reserve price
- Tracking the market history (only applicable for agents capable of learning)

In addition, the buyer will be responsible for the following knowledge:

- Bidding rules
- Reservation price
- Current bid price
- Quantity of products planning to buy
- Market history (only applicable for agents capable of learning)

The Seller Agents:

The seller agents will be responsible for selling items to the buyers using the current asking rules and an internal asking strategy. Although there will different seller agents that implement different asking strategies, all sellers will be responsible for the following processes:

- Present a current ask
Update its ask based on its current knowledge
Update the agent’s current reserve price
Tracking the market history (only applicable for agents capable of learning)

In addition, the seller will be responsible for the following knowledge:

- Asking rules
- Reservation price
- Current ask price
- Quantity of products planning to sell
- Market history (only applicable for agents capable of learning)

**Diagram to illustrate Agent Architecture and Knowledge Model:**

Abbreviation of processes:
P(x)— present a current ask or bid, e.g. x=a for ask, x=b for bid
U(x)— update an ask or a bid.
Ur— update current reserve price
T— Tracking history of market
Col— Collecting current bids and asks
Che— Checking for compatibility between these bids and asks
CT— Conducting a transaction if a compatible bid/ask is found
A— Announcing the highest bid and the lowest ask to all buyers and sellers if no compatibility exists between the current bids and asks
dM— Determining when the market is open or closed for business

Abbreviation of knowledge:
cP(x)— current price of a bid or ask
rP— reservation price
R(x)— rules for trading, bidding or asking
M— status of market
Ms— market history
Q(x)— quality of products planning to sell or buy
Choice of Tools

According to the previous investigation of several MAS platforms, we decided to use CIAgent as our multi-agent system (MAS) implement tool.

Prerequisites of choosing the tool
The reason for choosing this tool is that, first of all, it met all the following prerequisites:

1. It is a portable MAS development environment
2. The environment is based on Java, a programming language that the group members are familiar with and is accessible using the language via an API.
3. It contains both an agent framework and a rule-based inference engine, as opposed to integrating two distinct tools. When adding learning and reasoning capabilities to our agent, we need to figure out how much code could be reused from the provided inference engine.
4. It allows the creation of autonomous agents that are capable of running under their own thread of control and can communicate to other autonomous agents using messages.
5. It is free, easy to use, stable, and well documented. There are several comprehensive examples using the framework we chose.
Drawbacks of the tools not chosen

The survey conducted previously has identified three tools that all appear to be sufficient for use in reaching the goals outlined in this design. Both ABLE and Zeus provide excellent graphical development environments, however they also provide additional functionality that is not needed for this project and may complicate the use of the tool.

The weaknesses of ABEL are the 90-day evaluation period (be termed as only adequate), lack of exceptional set of documentation and examples. In addition, the environment’s power also adds a level of complexity that could result in a longer learning curve and difficulty diagnosing problems. For the purposes of this research project, many of the additional features provided by ABEL may not be required.

The weaknesses of Zeus are also a product of its highly developed graphical development environment. This environment seems to puts the developer at the mercy of the graphical tools, and could make it difficult to diagnose and fix problems that are related to the tool. In addition, Zeus appears to contain several capabilities that are not required by this research project. These capabilities increase the complexity of the tool and may increase the learning curve. Finally, Zeus’s documentation and examples are adequate but cannot be considered exceptional.

CIAgent Framework

http://www.bigusbooks.com/ciagent2.html

The CIAgent framework, on the other hand, is the easiest to learn and understand since it is the simplest framework and comes with the best documentation and examples of the three. It does not provide any graphical development tools but the design and feature set is the simplest. The main disadvantage of the CIAgent framework is that it will require the most Java programming of the three frameworks.

The CIAgent Framework is a library of Java interfaces and classes that enable the development of intelligent multi-agent system. This library supports the creation of autonomous agents that operate under their own thread of control. These agents can perform reasoning, learning, and communication with other autonomous agents. Although this framework does support Java beans, it does not include a graphical user interface for the construction of agents. In theory the developed agents can be plugged into a “Java bean aware” IDE but such an IDE must be obtained separately. All agents developed using this framework must be constructed in code by extending the classes provided by the framework.

The CIAgent Framework was introduced in the book “Constructing Intelligent Agents Using Java: Professional Developer's Guide, 2nd Edition”, by Joseph P. Bingus and Jennifer Bingus[11]. Excellent documentation and comprehensive examples are provided with the book, along with a CD containing the software. Use of this software is free and unrestricted with the purchase of the textbook.

The strengths of this environment appear to be the simplicity and excellent documentation and examples provided in the book. The framework is described in detail and many examples are provided, including one that is related to the goal of this research project. The design of this framework is relatively simple and with knowledge of object-oriented programming, should result in a relatively short learning curve. The feature set of this simple framework appears to match the requirements of the project and do not introduce additional complexities that are not related to the projects needs. Finally, other than the purchase of the book there are no restrictions on the use of the software.

The weaknesses appear to be the need to write a fair amount of Java code in order to implement a multi-agent system. Although the framework provides an excellent infrastructure to build agents with, code must be written...
to provide the agent logic, and the interface used to evaluate the agents. The lack of a graphical environment to build, execute, and debug the agents could introduce problems.

**Implementation Report:**

**Description of the Java Source Code Added**
The code created for the double auction multi-agent system consists of ten classes that can be broken into three categories: system classes, communication classes and agent classes. The system category currently only contains a single class called Main. The communication category represents the different messages used by the agents in the double auction market. Finally, the agent category contains the buyer, seller, and the facilitator agents and related classes. All ten of these classes reside in the edu.psu.mas package. The following is a list of these ten classes, along with a short description of each class.

**Communication Protocol:**

1. Register --- Buyer agents and seller agents register with the facilitator agent to participate in the auction
   (TELL, agent name)

2. Offer Request --- The facilitator request buyer and seller agents to submit their bids and asks.
   (ASK)

3. Offer --- Buyer agents and seller agents submit their bids or asks to the facilitator agent
   (TELL, agent name, buy/sell, offer price)

4. Offer Response --- The facilitator agent inform all the registered agents respectively the market status and their bid status.
   (REPLY, win/loss, transaction price, highest bid, lowest ask)

**System Classes**

**Main**
The main class serves as the main driver for the double auction multi-agent system. It contains only a single method; main(). The main method is the method that is called when this class is run. It is helpful to look at the implementation of this main method to get an understanding of the structure of the MAS code:

```java
public static void main(String args[]) {
    Facilitator facilitator = new Facilitator();
    facilitator.initialize();
    facilitator.startAgentProcessing();

    BuyerAgent agent1 = new BuyerAgent(facilitator);
    agent1.initialize();
    agent1.startAgentProcessing();

    BuyerAgent agent2 = new BuyerAgent(facilitator);
```
agent2.initialize();
agent2.startAgentProcessing();

SellerAgent agent3 = new SellerAgent(facilitator);
agent3.initialize();
agent3.startAgentProcessing();

SellerAgent agent4 = new SellerAgent(facilitator);
agent4.initialize();
agent4.startAgentProcessing();
}

The code above is quite simple. First it creates a Facilitator agent and initializes it. After initialization it asks it to start processing. Once the Facilitator is up and running various buyers and sellers can be created. Notice that the constructor of these agents requires a reference to the Facilitator. Each buyer and seller is created, initialized, and executed. At this point the agents begin to communicate and after everyone registers with the Facilitator, the auction will begin!

Communication Classes

KQMLMsg
This is the class used to represent a KQML message. All communication in the system is done using instances of this class. A KQML message contains a performative and a domain specific message object. The remaining four communication classes are examples of the domain specific messages objects that are included in a KQML message. An example of how to create and send a KQML message follows:

```java
CIAgentEvent registerEvent =
    new CIAgentEvent(this, new KQMLMsg("tell", new Register(this)));
m_facilitator.postCIAgentEvent(registerEvent);
```

The code above creates a CIAgentEvent object. This event object is the object provided by the CIAgent framework for sending messages. The first parameter of the CIAgentEvent constructor is a reference to the sender and is used by the Java event dispatcher. The second parameter is the object to be sent. In this system this object will always be a KQML message. The KQML message constructor takes a performative (in this example “tell”) and an instance of a domain specific message object. In this example a “register” message is sent to the Facilitator using the Register class. The constructor of the Register class takes a reference to the agent that is registering.

Register
A domain specific message class used in a KQML message. This message is used by agents when the wish to register with the Facilitator.

OfferRequest
A domain specific message class used in a KQML message. The Facilitator uses this message in order to request that registered agents present their offers.

Offer
A domain specific message class used in a KQML message. This message is used by agents when the wish to submit an offer to the Facilitator.
OfferResponse

A domain specific message used in a KQML message. The Facilitator uses this message to notify all registered agents of the results from the most recent round of offers.

Agent Classes

Facilitator

The Facilitator class is responsible for coordinating the double auction. Buyer and seller agents register with the Facilitator before the auction begins. After registration, the Facilitator asks registered agents for offers. Once the offers have been received the Facilitator calculates the result of the offers and then notifies registered agents of these results. The three important methods in the Facilitator class are:

processTimerPop() – This method is called every SLEEP_TIME milliseconds and provides the Facilitator with an opportunity to act. The previous phase of the Facilitator (WAIT_FOR_REGISTER or WAIT_FOR_OFFERS) determines how it will behave. If the last phase was WAIT_FOR_REGISTER the Facilitator will terminate its wait for agents to register, send a request for offers, and enter the WAIT_FOR_OFFERS phase. If the previous phase was WAIT_FOR_OFFERS the Facilitator will terminate its wait offers, calculate and send the result of all of the offers, and then request the agents to send the next round of offers.

processCIAgentEvent() – This method is called every MESSAGE_CHECK_TIME milliseconds and provides the Facilitator with an opportunity to respond to messages. If the next message in the message queue is “tell facilitator to register me” the Facilitator will add the sender of this message to its list of registered agents. If the next message in the queue is a “reply to the request of offers” the sender’s offer will be added to the list of current offers.

sendOfferResponse() – This method is called when the Facilitator needs to calculate the result of the current offers and send this result to all registered agents. The Facilitator will go through all the current bids and asks, and decide the values of highest bid and lowest ask. The highest bid of the current round is not allowed to be lower than that of the previous round, while the lowest ask of the current is not allowed to be higher than that of the previous round. The Facilitator will then compare the highest bid with the lowest ask and determine whether there is a transaction.

MarketAgent

This abstract class serves as the base class for all buyer and seller agents in the system. The goal of this class is to make it easy to create new buyer and seller agents. This class only needs to respond to events therefore it does not do anything in its processTimerPop() method. Every MESSAGE_CHECK_TIME milliseconds the processCIAgentEvent() method is called giving the MarketAgent an opportunity to handle any pending messages. All MarketAgents must respond to messages from the Facilitator asking for offers or announcing the result of that last round of offers. The internal algorithm used by the specific agent determines the way each agent responds to these messages. For this reason the MarketAgent dispatches these messages to the sendOffer() and handleOfferResponse() abstract methods. These are the methods that must be overridden by all buyers and sellers to specify the specific bidding algorithms used.

BuyerAgent

This class is a simple implementation of a buyer. It extends the MarketAgent class and implements a very trivial bidding strategy by overriding the sendOffer() and handleOfferResponse() methods. This class serves as an example of how to create a new buyer agent and is only used for testing the framework.
SellerAgent

This class is a simple implementation of a seller. It extends the MarketAgent class and implements a very trivial bidding strategy by overriding the sendOffer() and handleOfferResponse() methods. This class serves as an example of how to create a new seller agent and is only used for testing the framework.

ZiBuyerAgent

This class acts as a ZI buyer agent. It randomly selects a profit margin which is between zero and one using the “Zero Intelligence” algorithm described in the Bidding Agent Models part.

ZiSellerAgent

This class acts as a ZI seller agent. It randomly selects a profit margin which is between zero and one using the “Zero Intelligence” algorithm described in the Bidding Agent Models part.

Compiling and Running the Application

All of the source code for this MAS is contained in two different packages; ciagent and edu.psu.mas. The ciagent package contains the multi-agent framework code provided by Joseph Bigus and Jennifer Bigus in the book “Constructing Intelligent Agents using Java”. This framework is well described in the text and will not be elaborated on here. The edu.psu.mas package, on the other hand, contains all of the source code described in this document in order to simulate the double auction multi-agent environment.

To compile and execute this code you first need to unzip the source code into a working folder. For the purposes of these instructions assume that you have unzipped the software into c:\ist597b\src. Be sure to maintain the folder structure contained in the zipfile when you unpack the software. If you choose to unpack the code into a different working folder, just substitute it place of c:\ist597b\src while reading these instructions.

After unpacking the code you should have the following folder structure:

C:  
    \ist597b  
      \src  
        \docs  
        \ciagent  
        \edu  
          \psu  
            \mas

The ten classes described in this document are located in the edu\psu\mas folder. To compile and run these classes open a command prompt and make sure that c:\ist597b\src is the current working directory. If it is not use the cd c:\ist597b\src command to change to the correct directory. Next, type the following command to compile the source code:

    javac -classpath . edu\psu\mas*\.java

Be sure to include the “.” character after -classpath. When compilation is complete you should be able to run the code using the following command:

    java -classpath . edu.psu.mas.Main
Javadocs

In addition to this document and HTML based description of the source code is available in the docs folder created when you unpacked the source code. To view this documentation just open the docs/index.html file in your web browser.

Appendix 2 – ZIP and RA Pseudo Code

Based on our literature review, the logic process of the ZIP and RA agents and transformed them into the pseudo code listed below:

ZIP (Zero Intelligence Plus) Agents:

For SELLERS

If (the transaction happens)
Then
If ask <= lowest ask then raise profit margin
else lower profit margin
Else
If (there is no transaction happening)
Then
Any active seller should lower its margin

For BUYERS

If (the transaction happens)
Then
If bid >= highest bid then raise profit margin
else lower profit margin
Else
If (there is no transaction happening)
Then
Any active buyer should lower its margin

Psuedo Code for altering profit through learning:

Void profit alter (Agent *a, price p)
{ c, diff, change, newprofit;
diff=(p-a\rightarrow p)
//\Gamma(t)\equiv((1.0- \gamma)*\beta(\Gamma(t)-pi(t)) +\gamma*\Gamma(t-1))
change =((1.0-a\rightarrow momntm)*(a\rightarrow beta*differ)+((a\rightarrow momntm)*(a\rightarrow last_d));

//\Gamma(t-1)\equiv \Gamma(t)
 a\rightarrow last_d=change;

//set new price by altering profit margin
//\mu(t+1)=pi(t) +\Gamma(t))/\lambda, j -1
new profit = (a\rightarrow p +change)/(a\rightarrow limit)-1;

if agent is a seller
{ if (newprofit>0) a\rightarrow profit= newprofit}
else //agent is a buyer
{ if (newprofit<0) a\rightarrow profit -newprofit}
set_price (a);
RA (Rational Agent):
--randomly choose a bid or ask at the first round (make sure not to have negative profits)
--determine the strategy that the competitor agent is using by observing its transaction history

```
private void DetermineStrategy()
{
    double mean=0;
    double variance=0;
    for(int i=0;i<numRounds;i++)
    {
        mean=mean+lowestAskList[i];
    }
    mean=mean/numRounds;
    for(int i=0;i<numRounds;i++)
    {
        variance=variance+(lowestAskList[i]-mean)*(lowestAskList[i]-mean);
    }
    variance=variance/numRounds;
    if(variance<THRESHOLD)
    {
        opponent_type=ZIP;
    }
}
```

GD (Gjerstad-Dickhaut) Agents:

- This is an agent design strategy we didn’t choose to implement in our experiment, the pseudo code is included here as a reference to our arguments why we didn’t choose this type of agent.

Some denotations:
Hm--of recent market activity (the bids and asks leading to the last M successful trades) current price-P
Cost (seller)
MaxVal (buyer)

"belief" probability p will be accepted: $f(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)}$

AAG (p)--the number of bids accepted in HM with price >=p.
BG (p)-- the number of bids in HM with price >=p
UAL (p)-- the number of unaccepted asks in HM with price <=p

**Pseudo Code For SELLERS**

If (the last shout was accepted at price q)
    Then
        If p<=q raise profit margin ASK Pnew=hi
If (last shout was a bid)
Then
   Any active seller for which \( p \geq q \) should lower its margin ASK \( P_{\text{new-lo}} \)
Else
   If (the last shout was an offer)
   Then
      Any active seller for which \( p \geq q \) should lower its margin ASK \( P_{\text{new-lo}} \)

• Pseudo Code For BUYERS

If (the last shout was accepted at price \( q \))
Then
   If \( p \geq q \) raise profit margin ASK \( P_{\text{new-hi}} \)
   If (last shout was an offer)
      Then
         Any active buyer for which \( p \leq q \) should lower its margin ASK \( P_{\text{new-lo}} \)
   Else
      If (the last shout was a bid)
      Then
         Any active buyer for which \( p \leq q \) should lower its margin ASK \( P_{\text{new-lo}} \)
**ATTACHMENT 3 – JAVA SOURCE CODE**

Description of the Java Source Code........................................................................................................29
System Classes........................................................................................................................................29
  Main..................................................................................................................................................29
Communication Classes........................................................................................................................30
  KQMLMsg ........................................................................................................................................30
  Register ...........................................................................................................................................31
  OfferRequest ....................................................................................................................................31
  Offer ................................................................................................................................................31
  OfferResponse .................................................................................................................................31
  NextAuction ....................................................................................................................................31
  Close .............................................................................................................................................31
Agent Classes ........................................................................................................................................31
  Facilitator .......................................................................................................................................31
  MarketAgent ..................................................................................................................................32
  BuyerAgent .....................................................................................................................................32
  SellerAgent .....................................................................................................................................32
  ZiBuyerAgent ..................................................................................................................................32
  ZiSellerAgent ..................................................................................................................................32
  ZipBuyerAgent .................................................................................................................................33
  ZipSellerAgent .................................................................................................................................33
Compiling and Running the Application .................................................................................................33
Javadocs .................................................................................................................................................36
Description of the Java Source Code

The code created for the double auction multi-agent system consists of ten classes that can be broken into three categories: system classes, communication classes and agent classes. The system category currently only contains a single class called Main. The communication category represents the different messages used by the agents in the double auction market. Finally, the agent category contains the buyer, seller, and the facilitator agents and related classes. All ten of these classes reside in the edu.psu.mas package. The following is a list of these ten classes, along with a short description of each class.

System Classes

Main

The main class serves as the main driver for the double auction multi-agent system. It contains only a single method; main(). The main method is the method that is called when this class is run and it expects two parameters. The first parameter indicates the type of buyer agents to be used and the second parameter indicates the type of seller agents to be used.

It is helpful to look at the implementation of this main method to get an understanding of the structure of the MAS code:

```java
public static void main(String args[]) throws Exception {
    if (args.length != 2) {
        System.out.println
            ("Usage: edu.psu.mas.Main [BuyerType] [SellerType]");
        System.out.println
            ("Legal types are: ZI or ZIP");
        System.exit(0);
    }

    Facilitator facilitator = new Facilitator(args[0]+args[1]);
    facilitator.initialize();
    facilitator.startAgentProcessing();

    int[][] buyer_limit_prices = {
        {120,100,80,60,40,20},
        {140,120,100,80,60,40},
        {100,80,60,40,21,1}
    };

    int[][] seller_limit_prices = {
        {40,60,80,100,120,140},
        {20,40,60,80,100,120},
        {60,80,100,120,140,160}
    };

    for (int i = 0; i < buyer_limit_prices.length; i++)
    {
```
CIAgent buyer =
createBuyer(facilitator, buyer_limit_prices[i], args[0]);
buyer.initialize();
buyer.startAgentProcessing();
}

for (int i = 0; i < seller_limit_prices.length; i++)
{
    CIAgent seller =
seller.initialize();
seller.startAgentProcessing();
}

The code above is quite simple. First it creates a Facilitator agent and initializes it. After initialization it asks it to start processing. Once the Facilitator is up and running various buyers and sellers can be created. Notice that the agents are created via a calls to either the createBuyer() or createSeller() methods. These method take the agent type specified on the command line and uses it to create the correct agent. For example, here is the createBuyer() method:

private static CIAgent createBuyer(Facilitator facilitator,
int[] limits,
String type) throws Exception
{
    CIAgent agent = null;
    if (type.equals("ZI"))
        agent = new ZiBuyerAgent(facilitator, limits);
    else if (type.equals("ZIP"))
        agent = new ZipBuyerAgent(facilitator, limits);
    else
        throw new Exception("Illegal Buyer Type!");
    return agent;
}

Notice that the agent constructor requires a reference to the Facilitator and a reference to an array of limit prices. Each agent uses the limit prices to determine the value of their bids. After each buyer is created it is initialized and executed. At this point the agents begin to communicate, and once everyone registers with the Facilitator, the auction will begin.

**Communication Classes**

**KQMLMsg**

This is the class used to represent a KQML message. All communication in the system is done using instances of this class. A KQML message contains a performative and a domain specific message object. The remaining four communication classes are examples of the domain specific messages objects that are included in a KQML message. An example of how to create and send a KQML message follows:

CIAgentEvent registerEvent =
new CIAgentEvent(this, new KQMLMsg("tell", new Register(this)));
m_facilitator.postCIAgentEvent(registerEvent);

The code above creates a CIAgentEvent object. This event object is the object provided by the CIAgent framework for sending messages. The first parameter of the CIAgentEvent constructor is a reference to the sender and is used by the Java event dispatcher. The second parameter is the object to be sent. In this system this object will always be a KQML message. The KQML message constructor takes a performative (in this example “tell”) and an instance of a domain specific message object. In this example a “register” message is sent to the Facilitator using the Register class. The constructor of the Register class takes a reference to the agent that is registering.

Register
This is a domain specific message class used in a KQML message. This message is used by agents when the wish to register with the Facilitator.

OfferRequest
This is a domain specific message class used in a KQML message. The Facilitator uses this message in order to request that registered agents present their offers.

Offer
This is a domain specific message class used in a KQML message. This message is used by agents when the wish to submit an offer to the Facilitator.

OfferResponse
This is a domain specific message used in a KQML message. The Facilitator uses this message to notify all registered agents of the results from the most recent round of offers.

NextAuction
This is a domain specific message class used in a KQML message. The facilitator sends this message to notify that the current auction session has ended and a new auction session has started.

Close
This is a domain specific message class used in a KQML message. The facilitator sends this message to notify that all auctions have terminated and the auction is closed. When an agent receives this message it will stop all processing and the thread will terminate.

Agent Classes

Facilitator
The Facilitator class is responsible for coordinating the double auction. Buyer and seller agents register with the Facilitator before the auction begins. After registration, the Facilitator asks registered agents for offers. Once the offers have been received the Facilitator calculates the result of the offers and then notifies registered agents of these results. The three important methods in the Facilitator class are:
processTimerPop() – This method is called every SLEEP_TIME milliseconds and provides the Facilitator with an opportunity to act. The previous phase of the Facilitator (WAIT_FOR_REGISTER or WAIT_FOR_OFFERS) determines how it will behave. If the last phase was WAIT_FOR_REGISTER the Facilitator will terminate its wait for agents to register, send a request for offers, and enter the WAIT_FOR_OFFERS phase. If the previous phase was WAIT_FOR_OFFERS the Facilitator will terminate its wait offers, calculate and send the result of all of the offers, and then request the agents to send the next round of offers.

processCIAgentEvent() – This method is called every MESSAGE_CHECK_TIME milliseconds and provides the Facilitator with an opportunity to respond to messages. If the next message in the message queue is “tell facilitator to register me” the Facilitator will add the sender of this message to its list of registered agents. If the next message in the queue is a “reply to the request of offers” the sender’s offer will be added to the list of current offers.

sendOfferResponse() – This method is called when the Facilitator needs to calculate the result of the current offers and send this result to all registered agents.

**MarketAgent**

This abstract class serves as the base class for all buyer and seller agents in the system. The goal of this class is to make it easy to create new buyer and seller agents. This class only needs to respond to events therefore it does not do anything in its processTimerPop() method. Every MESSAGE_CHECK_TIME milliseconds the processCIAgentEvent() method is called giving the MarketAgent an opportunity to handle any pending messages. All MarketAgents must respond to messages from the Facilitator asking for offers or announcing the result of that last round of offers. The internal algorithm used by the specific agent determines the way each agent responds to these messages. For this reason the MarketAgent dispatches these messages to the sendOffer() and handleOfferResponse() abstract methods. These are the methods that must be overridden by all buyers and sellers to specify the specific bidding algorithms used.

**BuyerAgent**

This class is a simple implementation of a buyer. It extends the MarketAgent class and implements a very trivial bidding strategy by overiding the sendOffer() and handleOfferResponse() methods. This class serves as an example of how to create a new buyer agent and is only used for testing the framework.

**SellerAgent**

This class is a simple implementation of a seller. It extends the MarketAgent class and implements a very trivial bidding strategy by overiding the sendOffer() and handleOfferResponse() methods. This class serves as an example of how to create a new seller agent and is only used for testing the framework.

**ZiBuyerAgent**

This class is a more specialized implementation of a buyer. It extends the MarketAgent class and implements the “Zero Intelligence” bidding strategy by overiding the sendOffer() and handleOfferResponse() methods.
ZiSellerAgent
This class is a more specialized implementation of a seller. It extends the MarketAgent class and implements the “Zero Intelligence” bidding strategy by overriding the sendOffer() and handleOfferResponse() methods.

ZipBuyerAgent
This class is a more specialized implementation of a seller. It extends the MarketAgent class and implements the “Improved Zero Intelligence” bidding strategy by overriding the sendOffer() and handleOfferResponse() methods.

ZipSellerAgent
This class is a more specialized implementation of a seller. It extends the MarketAgent class and implements the “Improved Zero Intelligence” bidding strategy by overriding the sendOffer() and handleOfferResponse() methods.

Class Diagram
Compiling and Running the Application

All of the source code for the MAS is contained in two different packages: ciagent and edu.psu.mas. The ciagent package contains the multi-agent framework code provided by Joseph Bigus and Jennifer Bigus in the book “Constructing Intelligent Agents using Java”. This framework is well described in the referenced text and will not be elaborated on here. The edu.psu.mas package, on the other hand, contains all of the source code described in this document in order to simulate the double auction multi-agent environment.

To compile and execute this code you first need to unzip the source code into a working folder. For the purposes of the following instructions, assume that you have unzipped the software into c:\ist597b\src. Be sure to maintain the folder structure contained in the zipfile when you unpack the
software. If you choose to unpack the code into a different working folder, just substitute your folder in place of c:\ist597b\src while reading these instructions.

After unpacking the code you should have the following folder structure:

C:
  \ist597b
  \src
  \docs
  \ciagent
  \edu
    \psu
      \mas

The classes described in this document are located in the edu\psu\mas folder. To compile and run these classes open a command prompt and make sure that c:\ist597b\src is the current working directory. If it is not use the cd c:\ist597b\src command to change to the correct directory. Next, type the following command to compile the source code:

    javac -classpath . edu\psu\mas\*.java

Be sure to include the "." character after -classpath. When compilation is complete you should be able to run the code using one of the following four commands:

    java -classpath . edu.psu.mas.Main ZI ZI
    java -classpath . edu.psu.mas.Main ZIP ZIP
    java -classpath . edu.psu.mas.Main ZI ZIP
    java -classpath . edu.psu.mas.Main ZIP ZI

The first two commands run auctions using ZI buyers and sellers or ZIP buyers and sellers. The third command runs an auction using ZI buyers and ZIP sellers. The final command runs an auction using ZIP buyers and ZI sellers.

When an auction completes two comma separated data files will be created. The first file will contain the total profit for each agent in the auction. The second file will contain the transaction history for the entire auction. The name of each of these files will start with the agent types and will be followed by either TxHistory or Profits. Finally, a number is appended to the end of the filename that designates the number of times a particular combination of buyers/sellers has been executed. For example, if ZI buyers and ZIP sellers are used, and two different trials are run, the following four files will be generated:

- ZIZIPTxHistory1.csv
- ZIZIPProfits1.csv
- ZIZITxHistory2.csv
- ZIZIPProfits2.csv
**Javadocs**

In addition to this document and HTML based description of the source code is available in the `docs` folder created when you unpacked the source code. To view this documentation just open the `docs/index.html` file in your web browser.