DEMONSTRATING BID RIGGING WITH A SHOPPING AGENT APPLICATION USING JADE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Dinesh Arun Sivanandam

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2011

Fargo, North Dakota

ABSTRACT

Sivanandam, Dinesh Arun, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, November 2011. Demonstrating Bid Rigging with a Shopping Agent Application Using Jade. Major Professor: Dr. Kendall Nygard.

This paper presents the implementation of a shopping-cart agent that demonstrated collusion in online bidding. Collusion is an agreement between two or more persons, sometimes illegal and therefore secretive, to limit open competition by deceiving, misleading, or defrauding others of their legal rights; or to obtain an objective forbidden by law, typically by defrauding or gaining an unfair advantage [15]. It is an agreement among firms to divide the market, set prices, or limit production. The paper has the shopping agent and its bid-rigging methods implemented using the Java Agent Development Framework (JADE), which helps develop agent-based applications. JADE uses agent communication language as part of its framework to provide message transportation between agents. Collusion in online bidding is caused by bid rigging between the vendors. Some basic types of bid-rigging implementation are described in this paper. Each bid rigging shown is different from the other, but all exhibit collusion and profit making by illegal, backdoor communication before a product price and availability are quoted to a potential buyer. Collusion removes any chance of competitive price benefits that a potential buyer might enjoy. The bidding method and its algorithm are explained in detail with the help of diagrams and flow charts. Each agent's communication with the others is also briefly explained. The bid-rigging algorithm for the three flavors of rigging is explained in detail. Graphs and tables are drawn from the output for each bid-rigging method. Bidding run are run for a product in multiple iterations to depict the winning strategy that vendors use as the result of backdoor vendor-to-vendor communications.

To demonstrate the bid-rigging types, a working model of a shopping agent is developed using JADE. The program has the option to select one of the bid-rigging types at a time, say bid rotation, bid suppression, or complementary bidding, and its outputs is saved for analysis.

ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Kendall Nygard, for his support, guidance, and expertise. My sincere thanks to Dr. Jun Kong, Dr. Simone Ludwig, and Dr. Harlene Hatterman-Valenti for serving on the committee.

Finally, I would like to thank my parents; my sister; my wife; my host family, the Vollas of Moorhead; Satheesh Chakravarthi; and other friends for their continuous encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Java Agent Development Framework (JADE)

Java Agent Development Framework, referred to as JADE, is computer software that connects software components and/or applications to multiple processes running on one or more machines, making them interact. JADE is an agent development system that has three main components that facilitate agent development: Agent Platform, Agent Management Services, and Directory Facilitator. JADE (Eclipse) is the Java-based JADE development where agents are created and contained [8].

JADE has a runtime environment, library, and graphic tools. A runtime environment is the base where all the JADE agents reside, execute, and interact with one or more agents. A library is a built-in archive of programs that the agents can use for a direct or some advanced functionality without having to create them every time they are needed. A set of graphic tools are available for the runtime environment to monitor, start, stop, and manipulate the agent activity in NetBeans IDE.

Containers and Platforms

A container is where a group of agents live. A container might have several active and passive agents running. Agents inside a container can communicate between themselves.

Two or more active containers would make a platform (Figure 1). Every platform has a main container. The main container in the platform should be active, and all the other containers in the platform would register to the main container as they start. With the exception of the main container, all other containers in a platform are considered non-main.

The main container has two special agents that are automatically started when it starts. Having the two special agents is a feature that makes it different from other containers because it supports the agents and maintains its functionality. The Agent Management System manages the naming service for all the agents in a platform so that each container in the platform does not have a duplicate name. The Directory Facilitator, or DF, provides the "yellow pages" service through which the agent and the services it provides is listed, so other agents can find agents if needed. We look at Directory Facilitator in more detail in the forth coming section.

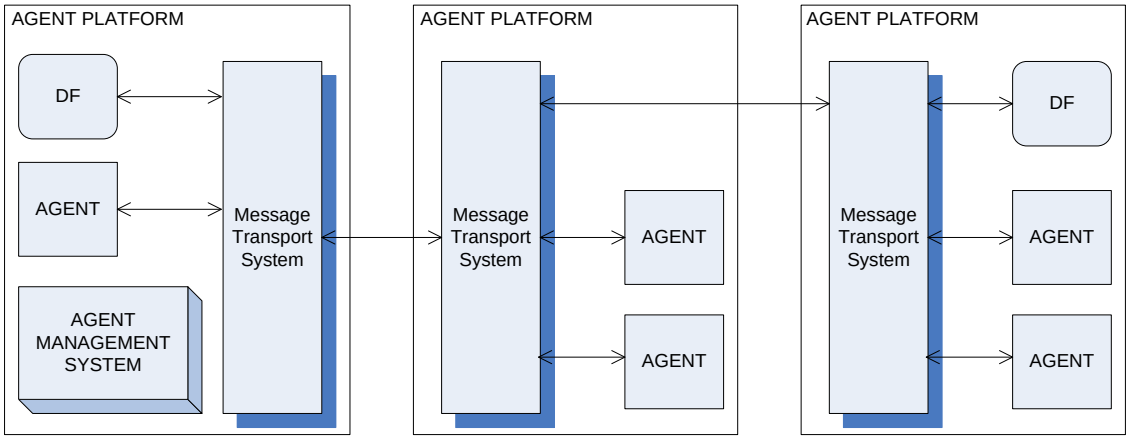

Figure 1. Agent Platform Interactions.

The Agent Management Reference Model gives a formal model in which intelligent physical agents can coexist and operate. It helps in the management of agent creation, agent registration, agent location, agent communication, agent execution, and agent termination.

An agent is a module that is designed to execute certain functionality when its intended computation process occurs in the application. An agent communicates with the rest of the application through agent Communication Language (ACL). Every agent has an owner with a specific identity. This identity is established using an agent identifier called

AID that gives it a unique name inside the agent platform (AP). In the AP, an agent is a player along with other agents where it is performs a single service or a combination of services. These agent services are published in the service description, thereby all agents unite to make an integrated facility model for cumulative services provided by the AP. Each agent makes referrals to the service provider for the description of agents and what they offer, so depending on their requirements, they can communicate with other agents using ACL and can use what other agents have to offer.

Directory Facilitator

Directory Facilitator (DF) is a component that provides yellow-page services to other agents. DF is an optional component, but when it is present, all the agents will register their services to the DF. The DF is where other agents will query for services offered by other agents. There can be multiple DFs operating within as AP and such process is known as a federation.

The DF will always try to maintain a recent and updated list of agents and services. As a result, a DF will have the most current information in an unbiased way for all its authorized agents. While forming a federation, the various DFs register with each other. Each agent should find the appropriate DF and request registration. An agent can also refuse to advertise its service through a DF. The deregistration function can be used by an agent at any moment to break all ties with the DF, and an agent can modify its service description at any time for any reason.

DF supports the following functions for the yellow-page service it offers:

1. Register: Advertise the agent service through the DF with a description.
2. Deregister: An agent breaks ties and removes the agent description.

3. Modify: The agent can modify its description at any time and for any reason.
4. Search: Searches are allowed for a registered agent and its service description.

Agent Management Services

Agent Management Services (AMS) is the second component of an AP. Each AP should always have an AMS. There can be only one AMS per agent platform, and it controls and supervises the communication, access, and message transportation to agents of the AP. AMS has a directory of all agent IDs in the AP. AMS is responsible for issuing the agent identifier (AID) to each agent that register with it.

AMS manages the agent platform (AP), agent creation, agent deletion, and agent life cycle; including agent migration from one AP to another if agent mobility is allowed. An AP can have multiple machines in a single platform, hence allowing AMS managing authority to spam across multiple machines. An AMS can be queried to get the description of an AP.

AMS maintains an index that has all the agents currently living in the AP. This index has the entries of all the agent AIDs.

AMS supports these additional functions to manage the agents residing in a platform as a part of the agent life cycle (Figure 2).

1. Create agent: Creation of a new agent.
2. Invoke agent: Invocation of a new agent.
3. Suspend agent: Freeze the agent from its activities.
4. Resume agent execution: Wake up agent for execution.
5. Execute agent: Trigger an agent to perform its advertized services.
6. Terminate agent: Kill an agent that is at end of its life cycle.

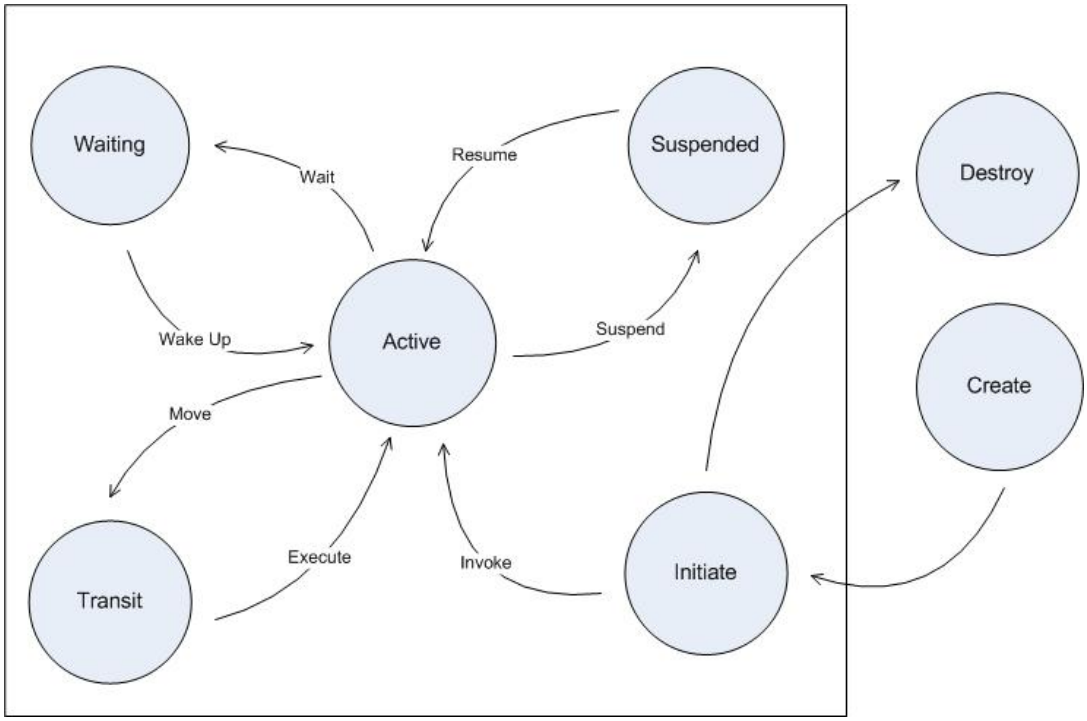

Figure 2. Agent Life Cycle Model.

Agent Platform

Agent platform (AP) is the physical infrastructure that acts like a container, where all the agents, machine hardware, machine operating system, and software that support the agents reside. An AP can have its agents talk to other agents on different agent platforms using message transportation services.

Agent Naming

The Foundation for Intelligent Physical Agents, also called FIPA, set standards for agent naming. Agent naming helps identify the agent using an agent identifier. An agent identifier is a combination of parameter and value pairs. An agent identifier can also have parameters, such as nicknames, roles, etc., to make the agent naming and AP more precise.

The naming parameter is the global unique identifier. It combines the agent and its home AP address using the "@" character.

The addresses parameter lists the transport addresses where a message can be delivered and could be in the URL format. The resolver parameter is where name resolution services are located.

The parameter values of an agent ID can be edited or modified by an agent to update more naming resolution servers and/or transportation addresses. Mandatory parameters can only be changed by the owner of the agent.

Name Resolution

Name resolution is a service is provided by AMS. Name resolution involves the search function resolving the parameters in the AID, thereby resolving the transport address of the agent.

Transportation Address

The transportation address is a physical address where an agent is reachable and cab be reached by specifications through the transport message protocol. An agent can have more than one way of communicating. It is also possible for the agent to have multiple transport address values assigned as parameters in an AID.

Literature Review

The Foundation for Intelligent Physical Agents [1] (FIPA) is the international organization that develops and specifies the design and architectural standards of the agent-based framework. The organization gives specifications to standardize the agent-based application so they can have interoperability and portability across multiple APs. Organization helps avoid multiple localized or platform-dependent applications that cannot communicate with other APs of the same scope but different standards. The goals of FIPA are to deliver specifications for “Agent Management and platform services, Agent

Communication Model and Language and set of Common Interaction Protocols" [2]. The JADE platform offers support for distributed architecture and agents can choose among many message transportation protocols to communicate with agents on the same or different APs. Agent platform makes inter-machine communications possible and application agents portable. Agents are autonomous and control their own thread of execution. Agents can also execute several conversations simultaneously, creating the need for concurrency among them.

An antitrust primer for agents and procurement officials talks [4] about the commitment of the Antitrust Division, FEMA, and other federal law enforcement agencies to ensure clean operation of online or e-based business operations. Federal law enforcement agencies monitor online and e-based business operations to detect and deter anticompetitive conduct in bidding. They design methods to find and charge businesses organizations that engage in bid rigging, fixing prices, and anticompetitive activities. The paper discusses basic types of bid rigging and how the winner is determined in each bid-rigging type. Each bid-rigging model has a way to benefit all the players in some way and when one of the methods is used, it helps all players make profit but deters the bidder from gaining any competitive advantage. The payback method is the most common type chosen. No matter what bid-rigging type is adapted, there are suspicion indicators that point out wrongdoing and price fixing. The paper [4] also discusses what conditions favor collusion and what procurement officials can do about stopping it.

"A Mobile Agent Platform for Supporting Ad-hoc Network Environment" [6] is a research paper that is focused on adopting the AP architecture for implementing broadband mobile communication. Because independent agent-based applications are autonomous,

portable, and can use a variety of communication protocols, they are well suited for mobile implementation. Mobile ad-hoc networks (MANETSs) can use independent physical agents as mobile devices, and they need few physical resources; agents are designed to run on a single thread, thus reducing resource consumption.

"An Ambient Intelligence Application Integrating Agent and Service-Oriented Technologies" [7] presents the agent-based approach in developing a service-oriented architecture for mobility-impaired people. In order to create this function, the paper proposes integrating FIPA-based agent platforms into a service-oriented framework. "Online Bidding" for the construction council [9] discusses the online bidding process used for contract biddings, the advantages and benefits of using unbiased and genuine bidding methods, and the problems of bid rigging and other compromising factors that could cause loss to the business. The cost of training, monitoring and exercising the online bidding safely, as well as methods for improving the processes and procedures of online bidding are discussed.

"Practical Secrecy-Preserving, Verifiably Correct and Trustworthy Auctions" [10] presents the option of using sealed bid auctions where the full details of a bid are not provided to parties to attain a correct and trustworthy auction. The system creates a simple and effective means of controlling collusion by not letting the parties know who all the players and what the complete details are.

"On Cheating in Sealed-Bid Auctions" [11] explores two form of cheating in sealed bid auctions. One scenario is the when a seller illegally accesses the bid price and inserts a fake bid to increase the payment for a winning bidder. Another scenario is when the seller illegally accesses the bid price of a second bidder and uses it for personal winning bid. This

paper discusses the strategies of how bidders are conscious about the possibilities of cheating.

"Coalition Formation in Proportionally Fair Divisible Auctions" [12] discusses a fairly interesting topic of the same owner creating and operating many agents. This situation creates multiple ways collusion could happen because all the agents have an owner. Even though they resemble competing agents, any wins create a mutually beneficial result.

"Online Auctions Efficiency: A Survey of eBay Auctions" [13] studies eBay's online auctioning properties. Auctioning is characterized using common features in this paper: private, where a bidder remains anonymous, vs. public, where bidders' identity is open; open, where all bid amounts are exposed, vs. closed, where bid amounts are concealed; first price vs. second price, where the bidder pays the second highest fair; fixed time vs. auto-extend end time; and hidden reserve price vs. no reserve price.

"Five Sealed-Bid Auction Models" [14] presents four models and how they deal with bid privacy. The article defines some basic properties and optional properties. The basic properties are correctness, confidentiality, and fairness. The optional properties are anonymity, privacy, public verifiability, robustness, price flexibility, and rule flexibility.

CHAPTER 2. PROBLEM STATEMENT

Paper Topic

The goal of the paper is to develop a prototype system using multiple agents with characteristics of pervasive computing (ambient systems, ubiquitous computing). The agent should communicate in a loosely coupled fashion using an agent communication language (ACL). ACL is part of the Java Agent Development Framework (JADE) provided for communication between agents [5].

With my adviser's guidance, I have included a demonstration of what can happen if vendors who set prices are not completely independent but, rather, engage in some type of collusion. This situation would then involve inter-agent message passing among the vendors and involving in collusion and bid rigging to make profit by illegal means. See the following web site [4] for some of the ideas implemented in the shopping agent program in this paper.

Problem Statement

E-Commerce is growing rapidly with the aid of the internet, offering a wide variety of products for online shopping. The problem is that customers who want to shop for a specific product cannot visit each and every vendor's website to find the best possible price customers can get. Potential buyers would like a one-stop shop where the buyer can enter product information and get a list of vendors offering the best price. This paper addresses the issue by developing a Shopping Agent application where the buyer can enter product information and get a recommendation for the best price, including necessary vendor information.

Solution

Create a Shopping Agent Application Using JADE

A shopping agent is created with a buyer agent, seller agent, and vendor agents as in Table 1.

Demonstrate Three Kinds of Bid-Rigging Methods

1. Bid rotation: Everyone in the collusion team gets turns selling his/her product.
2. Bid suppression: One or more vendors withdraw their product from the bidding or shopping list.
3. Complementary bidding: One or more vendors post very high prices that make the intended vendor product look cheaper while the product is still priced higher than what the product worth.

Explanation

The shopping agent will be able to compare prices from various vendor agents where the currently searched product (item) is available. The shopping agent will then provide recommendations to the buyer after comparing the prices. The vendor agents interact with the database to check whether the searched item is present in the database; if yes, the vendor agent returns the best price possible. After retrieving the prices from different vendor agents, the seller agent recommends the best price to the user after comparing the prices offered.

To support the collusion, there are additional inter-agent messages passing among the vendors. The vendors, in turn, manipulate the price to facilitate bid rigging and send altered prices back to the seller agent based on the collusion strategy.

Tasks Involved

Designing a User Agent

The following steps are taken to design a user agent:

1. Design a user interface (using Java swings/web-based interface) to provide a search box and to display the recommended results.
2. Design the user agent using agent communication language (ACL): JADE delegates the request (searched product) and the response (best price after comparing) to and from the seller agent.

Designing Different Vendor Agents

The following steps design the vendor agents:

1. Design a database table for multiple vendor agents by using MySQL for storing the vendor product information along with the offered price.
2. Retrieve the product information from multiple vendor agents that have characteristics of pervasive computing by writing a SQL query.

Vendor agents receive the query from seller agent and they proceed in bid rigging. Vendor agents pass inter-agent messages that help them to fix prices and to eliminate competitive and fair consumer benefits for buyers.

The vendor agents would use a round-robin mechanism to take turns in bid rotation. They may do a combination of bid suppression and complementary bidding in a random fashion. Each agent sends inter-agent messages and sets a price that would benefit or aid one of the vendors. Vendor agents might occasionally opt not to participate in the selling. Process the retrieved SQL query result, and return the altered/tampered price to the seller agent.

Designing a Seller Agent

The following steps are taken to design a seller agent:

1. Communicate using JADE with multiple vendor agents in a loosely coupled fashion.
2. Process the result from multiple vendor agents, compare prices, and return the best price to the user agent.

CHAPTER 3. DESIGN AND ARCHITECTURE

Bidding Algorithm

The bidding algorithm is outlined here and in Figure 3:

1. User agent gets a product from the user.
2. The received product is sent to the seller agent for pricing.
3. The seller agent broadcasts the product information to all the vendor agents.
4. Upon receiving the product information, the seller agent and vendor agents process the request. All the vendor agents submit their prices to the seller agent.
5. The seller agent, after receiving the price, sends the prices to the user agents.

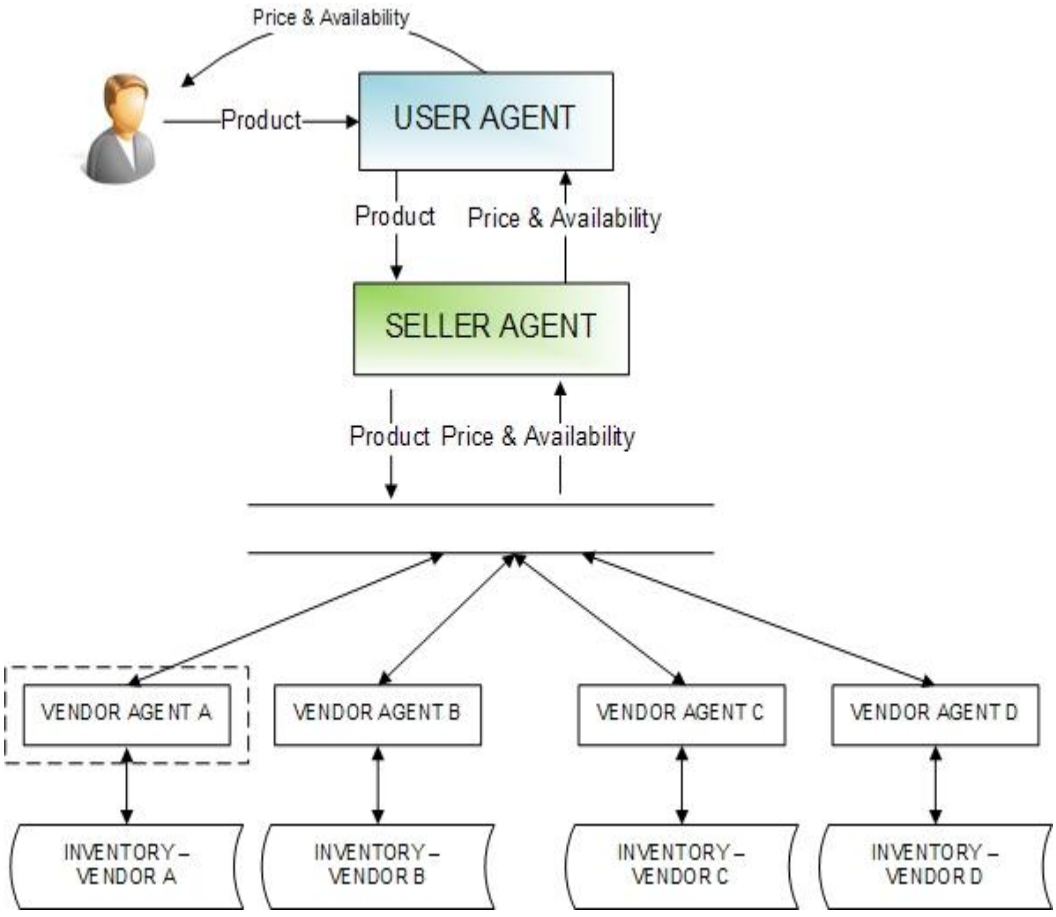

Figure 3. Bidding Algorithm Diagram.

Bid-Rigging Algorithm

The bid-rigging algorithm is outlined here and in Figure 4:

1. User agent gets a product from the user.
2. The received product is sent to the seller agent for pricing.
3. The seller agent broadcasts the product information to the vendor agents.
4. After receiving product information, the seller agent and vendor agents process the request.
5. The vendor agent finds the type of bid rigging to implement from the three choices: bid suppression, bid rotation, and complementary bidding.
6. For the particular bidding type, the dominant vendor agent from the specific bid rigging is read from the database.
7. The dominant vendor agent secretly communicates with other agents using inert communication messages and sends his/her price to the product user agent.
8. Other vendor agents manipulate their prices to favor the dominant vendor agent's price by utilizing the appropriate bid-rigging strategy.
9. All the vendor agents submit their manipulated prices to the seller agent.
10. The seller agent, after receiving the pricing information from vendor agents, sends the prices to the user agent.

User Agent

Functions of a User Agent are shown in Figure 5 and as follows:

1. User agent gets product from the user.
2. The user agent waits and listens for the user to request product information and pricing.

3. The user agent's mechanism of waiting and listening is done through a JADE agent cyclic behavior.
4. Once the product information from the user is received, the user agent creates a message with the product information.
5. And sends the product information to the seller agents.

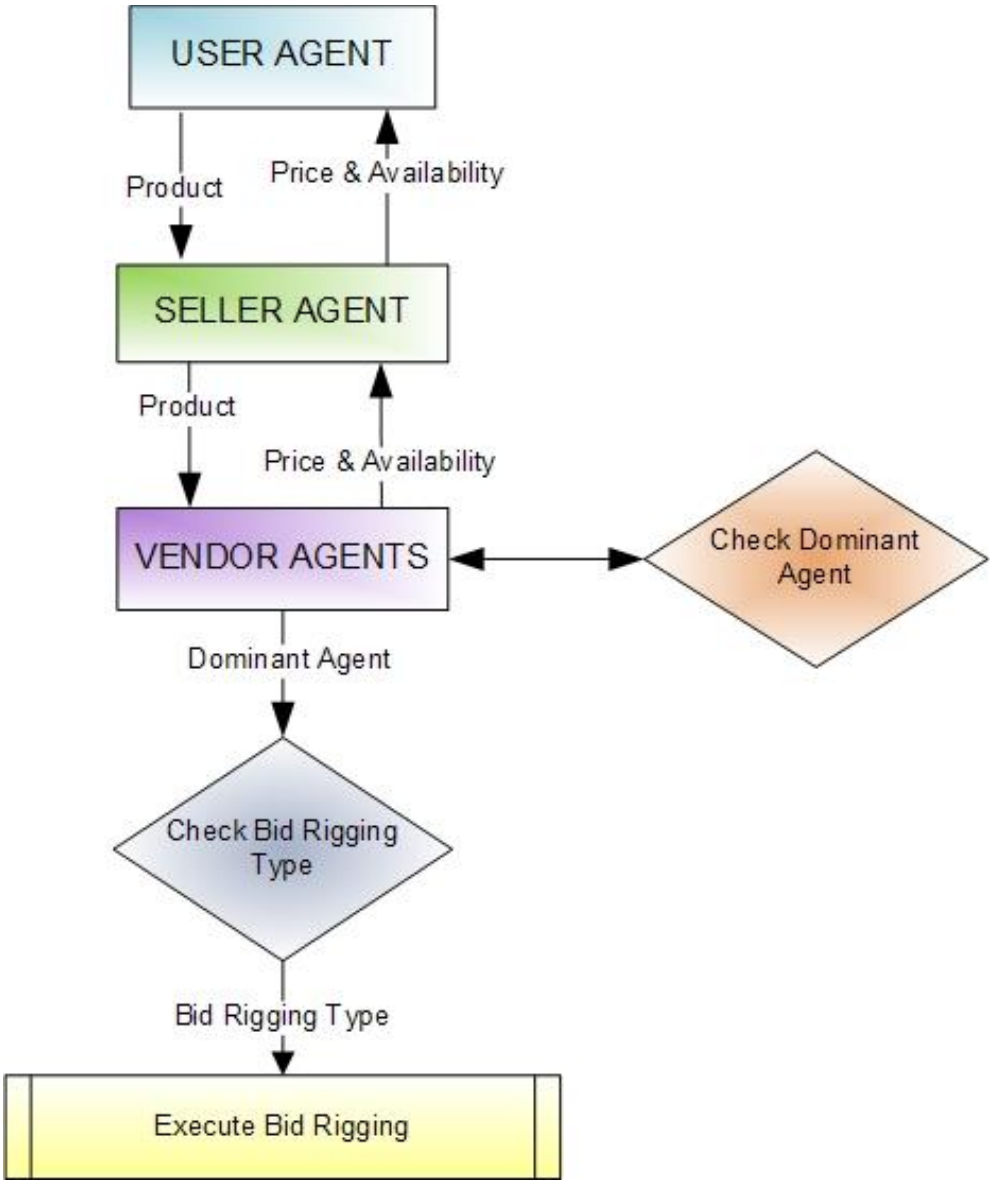

Figure 4. Flowchart of Bid-Rigging Algorithm.

Table 1. Agent Types and Their Functionalities.

Agent Type	**Functionality**
User Agent	Gets product request from user, and requests prices and availability from the seller agent.
Seller Agent	Processes requests from the user agent and enquires about pricing and availability from vendor agents.
Vendor Agents	Gets a product request from a seller agent, checks inventory, and replies to the seller agent with availability and pricing.
Number of Vendors	Vendor A, Vendor B, Vendor C, and Vendor D are the four vendors. They each perform the same function.

Figure 5. User Agent's Diagram.

Seller Agent

Functions of Seller Agent are shown in Figure 6 and as follows:

1. Seller agent waits and listens for a product from the user agent.
2. The seller agent exhibits a JADE agent cyclic behavior until the agent is stopped.
3. The seller agent creates a broadcast message, sends it to all the vendor agents, and then waits for a response.
4. After receiving the price, the seller agent sends the price to the standalone output user console.

Figure 6. Seller Agent's Diagram.

Vendor Agent

Functions of Vendor Agent are shown in Figure 7 and as follows:

1. The vendor agent gets the product information from the seller agent.
2. The vendor agent checks what collusion type is being requested in bid config-en properties.
3. Based on the collusion type, the vendor agent chooses which module of bid rigging to be demonstrated: bid-rotation, bid suppression, or complementary bidding.
4. Once received, prices are sent back to seller agent.

No Rigging

No rigging is shown in figure 8 and explained below:

1. A no rigging bid run is when there is no collusion between the vendors.
2. The user sends the product he wants to the user agent.
3. The received product is sent to the seller agent for pricing.

Figure 7. Vendor Agent's Diagram.

4. The seller agent broadcasts the product information to all the vendor agents.
5. Upon receiving the product information, the seller agent and vendor agents process the request. All the vendor agents submit their prices to the seller agent independently.
6. The seller agent, after receiving the pricing information, sends the prices to the user agents.
7. The user has the advantage of finding the cheapest price when the vendors compete against each other to sell the product.

Types of Bid Rigging by Vendor Agents

This paper discusses about the three bid-rigging methods bid rotation, bid suppression and complementary bidding. The three bid-rigging algorithms and their operation steps are briefly explained in the following sub sections and their respective figures. Each rigging method is different from one other in the way collusion and rigging is implemented.

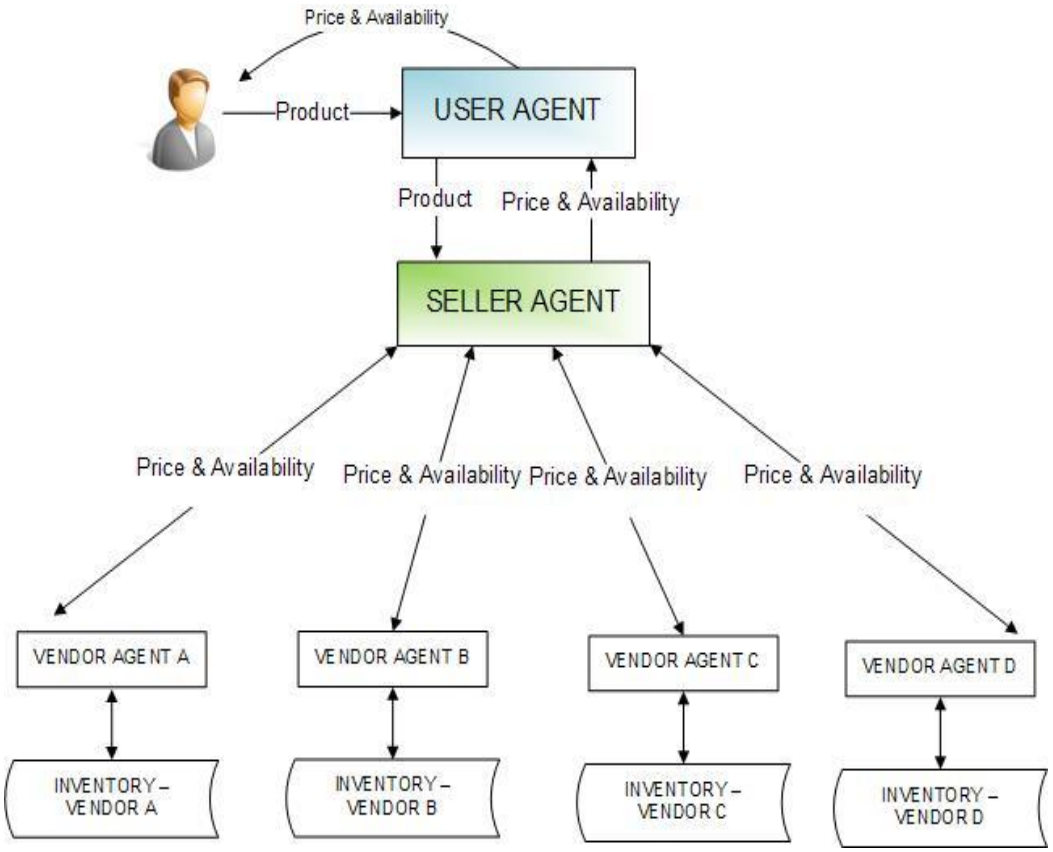

Figure 8. No Rigging Diagram.

Bid Rotation

Bid rotation is shown in figure 9 and explained below:

1. The vendor agent checks bid config-en properties and finds the bid-rigging method currently demonstrated.
2. If bid rotation is selected, the vendor agents executes that collusion module and checks for the superior agent from the database table.
3. Once chosen, the superior vendor agent dominates the bid and secretly sends its posted price to all the other inferior agents.
4. All other inferior agents correspondingly post a relatively higher price than the price posted by the superior vendor agent.

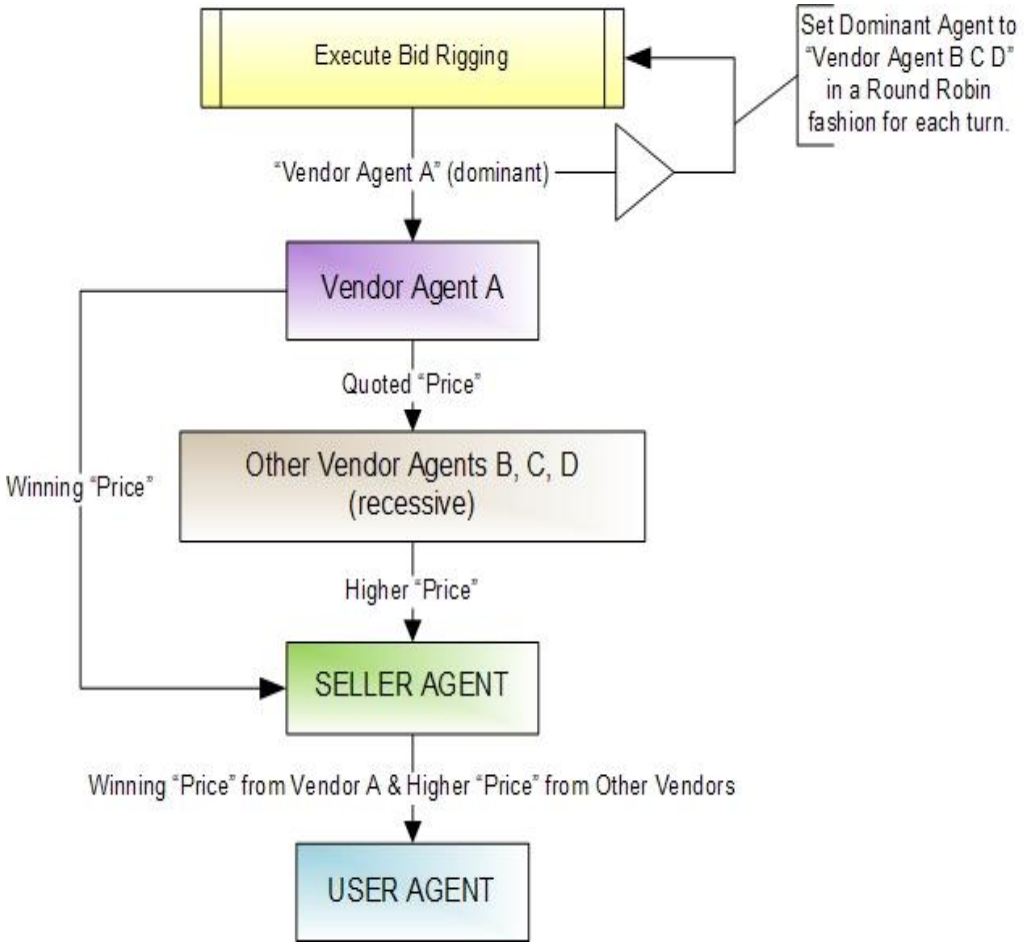

Figure 9. Bid Rotation Diagram.

5. In bid rotation, the next inferior agent dominates when the next bidding occurs. The database is reset with the next agent as the superior agent for the next bidding.

Bid Suppression

Bid suppression is shown in figure 10 and explained below:

1. The vendor agent checks bid config-en properties and finds the bid-rigging method currently demonstrated.
2. If bid suppression is selected, the vendor agent executes that collusion module and checks for the superior agent from the database.

3. The superior vendor agent dominates the bid and secretly sends its profitable price to all the other inferior agents.
4. Some inferior agents then randomly choose to withdraw from posting a price and pretend the product is out of stock.
5. This randomized inferior agent withdrawal choice is generated at each bid run to keep the rigging undetectable by pattern analysis.
6. The superior agent suppresses some of its competition and makes agents bow out of bidding with "product out of stock" messages.
7. Superior agent then remains the only vendor to offer the product in demand at a profitable price.
8. In bid suppression, the next inferior agent dominates when the next bidding occurs.
9. The database is reset with the next agent as the superior agent for the next bid so other vendors can have winning bid run.
10. Bid suppression uses withdrawal and profitable price posting from the superior agent which makes it one of the toughest rigging methods to be identified by pattern analysis of bidding data.

Complementary Bidding

Complementary bidding is shown in figure 11 and explained below:

1. The vendor agent checks bid config-en properties and finds the bid-rigging method currently demonstrated.
2. If complementary bidding is selected, vendor agents execute that bidding module and check for the superior agent from the database.

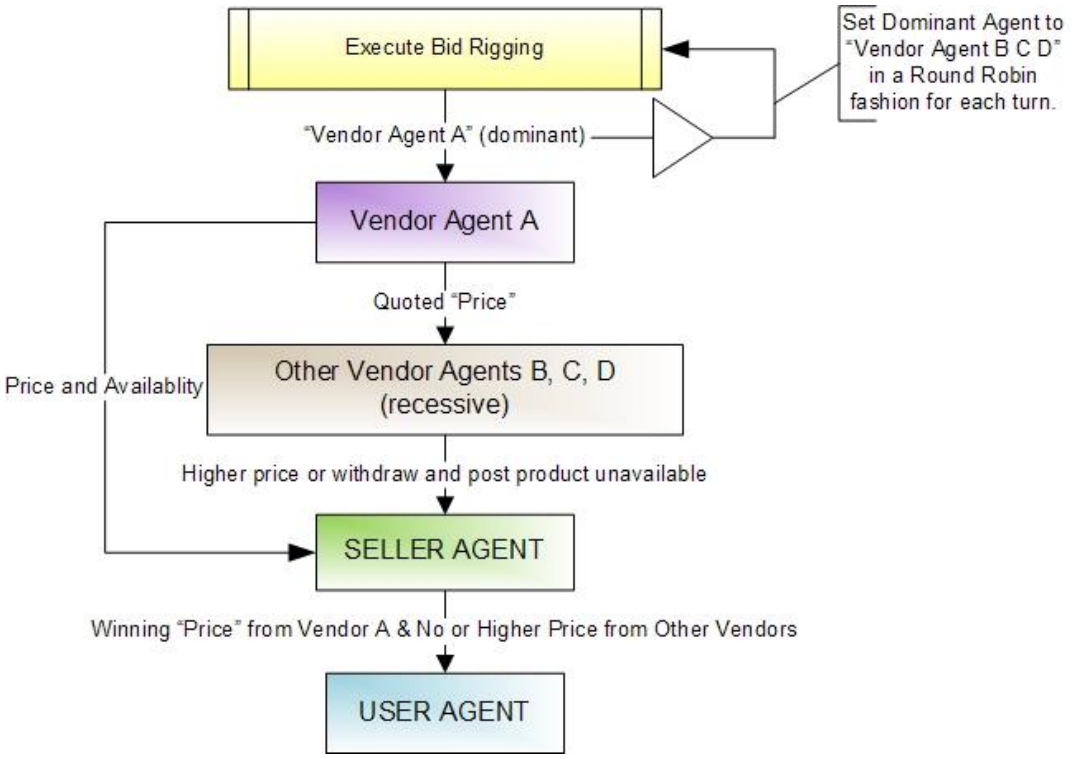

Figure 10. Bid Suppression Diagram.

3. The superior vendor agent dominates the bid and one agent becomes the superior agent as the database describes.
4. The superior vendor agent secretly sends its posted price to all the other inferior agents who are waiting to receive it.
5. These inferior agents correspondingly post a higher price than the price posted by the superior vendor agent.
6. These inferior agents randomly withdraw from posting a bid.
7. This helps the superior agent to post a winning bid.
8. Once the bid run is over, the database is reset with a random vendor agent as the superior agent for the next bidding run.

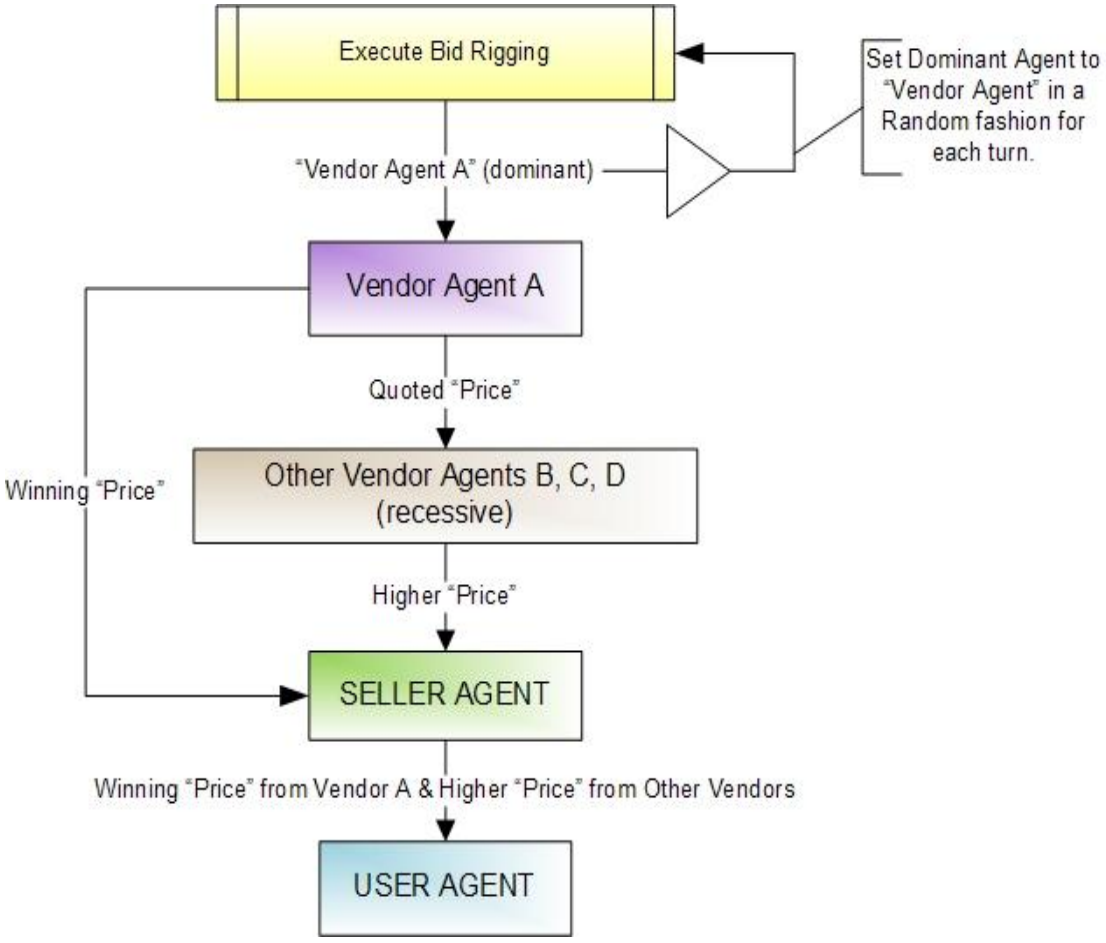

Figure 11. Complementary Bidding Diagram.

CHAPTER 4. IMPLEMENTATION

A shopping agent application has been created, demonstrating what can happen with prices if vendors are not completely independent but, rather, engage in some type of collusion. All the agent classes are created using the JADE programming language. Each agent is built with appropriate behaviors and executes them when engaged.

Every agent gets an agent identifier, gets initialized when required, and is taken down after its intended purpose is completed. NetBeans is the integrated development environment (IDE) used for developing JADE. It comes with built-in templates that help create the classes and code easily. It also helps in compiling and running the code. NetBeans has a GUI which aids in showing when a main container is started and other agents when they are started.

Agent Class

Creating an Agent Class

A JADE class is created by extending the jade.core.Agent class and invoking the setup method. Example is shown in the sample seller agent code below, where the class Seller Agent extends Agent.

```
// creating a JADE agent
public class UserAgent extends Agent {
    @Override
    protected void setup() {
     // implementation
 }
}
```

Agent Identifiers

Agent identifiers are instances of the jade.core.AID class. The agent identifier has a structure of <agent name>@<platform-name>, so the agent will have a unique global ID.

The AID is created using the configuration template on the NetBeans IDE as shown in the following code.

```
// agent identifier
AID agentID = new AID("VendorAgentB", AID.ISLOCALNAME);
```

Agent Initialization

Agent initialization was done using NetBeans IDE, where the run template takes the agent names of the AID and compiles them.

```
// initializing the JADE ***
Apr 21, 2011 1:29:34 PM jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 3.6 - revision 6032 of 2008/05/05 14:07:10
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
----------------------------------------
Apr 21, 2011 1:29:35 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement
initialized
Apr 21, 2011 1:29:35 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Apr 21, 2011 1:29:35 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Apr 21, 2011 1:29:35 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Apr 21, 2011 1:29:35 PM jade.core.messaging.MessagingService
clearCachedSlice
INFO: Clearing cache
Apr 21, 2011 1:29:36 PM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser
com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAX
Parser
Apr 21, 2011 1:29:36 PM jade.core.messaging.MessagingService
boot
INFO: MTP addresses:
http://134.129.225.203:7778/acc
Apr 21, 2011 1:29:36 PM jade.core.AgentContainerImpl
joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@hex is ready.
--------------------------------------------
```

When the main class is run, a disclaimer that the JADE runtime has started is printed. All kernel services are started when the platform starts. Finally, a message saying "Main-Container is ready" appears.

Agent Takedown

When the agent has to be terminated, a method named doDelete () is called. This method does the opposite of what a setup () method would do. This doDelete () initiates the takedown () method that terminates agents and does the cleanup operation.

Behavior Class

The behavior class has the actual task an agent performs. A behavior represents the task that an agent would execute when the conditions occur. A behavior can be created to function at the start of an agent, or it can be placed inside another behavior. Behavior is implemented as a class that extends jade.core.behaviors.Behaviour, and the addBehavior () method is used to add a new behavior. There are many types of behavior a class can choose from for its intended purpose.

Behavior Execution

Scheduling of an agent's behavior is not pre-determined, but more in tune for performing its action () method and returning a value. So depending on the agents function the agent behavior is determined.

Advantages of JADE Behavior

1. JADE behavior performs well with limited resources. It uses one Java thread per user, so machines with small resources benefit.
2. It provides better performance than Java threads switching because the behavior switch is much faster.

3. No two behaviors get into a deadlock trying to access the same resources because all behaviors run on a single thread.

Types of Behaviors

One-Shot Behavior

The one-shot behavior action method is executed only once and completes after that specific execution. I will not go into detail about this behavior because it was not used it for this project.

Cyclic Behavior

Cyclic behavior is used in this shopping agent project. A cyclic behavior never stops and goes on executing the action () method every time it is called. The cyclic class is called by extending the CyclicBehavior class. When invoked, it repetitively executes the operation. All the vendors exhibit the cyclic behavior in this paper constantly waiting for the seller agent to send them a product request.

Other Behaviors

Generic Behavior, Sequential Behavior, Parallel Behavior, and FSM Behavior are JADE behaviors we did not use but they can be used in the future work if need arises for more behaviors.

Agent Communication Language (ACL)

The Foundation for Intelligent Physical Agents is an IEEE Computer Society that seeks to oversee the fit with other technologies through standardizations [3]. FIPA provides the standard specifications for ACL. The specification includes the high-level communication protocols, requesting for an action, and the response to the specific action. The FIPA agent communication language is based on speech act theory [2].

ACL Message Class

JADE agents communicate with each other through a specialized message passing API referred to as ACL Message and follow the agent communication language (ACL) paradigm. Using the setContentObject () method and the getContentObject () methods, one can send serialized Java objects over the content of an ACL Message. The code snippet below shows the various ways an ACL Message can be created; an agent can propose a message using the PROPOSE message type, and the receiving agent can either accept or reject the proposal of the message using the ACCEPT_PROPOSAL or REJECT_PROPOSAL message types, respectively.

```
ACLMessage proposeMessage = new
ACLMessage(ACLMessage.PROPOSE);
ACLMessage  acceptProposalMessage = new
ACLMessage(ACLMessage. ACCEPT_PROPOSAL);
ACLMessage  acceptProposalMessage = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
ACLMessage informMessage = new ACLMessage(ACLMessage.INFORM);
ACLMessage agreeMessage = new ACLMessage(ACLMessage.AGREE);
ACLMessage cancelMessage = new ACLMessage(ACLMessage.CANCEL);
```

Sending Messages

To pass a message between agents, one agent needs to create an instance of an ACL Message and then add the intended recipients using the addReceiver () method. Finally, the agent sends the message using the send () method, See Figure 12 and listed in code below.

```
// create an instance of ACLMessage.
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
// message to be sent.
msg.setContent("IPOD Touch 16 GB");
// whom to send.
msg.addReceiver(new AID("SellerAgent", AID.ISLOCALNAME));
// send the message.
send(msg);
```

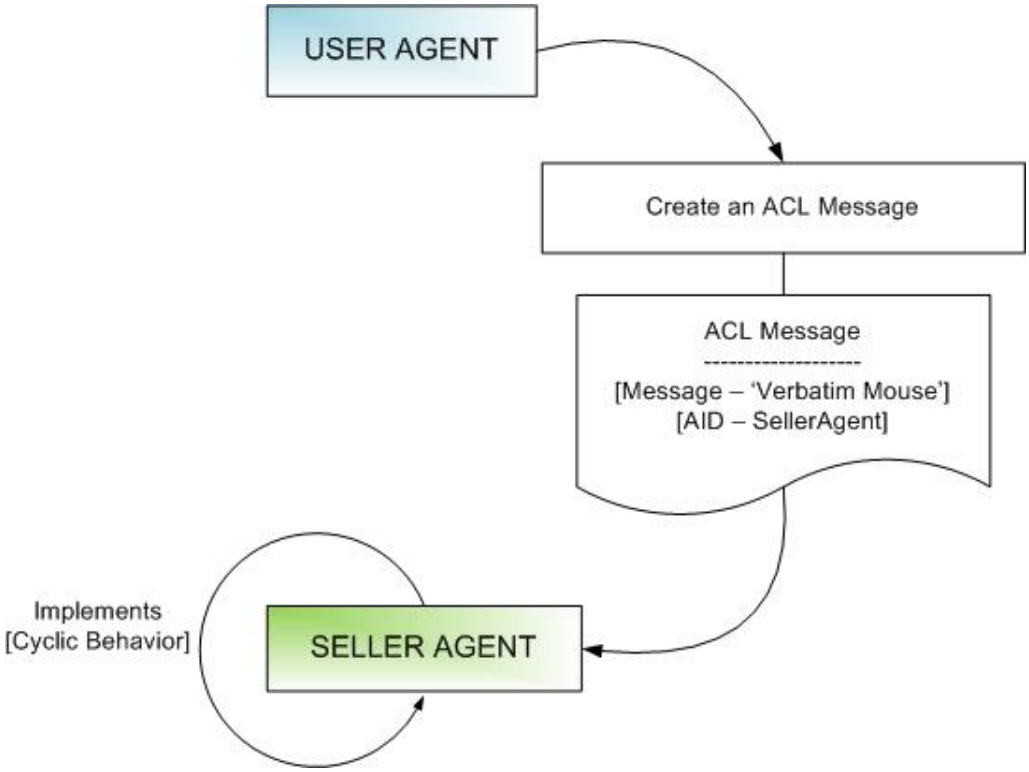

Figure 12. Sending ACL Message.

Receiving Messages

To receive a message, an agent needs to watch for an intended message. To listen constantly, I created an agent with cyclic behavior which listens for a message at all times. After receiving a message, the agent extracts the address information and the content. Based upon the address information, the agent takes the necessary action.

The recipient agent implements a cyclic behavior which *waits* for a message to be received, and waiting is implemented by calling the block () method. The message itself is received by calling the receive () method which returns an instance of the ACL Message (Figure 13). Once the message is received, the action () method can process the message and communicate to various other agents (if needed).

Listed below is the sample code for receiving ACL message.

```
addBehaviour(new CyclicBehaviour(this) {
     public void action() {
          //get messages from RECEIVER, if it sends any.
           ACLMessage msg = receive();
           if (msg != null) {
                    System.out.println("\n - " +
myAgent.getLocalName() + " received: " + msg.getContent());
           }
           block(); // wait for a message to receive.
     }
});
```

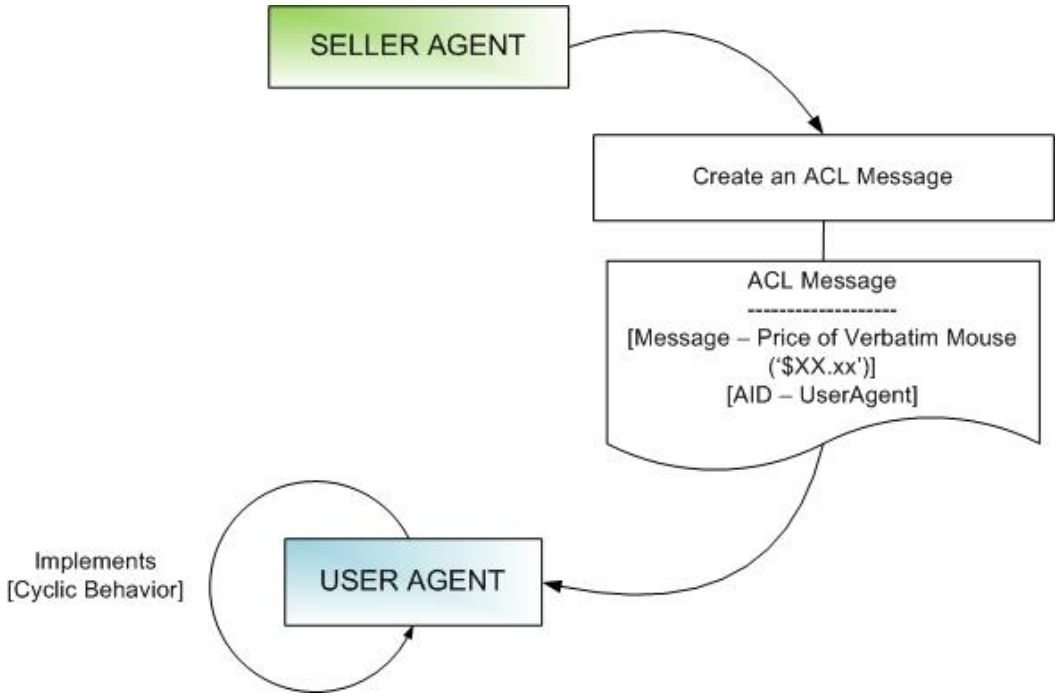

Figure 13. Receiving ACL Message.

Building the Database

A database was created with list of products for each vendor. Each vendor has products in common and products unique to itself. The database for each vendor has the repository details of product name, product description, availability, unit price, and product

ID. When a buyer agent sends a request for product to vendors, they check their databases for product price and availability.

Making the database more realistic was important, so we can run the experiments and collect data that would resemble a real-life bidding transaction. The realistic database was created by browsing the Amazon website for a list of products. Amazon is a one-stop shop where a user can look for a product and see the different prices posted by a group of sellers. The prices are posted on the same page, so the buyer can select the best seller and the lowest price for a deal. For each product, the product price, description, and availability were copied from a seller and assigned to a vendor's database. The same steps were repeated until a list of products, prices, and other details was gathered to make four vendor databases.

MySQL was used to create an online database for the project on NDSU's obiwan server. The vendors were programmed to check their online inventory and to post product price and availability. The screenshot Figure 14 shows how product, price and availability were entered.

Contract Superior Agent Table

MySQL was also used to create the contract tables as shown in the Figure 15. Each contract table told the appropriate bid-rigging scheme who was the superior vendor agent was for that particular bid run. The vendor agents were programmed to check their corresponding contract table for the superior agent on each bid run. Vendor agents also update the table with the new superior agent at the completion of each biding run. The updated contract table is used for identifying the superior agent on the next bid-rigging run.

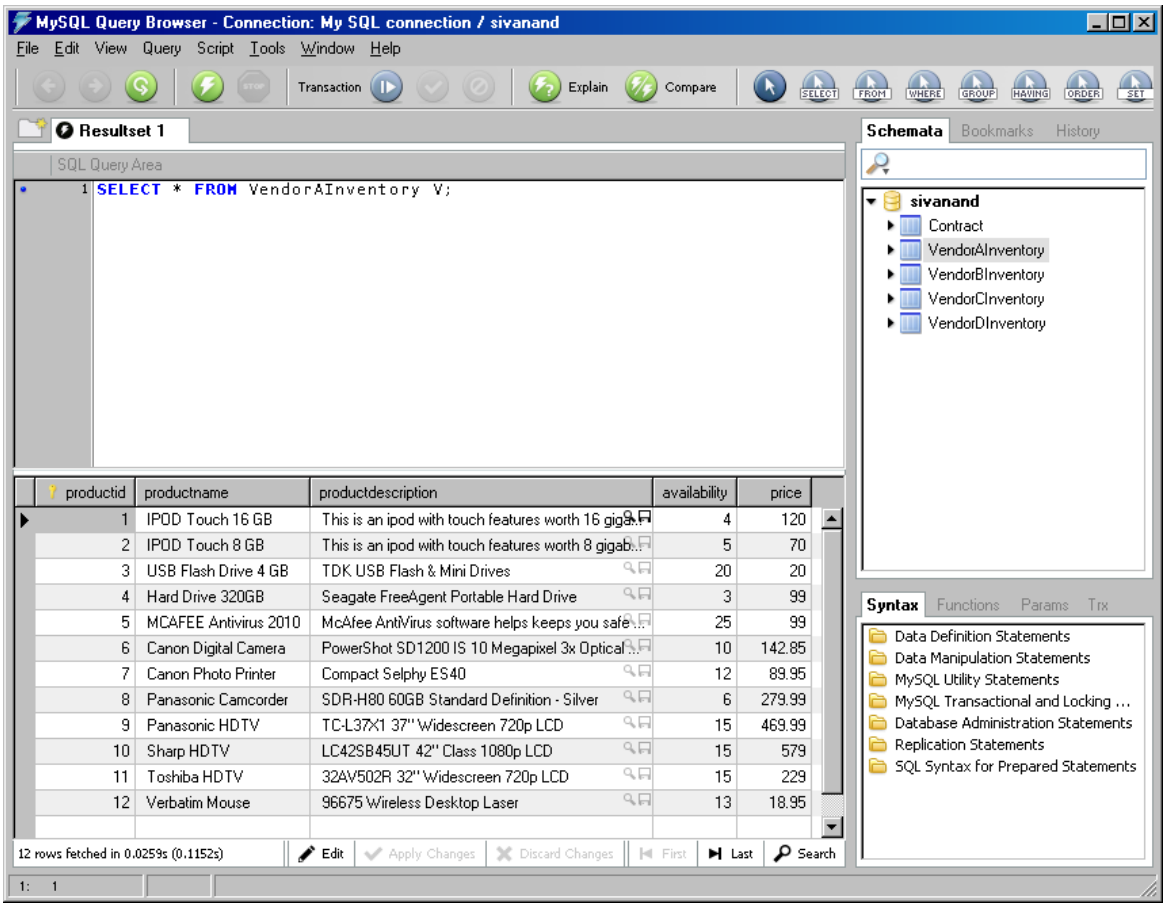

Figure 14. Sample Vendor (A) Inventory Table.

The algorithm for updating the contract table is different based on each bid-rigging scheme and its method of execution.

Inventory Tracker Webpage

A webpage was created using Perl script that displayed the database repository. The webpage was hosted at obiwan.cs.ndsu.nodak.edu. A dropdown box gave the option to click and view any one of the vendors in the database (Figure 16 and 17). The webpage displayed the vendor's database with the product name, description, unit price, and availability when each vendor agent is selected.

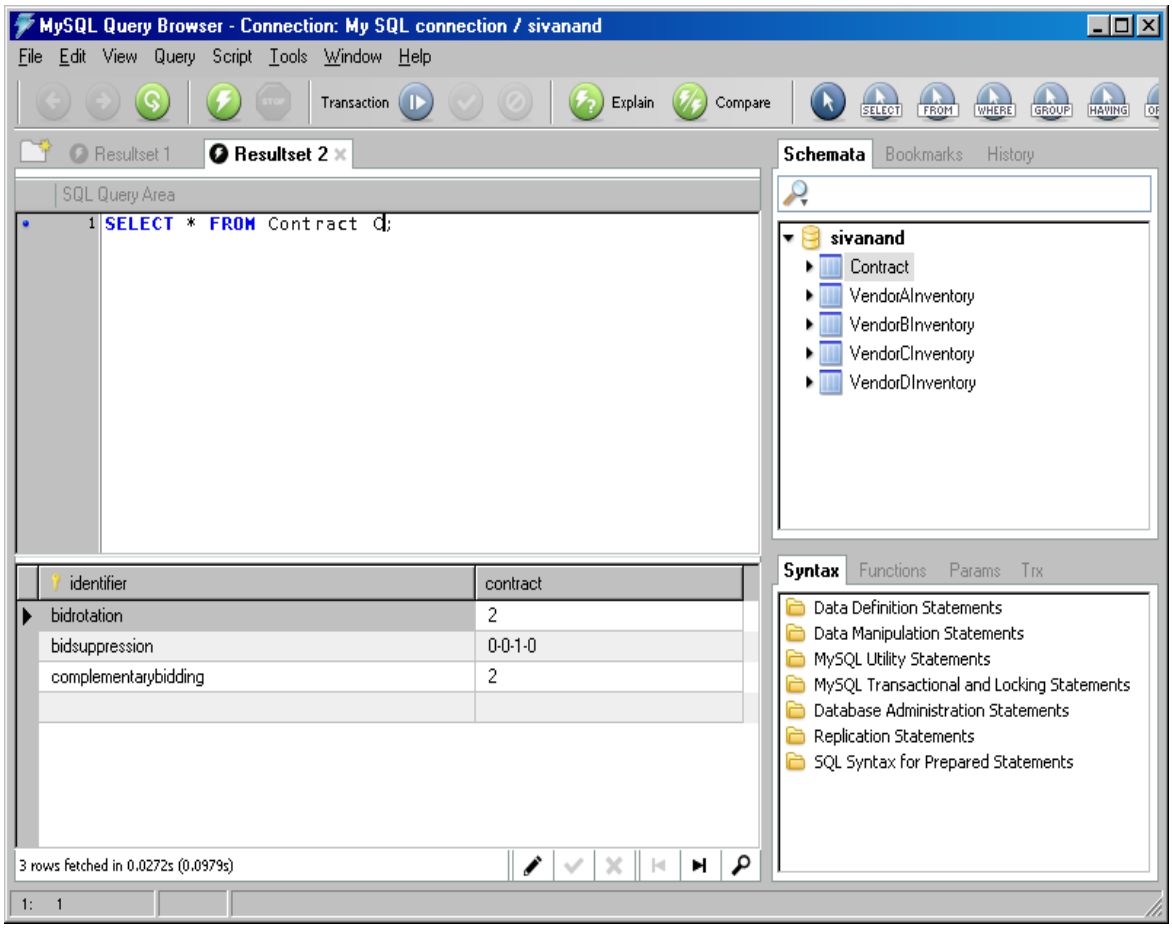

Figure 15. Contract Database Table.

Selecting the Bid-Rigging Method

The bid-rigging methods demonstrated in this paper can be easily configured using a properties file. When the user wants to run a specific bid-rigging method, for example "bid suppression," the specific method is enabled and the other methods are disabled as shown in the Figure 18.

This allows the user to choose between the different bid-rigging methods he could use for each bid run. We would discuss the tool used to create the shopping cart program in the next section.

Tools Used in this Project

The tools used in this project are NetBeans version 6.8; JDK version 1.6; JADE API; JADE Tools API for coding and running the shopping cart agent program. Dreamweaver 8.0 was used for developing PHP, HTML, and CSS code for the webpage.

MySQL database was hosted at obiwan.cs.ndsu.noda.edu as a webpage. MySQL Query Browser for creating and viewing the database. Operation system in which all these programs were installed and used was Windows XP.

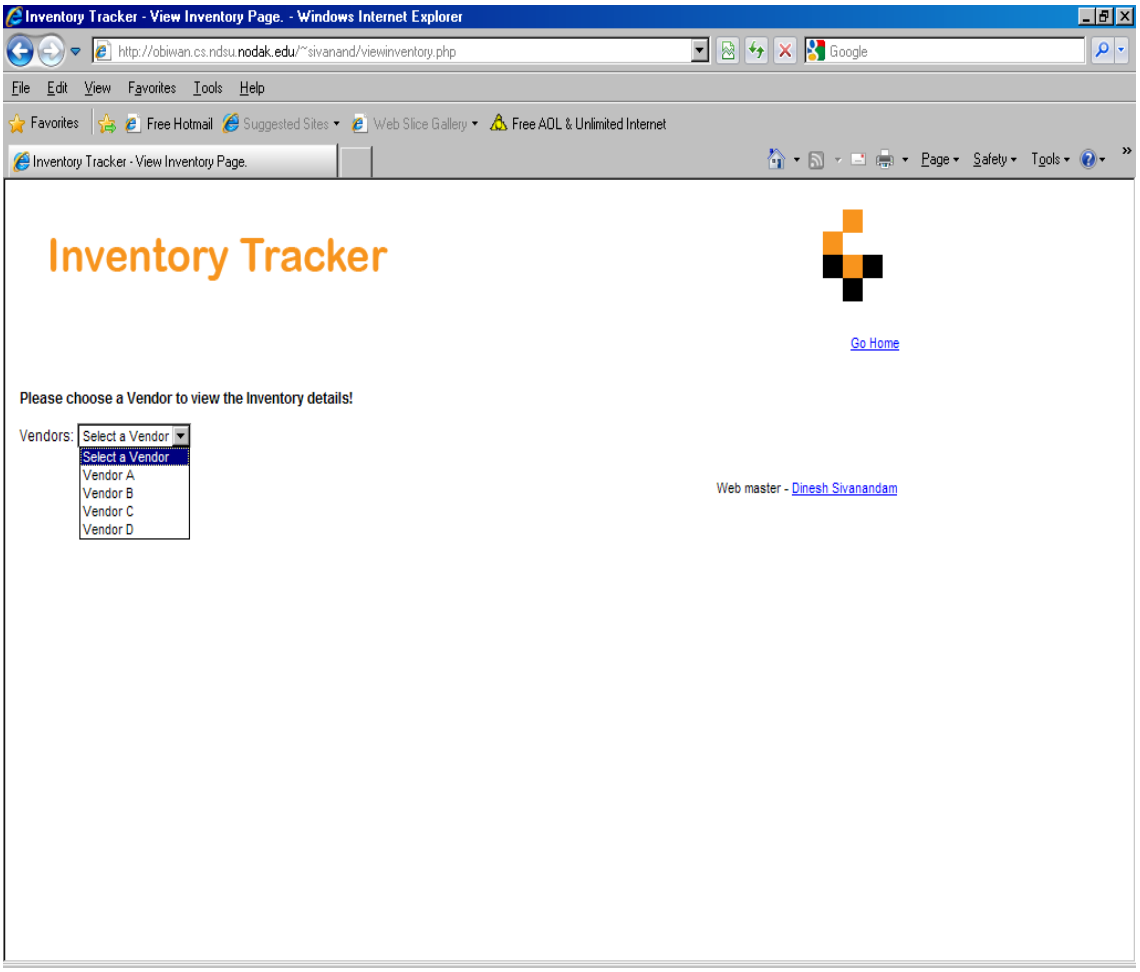

Figure 16. Inventory Tracker Webpage.

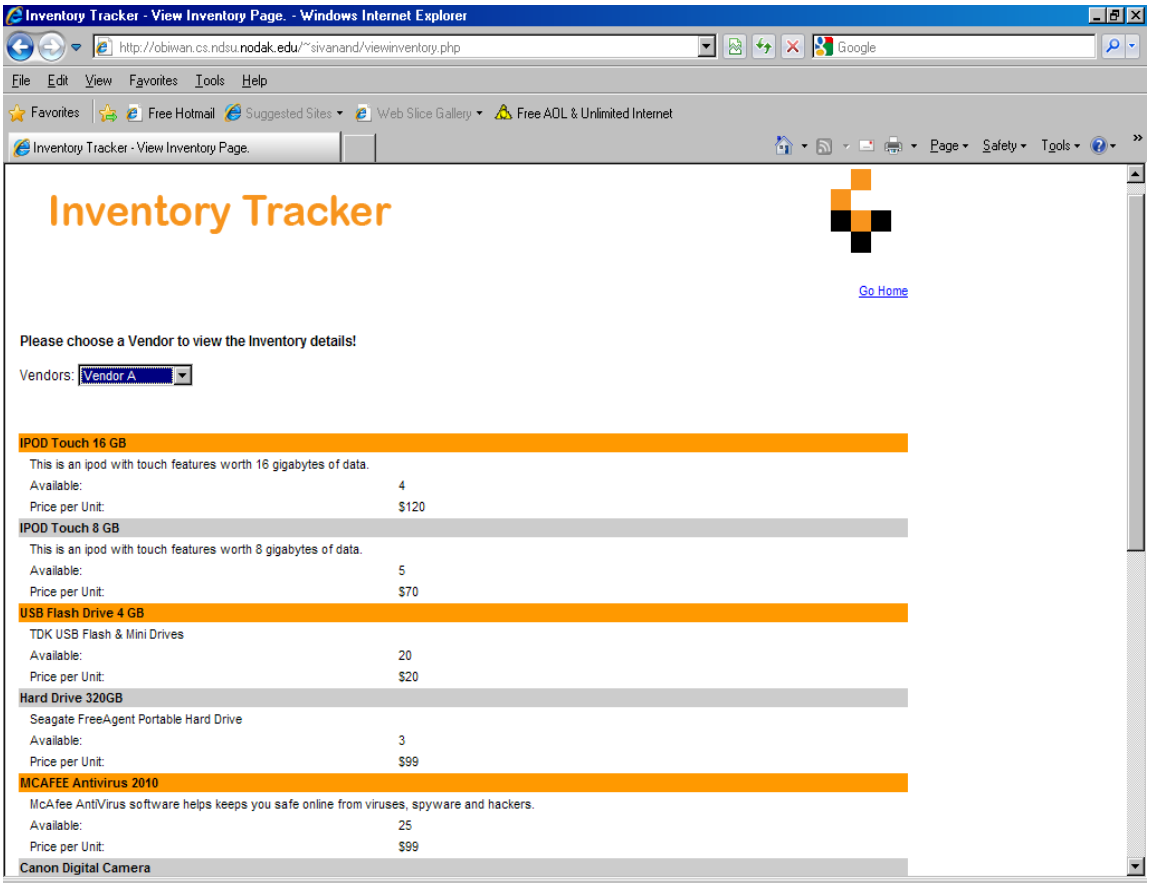

Figure 17. Vendor A's Repository on Inventory Tracker Webpage.

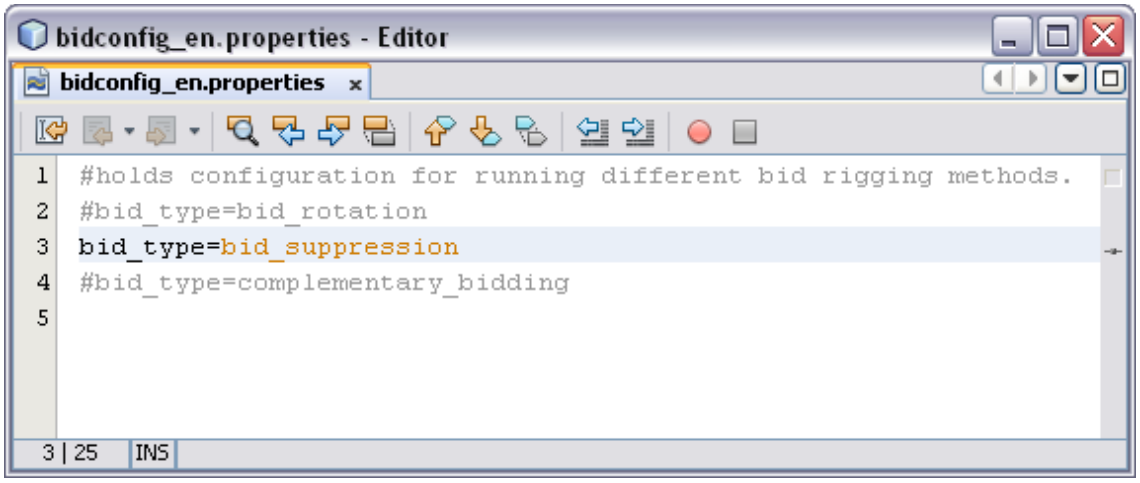

Figure 18. Bid Configuration Properties.

CHAPTER 5. EXPERIMENTS, RESULTS, AND ANALYSIS

Experiments

The results of the no-rigging experiment are used as the control output. The control output is used to compare the other bid-rigging experiments' sales and profits. Experiments were run for one product at a time, under no rigging, to acquire the control outputs. These outputs include the profit and winner of the bidding run. Experiments were run for multiple products under no rigging; then mean profit was calculated for each vendor.

Experiments were run for one product at a time under each bid-rigging method. The output included the profit and the winner of the bidding run. Once experiments were run for multiple products under each bid-rigging method, the mean profit was calculated for each vendor.

Agent Initialization Outputs and Sample Test Results

Bid Rotation Vendor Agent Initialization and Table Outputs

Sample output that demonstrates vendor agent initialization for collusion-type bid rotation.

```
Vendor-B started listening to client-request(s)...
Vendor-D started listening to client-request(s)...
Vendor-C started listening to client-request(s)...
Vendor-A started listening to client-request(s)...
Vendor-A received: Verbatim Mouse
DEMONSTRATING...BID ROTATION
Vendor-C received: Verbatim Mouse
DEMONSTRATING...BID ROTATION
Vendor-B received: Verbatim Mouse
DEMONSTRATING...BID ROTATION
Vendor-D received: Verbatim Mouse
DEMONSTRATING...BID ROTATION
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
```

```
[MESSAGE]: database connection successfully established!
[MESSAGE]: updated contract for 'bidrotation' with a value of
'4'
Contract successfully updated!
```

Vendor Agents Posting Prices and Availability for Bid Rotation

Sample output that demonstrates vendor agents posting price and availability for collusion-type bid rotation.

```
[MESSAGE]: database connection successfully established!
PRODUCT RECEIVED FROM USERAGENT: Verbatim Mouse
PRICE RECEIVED FROM Vendor C IS: 21.75
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor B IS: 24.58
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor A IS: 25.01
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor D IS: 25.23
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
```

Following tables 2, 3, 4 and 5 show four iterations of bidding that demonstrate collusion-type bid rotation.

Table 2. Output of Bid Rotation: Iteration 1 (Vendor A is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	129.6	129.6
Vendor B	IPOD Touch 16 GB	118	146.45	
Vendor C	IPOD Touch 16 GB	115	142.56	
Vendor D	IPOD Touch 16 GB	118	145.15	

Table 3. Output of Bid Rotation: Iteration 2 (Vendor B is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	158.12	137.5
Vendor B	IPOD Touch 16 GB	118	137.5	
Vendor C	IPOD Touch 16 GB	115	151.25	
Vendor D	IPOD Touch 16 GB	118	154	

Table 4. Output of Bid Rotation: Iteration 3 (Vendor C is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	148.12	128.8
Vendor B	IPOD Touch 16 GB	118	145.54	
Vendor C	IPOD Touch 16 GB	115	128.8	
Vendor D	IPOD Touch 16 GB	118	144.26	

Table 5. Output of Bid Rotation: Iteration 4 (Vendor D is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	150.63	130.98
Vendor B	IPOD Touch 16 GB	118	148.01	
Vendor C	IPOD Touch 16 GB	115	144.08	
Vendor D	IPOD Touch 16 GB	118	130.98	

Complementary Bidding Vendor Agent Initialization and Table Outputs

Sample output that demonstrates vendor agent initialization for collusion-type complementary bidding.

```
Vendor-D started listening to client-request(s)...
Vendor-A started listening to client-request(s)...
Vendor-C started listening to client-request(s)...
Vendor-B started listening to client-request(s)...
Vendor-D received: Verbatim Mouse
DEMONSTRATING...COMPLEMENTARY BIDDING
Vendor-C received: Verbatim Mouse
DEMONSTRATING...COMPLEMENTARY BIDDING
Vendor-A received: Verbatim Mouse
DEMONSTRATING...COMPLEMENTARY BIDDING
Vendor-B received: Verbatim Mouse
DEMONSTRATING...COMPLEMENTARY BIDDING
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: updated contract for 'complementarybidding' with a
value of '3'
Contract successfully updated for 'Complementary Bidding'!
```

Vendor Agents Posting Prices and Availability for Complementary Bidding

Sample output that demonstrates vendor agents posting price and availability for collusion-type complementary bidding.

```
[MESSAGE]: database connection successfully established!
PRODUCT RECEIVED FROM USERAGENT: Verbatim Mouse
PRICE RECEIVED FROM Vendor B IS: 19.74
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor A IS: 22.7
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor D IS: 22.9
[MESSAGE]: database connection successfully established!
PRICE RECEIVED FROM Vendor C IS: 22.5
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
```

Following tables 6, 7, 8 and 9 show four iterations of bidding that demonstrate collusion-type complementary bidding.

Table 6. Output of Complementary Bidding: Iteration 1 (Vendor A is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	Verbatim Mouse	21.95	23.71	23.71
Vendor B	Verbatim Mouse	14.95	26.79	
Vendor C	Verbatim Mouse	19.95	26.08	
Vendor D	Verbatim Mouse	14.95	26.56	

Table 7. Output of Complementary Bidding: Iteration 2 (Vendor B is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	Verbatim Mouse	21.95	22.7	19.74
Vendor B	Verbatim Mouse	14.95	19.74	
Vendor C	Verbatim Mouse	19.95	21.71	
Vendor D	Verbatim Mouse	14.95	22.11	

Table 8. Output of Complementary Bidding: Iteration 3 (Vendor C is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	Verbatim Mouse	21.95	25.69	22.34
Vendor B	Verbatim Mouse	14.95	25.24	
Vendor C	Verbatim Mouse	19.95	22.34	
Vendor D	Verbatim Mouse	14.95	25.02	

Table 9. Output of Complementary Bidding: Iteration 4 (Vendor D is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	Verbatim Mouse	21.95	19.08	16.59
Vendor B	Verbatim Mouse	14.95	18.75	
Vendor C	Verbatim Mouse	19.95	18.25	
Vendor D	Verbatim Mouse	14.95	16.59	

Bid Suppression Vendor Agent Initialization and Table Outputs

Sample output that demonstrates vendor agent initialization for collusion-type bid suppression.

```
Vendor-D started listening to client-request(s)...
Vendor-C started listening to client-request(s)...
Vendor-A started listening to client-request(s)...
Vendor-B started listening to client-request(s)...
Vendor-C received: Verbatim Mouse
DEMONSTRATING...BID SUPPRESSION
Vendor-A received: Verbatim Mouse
DEMONSTRATING...BID SUPPRESSION
Vendor-B received: Verbatim Mouse
DEMONSTRATING...BID SUPPRESSION
Vendor-D received: Verbatim Mouse
DEMONSTRATING...BID SUPPRESSION
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
VendorAgentB AGENT ID: 3
VendorAgentC AGENT ID: 3
VendorAgentA AGENT ID: 3

unavailable from VendorAgentB
unavailable from VendorAgentA
```

```
[MESSAGE]: database connection successfully established!
available from VendorAgentC
[MESSAGE]: database connection successfully established!
VendorAgentD AGENT ID: 3
unavailable from VendorAgentD
```

Vendor Agents Posting Prices and Availability for Bid Suppression

Sample output that demonstrates vendor agents posting price and availability for collusion-type bid suppression.

```
[MESSAGE]: database connection successfully established!
PRODUCT RECEIVED FROM USERAGENT: Verbatim Mouse
Product Unavailable from Vendor B
Product Unavailable from Vendor A
PRICE RECEIVED FROM Vendor C IS: 21.75
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
[MESSAGE]: database connection successfully established!
Product Unavailable from Vendor D
```

Following tables 10, 11, 12, 13 show four iterations of bidding that demonstrate collusion-type bid suppression.

Table 10. Output of Bid Suppression: Iteration 1 (Vendor A is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	129.6	129.6
Vendor B	IPOD Touch 16 GB	118	146.45	
Vendor C	IPOD Touch 16 GB	115	142.56	
Vendor D	IPOD Touch 16 GB	118	145.15	

Table 11. Output of Bid Suppression: Iteration 2 (Vendor B is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	158.12	137.5
Vendor B	IPOD Touch 16 GB	118	137.5	
Vendor C	IPOD Touch 16 GB	115	151.25	
Vendor D	IPOD Touch 16 GB	118	154	

Table 12. Output of Bid Suppression: Iteration 3 (Vendor C is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	148.12	128.8
Vendor B	IPOD Touch 16 GB	118	145.54	
Vendor C	IPOD Touch 16 GB	115	128.8	
Vendor D	IPOD Touch 16 GB	118	144.26	

Table 13. Output of Bid Suppression: Iteration 4 (Vendor D is Dominant.).

Vendor	Product	Actual Price	Posted Price	Least Price
Vendor A	IPOD Touch 16 GB	120	150.63	130.98
Vendor B	IPOD Touch 16 GB	118	150.63	
Vendor C	IPOD Touch 16 GB	115	144.08	
Vendor D	IPOD Touch 16 GB	118	130.98	

Comparison Study

A comparative study is aimed at evaluating the performance of the experimental design. The study does the evaluation of each bidding method one at a time against the no-rigging bidding scenario. The graphs are drawn with the final price paid by the buyers and the profit made by each vendor under each bidding strategy and the no-rigging bidding. The graphs are used to find pattern and analyze each rigging methods pros and cons with pictorial representation. Finally, the method of settlement that is required among the vendors under each bid rigging methodology is discussed in the following section later in this chapter.

Comparing Results with Aggregate Data Tables and Graphs

Tables with aggregate values have been drawn for control, bid rotation, bid suppression, and complementary bidding. Each table has data for the mean profit per vendor from multiple product-bidding runs.

Table 14. No Bid Rigging Aggregate Data Table.

No Rigging	Profits			
Iteration	1	2	3	4
Vendors	Vendor A	Vendor B	Vendor C	Vendor D
Verbatim Mouse	$ 0.00	$ 0.00	$ 0.00	$ 2.03
IPOD Touch 8 GB	$ 5.60	$ 0.00	$ 0.00	$ 0.00
MCAFEE Antivirus 2010	$ 0.00	$ 0.00	$ 8.82	$ 0.00
Panasonic HDTV	$ 18.80	$ 0.00	$ 0.00	$ 0.00
Sum	$ 24.40	$ 0.00	$ 8.82	$ 2.03

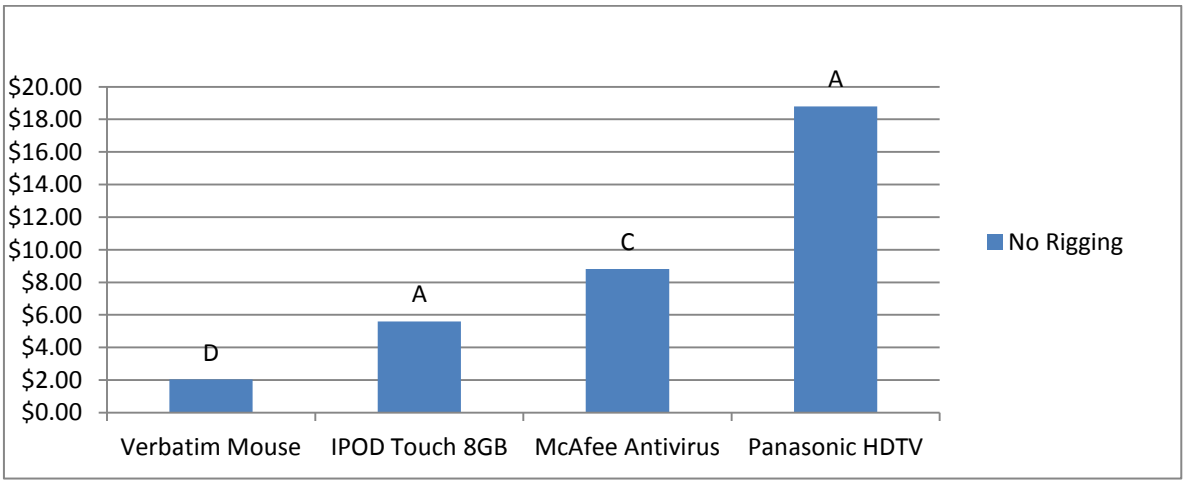

Figure 19. No Bid Rigging Graph with Multiple Products, Winning Vendors and Profits.

Graph Drawn from the Tables with Aggregate Values Data of No Rigging

Here in table 14, we see that, for each bid run for a product, the vendor who posted the smallest price won. Graphs as shown in Figure 15 show that the profit margins are highly varied and that no single vendor has the total advantage for selling his/her product. The profit margins are also cheaper, hence proving to be advantageous to the user who wants to buy products.

Comparing No Rigging and Bid Rotation Graphs

Graphs are drawn to compare the aggregate data tables between no-rigging and bid-rotation. Here in table 15, we could see that each vendor is making an equal number of

sales and that the profit margin is also similar. Each vendor gets a chance to sell their product and to make profit as shown in figure 20. Compared to the profit margins of each vendor with no rigging, bid rotation brings in more profit for all vendors evenly.

Table 15. Bid Rotation Aggregate Data Table.

Bid Rotation	Profits			
Iteration	1	2	3	4
Vendors	Vendor A	Vendor B	Vendor C	Vendor D
Verbatim Mouse	$ 1.52	$ 2.79	$ 1.80	$ 2.03
IPOD Touch 8 GB	$ 5.60	$ 4.01	$ 6.39	$ 8.76
MCAFEE Antivirus 2010	$ 7.92	$ 9.90	$ 8.82	$ 11.88
Panasonic HDTV	$ 37.60	$ 58.56	$ 42.30	$ 56.40
Sum	$ 52.64	$ 75.26	$ 59.31	$ 79.07

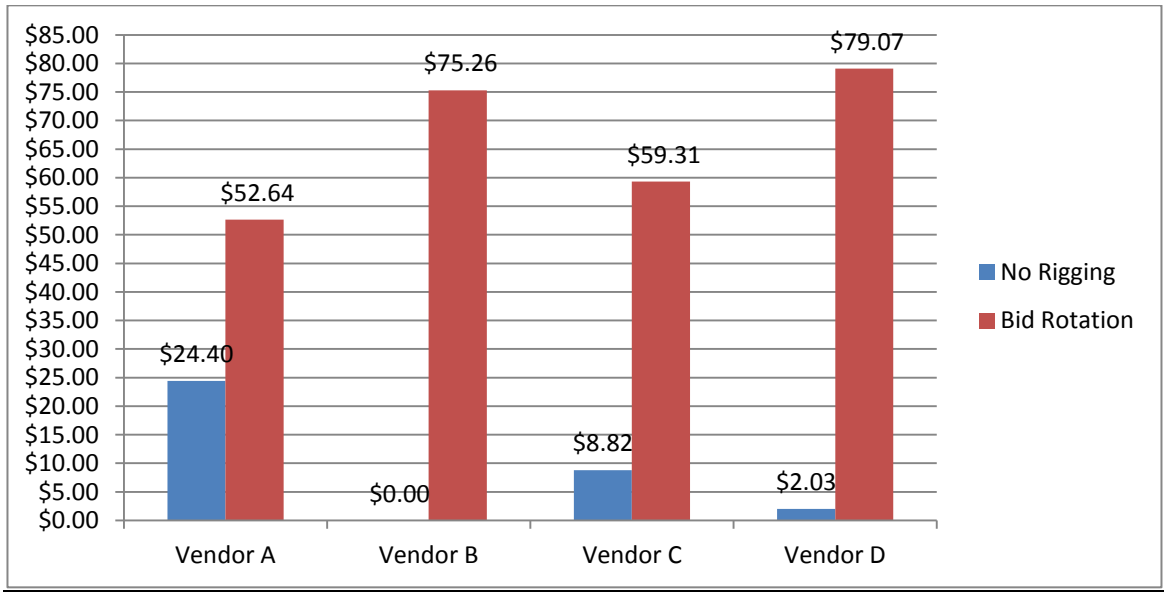

Figure 20. Bid Rotation vs. No Rigging Graph with Vendor and Profit.

Comparing No Rigging and Bid Suppression Graphs

Graphs are drawn to compare the aggregate data tables between no-rigging and bid-suppression. Compared to the profit margins of each vendor with no rigging (Table 16), bid suppression brings in more profit for all vendors by denying the availability of a product;

the vendors force the user to buy it from the dominant vendor at a much higher price. Thus boosting the profit margin greatly compared to the no-rigging profit margins which can be seen in graph (Figure 21).

Table 16. Bid Suppression Aggregate Data Table.

Bid Suppression	Profits			
Iteration	1	2	3	4
Vendors	Vendor A	Vendor B	Vendor C	Vendor D
Verbatim Mouse	$ 1.52	$ 0	$ 1.8	$ 4.06
IPOD Touch 8 GB	$ 11.2	$ 4.01	$ 6.39	$ 0
MCAFEE Antivirus 2010	$ 7.92	$ 9.9	$ 8.82	$ 11.88
Panasonic HDTV	$ 37.6	$ 0	$ 42.3	$ 112.8
Sum	$ 58.24	$ 13.91	$ 59.31	$ 128.74

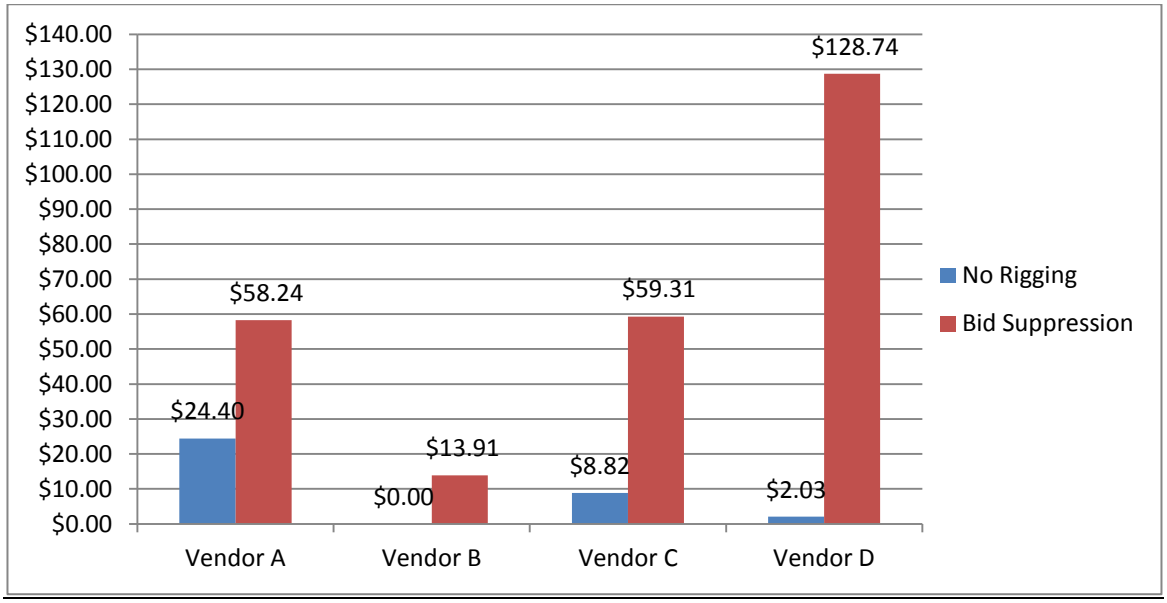

Figure 21. Bid Suppression vs. No Rigging Graph with Vendor and Profit.

Comparing No Rigging and Complementary Bidding Graphs

Graphs are drawn to compare the aggregate data tables between no-rigging and complementary-bidding. Compared to the profit margins of each vendor with no rigging,

complementary bidding, too, brings in a high profit for all vendors when their turn to sell comes (Table 17). By pricing the product very high, the vendors force the user to buy it from the dominant vendor, making his/her price look cheaper. In reality, the price is still higher than a no-rigging bid; hence the user is paying more money than the value of the product (Figure 22).

Table 17. Complementary Bidding Aggregate Data Table.

Complementary Bidding	Profits			
Iteration	1	2	3	4
Vendors	Vendor A	Vendor B	Vendor C	Vendor D
Verbatim Mouse	$ 1.76	$ 4.79	$ 2.39	$ 2.03
IPOD Touch 8 GB	$ 11.2	$ 4.01	$ 0	$ 8.76
MCAFEE Antivirus 2010	$ 15.84	$ 9.9	$ 0	$ 11.88
Panasonic HDTV	$ 37.6	$ 58.56	$ 42.3	$ 56.4
Sum	$ 66.40	$ 77.26	$ 44.69	$ 79.07

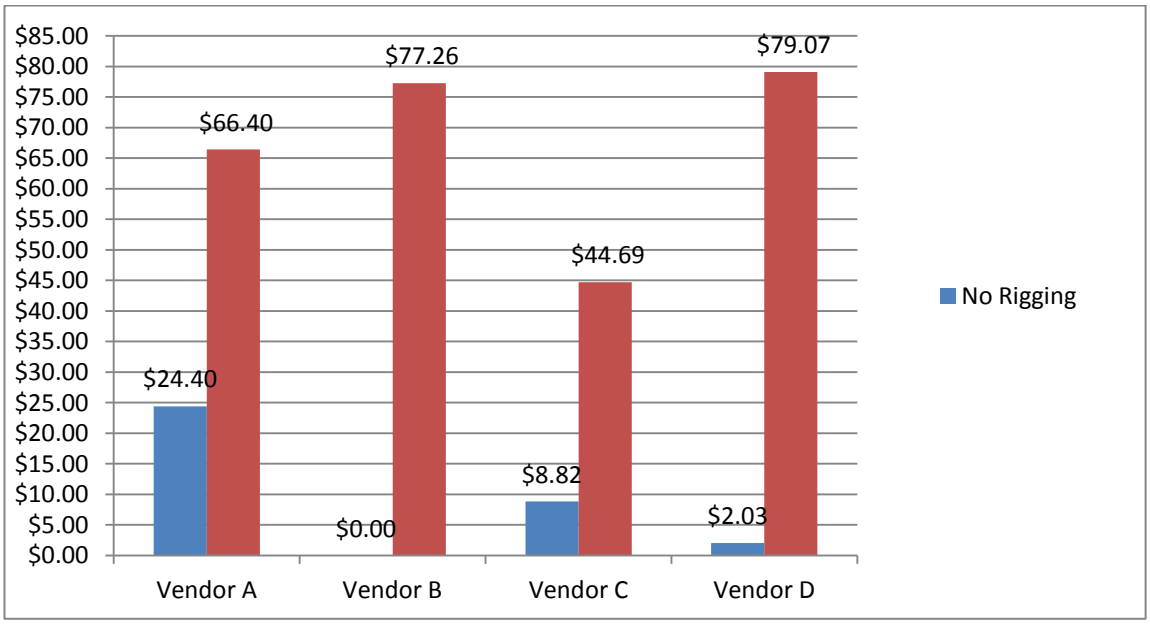

Figure 22. Complementary Bidding vs. No-Rigging Graph with Vendor and Profit.

Method of Settlement

Methods of settlements might vary based on the type of bidding or any method the parties involved used in the collusion. Some of common ways of settling are stated below for each bid-rigging method.

1. Bid rotation.
    - Rotating the winning bid evenly so that each vendor has an equal number of turns to win.
    - Rotating the bid in a way that each vendor gets an equal amount of profit or the chance to sell each product once to make the same margin of profit.
    - Dividing products in such a way that the cumulative profit would be equal at the end of all bidding runs.
2. Complementary bidding.
    - Methodical ways of increasing or decreasing the price so that one of vendors can profit when selling a product over a given period of time.
    - There can also be direct payoff in the form of goods, money, or other means of dividing profit in a legal tender.
3. Bid suppression.
    - One is allowed to win while the other becomes the loser; losing bid is done with a promise that the other will be allowed to win a different bid.
    - The vendors who withdraw from posting a competing price are promised a chance to win a later bid or a different product to make a profit.

- The wining vendor could settle by dividing his/her profit and sending it to the other vendor by different means of payment, such as products, licenses, or money.
- The winning vendor can also trade products to the losing vendors for a loss, letting them gain profit by an internal trade.

Thus, there are numerous ways the profit could be divided within the collusion group. It takes a lot of intelligence to crack the way such an inside trade can happen. Government agencies are trained extensively to monitor and analyze such patterns to stop bid riggings. But still they are in the process of learning and formulating schemes since online shopping is fairly a new concept with few years in existence compared to other means of commerce that we have used for ages.

CHAPTER 6. CONCLUSION AND FUTURE WORK

Conclusion

As the demand for online shopping for business, computing, and communication surges, the need to make it safe also increases. An independent agent-based system requires low resources for its operation and it is portable. It can run on multiple machines with different hardware and operating systems. It is a flexible system that performs multiple functions autonomously.

The program was developed to exhibit some different bid rigging in the online shopping-cart scenario where a buyer bids for a product to get best price quotes from a group of vendors. Non-rigged bidding would be where vendors compete; the vendor posting the lowest price would win the bid. However, with bid rigging, inter-vendor communication updates other vendors with price-eliminating completion, and based on the collusion they choose, vendors withdraw from competition, post a higher price, or take turns winning the bid. This paper is only concerned with an online-based rigging scenario while there are other non-electronic methods that could accomplish the same outcome.

Vendors can always settle their profit sharing in other ways. The losing company can negotiate the reward or payback in different ways. The intentional loser can win a different contact at a different site while the previous winner fails or withdraws intentionally. The winning vendor can decide to lose on certain products, thereby creating a divide between the product hierarchy for who sells and profits from what products. There can also be a direct payoff outside the online services with cash, goods, or other forms of payment which are not electronically traceable by output or sales profit analysis. No matter

what methods and means are used, bid rigging is illegal and leaves evidence that could bring wrongdoers punishment under federal law.

Collusion-based bid rigging only works when the settlements are followed properly. The profits need to be divided in some fashion that all parties in the scheme benefit. If some parties decide to cheat, then there is a chance that the vendors that failed to profit might alert authorities about the rigging. Not dividing profits evenly is one of most common ways rigging schemes are exposed.

On observation of patterns on the bid-rigging schemes and their graphs help identify some of the pros and cons for each bid-rigging scheme. With fewer runs, the bid rotation brought better profits and an even number of turns for all involved parties to sell and make a profit. The profit was instantly realized, so none of the parties involved in collusion had to settle profits latter. Bid rotation forms a clear and better scheme on short bid-rigging runs as everyone has a better guarantee to win without others having to be involved in settlement. The cons of the scheme were on multiple runs where the patterns clearly show that the vendors are involved in collusion. This method will make collusion obvious if used over a longer period of time.

Bid suppression was one of the toughest rigging schemes with which to spot a pattern to identify collusion. It has a very random pattern, even on a long period of bid runs. Because the inventory for each vendor is something only he/she knows, if he/she decides to withdraw from participating from a bid, it is not easily identified. A vendor can decide not to participate in bidding with a non-collusion scenario, too, and the results would not look any different from one who does it based on collusion. The disadvantage of this scheme is that all colluding parties should divide the profit in a proper way, or this

scheme can lead to failure or even tip authorities about the wining vendor's involvement in illegal schemes.

Future Work

This paper focused on implementing basic types of bid rigging in a shopping-agent program, and it did not get into cross analysis, the patterns exhibited by different types of bid rigging. As the need for better and safer online shopping increases, such studies could help develop tools and strategies that keep fair-market competition open, thus helping the buyer benefit from competition among vendors. When vendors rig prices before they post information for potential buyers, they have to follow a bidding method to establish this collusion and its resulting profit making. Any such method would leave evidence; such evidence can be analyzed and documented to bring about justice. JADE and the agent framework can be utilized for mobile technology, where independent, autonomous agents can provide services using a single thread for executing multiple functions, thus using the limited resources of mobile devices.

One of main requirements for the bid-rigging scheme to work in a collusion scenario is if all the parties decide to participate. Only when all parties decide to manipulate the price can the rigging really work. If a single vendor decides not to participate in collusion and they decide to set price honestly, the chances are they would have the lowest price in the entire bidding arena, hence winning all the bids without any illegal support from others.

One way of challenging the entire bidding scheme illustrated in this paper is to introduce a fifth vendor who does not participate in collusion. The fifth agent would post his/her real price and the item availability on each bid run. The expectation is that he/she

will end up posting a lower price than the others who are colluding because he/she is not increasing the price or manipulating data to help someone. Thus, this introduction of a fifth agent would have a big chance of toppling the entire bid-rigging scheme.

REFERENCES

[1] FIPA.Org. (2004, March 18). Foundation for Intelligent Physical Agents (FIPA). *FIPA Agent Management Specification*. Retrieved February 23, 2010, from http://www.fipa.org/specs/fipa00023/SC00023K.pdf

[2] Bellifemine, F. (2001, April 7). Java Agent Development Framework (JADE). *JADE TILAB*. Retrieved October 10, 2010, from http://jade.tilab.com/papers/JADEEtaps2001.pdf

[3] Caire, G. (2009, June 30). JADE Tutorial - JADE Programming for Beginners. *JADE TILAB*. Retrieved December 29, 2009, from http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

[4] Preventing and Detecting Bid Rigging, Price Fixing, and Market Allocation in Post-Disaster Rebuilding Projects. (n.d.). *Justice GOV*. Retrieved April 14, 2009, from http://www.justice.gov/atr/public/guidelines/disaster_primer.htm

[5] Taveter, K. (n.d.). Java Agent Development Framework (JADE). *University of Melbourne, Computer Science Department*. Retrieved March 17, 2010, from http://ww2.cs.mu.oz.au/682/Week6b.pdf

[6] Park, J., Youn, H., & Lee, E. (n.d.). A Mobile Agent Platform for Supporting Ad-hoc Network Environment. *International Journal of Grid and Distributed Computing*, 9-16.

[7] Spanoudakis, N., & Moraitis, P. (n.d.). An Ambient Intelligence Application Integrating Agent and Service-Oriented Technologies. *JADE TILAB*. Retrieved July 23, 2010, from http://jade.tilab.com/papers/AI07-Spanoudakis-Moraitis.pdf

[8] Caire, G. (2004). JADE : The New Kernel and Last Developments. *JADE TILAB*. Retrieved August 26, 2010, from http://jade.tilab.com/papers/Jade-the-services-architecture.pdf

[9] SEC Group. (n.d.). Online Bidding. *Construct IT Org*. Retrieved October 28, 2010, from http://www.construct-it.org.uk/pages/sources/CIC_online_bidding_briefing_note.pdf

[10] Parkes, D., Rabin, M., Shieber, S., & Thorpe, C. (2008). Practical Secrecy-Preserving, Verifiably Correct and Trustworthy Auctions. *Electronic Commerce Research and Applications, 7*(3), 294-312. doi: 10.1016/j.elerap.2008.04.006

[11] Porter, R., & Shoham, Y. (2005). On Cheating in Sealed-Bid Auctions [Abstract]. *Decision Support Systems, 39*(1), 41-54.

[12] Maheswaran, R., & Basar, T. (2003). Coalition Formation in Proportionally Fair Divisible Auctions. *AAMAS '03 Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 25-32. doi: 10.1145/860575.860580

[13] Wenyan, H., & Bolivar, A. (2008). Online Auctions Efficiency: A Survey of eBay Auctions. *WWW '08 Proceedings of the 17th International Conference on World Wide Web*, 925-934. doi: 10.1145/1367497.1367621

[14] Peng, K., Boyd, C., Dawson, E., & Viswanathan, K. (2003). Five Sealed-Bid Auction Models. *ACSW Frontiers '03 Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers*, *21*, 77-86.

[15] "Collusion." Wikipedia, the Free Encyclopedia. Web. 28 Nov. 2011. <http://en.wikipedia.org/wiki/Collusion>.