Abstract

SINGH, RATNA. Design and Implementation of fauxBay: Test-Bed for Bidding Agents in Online Auction Markets. (Under the direction of Dr. Peter Wurman).

Online auctions provide a very competitive market for a variety of goods and services and have gained a lot of popularity over the last few years. The increasing participation of buyers and sellers in these auctions has triggered the development of software bidding agents. These agents are automated agents that are designed to place bids on behalf of the user. In recent years, simulation of online auction markets has become a major research area in the field of electronic commerce. This is because testing different bidding strategies in a real auction environment involves risks and can result in heavy losses. Moreover, testing in a real life auction involves a substantial amount of time and participation, in order to derive a definite conclusion about the bidding agent's performance. We propose to solve this problem by designing and implementing a tool for simulating online auctions. Our simulation tool called *fauxBay* can be configured to develop test scenarios for testing bidding agent strategies. Currently, *fauxBay* supports the reserve price auction mechanism found on eBay. However, support for other auction mechanisms can be easily integrated with *fauxBay* in the future. We have designed *fauxBay* to offer two modes of simulating auction environments for testing bidding strategies: event-based and clock-based. Event-based simulation is used to obtain a quicker analysis of the bidding agent's performance, by moving forward to each event rather than moving linearly by the clock. In contrast, the clock-based simulation offers to test and gauge a bidding agent's performance in an auction environment where time moves linearly by the clock similar to real life online auctions. A key characteristic of *fauxBay* is that it tests bidding agent's strategies against real eBay data on completed auctions. Therefore, the results obtained by testing bidding strategy in *fauxBay* give a better insight into its anticipated performance in a real life online auction. Overall, *fauxBay* is designed to be extensible, configurable and give close to realistic results on the performance of bidding agents.

Design and Implementation of fauxBay: Test-Bed for Bidding Agents in Online Auction Markets

by

Ratna Singh

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2006

Approved By:

Dr. R. Michael Young	Dr. Laurie Williams

Dr. Peter R. Wurman
Chair of Advisory Committee

To my parents...

Biography

Ratna Singh was born in Lucknow, India, in 1981. She received her Bachelor's degree in Computer Science from Dr. B.R. Ambedkar University, Agra, India in 2003. Since 2003, she has been a Master of Science student in the Department of Computer Science at North Carolina State University. In Fall of 2004 she worked at IBM Corp., RTP, NC, as a software development intern.

Acknowledgements

I acknowledge the efforts of Dr. Peter R. Wurman, my thesis advisor, for providing me direction and guidance when required. He helped me remain focused and persistent. Working with him has been a pleasure and a great learning experience.

I am grateful to my committee members Dr. Laurie Williams and Dr. R. Michael Young for devoting time and providing valuable input.

Special thanks to my brothers, Amit and Atul, and my sister-in-law, Ashu, for being a constant source of motivation and encouragement. I also appreciate the help and immense support I received from Jaideep and Pratik , my best pals since my first semester at NC State. Thanks to Kunal, Richa and Batul for being there with me through thick and thin.

I would also like to thank my roommates and all my friends in Raleigh for making my life at NCSU wonderful and memorable.

Contents

List of Figures

Chapter 1

Introduction

Online auctions are among the numerous revolutions brought to us by innovation in e-commerce. They have acquired immense popularity during the last few years, facilitated by major online auction sites like eBay[1]. With the ease of being able to bid over the internet at any time of the day, online auction sites have attracted an increased participation from online buyers as well as sellers. This has also caused a simultaneous increase in the complexity and dynamic nature of online auctions. Therefore, there is a mounting need to study and understand the functioning of online auctions for designing a system that addresses these issues. The key determinants of auction dynamics are: interactions between an auction mechanism and bidding strategies, the ensuing allocation of goods to the bidders, and the allocation of money to the sellers.

A bidding agent is a software entity that bids on behalf of a human auction participant. It is designed to employ bidding strategies to win an auction in accordance with the preferences of the user. The interaction between an auction and an agent is defined by a set of rules that describe the auction mechanism. These rules define the validity of item price set by the seller and validity of bids placed by the bidders. These rules also define the winning bidders and the final price of the item. The allocation of goods to winning bidders and allocation of money to sellers reflects the result of bids that were placed in a completed auction. A completed auction is one which has ended. However, a completed auction does not necessarily mean that it resulted in a sale.

[1] eBay is a registered trademark of eBay Inc.

The automation of bidding agents will be a close reflection of human bidding behavior if these agents intelligently employ the strategies built in them by the agent developers. Autonomous bidding activity is of great importance in a scenario where participation is required in multiple heterogeneous auctions. This thesis is motivated by the need for a tool that simulates online auctions in order to provide a research tool for studying and testing strategies of bidding agents.

Bidding agents can be tested in real life online auction sites. However, it is impractical because winning in real life auctions while testing the bidding strategy can result in heavy losses. Moreover, testing a certain strategy which will place winning bids for the maximum number of times in an online auction environment still remains a challenge. Time is a big factor that affects simulation of the auctions and evaluation of the bidding strategies. Auctions in real life can last for a duration of hours, days or even weeks. The result of the strategies employed are visible only after the auction has ended. Consequently, analysis of the performance of bidding strategy in an online auction system can only be done after a considerable amount of time and participation, which is impractical. In order to overcome this problem the simulation tool developed in this thesis will simulate auctions that were completed on eBay with their times scaled down. This will facilitate faster testing of bidding strategies while still maintaining the essence of real eBay auctions.

1.1 Motivation

Simulation of multiple heterogeneous auctions is a rapidly emerging research area because of two key motivations. The first is the requirement to offer a development testbed for the study of bidding agents involved in online auction markets. The second is the strong need to understand the patterns that emerge by employing different bidding strategies followed by an analysis of consequent results.

This thesis is motivated by the above mentioned requirements. The simulated auction environment in this thesis, backed by real eBay auction data on completed auctions, will aid in the following research purposes:

- Studying the behavior of bidding agents across multiple homogeneous or heterogeneous auctions. The quantitative attributes of this behavior will be defined by the end user of this tool. For example, the attributes that define good outcome versus that

defining bad outcome of a test simulation. The results obtained will also depend upon the number of auctions taken into account. Analysis can consider the result based on the outcome as a percentage over a range of auctions as well. Overall, this tool serves as a building block that is extensible and configurable depending on the requirements of the agent developer. These extensions will involve writing and integrating code.

- Studying behavior of bidding agents with similar or varying strategies of searching for items and bidding on items. End-users of this tool can devise test plans by designing a bidding agent which uses different combinations of strategies keeping the rest of the variables same e.g. auction item, its perceived value by the bidders, time constraints etc. This will allow them to analyze the auction results over a number of tests and to derive some qualitative characteristics of the agent under test. This study will also include analyzing the results of employing a single or a combination of bidding behaviors.

- Studying negotiations between bidding agents over multiple issues and constraints like the remaining time for auction to close, individual price valuation, remaining number of desired auctions or desired time in which the item is required.

- Studying the behavior of bidding agents in the presence of other bidding agents.

- Studying various types of bidding behavior on eBay and their implications. This will provide further insight into all bidding behaviors on eBay such as sniping, and shilling. Sniping and shilling behaviors are explained in Chapter 3.

1.2 Contributions

We make the following contributions in this thesis:

- Describe the design and implementation of an online auction market simulator, *fauxBay*, which provides a configurable test-bed for testing strategies of bidding agents.

- Implement an auction manager and its components that orchestrate real eBay auctions with test agent participation in a simulated auction environment.

- Provide Web Service API with fauxBay for interacting with test agents. The Web Service API is well documented making it simple for developers to use. The Web Service API also gives the agent developers freedom to develop their agent in any programming language.

- Provide two modes for simulation of an online auction market through fauxBay:

  a) Event-Based – In this simulation mode, all timed attributes of an auction are handled as specific events in time. Timed attributes of an auction are – auction start, auction end, and bids of the participants. Bids by the agent being tested are also maintained as timed events. A test scenario is configured by the tester by choosing from a set of desired auctions against which the bidding agent is to be tested. The time in this simulation is accelerated by moving to the next event in time rather than accelerating it by the clock. This simulation mode is useful when a faster test of bidding strategy is required.

  b) Clock-Based – In this simulation mode, the timestamps of all timed attributes of an auction are preserved. A test scenario is configured by the tester as discussed in the event-based simulation. The test scenario is maintained as a set of auction objects. Each auction object represents an auction with all its attributes. The time in this simulation moves linearly with an accelerated clock. This simulation mode is useful when test of a bidding strategy involves the evaluation of real-life timing issues.

- Provide a platform for agent developers to test the bidding strategies of their agents against real eBay auction data. This will give a better insight into the real life outcome of the implemented strategy.

1.3 Thesis Organization

In Chapter 2, we review the related work in online auction market simulation . In Chapter 3, we provide the background information on the types of auctions alongwith a case study of various eBay auction mechanisms and its bidding format. Chapter 4, gives an overview of the *fauxBay* system, its different components and the interactions between

them. In Chapter 5, we explain the implementation details of the system followed by our conclusion and discussion of future work in Chapter 6.

Chapter 2

Related Work

Online auction markets are a hub for trading for millions of sellers and buyers. As a result, online auction markets have attracted intense research in the area of electronic commerce. Software agents are used as a means to automate trading with the predefined preferences and trading strategies of their users. Despite their utility in facilitating commercial transactions in online auction markets, the enormous potential of the trading agents has not been fully understood [18]. This is due to the fact that testing trading agents and their trading strategies in real online auction markets involves huge risks. Testing in real life online auctions can turn out to be a very expensive affair if the agent wins in the auctions. It is also impractical because of the time and participation required to come to a definite conclusion about a trading agent's performance. A simulated auction market environment offers a practical solution to these problems. Efforts have been made to build applications that incorporate the design of auction mechanisms, bidding strategies and auction servers. In this chapter, we discuss some of the efforts made in this direction.

2.1 The Michigan Internet AuctionBot

The AuctionBot [13] was a research-oriented centralized auction server system that provides simulation of simultaneous auctions. One of the hundreds of auction mechanisms supported by the AuctionBot is the second-price sealed bid auction mechanism, also called

the Vickery auctions. In this type of auction, all bidders specify their highest bid to the server. The bidder who places the highest bid wins the auction but pays price equal to the second highest bid. This bidding process, similar to the proxy bidding at eBay [6], in AuctionBot reduces a lot of effort in continuous monitoring of the price in a single item auction. *fauxBay* differs from the AuctionBot in that it runs real eBay auction data to simulate an online auction environment. The AuctionBot provides two separate interfaces for human and software agents web interface for human agents and TCP/IP interface for software agents.

2.2 eMediator

eMediator [17] is an eCommerce server developed at the Washington University in St. Louis. It was an effort to make the process of online bidding and eCommerce more intelligent and efficient. eAuction house, one of the major components of the system supports several interesting features including combinatorial auctions and bidding through software agents. eAuction house is one of the first prototypes to support combinatorial auctions. In a combinatorial auction a bidder can place bids on a combination of items instead of just a single item as is the case in sequential auctions. It also supports bidding through price-quantity graphs.

2.3 JASA

Java Auction Simulator API (JASA) [10] offers auction simulation using different auction mechanisms. JASA supports the double-auction mechanism, in which there are multiple sellers and buyers for a single item. In a double-auction, buyers and sellers both submit bids, the buying bid and the selling offer.

A demand and supply profile is created by ranking the bids from highest to lowest. These profiles are used to determine the maximum quantity exchanged between the seller and the buyer by matching the buyer bids and the selling offers. A double-auction can be periodic or continuous. In a periodic double auction the bids are matched at a predetermined interval of time known as the bidding interval. Whereas in a continuous double auction bids are matched as soon as a compatible buyer bid and seller offer is found.

In JASA, simulation of auctions is targeted at research in the area of agent based computational economics and microeconomics of market places where both supply-side and demand-side markets coexist, for example stock markets. JASA provides test simulations for use of bidding strategies in standalone auctions.

2.4 TAC

The Trading Agent Competition (TAC) [12] was started as a test-bed for software agents competing in auctions to acquire services for customers. TAC is an annual international competition initiated by the University of Michigan's Artificial Intelligence Laboratory and is now operated by the Swedish Institute of Computer Science (SICS).

TAC was designed to promote research in online auction and software trading agents and has proven to be successful. First TAC was a simulation of an online marketplace scenario where trading agents had to bid on hotel rooms. The agents also had to package the rooms with flight tickets and entertainment tickets in order to maximize a set of pre-defined utility functions.

TAC uses a centralized market server used by the software agents for various services, communication, coordination and commitment. The software agents communicate with the market server through network sockets using pre-defined XML based messages. The simplicity and global accessibility of TAC made it a very good platform for testing and analyzing various bidding strategies.

2.5 TAGA

In [19] Zou et al. describe an open source game for testing and analyzing agent based applications. Travel Agent Game in Agentcities (TAGA) is an enhancement to Trading Agent Competition (TAC) [12]. TAGA, is based on the Agentcities [9] infrastructure utilizing the Foundation for Intelligent Physical Agents (FIPA) [7]. FIPA produces standards for interaction of heterogeneous agents and agent based systems. Agentcities defines a distributed network of heterogeneous agents and services. This creates a dynamic environment where there are compound services competing for fulfilling changing business tasks.

TAGA is one such test-bed for market simulations and auction agents. TAGA simulates a real life travel market where a customer wants to buy a travel package with specific requirements and there are several travel agents that want to sell the package. In this game, various travel agents compete to fulfill more and more customer requests to maximize their profit. As TAGA is based on the open standards, FIPA, it makes the system a good platform for heterogeneous software agents to compete and analyze their strategies.

This chapter discussed some implementations of simulated auction markets that alleviate the problem of testing trading agents in real life online auction markets. All these simulated auction environments are driven by a key motivation - a need to have simulation of market-based scenarios where strategies implemented in trading agents can be tested for performance and tested against strategies implemented in other trading agents. As discussed in the beginning of the chapter, testing trading agents in real online auction markets is impractical and involves high risks.

Chapter 3

Background

3.1 Online Auction Markets

3.1.1 Auctions

An auction is a process wherein items are bought and sold by offering goods for sale by accepting bids from buyers and then selling the goods to the highest bidder. This is one of the oldest business models. A bid is the price a buyer is offering to pay for the item. The online auction business model is one which involves auctions where buyers bid for items offered by sellers over the internet. Online auctions have found their way into homes around the world because of their defiance of geographical boundaries and 24x7 availability over the internet. Online auctions have become a very successful means of electronic commerce. They are especially useful for selling items that do not have pre-determined market value yet. They enable dynamic pricing by determining a price for such items based on their demand and supply in the market.

eBay Auctions

Since its inception in 1995, eBay [6] has become a congregation of millions of sellers and buyers. Moreover, eBay being one of the leading online auction sites provides a good subject for studying the intricacies of working of online auction markets. eBay is also a large

source of very valuable auction data that can serve as a testing ground for many bidding strategies. eBay has thousands of unique categories and thousands of items listed for sale within a single category. Three auction mechanisms common in eBay online auctions are the Reserve-Price auction, the Dutch auction and the Fixed-Price auction. This thesis implements reserve price auctions.

3.1.2 Reserve Price

A Reserve-Price Auction [6] is characterized by the seller setting a minimum price for which she is willing to sell her item. This minimum price is called the "reserve price". The presence of a reserve price is known to the bidders, but its value is not disclosed to them. The highest bidder must meet or beat the reserve price set by the seller in order to win an auction. At the end of a reserve price auction, if there are no bidding offers that meet the reserve price, neither the seller nor the highest bidder is under any compulsion to complete the transaction. Also, if there is more than one bidder who bids the highest, the earliest bidder wins the auction.

3.1.3 Dutch

In a Dutch Auction [11], a seller offers multiple identical items for bidding. In this type of auction, the seller sets a starting price per unit item and the number of identical items for sale. A bidder specifies a bid price per unit item as well as the number of items she wishes to buy. Unlike the reserve price auction, all winning bidders have to pay the lowest successful bid. If the number of bidders exceeds the number of items for sale, earlier bidders get the items.

3.1.4 Fixed Price

In a Fixed-Price Auction [6, 14], a seller offers their item at a fixed price. In a fixed price auction, the first bidder who bids at the fixed price set by the seller wins the auction and the auction closes immediately. At eBay this auction mechanism is often implemented and exercised by agents as the "Buy It Now" option offered in auctions.

3.2 Bidding Behaviors

3.2.1 Sniping

Sniping [15] is the bidding behavior where bidders place high bids in the closing minutes or seconds of an auction so that competing bidders do not get enough time to outbid them. However, eBay does provide a way to avoid getting outbid by sniping bidders during the last few minutes of an auction by employing the system of proxy bidding. Proxy bidding allows bidders to enter their maximum bid which is kept confidential from other bidders and the seller for the item. Proxy bidding system makes an automatic bid on behalf of the bidders to maintain their highest bidder position using only a part of their maximum bid. Though snipe bidding helps in winning the auction, it is not necessary that a sniping bidder will always win the auction. To win the auction, a sniping bidder must be willing to pay more than the maximum bid entered for the item through the proxy bidding system. In case a bidder submits a high enough maximum bid to the proxy bidding system, the sniping bidder might not be able to meet the maximum bid or may not be able to get feedback in time to snipe before the auction ends. At eBay, sniping is not illegal. Any bid that is submitted before the auction ends is legal.

3.2.2 Shilling

Shilling [16] is the bidding behavior where sellers' associates or sellers obtain a different identity to bid on the items the sellers themselves put up for sale. They do this in order to raise the item's price to the seller's desired ask price. This behavior is not easily identified and bidders are often conned into believing that the item is in good demand. This leads the bidders to bid more for the item. Shilling helps sellers sell their items on much higher prices than their actual market value by artificially increasing their demand in the market. At eBay, shilling is illegal.

3.3 Bidding Format used at eBay

3.3.1 Proxy Bidding

eBay offers an automatic bidding format called proxy bidding. Proxy bidding [6, 8] is a dynamic variant of the Second Price Sealed Bid auction. In proxy bidding, bidders specify their maximum valuation of an item. A bidder's maximum valuation for an item is the maximum amount or maximum bid that a bidder is willing to pay for that item. These maximum bids or maximum valuations are called "actual bids" of the bidder as opposed to the proxy bids placed by eBay on behalf of the bidders. The proxy bidding format allows eBay to automatically increase a bidder's bid up to their maximum bid to prevent them from getting outbid. If the proxy bid for a bidder is not outbid until the auction closes, the bidder wins the auction and pays the amount equal to proxy bid and not her highest bid. Proxy bidding is particularly helpful to bidders in case of sniping. Proxy bidding allows bidders to enter a higher maximum bid in order to avoid the chances of getting outbid due to sniping in the closing seconds of an auction. Proxy bidding works as follows:

1. When a bidder B, places her bid, she enters her maximum bid for the item. The maximum bid is the maximum price B is willing to pay for the item. This maximum bid is not disclosed to any other bidder or the seller.
2. B's maximum bid is compared to those of the other bidders for the item.
3. A proxy bid is placed on behalf of B, utilizing only the amount of the maximum bid that is necessary to maintain the highest bidder position for B or to meet the reserve price set by the seller.
4. Proxy bids are placed only up to B's maximum bid.
5. If another bidder places a higher bid than B's maximum bid, B is outbid.
6. If no other bidder places a higher bid than B's maximum bid, B wins the item. In this case B might end up paying less than her maximum bid.

Proxy bidding facilitates bidding in the sense that it does not require the bidder to keep revising the bid every time there is another bid for the item. This saves a lot of effort for the bidder.

Chapter 4

System Architecture

4.1 Overview

fauxBay is a Java application that provides a simulated auction environment as a test-bed for bidding agents through a web service interface. It acts like a central managing component for the orchestration of multiple auctions in a scenario. We use the term scenario to represent an instance of online auction market simulation having multiple auctions. Figure 4.1 gives a top level view of the system architecture, displaying the interplay between the main entities. The different entities in the system are described in the following section.

4.1.1 Tester

We will use the term tester throughout this thesis for the agent developer who is using *fauxBay* to test the bidding strategy of her agent. Tester sets up the test scenario by selecting a set of auctions based on the tester's preferences. These auctions are selected from a pool of real eBay auctions. The selected auctions form the test scenario for the testing the bidding agent. The bidding agent bids on these selected auctions based on its user's preferences. Finally, after the simulation, the tester has access to bid history generated from running the test scenario as against that of the historical eBay bid history data. The bid history obtained after the test simulation allows the tester to evaluate the

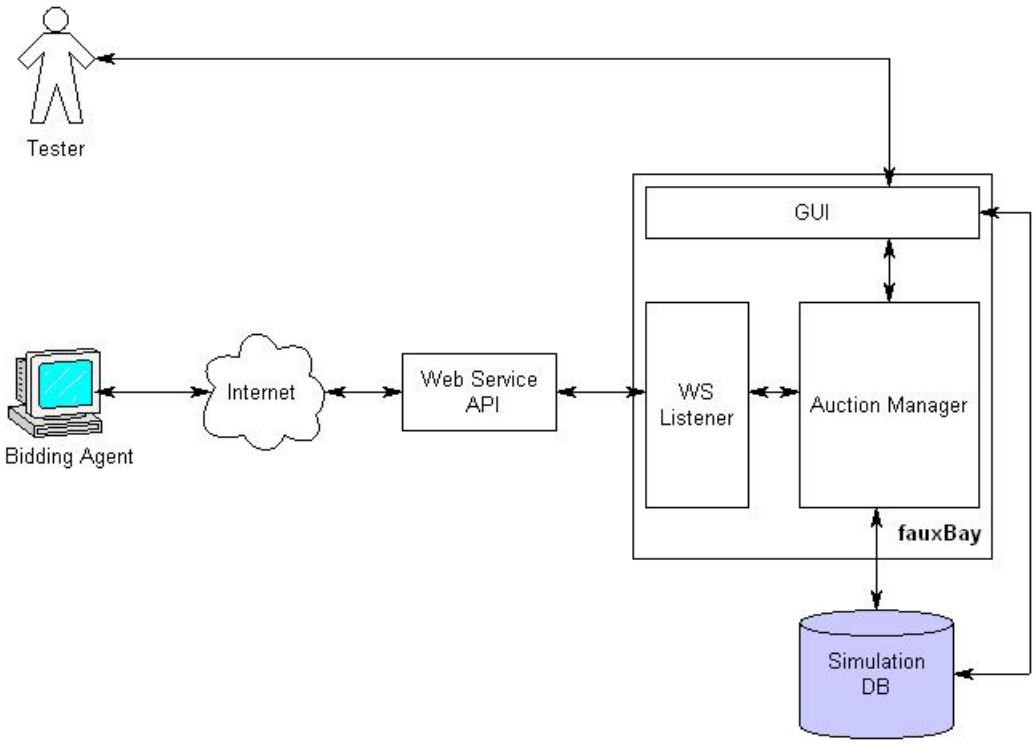

Figure 4.1: fauxBay:System Architecture Overview

bidding agent's performance. The figure 4.2 describes the steps required by the tester to setup a test scenario using the GUI component.

4.1.2 Bidding Agent

The bidding agent is the agent to be tested for its bidding strategy through the test-bed provided by *fauxBay*. Strategies of the bidding agent are tested against the selected set of real eBay auctions with their bid history data simulated as other bidders in these auctions. The bidding agent sends requests to the Web Service (WS) for participating in an auction and receives appropriate response detailing the result of its request.

The response obtained by the bidding agent for its request to the WS API calls is always in the form of string. The string response is in the form of concatenated sequence of variables and their associated values. All variables in the response string are separated by "|". The variables and their values are separated by "=". The variables and their possible

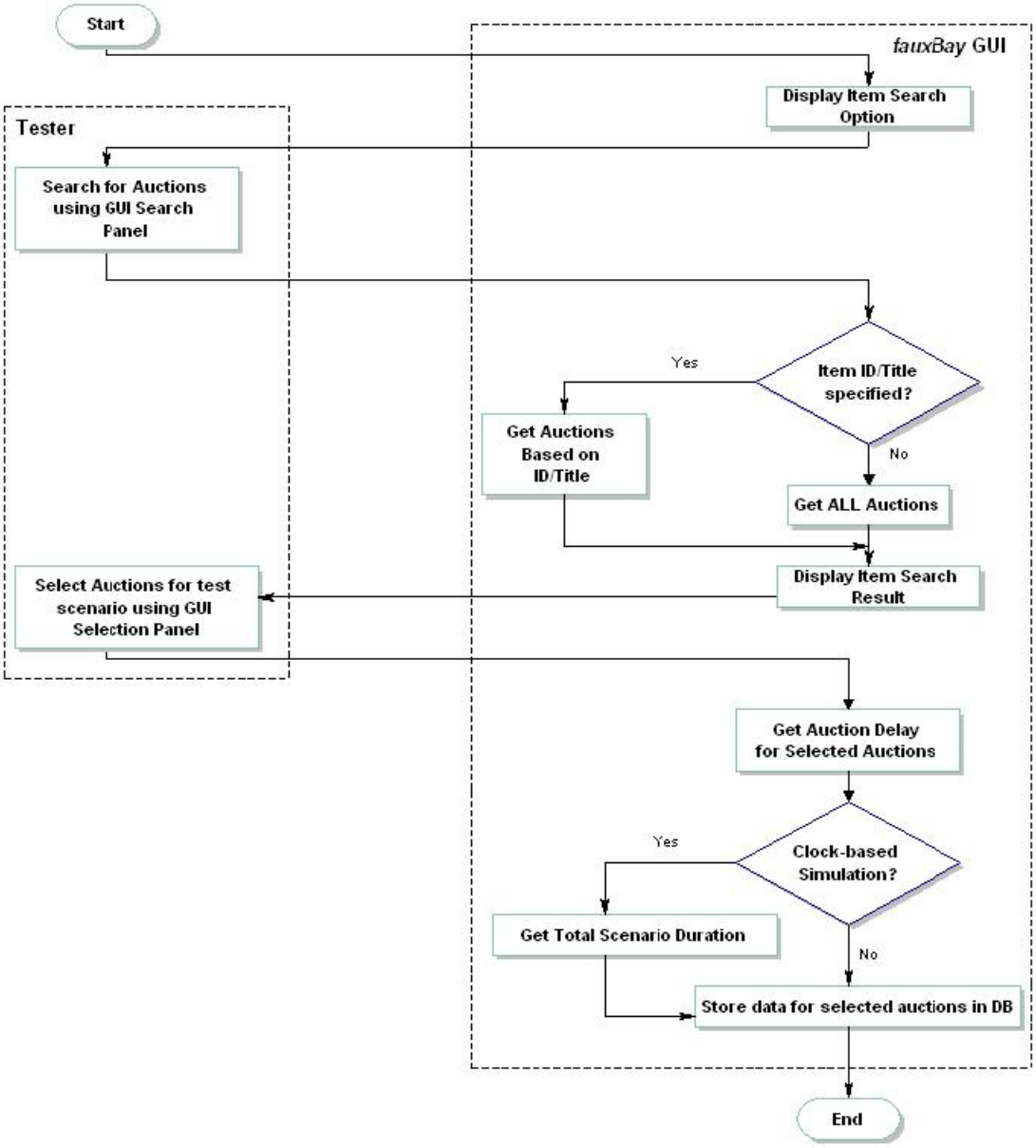

Figure 4.2: Scenario setup by Tester using GUI

values that occur in a string response from the web service methods are:

1. SIMULATION_STATE

   - 0 = simulation has not started yet.

- 1 = simulation has started.
- 2 = simulation has ended.

2. CURRENT_TIME
   - time elapsed from the scenario start time.
3. EVENT
   - AUCTION_START = auction start event.
   - AUCTION_END = auction end event.
   - BID = historical bid event.
   - AGENT_BID = bidding agent bid event.
4. ITEM_ID
   - item's unique identification number.
5. ITEM_DESCRIPTION
   - item's description.
6. AUCTION_START_TIME
   - start time of an auction in terms of delay from the scenario start time.
7. AUCTION_END_TIME
   - ending time of an auction in terms of delay from the scenario start time.
8. START_PRICE
   - start price for the item.
9. CURRENT_BID
   - current bid for an item.
10. BID_STATUS
    - 0 = bidding agent's bid too low to submit.

- 1 = bidding agent's bid is outbid.
- 2 = bidding agent's bid is not outbid yet.

11. FINAL_PRICE
    - final price for the item.
12. WON_AUCTION
    - true = bidding agent wins the auction.
    - false = bidding agent loses the auction.

4.1.3 fauxBay

The architecturally significant components of *fauxBay* are:

1. Graphical User Interface
2. Web Service Listener
3. Auction Manager

1. **Graphical User Interface**

   The GUI is the *fauxBay* interface available to the tester. The tester can search and select auctions through GUI to set up a test scenario. The *fauxBay* GUI consists of:

   (a) Search Panel - This panel provides the tester the option of search on auction based on keywords or item ID (unique item identification number) from the simulation database. The search is performed on the item ID and the item description fields in the *fauxBay* simulation database.

   (b) Scenario Set Up Panel - This panel provides the tester the option of selecting auctions from the set of auctions in the search result. This selection forms the test scenario against which the bidding agent's strategy is tested.

   (c) Simulation In Process Panel - This panel shows that the simulation is in process.

   The GUI offers functionality to the tester to perform a keyword search for items in the *fauxBay* simulation database using the Search panel. The GUI allows tester to select

auctions from the Scenario Set Up panel. Key attributes of all selected auctions, unique item identification numbers and starting time, are stored in the simulation database by the GUI component. The scenario start time is the start time of the earliest auction selected by the tester. In the case of clock-based simulation, the tester uses the GUI to set up the test scenario as well as start the test scenario while in event-based simulation GUI is only used by the tester to set up the test scenario.

2. **Web Service Listener** The Web Service (WS) Listener component of the *fauxBay* provides an interface between the Auction Manager and the web methods exposed as Web Services. Whenever any of the web methods are called by the bidding agent, the WS Listener invokes the appropriate method in the Auction Manager. The result of the method invocation on Auction Manager is returned to the Web Service through the WS Listener.

3. **Auction Manager** This module manages the orchestration of all auctions in the scenario. It provides the actual services that are exposed via Web Service APIs.

   The Auction Manager loads up the scenario data from the database when the simulation is started. This scenario data consists of the data on auctions that were selected by the tester. The tester's selection of auctions form the test scenario against which the bidding agent has to be tested. The Auction Manager is either started by the tester through the GUI (in Clock-Based Simulation) or the bidding agent through WS API (in Event-Based Simulation). The Auction Manager then starts the WS Listener which listens to the requests coming from the web service to invoke appropriate methods in Auction Manager.

   The Auction Manager simulates the auctions that were selected to run in the test scenario. It also simulates the presence of other bidders in the auctions with the help of historical bid data of real eBay auctions. We use proxy bidding system in *fauxBay* and it is implemented as a part of the Auction Manager. Proxy bidding is applied to all historical bids as well as the bids from the bidding agent. The proxy bidding system in Auction Manager uses the start price of the auction item to determine the first current bid of the auction when an auction starts.

4.1.4 Web Service API

The *fauxBay*'s interface that is exposed to bidding agents is chosen to be Web Service APIs thereby making it independent of the language in which agents are developed. In the case of event-based simulation, the bidding agent uses the Web Service (WS) API to start the test scenario.

The WS API is different for the two different types of simulation offered by *fauxBay*. In event-based simulation, the tester configures the agent to start a particular scenario by giving it the scenario identification name (scenarioID). The agent then makes the web method call of the appropriate web service for event-based simulation. The agent also identifies itself with the simulation environment of *fauxBay* by giving out its own agent identification name (agentID). In the event-based simulation, there are other web service methods besides starting scenario that are provided for the test agent to a) place bids in an auction, and b) end the test scenario when all auctions are completed. In clock-based simulation, the scenario is not started by the agent through the WS API. It is started by the tester through the GUI component. In the Clock based simulation, there are web service methods that are provided for the test agent to a) perform a keyword search for auctions, b) place bids on auctions, c) get current bid in a running auction, and d) check if the reserve price is met in a running auction.

4.1.5 Simulation Database

The Simulation Database holds the historical eBay auction data. It also holds the results for the scenario test run.

Data Collected From eBay

1. **Variables collected for simulating auctions from eBay auction data**

   Each auction has an item for sale. Every item has a unique number associated with it called the item ID. Hence, every auction is also categorized by a single item ID.

   - ITEMID - unique item identification number.
   - ITEMTITLE - item description.
   - AUCTSTARTDATE - auction start date.

- AUCTENDDATE - auction start date.
- AUCTTYPECODE - auction type code.(Reserve-Price/Dutch/Fixed-Price)
- STARTPRICE - start price of the item.
- RESERVEPRICE - reserve price of the item.
- FINALPRICE - final price of the item.
- QTYSOLD - quantity of the item sold.
- QTYAVAIL - quantity of the item available for sale.
- BIDCOUNT - number of bids for the item.
- SUCCESSFLAG - true if item was sold, false otherwise.(Applicable only for reserve-price auctions)
- RESPRICEFLAG - true if there is a reserve price, false otherwise.
- CATDESC1 - category of the item in eBay.
- CATDESC2 - subcategory of the item in eBay.

2. **Variables collected for simulating bidders from eBay bid history data**

   Each item has a bid history associated with it. It represents the bids that were placed for the item in an auction.

   - ITEMID - unique item identification number.
   - BIDDATE - actual timestamp when the bid was placed.
   - BDRID - unique bidder's identification number/name.
   - BIDTYPECODE - bid type code. (Reserve-Price/Dutch/Fixed-Price)
   - QTYBID - item quantity for which the bid is placed.
   - BIDAMTPERUNIT - bid value per unit.
   - BIDAMTTOTAL - total amount of the bid. (bid value per unit * quantity bid)
   - BIDSUCCESSFLAG - true if the bid wins, false otherwise.
   - FINALUNITPRICE - final unit price of the item.
   - WINNINGQTY - item quantity won.

Parser Used To Collect Data From eBay

The real eBay auction data is collected using a parser written as a collection of Python scripts for retrieving data from eBay. The auction data is collected from completed auctions on eBay for both the Reserve-price and Dutch type auctions. However, only the data collected for reserve-price auction is used for simulation as *fauxBay* currently supports the reserve-price auction mechanism only.

4.2 Modes of Simulation

fauxBay offers two possible types of simulation for the chosen set of auctions – event-based and clock-based.

4.2.1 Event-Based Simulation

Every auction has attributes like the item identification number, item description, auction start time, auction end time, start price, auction type (reserve price/ dutch/ fixed price), reserve price (if applicable), and bids placed in that auction.

Figure 4.3 gives an overview of the Event-Based Simulation. For event-based simulation purposes, we process timed attributes – auction start, auction end and bids in the auction, as distinct events in time. Static attributes – item identification number, item description, auction type and start price – are used by the bidding agent to decide whether or not to bid in that auction. All events are maintained in a sequential queue of events in time.

In the event-based simulation, *fauxBay* interacts with the test agent whenever an event occurs. The test agent then takes an action using the WS API of *fauxBay*, thereby driving the scenario forward. Instead of moving through time in a linear fashion, *fauxBay* jumps from event to event on the time scale, keeping the test agent informed of the event time as well. When the last auction in the scenario ends, the scenario bid data is saved in the database and the test agent is informed.

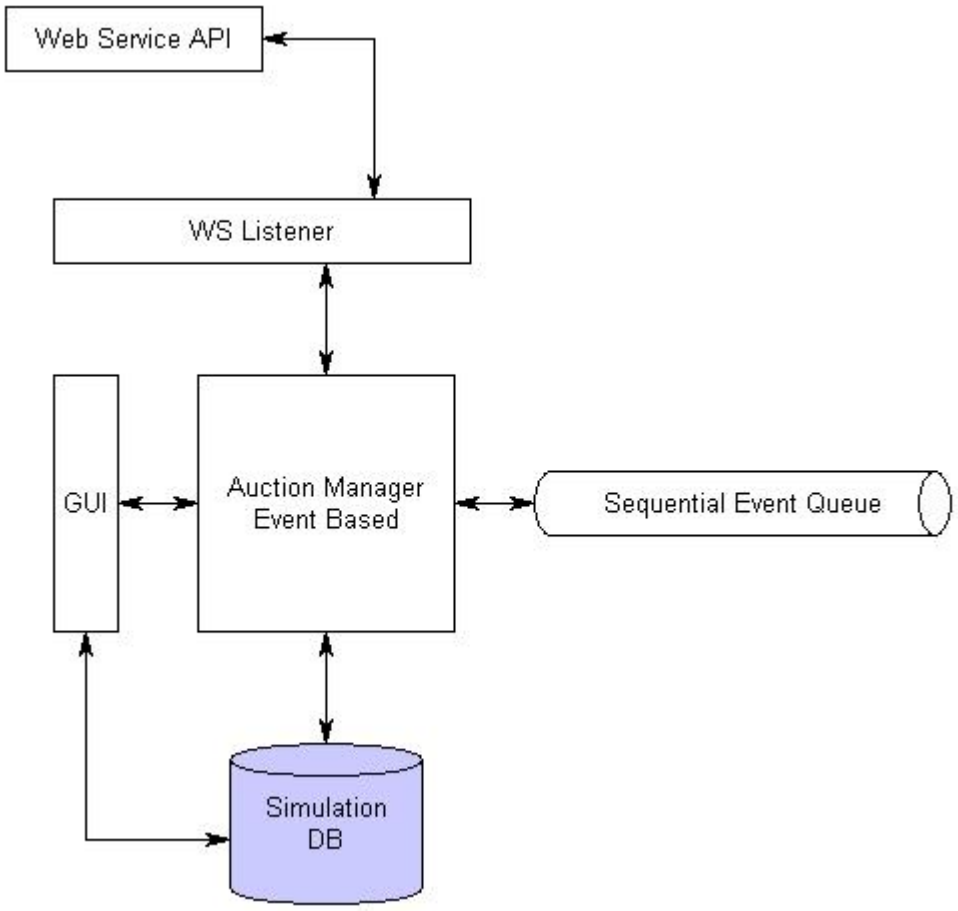

Figure 4.3: Event-Based Simulation Overview

4.2.2 Clock-Based Simulation

For the clock-based simulation purposes, each auction is represented by an auction object. Each auction object encapsulates both the timed and static attributes of an auction. Figure 4.4 illustrates the working of the clock-based simulation.

In this simulation, there are added web service methods that are provided for the test agent to:

- perform a keyword search for desired auction items,
- get current bids of currently running auctions,

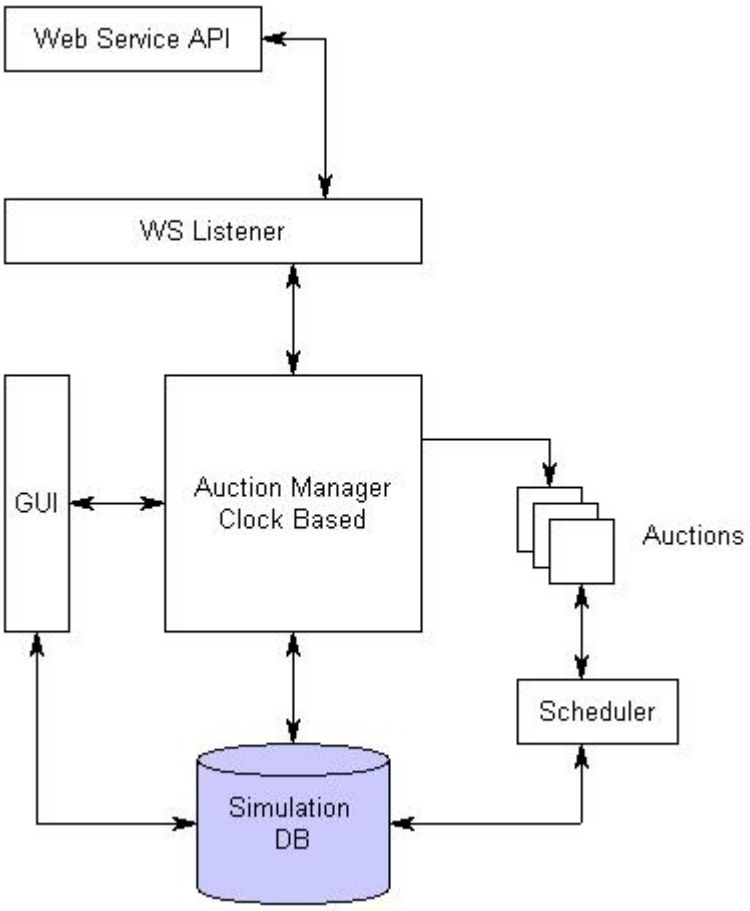

Figure 4.4: Clock-Based Simulation Overview

- place a bid in an auction and
- check if the reserve price is met in a running auction.

In this simulation, *fauxBay* moves forward in a linear fashion with time scaled down by a configurable factor. The relative timing of events is preserved even though the time is scaled down. When the scenario duration fixed by the tester expires, the scenario bid data is saved in the simulation database.

Chapter 5

System Implementation Details

5.1 Basic Components

In this chapter, we describe the working of each of the modules in both the simulation modes.

5.1.1 Event-Based Simulation

Event-based simulation provides a faster option of testing an agent strategy by accelerating time to every next event in the scenario. An event represents any one of the attributes of the auctions that have time associated with them - auction end, auction start, and bids. Here, all events are in sequential order of their time delays in reference to the start of the test scenario.

We will take a sample scenario to explain the working of an event-based simulation in *fauxBay* starting from the scenario set up using the GUI in the event-based simulation and moving on to describing the processing of bidding agent's action by the test-bed.

The *fauxBay* modules that process the actions taken by the bidding agent are discussed in more detail below.

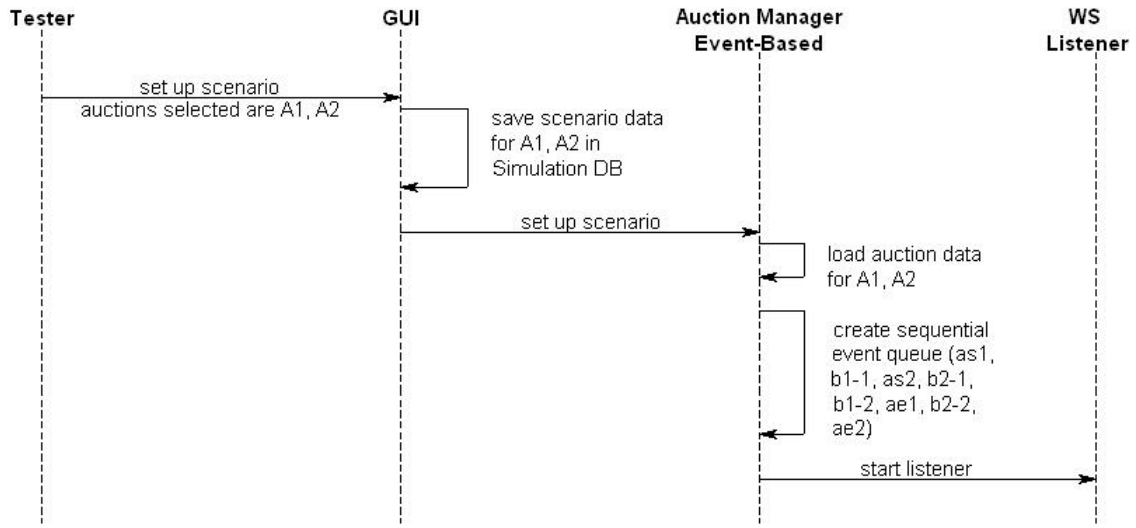

Figure 5.1: Event-Based Simulation: Scenario Set up

AuctionGuiEB

Purpose: *AuctionGuiEB* helps tester to search for auctions and select auctions from the search result to set up the test scenario for testing the bidding agent. The three panels used in scenario set up are shown in figures B.1, B.2, B.3. The input and output for each of the panels in the GUI is listed below:

Input:

- Search interface - keyword or item ID.
- Selection interface - scenario ID and auction selection.
- Monitor Scenario interface - No Input.
- Ending Scenario Options - Window close event before the end of simulation.

Output:

- Search interface - List of matching auctions with all attributes in a table format.
- Selection interface - Stores the itemID for selected auctions and the scenario ID in the *fauxBay Simulation Database* (SDB).

- Monitor Scenario interface - No output.
- Ending Scenario Options - Saves current bid data if the simulation has not ended.

Suppose, using the *AuctionGuiEB*, a tester chooses two auctions, A1 and A2, for simulation as shown in figure 5.1. The auction properties for the two selected auctions are:

- Auction A1
  a) Auction Start: as1
  b) Bids: b1-1, b1-2
  c) Auction End: ae1
- Auction A2
  a) Auction Start: as2
  b) Bids: b2-1, b2-2
  c) Auction End: ae2

AuctionManagerEB

Purpose *AuctionManagerEB* manages the simulation of all auctions in the scenario. It maintains a sequential queue of events in time. The sequential event queue generated by the AuctionManagerEB in the case of sample scenario is: as1, b1-1, as2, b2-1, b1-2, ae1, b2-2, ae2 – in order of their occurrence in time. The sequence of actions taken by bidding agent and their processing done by *fauxBay* for the sample scenario is shown in the figure 5.2.

Input: The inputs to the Auction Manager for starting simulation through method *startScenario()* are the agent ID and the scenario ID of the test scenario set by the tester. The inputs for placing a bid through the *placeOffer()* method are item ID of the item for which the bid is intended as itemID, value of the bid as bidValue, and the time delay in reference to current time as bidTime. The input bidTime indicates the time delay after which the bidding agent wants to place its bid on the item. The AuctionManagerEB also informs the bidding agent of simulation end after processing the last event in the sequential event

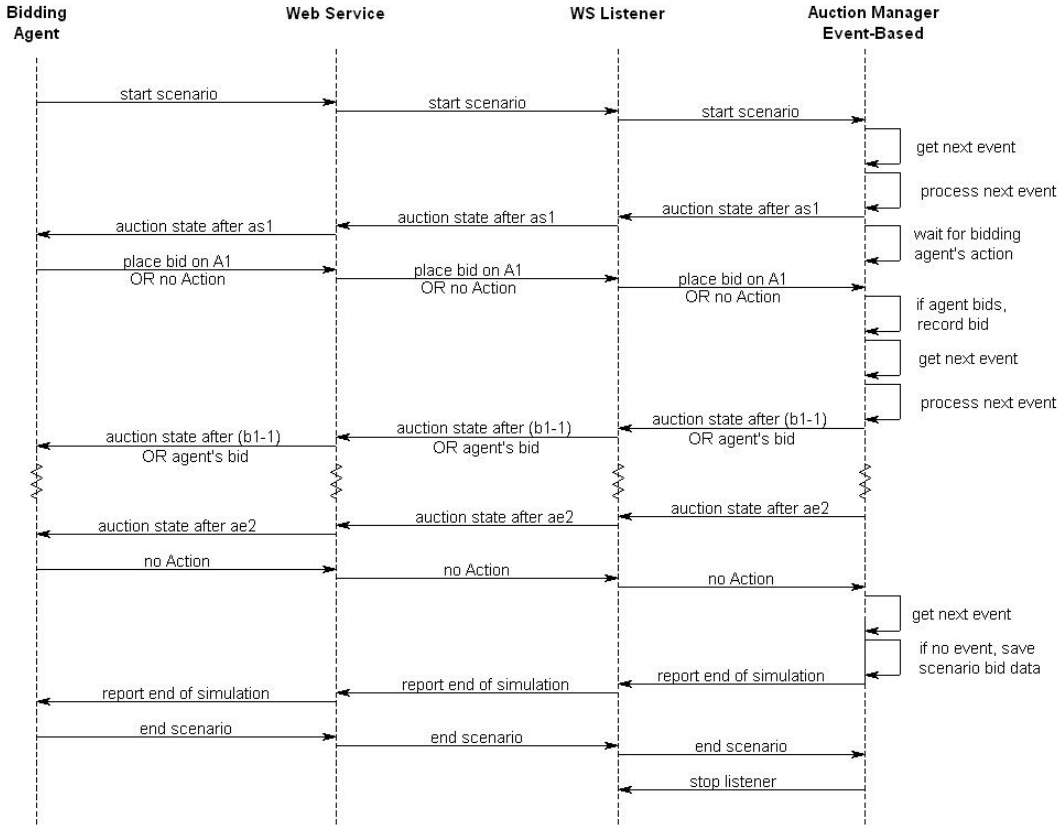

Figure 5.2: Event-Based Simulation

queue. There is no input for *endScenario()* method which is used by the bidding agent to end the scenario after AuctionManagerEB reports the end of simulation. After *endScenario()* method is invoked, AuctionManagerEB stops the Web Service Listener in order to stop any subsequent calls to the methods exposed through the Web Service interface.

Output: The output of the Auction Manager depends on the method which is invoked. For the *startScenario()* method, the Auction Manager returns the first event of the test scenario and waits for the bidding agent's next action. In case of *placeOffer()* method, the Auction Manager returns the next event in the sequential queue of events after recording the bid placed by the bidding agent. In case, there is an event in the queue which occurs before the bidTime specified by the bidding agent, *placeOffer()* returns this event and the

agent's bid is recorded but the bid is not placed until bidTime.

EBWS APIs

Purpose: The bidding agent uses the Web Service (WS) API to start the test scenario and place bids on auctions. The Web Service is deployed on the Tomcat Container and a URL is available for the bidding agent to find it and use it.

Input: For method *startScenario()*, the inputs are agent ID and scenario ID. These inputs are supplied to EBWS and are needed by the Auction Manager to load auction data from the Simulation DB for the given scenario ID. For method *placeOffer()* inputs are item ID of the item for which the intended bid as itemID, value of the bid as bidValue, and the time delay in reference to current time as bidTime. The input bidTime indicates the time delay after which the bidding agent wants to place its bid on the item.

Output: In case of *startScenario()* method, the WS API returns the first event of the test scenario received from the Auction Manager and waits for bidding agent's next request. In case of *placeOffer()* method, the WS API returns the next event it receives from Auction Manager in the sequential queue of events after recording the bid placed by the bidding agent.

5.1.2 Clock-Based Simulation

Clock-based simulation provides an option where the bidding agent strategy is tested against a real life representation of an online auction market where time moves linearly by the clock. Here, all events are simulated in accordance to their actual time (as in historical eBay data) with time scaled down by a configurable factor e.g. hours scaled down to minutes and minutes to seconds. The functioning of *fauxBay* modules for setting up a test scenario and running the test scenario in a clock-based simulation is described below.

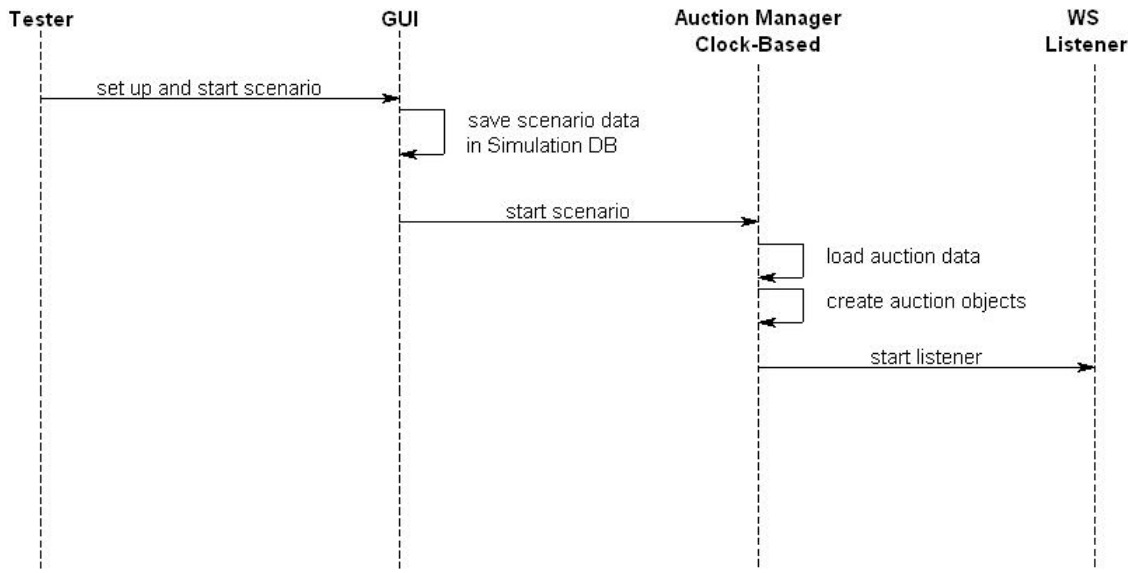

Figure 5.3: Clock-Based Simulation: Scenario Set up

AuctionGuiCB

Purpose: *AuctionGuiCB* helps tester to search for auctions and select auctions from the search result to set up the test scenario for testing the bidding agent. The scenario set up in clock-based simulation is done as shown in figure 5.3. It is different from *AuctionGuiEB* in that it also takes scenario duration as an input from the tester to set up the test scenario. This scenario duration is used by the Auction Manager to end the simulation whenever the scenario duration expires. Rest of the functionalities of *AuctionGuiCB* is same as the *AuctionGuiEB*. The three panels used in scenario set up in a clock-based simulation are shown in figures B.4, B.5, B.6.

AuctionManagerCB

Purpose: *AuctionManagerCB* manages the simulation of all auctions in the scenario. It maintains each auction as an auction object. Each auction object encapsulates both the timed and static attributes of an auction. The plausible set of disparate actions taken by bidding agent and the processing done by *fauxBay* modules for each of these actions is shown in the figure 5.4.

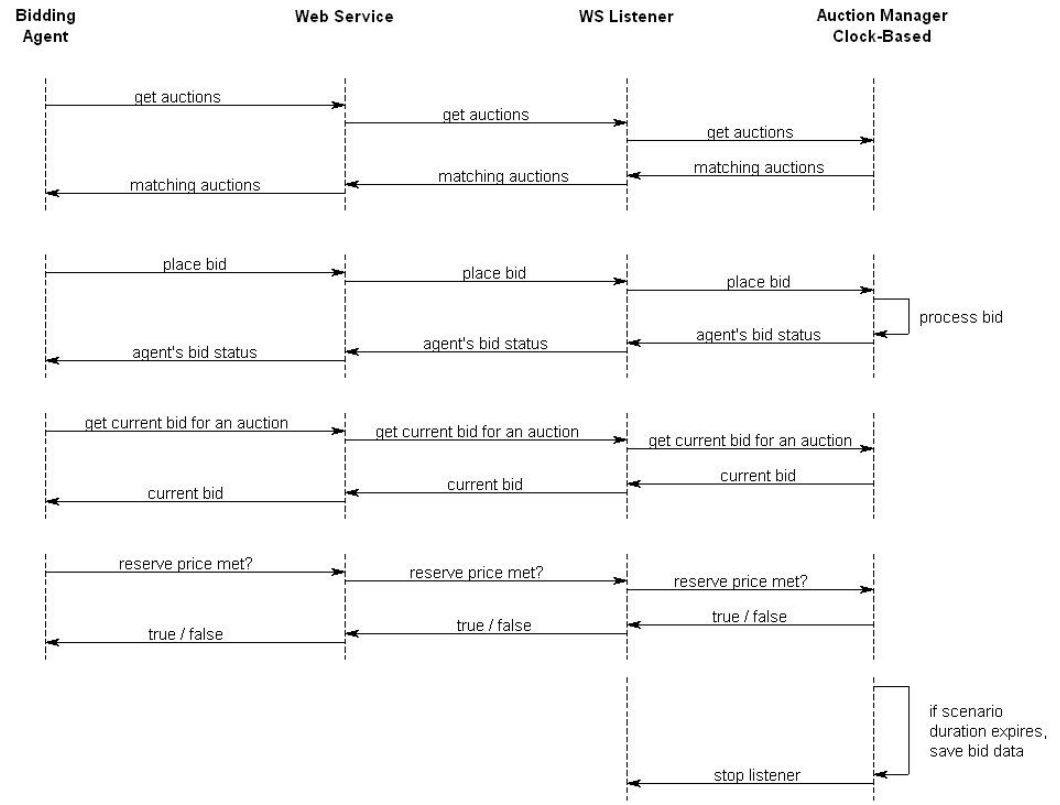

Figure 5.4: Clock-Based Simulation

Input: The inputs to the *AuctionManagerCB* for getting auctions through method *getSearchResults()* is the search keyword. The inputs for placing a bid through the *placeOffer()* method are item ID of the item for which the bid is intended as itemID, value of the bid as bidValue, and the quantity of items requested bidQuantity. The input bidQuantity indicates the quantity of item bidder agent wants to buy in an auction. This is 1 in case of reserve price auctions, which is the auction mechanism supported by *fauxBay* at this time. However, bidQuantity can a value greater than one when *fauxBay* is integrated with the support for Dutch auctions where multiple items are sold in an auction. The input for getting the current bid in a running auction through *getCurrentBid()* is item ID for the item. Similarly, the input for finding out whether the reserve price in a running auction has been met through the method *isReservePriceMet()* is item ID for the item.

Output: The output of the *AuctionManagerCB* depends on the method which is invoked. In case of *getSearchResults()* method, *AuctionManagerCB* returns all the matching auction based on the search keyword specified by the bidding agent. In case of *placeOffer()* method, the Auction Manager returns whether the bidding agent's bid was successful or not. The *placeOffer()* method returns after recording the bid placed by the bidding agent even if it is not successful. All bids (successful/unsuccessful) are recorded as actual bids placed by the bidders just like real eBay auction data on bid history. In case of *getCurrentBid()* method, *AuctionManagerCB* returns the current bid of a running auction based on the item ID specified by the bidding agent. In case of *isReservePriceMet()* method, the *AuctionManagerCB* returns true if the reserve price for the auction has been met and false in the other case.

AuctionCB

Purpose: This component encapsulates individual auction attributes and business rules relating to the bidding process. It also holds the historical bid data for the auction and the new bidding data from the scenario test run. This is only used in the clock-based simulation.

SchedulerCB

Purpose: This component runs scheduled task at a configured interval. It also keeps track of time for any other component to reference. It keeps track of simulation elapsed time, auction elapsed time, coordinates start and end of auctions, and schedules historical bids. A simulated auction ends when auction elapsed time is equal to auction duration set by the tester. A simulation ends when simulation elapsed time is equal to the scenario duration time also set by the tester. *SchedulerCB* also has the capability to scale down the running time for faster testing.

CBWS APIs

Purpose: The bidding agent uses the Web Service (WS) API to get auctions based on keyword search, place bids on auctions, find the current bid and whether the reserve price is met in the running auctions. The Web Service is deployed on the Tomcat Container and a URL is available for the bidding agent to find it and use it.

Input: The inputs to the CBWS for getting auctions through method *getSearchResults()* is the search keyword. The inputs for placing a bid through the *placeOffer()* method are item ID of the item for which the bid is intended as itemID, value of the bid as bidValue, and the quantity of items requested bidQuantity. The input bidQuantity indicates the quantity of the item the bidding agent wants to buy in an auction. The input for getting the current bid in a running auction through *getCurrentBid()* is the item ID for the item. Similarly, the input for finding out whether the reserve price in a running auction has been met through the method *isReservePriceMet()* is item ID for the item.

Output: The output of the CBWS depends on the method which is invoked. In case of *getSearchResults()* method, CBWS returns all the matching auctions received from *AuctionManagerCB* based on the search keyword specified by the bidding agent. In case of *placeOffer()* method, the CBWS returns whether the bidding agent's bid was successful or not. The *placeOffer()* method returns after recording the bid placed by the bidding agent even if it is not successful. All bids (successful/unsuccessful) are recorded as actual bids placed by the bidders just like real eBay auction data on bid history. For *getCurrentBid()* method, CBWS returns the current bid of a running auction based on the item ID specified by the bidding agent. For *isReservePriceMet()* method, the CBWS returns true if the reserve price for the auction has been met and false in the other case.

5.2 Parser

Purpose: The parser used for retrieving real auction data from eBay is a collection of Python scripts. The data is collected from completed auctions of both the Reserve price and the Dutch type auctions on eBay.

Input: The input to the parser is a list of item ids used by eBay for identifying the items. The file containing these ids is passed as an input argument to the script. This file is a text file containing one id on each line. It does not matter how the ids are gathered.

Output: The output of the script is two comma separated files (.csv extension) which can be directly loaded into a database such as MySQL. One of these files contains the

description of each item on each line. The other file contains the bidding history for each item.

Implementation Detail: The main components of the parser are

1. *Main:* The Main function parses the input text file and stores all the item ids from it in an array. For each item, it appends the id to a specific eBay URLs and retrieves the HTML pages containing its description and bidding history. The HTML page is retrieved as a data stream and fed to the HTML parser for parsing the required fields. The HTML parser is a wrapper to the in-built SGML parser. The HTML parser just extracts the required data from the page and stores them in a list separated by $'|'$. Once that is done, the list items are separated and stored appropriately in the output .csv file. The number of items to be parsed, the item from which to start, the two output files, and the URLs are specified at the start of the program.

   A certain type of auction, called the private auction, does not reveal the ID (identification) of the seller as well as the bidders. We search for the specific string "User Id kept private" and do not process those items further.

2. *HTML Parser:* Since the information presented by eBay in the page as a set of HTML tables, before passing the stream to the HTML Parser, the main function looks for specific strings such as "Winning Bid", "Buy It Now". and passes the stream data after that to the HTML Parser. The HTML parser then parses that specific table and calls the appropriate handlers whenever it encounters a new table row or a new table cell. Note that this is specific to our implementation and the current format used by eBay. The HTML parser further filters out white spaces and end of line characters and appends a $'|'$ between the field name and field data. Each of these name and data pairs are stored as separate items in a list which is returned back.

   Another implementation of the HTML parser parses the bidding history page and parses the bidder ids, bid time, bid amount and item id.

3. *ClientCookie and ClientForm:* These are external python libraries developed by John J. Lee for handling HTTP cookies on the client side and HTML forms. The ClientCookie module manages the exchange of cookies required for automatic login with eBay. Automatic login via the python script is required to retrieve the bidding his-

tory of Dutch types of auctions. eBay requires user authentication before providing that information. The ClientForm library provides the APIs for automated filling of forms elements such as Username and Password on the signon page.

5.3 Limitations

1. *fauxBay* does not support testing of selling strategies for seller agents. It currently supports testing of bidding strategies only.

2. *fauxBay* currently supports reserve price auctions only. The support for other auction mechanisms can be integrated with *fauxBay* easily by adding modules that implement the rules for these auction mechanisms.

5.4 Compatibility and Extensibility

fauxBay is independent of the language in which the bidding agents are developed as it interacts with them through the Web Service API. *fauxBay* is also designed to be highly extensible, so that new auction rules can easily be implemented. In event-based simulation, new auction rules can be added to the module *processNextEvent()* in AuctionManagerEB component where all events are processed depending on the auction mechanism. In clock-based simulation, new auction rules can be integrated as modules within the Auction class where all processing is done based on the auction mechanism of the Auction object.

Chapter 6

Conclusion and Future Work

We implemented *fauxBay* as a test-bed for testing the strategies of bidding agents. The purpose of *fauxBay* is to provide a simulation of real eBay auctions that would allow research on strategic bidding in online auction markets. *fauxBay* serves as a test-bed for bidding strategies employed by software agents in a single item reserve price auction mechanism. *fauxBay* is designed to be easily extensible to allow new auction mechanisms to be integrated with it. Currently, *fauxBay* is designed to support strategy testing for a single bidding agent. However, it can be extended to support multiple bidding agents as well. *fauxBay* allows the tester to set up a simulated online auction market environment for the bidding agent by choosing a set of auctions from real eBay auction data.

fauxBay offers two modes of simulating real eBay auctions. The event-based simulation of online auction market in *fauxBay* provides a quick test of the bidding agent's strategy. *fauxBay* also offers a clock-based simulation of real eBay auctions with actual running time scaled down by a configurable factor. This can be helpful in situations where agents have to be tested for their bidding strategy as well as their behavior and performance in real online auction environments that move linearly in time.

Another highlight of *fauxBay* is that it provides a single web service interface that can be used by both human bidders and software bidding agents over the web unlike [13]. By using a web service interface *fauxBay*'s simulation is not reliant on the language in which the bidding agents are developed. Hence, *fauxBay* does not restrict agent developers to develop their agents in a single development language and focuses solely on testing the bidding

strategy of the agents. Future work will include integrating different auction mechanisms in *fauxBay* along with extending support for testing of multiple bidding agents. *fauxBay* can also be extended in future to support testing of selling agents. Integrating different auction mechanisms will also involve collection of real auction data from eBay on these auction mechanisms.

Bibliography

[1] http://www.w3.org/TR/ws-arch/.

[2] http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[3] Soap. http://www.w3.org/TR/soap/.

[4] Webservice. http://esw.w3.org/topic/WebServices.

[5] Xml. http://www.w3.org/XML/.

[6] eBay. http://www.eBay.com.

[7] FIPA:Foundation for Intelligent Physical Agents. http://www.fipa.org.

[8] B. Kahng I. Yang, H. Jeong and A.-L. Barabasi. Emerging behavior in electronic bidding. *American Economic Review*, 68:016102, July 2003.

[9] J. Dale, S. Willmot, and B.Burg. Agentcities: Challenges and Deployment of Next-Generation Service Environments. In Proceedings of Pacific Rim Intelligent Multi-Agent Systems, Tokyo, Japan, August 2002.

[10] JASA:Java Auction Simulator API. http://www.csc.liv.ac.uk/~sphelps/jasa/.

[11] Stephen J. Rassenti Kevin A. McCabe and Vernon L. Smith. Auction institutional design: Theory and behavior of simultaneous multiple-unit generalizations of the dutch and english auctions. *American Economic Review*, 80(5):1276–1283, Dec 1990.

[12] Michael P. Wellman, Amy Greenwald, Peter Stone, and Peter R. Wurman. The 2001 trading agent competition, fourteenth innovative applications of artificial intelligence. Conference (IAAI-2002), Edmonton, August 2002, pp. 935-941.

[13] P.R. Wurman, M.P. Wellman and W.E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents.

[14] K. Rock. Why new issues are underpriced. *Journal of Financial Economics*, 15:187–212, 1986.

[15] Alvin E. Roth and Axel Ockenfels. Last-minute bidding and the rules for ending second-price auctions: Evidence from ebay and amazon auctions on the internet. *American Economic Review*, 92(4):1099–1103, Sep 2002.

[16] S. Barbaro and B. Bracht. Shilling, squeezing, sniping: Explaining late bidding in online second-price auctions. `http://www.sta.uni-mainz.de/barbaro/node5.html`. 2004.

[17] Tuomas Sandholm. eMediator:a Next Generation Electronic Commerce Server. In Proceedings of AAAI-99 Workshop on AI in Electronic Commerce, Orlando, FL, July 1999, pp. 46-55.

[18] N. Vulkan. Economic implications of agent technology and e-commerce. *The Economic Journal*, 109:67–90, 1999.

[19] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan. TAGA: Trading Agent Competition in Agentcities. Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, Acuulco MX, 11 August, 2003.

Appendix

Appendix A

Use of Web Services in fauxBay

We use web services as an interface exposed to the bidding agent. The use of web services has these key motivations:

1. Web services allow isolated applications to interact with each other and share information over the web [4].

2. Web services allow heterogeneous systems to connect using SOAP [3] and XML [5] based request-response protocol.

3. Web services allow loosely coupled functionality that permit compatibility with systems built in different languages like C, C++ or Java.

In fauxBay, the use of web service APIs allows the agent developer to develop agent in any programming language as long as it is capable of invoking the web services. It also makes fauxBay independent of the development language used to build the bidding agent.

Currently, fauxBay supports test scenario for a single bidding agent. However, in future fauxBay can be extended to support simultaneous testing of multiple bidding agents. This scenario will be particularly benefit from the use of web services interface of fauxBay. All participating human users and bidding agents will be able to use the test-bed by invoking web service methods over the internet.

A.1 Web Services Architecture

Figure A.1 illustrates the main components of Web Services Architecture [1][2]. A brief description of these three main components is as follows:

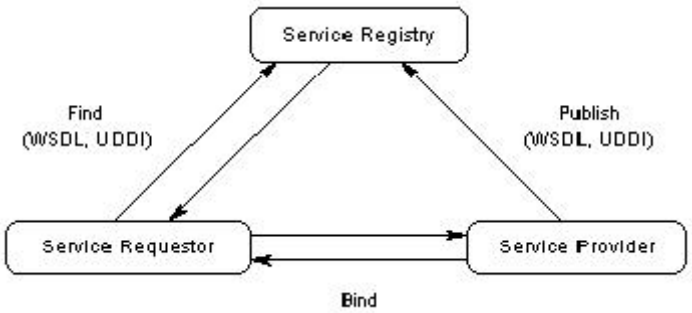

Figure A.1: Web Services Architecture

1. Service Provider:
   - Provides applications as services over the internet.
   - Publishes these services to the Service registry using WSDL (Web Services Description Language).
2. Service Registry:
   - Provides a registry for publishing service descriptions offered by service providers.
   - Provides a registry for service requesters to search for published services using UDDI (Universal Description, Discovery and Integration).
3. Service requestor:
   - Application which requires services over the internet.
   - Searches for published services (WSDL) on the Service Registry.

Appendix B

fauxBay's GUI Snapshots

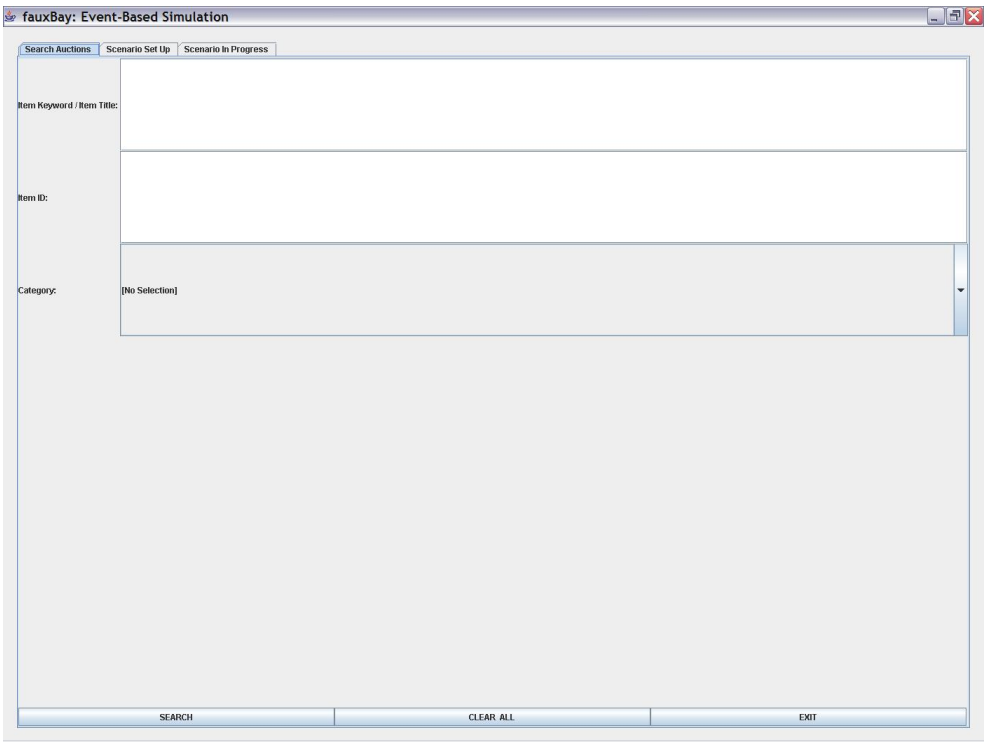

Figure B.1: AuctionGuiEB Search Interface

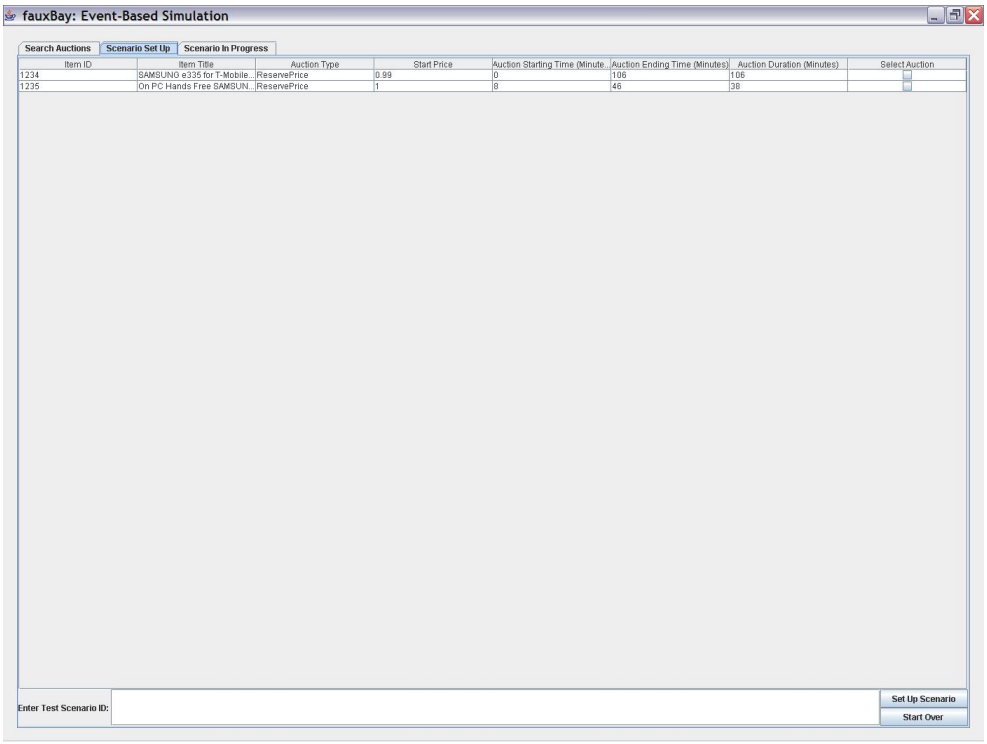

Figure B.2: AuctionGuiEB Selection Interface

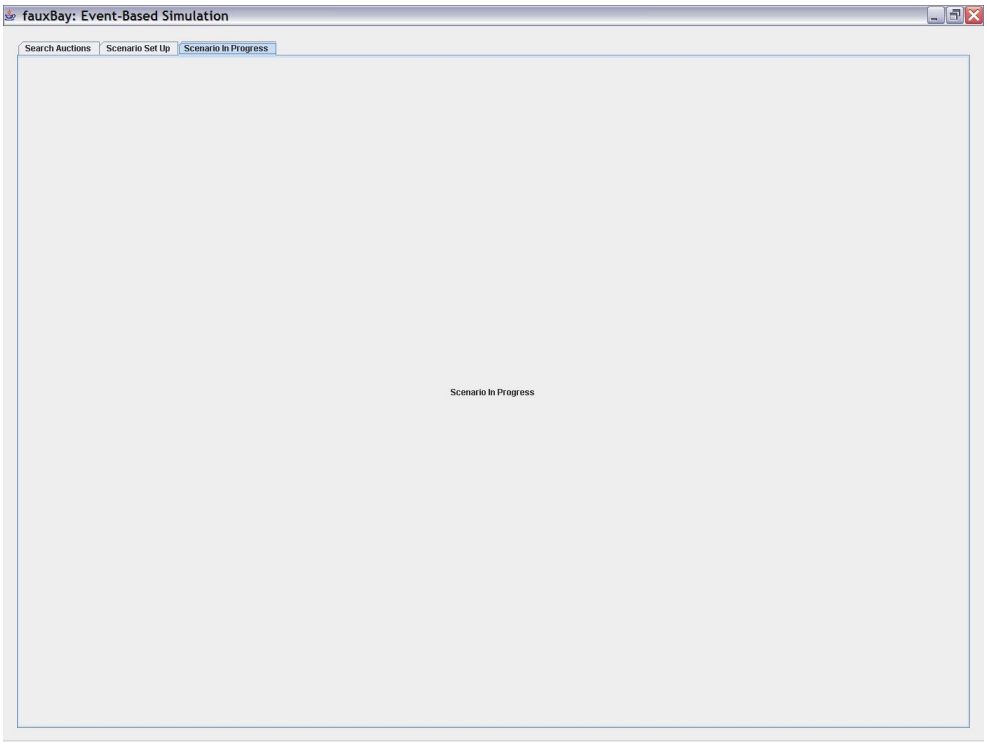

Figure B.3: AuctionGuiEB Monitor Scenario Interface

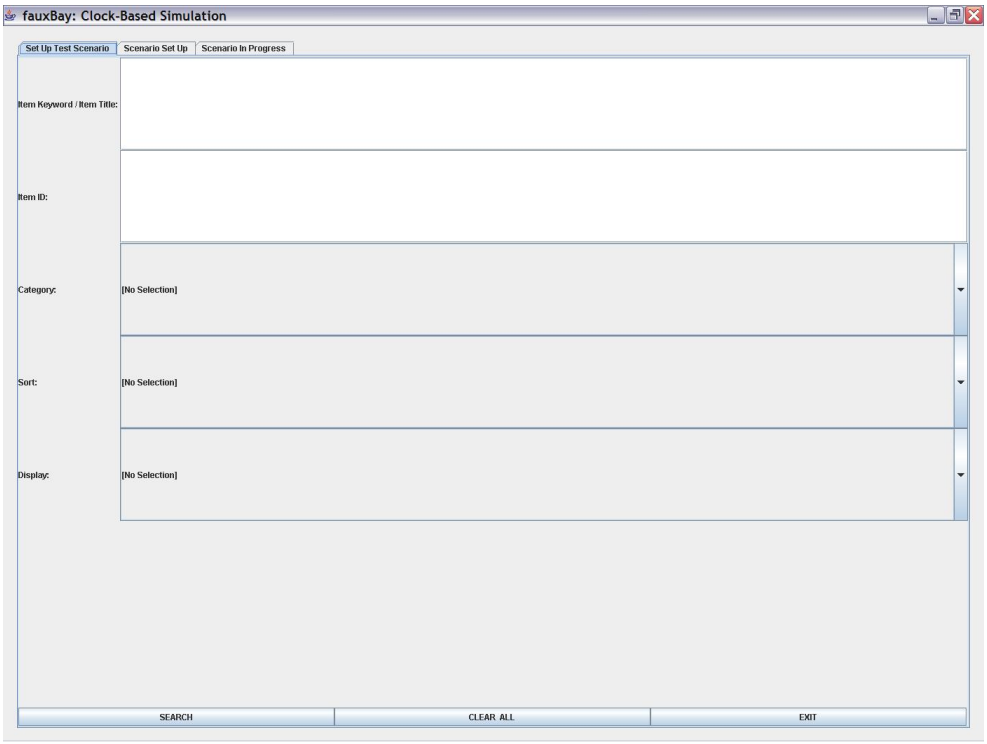

Figure B.4: AuctionGuiCB Search Interface

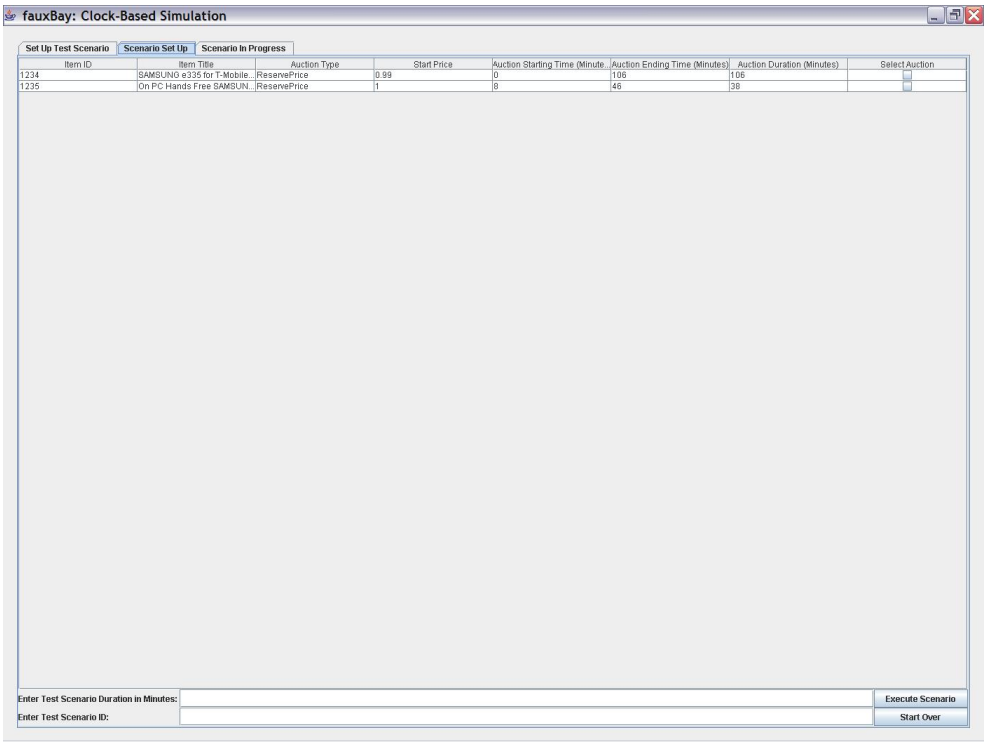

Figure B.5: AuctionGuiCB Selection Interface

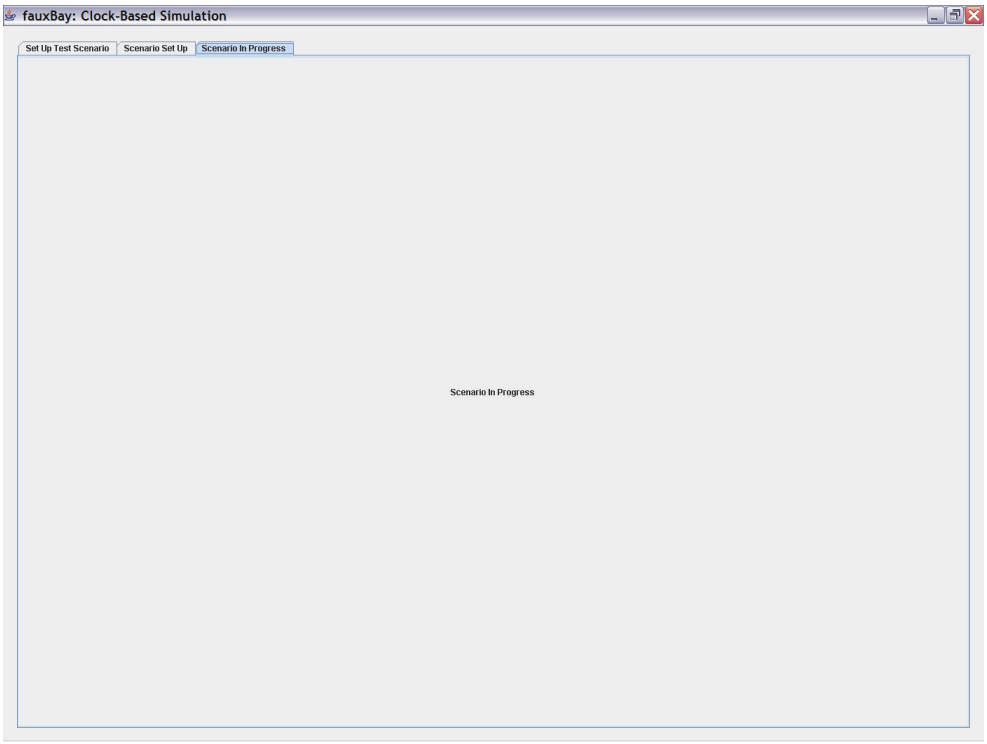

Figure B.6: AuctionGuiCB Monitor Scenario Interface