Design And Development Of A Framework For Testing And Developing Internet Auction Systems

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Ashish Kashyap

to the

Department of Computer Science & Engineering

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April, 2000

Certificate

Certified that the work contained in the report entitled "*Design And Development Of A Framework For Testing And Developing Internet Auction Systems*", by *Ashish Kashyap*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Dheeraj Sanghi

April, 2000

Preface

Internet Auctioning has gained immense importance in the last few years. However, the flexibility that Internet, as a medium, provides to auctioning is yet to be completely explored. Internet can allow every person to design and implement a new type of auctioning model that is better suited to the auction of his product. Moreover, sellers and buyers can decide on the auctioning strategies better suited to them. Automatic bidding agents can be used to make bids on your behalf. This project aims at providing a framework for addition of most auctioning models and development of automatic bidders and testing of auctioning strategies. This report is a description of the software that implements the framework and develops an interface on top of the framework to allow the users to make use of this software as a tool for development of Internet Auction Systems and testing of auctioning strategies.

Acknowledgements

I would like to thank my B. Tech. project supervisor Dr. Dheeraj Sanghi for his invaluable guidance and support which helped me in successfully completing this project. I would also like to thank my colleagues for their suggestions and constructive criticism throughout the course of this project.

Contents

List of Figures

Chapter 1

Introduction

Some of the technological innovations of the recent past have made a huge impact on us. They include Internet and a host of other web-related developments. *E-commerce* is one such facet of Internet-related developments. The rapid growth of E-commerce can be traced to the desire to bring trading activities to the home of the people, thereby making a huge improvement in the convenience and comfort related with marketing.

Trading can either be *fixed-price trading* or*negotiable-price trading*, better known as *auctioning*. The terms **Electronic Auctions**, **Online Auctions** and **Internet Auctions** refer to the negotiable-price trading activity carried out over the Internet.

Electronic auction model has some very obvious advantages over the conventional brick-and-mortar model of auctioning in which the whole auctioning takes place at a particular physical space in the presence of bidders and moderators. The most important of this is *greater reach*, both for the customer and for the merchant, which is restricted by both space and time in the conventional model. While a conventional auction can only take place between fixed durations at fixed time to a fixed set of bidders, there is no such restriction in case of Internet Auctioning. The customer can choose between merchants by comparing various auctions simultaneously taking place. The operation cost also reduces as the moderators can be replaced by software programs and there is no cost involved in transportation of bidders to the place of auctioning. Electronic auctions are also *less time-consuming* as the time wasted before and after auctions is greatly reduced. Moreover, electronic auctions can extend over longer durations, if need be, as physical presence is not involved.

The major hurdles restricting the growth of Electronic Auctions are *infra-structural requirements*, *security requirements* and *fairness and reliability requirements*. Infra-structural requirements include connectivity of all bidders and traders, servers to handle auctioning data, and payment module acceptable and available to all persons involved in the auction. Security requirements include authenticity and non-repudiability of users, privacy of users, integrity of data transferred between users and prevention of cartel formation. The fairness requirements imply that there should be no inherent bias for some of the same-status users (as decided by the auction policy), Reliability requirements imply that the software should be reliable. [5] [6]

Another major hurdle for the growth of Electronic Auctions is *generality requirements*. All the major commercial auction sites today support only two or three models of auctioning with minor variations. The emphasis is on service. However, there exist some conventional but not-so-popular auctioning models which are not supported by the commercial sites because of their relative lack of popularity. Moreover, there might be reasons for merchants to prefer one auctioning model over the other. This will gain further prominence if the merchant is allowed to weigh the two models by use of simulations. More importantly, people can develop auctioning strategies and test them using simulations to decide on a particular auctioning strategy to use for a particular case. The focus of this project is to develop such a framework that facilitates the task of testing and developing new auctioning models and strategies for those models.

Chapter 2

Internet Auctioning - Background

2.1 Services

Broadly speaking, **Internet Auction System** is a system software that allows conduct of negotiable-price trade over the Internet. In commercial use, the Internet Auction Systems also provide a lot of auxiliary features to make their site differentiable. [1] [2] The basic services provided by the popular commercial auction systems are:

- *Product Listing* - The consumer puts a commodity for auctioning and gives a description of the product to be auctioned. He selects a particular auction type and might specify a reserve price below which the product will not be sold. He might also specify the auction closing time. Depending on the particular auction type, there might be some other parameters which need to be specified by the seller.

- *Bidding* - The consumers might specify a price at which they are ready to buy the product or might agree to a price specified by the seller.

- *Browsing the list of active auctions* - The consumer might search for and select a particular product by browsing through the list of active auctions. Sometimes, products are put into various categories and the surfers have the option to browse through a particular category of product.

- *Explicit search for a product*
- *Notifications* - The consumer needs to notified about some important events from time to time. For this purpose short messages can be e-mailed to the consumers. The purpose of the notification can be pure information like in case of the consumer being declared a winner in some auction, or promotion of a new product put on auction. Generally, the messages are highly personalized and the users preferences are taken into account to avoid spamming.
- *Payment and shipping* - For this purpose, the auction site sometimes collaborates with other financial institutions and other related companies.

Some of the add-on services provided by them are:

- *Free Insurance* - Many e-auctioneers have tied up with insurers and provide limited insurance on the delivery and quality of the auctioned item. This insurance is generally limited to the order of few hundred dollars.
- *Escrow Services* - Escrow services refer to the service in which the buyer pays money into an escrow account but the money is forwarded to the seller when the buyer confirms that she has received the goods and is satisfied with it.
- *Authentication/Evaluation* - Grading is used to evaluate the physical condition of an item. Most sites employ the services of professional firms for this. For example, at Yahoo! Auctions the actual grading system is dependent on the type of item being graded. For example, trading cards are graded on a scale from A1 to F1, whereas coins are graded from poor to 'perfect uncirculated'.
- *Investigation*

2.2 Classification

Based on the consumer community at which the site is targeted, Internet Auction Systems are classified as *consumer-to-consumer*, *business-to-consumer* and *business-to-business.*

In the *consumer-to-consumer* electronic auction, both the major actors involved in the auction - the seller and the buyer - are consumers. An example of this may be: John gets bored of his old car and decides to auction it on the web where Jack buys the car after over-bidding the other fellow bidders. This is the most popular type of auctioning on the Internet as sales and purchase of second-hand goods has become very easy. The major hurdle is how to ensure the quality of the product being auctioned. But a large number of third party rating companies are appearing to rate the quality of used goods.
In the *business-to-business* electronic auction, both the major actors belong to the business community. This takes place, generally speaking, for sale of large quantities of goods from one merchant to another. This is catching up quite fast these days.
In the *business-to-consumer* electronic auction, a merchant sells a good, generally new product, to a consumer. This is not so popular as fixed-price trade is better suited for new products.

2.3 Auctioning Models

Most of the commercial auction sites only support the two most popular auctioning models - English Auction and Dutch Auction.

English Auction, also known as *Open Cry Auction*, is the most common type of auction. This is the common type of auctioning that we are all familiar with. In this case, there are a number of bidders. The bidders make a bid by specifying the price at which they are ready to buy the product. All the bids are, generally, known to all other bidders. The process continues till no new bid out-bidding the current highest bid is received for a specific period of time. In such a case, the person who made the highest bid wins. There are a number of variants of this auction type allowing the seller to specify a reserve price (minimum price at which he is ready to sell the product), or an the first bid price (price specified in first valid bid) and the time interval between bids and the time to wait before closing auction.
In **Dutch Auction**, there are a number of quantities of the same product and the seller keeps on decreasing the price at which he is ready to sell them. The buyers

specify if they are ready to buy the good at that price or not. The process continues till all the goods are sold.

Vickery Auction is designed for a seller offering a single unit. The highest bidder wins the good at the price offered by the second highest bidder. (A reservation price can optionally be specified by the seller).

In **Sealed-bid Auction**, the bids remain sealed from the other bidders and the sellers.

In **Multiple-round Anonymous Auctions** [3] [4], the bids are made on the prices offered by the sellers. There are a number of auctioneers for the auction of a single bid and the bidder send a bid-vector to each of them. This ensures that neither the auctioneers, nor the seller comes to know which bidder is making what bid. Finally, when the auction comes to an end, then the seller gets to know which bidder won the auction. This preserves the anonymity of the bidder.

In **(M+1)st-Price Auction** [7], both buyers and sellers make bids. Let M be the number of active sell bids, each for a single unit of the good. The clearing price for an (M+1)st-price auction is the (M+1)st highest price specified in any of the bids. Sells as well as buys are counted. All buy bids at prices greater than the clearing price are successful, as are all sell bids at prices less than the clearing price. While the highest remaining buy bid is greater than or equal to the lowest remaining sell bid, remove these from the set of outstanding bids and add them to the set of matched bids. For the case of exactly one seller, this auction is equivalent to the standard first-price auction, with the provision that the seller is allowed to specify a reservation price. The clearing price corresponds to the highest price such that the number of willing buyers equals the number of willing sellers. Given a fixed set of existing bids, the price offered in a sell bid determines whether the buy is successful, but if successful, does not affect the clearing price. The same is not true for buy bids.

Mth-Price Auction is similar to the (M+1)st-Price Auction except that the clearing price is the Mth highest price specified in any of the bids.

Continuous Double Auction (CDA) matches bids in the order received. When a new buy bid is processed, the auction checks whether the offered price would match the lowest existing sell bid, and vice versa. On detection of a match, the auction clears at the price of the existing bid, and generates a new price quote. The price quote at any time is the bid/ask spread, that is, the highest buy price and lowest sell price. The CDA is a common mechanism for organized exchanges, such as stock and commodity markets.

Chronological Match Auction is a generalization of CDA that allows for non-immediate clearing. Price quotes may reflect delayed information, and bids may be withdrawn within the clearing interval.

In **Capacity Auction**, the quantity of products available varies with time.

It is not possible to give an exhaustive list of auctioning models as just by slight variation of a rule, you can create a new auction model. Theoretically speaking, the number of auctioning models is infinite. Moreover, some of the auctioning models vary greatly in their basic structure. Hence, it is almost impossible to give a uniform structure for all the auctioning models.

2.4 Design Of Internet Auction System

All the internet auction systems consist of a front end web-based interface and a back end data processing. [5]

2.4.1 Front End

The front page generally consists of links to the various options: register, browse, search, bid, sell, help, etc. Moreover, some featured auctions, which are of higher importance (the seller having paid more for listing that auction), have their current status shown on the front page. A listing of the major categories of products is shown having links to the active auctions in those categories. A number of forms need to be handled, whether to register, login, list product, bid or report, and these are generally handled by CGI scripts. The forms are static and rarely need modifications. The web pages themselves are generally dynamic in nature as they have to show current status. The HTML based pull mechanism, wherein the web page reloads itself from a server after fixed intervals, is generally used, because of its inherent simplicity.

The web pages for the seller and the buyer are kept separate and provide different services. While the seller's web interface would allow him to add auctions, the buyer's interface won't. Moreover, the mechanism of bidding would be different in both the cases. Similarly, both will be view the status of auctions differently. The differences in the services provided makes separate web pages a neccessity.

2.4.2 Back End

The purpose of the database is to store auction information and to store click data (used to determine the popularity and hence, the advertisement rates for that site) The database contains information about registration of the users, product categories, product being bid upon, auction types and rules, auction, the bids placed on the auction, the current status of auctions, payment, shipping, notifications and a number of other related informations. Clearly, this means a huge database which is constantly being updated. There are generally a number of servers to handle this data processing job.

Chapter 3

Project Overview

The objective of the project is to develop a software. The utility of the software lies in the following:

3.1 Development of Internet Auction System

It is possible to use this software as a normal Internet Auction System. However, payment mechanism is not supported in the existing system. Hence, to make this software a properly functional Internet Auction System, payment module needs to be added. Moreover, the security issue has been overlooked in favour of other sought objectives. Hence, the software, as such, is not an Internet Auction System, but a few additions can make it so.

This software has a front end and a back end like most Internet Auction Systems. The front end is a *web interface*. There are separate web pages for the buyer and the seller. The very basic services like registration, bidding, viewing status and results and product listing is provided for. Due to unavailability of sufficient time *notification* could not be provided, though it was earlier planned and the database table exists for that. The back end is a data processing server that handles all auction related data processing. No additional data (like click information) is maintained.

3.2 Integration of New Auctioning Models

The software provides a **framework** for *most* auctioning models. There exist some auctioning models which cannot be accommodated in the existing framework of this software. The major contribution of the software lies in providing a *uniform platform* for most auctions and develop a method of integrating the new auctioning model in the existing software easily using a well-defined process of specifying auctioning rules.

For the purpose of developing a framework for most models, a uniform structure needs to be identified for those models. In this case, the auctioning models that have a fixed units of a single product being sold by a single seller, with a single auctioneer, is considered. All auctions in this class are identified as having a *starting phase*, *bidding phase* and *closing phase*. The starting phase involves auction listing and initialization. The bidding phase consists of the negotiations between the buyers and the seller. All messages conveyed between the buyers and the seller are identified as being part of a bid. The closing phase consists of closing the auction and storing and declaring results. Different auctions can have different starting phase, i.e. having different sets of additional variables involved in auction listing, different bidding phase, i.e. having different types of messages being communicated between the buyers and sellers and having different sets of people allowed to make a bid, and different closing phase, i.e. having different ways to determine when to close an auction, whom to declare winner in that case and what information to keep in that case. This uniform structure is used to create a framework for these auctions.

In this software, adding a new auctioning model consists of just specifying the rules and parameters involved in that model. Most of the work has been automated. The database is such that there are no separate tables for the new auctioning models. However, to facilitate the job of the developer of a new auctioning model, methods are provided to allow the developer to use his own parameters without bothering about the database. These parameters are automatically transferred to and from the database.

3.3 Development and Testing of Auction Strategies

A large number of attempts are being made these days to find out the *optimal strategies* to maximize the profit or the revenue or the quantity of items auctioned for a particular auctioning model. Take the case of Dutch auction. The rate at which the seller decreases the price at which he is ready to sell the product is a crucial factor in determining his revenue. So, he would like to know the optimal rate. This rate will be dependent on a number of factors like: the quantity of items remaining, the number of interested buyers and probably the rate at which the quality of the product decreases with time. One way in which this can be found out is by having a set of simulations with different values for the variable factors in each simulation. This project helps them by providing a *programming interface* where they can add some code to create automatic bidders. Also after they specify the functions governing the variable factors, simulations of auctions with those automatic bidders and using the values computed for the functions in that particular simulation are carried out and the results can be viewed to estimate the utility value of that strategy.

Chapter 4

Requirements Specification

This chapter describes the requirements specifications for the software product.

4.1 User Characteristics

There can be two types of users for this product - developers of auction systems and strategies and users of the auction systems.

- **Developers of Auction Systems**: They should be computer literate. They will install this product, add the new auction types and test them. They may also use the testing tool to test their strategies for the buyer or the seller for a particular auction type. For this purpose they should be able to code in Java and be familiar with **XML**.
- **Users of Auction Systems**: They should know how to browse and be familiar with form filling. No specialization of any sort is required. They are the buyers and sellers of the auction systems.

4.2 Constraints and Assumptions

- *All types of auctions cannot be supported.* The assumption made while designing the product is that there is central auctioneer who conducts all auctions and the auctioneer can be completely trusted. Moreover, there will be only one seller selling one or more quantities of a single product.

- *There are no in-built security measures.* Any security related stuff needs to be added by the developer of the auction system.
- *The system, at present, does not support the payment procedures.*

4.3 User Interface

There are two types of interface - the web interface for the auction system users and the testing tool interface for the purpose of conducting simulations.

Web Interface

This consists of web pages for the two types of users of the auction system - the sellers and the buyers. There are separate web interfaces for both of them.

Buyer's UI

This is the interface for the customers interacting with the auction system. Every interface is a web page, either pre-generated or generated based on the actions taken when some information is supplied through a HTML form.
The front page is for the sign-in purpose. If you are a new buyer, you must register first. Links to help pages and information on system are also present. Refer to fig.1 and fig.2.
After the sign-in, the next page appears. This has a menu on the left frame. Refer to fig.3. The possible actions that the user can take here are: view status of an active auction, bid on an auction, view results of a completed auction and (as yet incomplete) view/send personal messages. Refer to fig.4 to understand the process of bidding.

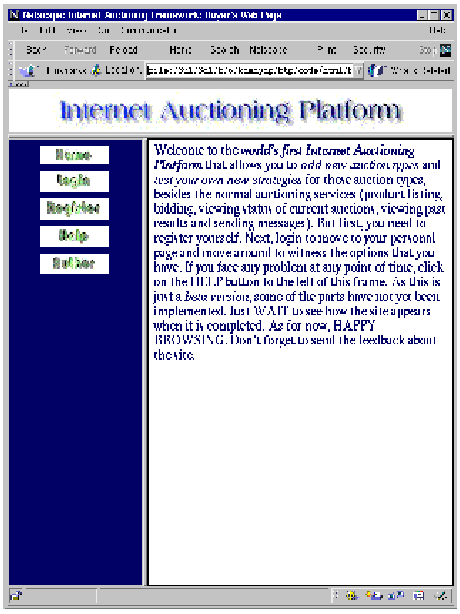

Figure 1: *Front Page For The Buyer*

Figure 2: *Registration page For The Buyer*

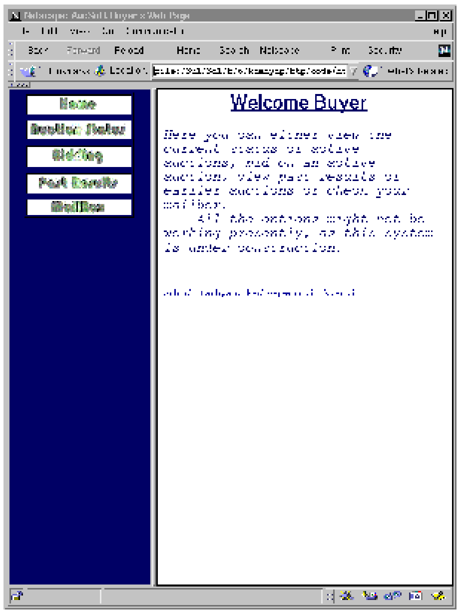

Figure 3: *Buyer's Personal Page*

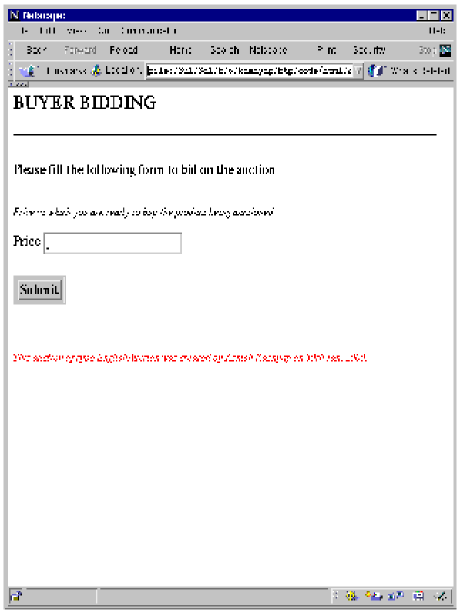

Figure 4: *Buyer Bidding Form For English Auction*

Seller's UI

This interface is for the seller. This is very similar to the buyer's UI. The only difference lies in an additional optional of adding a product for auction. The form for addition of auction is shown in fig.5.

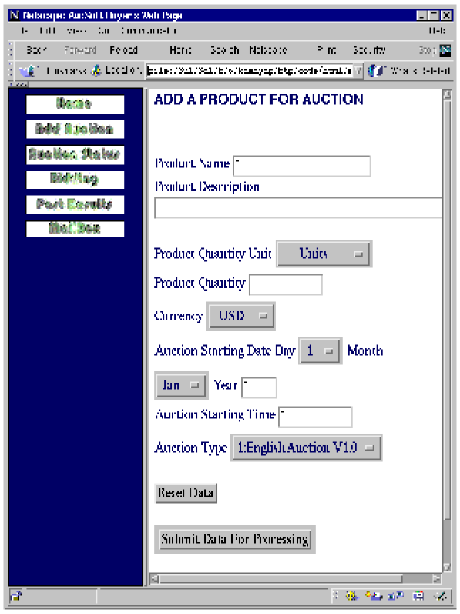

Figure 5: Auction Addition Form

Testing Tool Interface

This interface takes care of the testing tool. The interface starts with a menu with options to either create a new test run (i.e. simulate a new set of auctions with some auctioning strategy) or view past results. (Refer to fig.6.) If he wishes to create a new test run, a new screen appears that asks him to specify the number of simulations he wants to carry out and the auctioning model to be used for this case. (Refer to fig.7.) Next, he will be asked to specify the values to add a new product for auction. (Refer to fig.8 and 9.) Here, he can give the values as a function of N, the sequence number of simulation. Next, he needs to add automatic bidders. (Refer to fig.10.) For each automatic bidder, he will be asked to specify the values for the involved parameters which can again be functions of N. (Refer to fig.11.) While viewing the results, he selects a test run and the set of parameters whose values he wants to see. The values for those parameters are displayed.

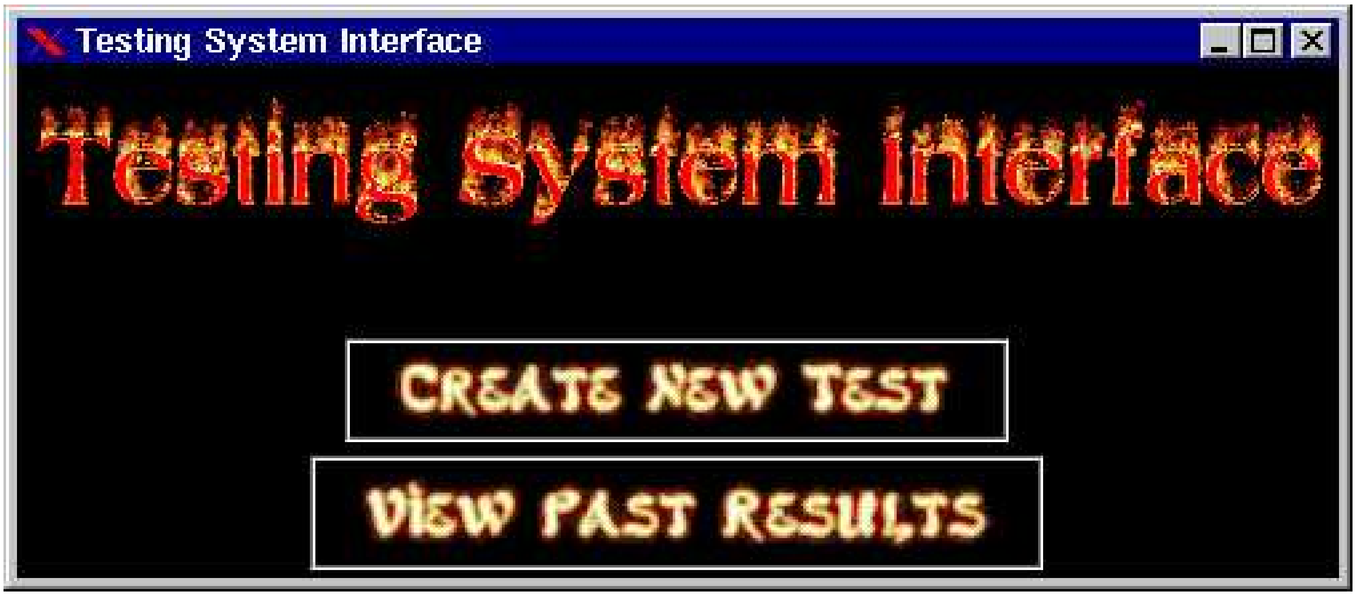

Figure 6: *Test System Interface*

4.4 Functional Requirements

The product has four basic functional requirements. They are detailed in the following subsections.

- *Requirements for development of an auction system*

  - The software should be able to *conduct different types of auction* at the same time—. For the conduct of an auction, it should accept bids from the sellers and buyers and then compare the bids according to some bid comparison rules specified for that auction type. The rules for ending the auction should also be implemented. Notifications regarding the auctions should be sent to appropriate persons as and when required.

Figure 7: *Test Creation Interface*

Figure 8: *Addition Of Auction*

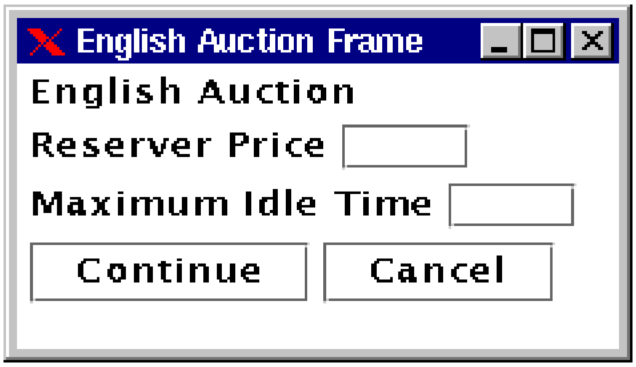

Figure 9: *English Auction Addition Form*

Figure 10: *Form For Addition Of Automatic Bidders*

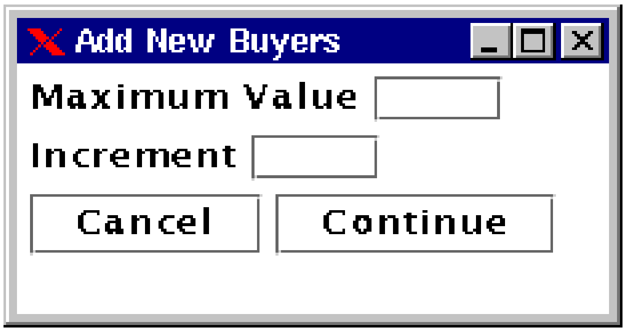

Figure 11: *Interface For Addition Of Desperate Bidder for English Auction*

- The code for an auction type should have some *generic structure*. A new auction type can be easily created just by giving the specifications for that auction type in some format (eg. corresponding to a DTD in XML) and writing code for some basic functions that express how the auction can be added (whether there can be two or more sellers, what should be the unit of product auctioned, how can the seller specify rule for close of an auction) and how it should be implemented (what should be the rule for bid acceptance, bid comparison, etc.). Some default auction types should be implemented and tested.

Requirements for seller

- The seller should be able to *register*. He has a login and a password.
- On his web page, he should be able to *add a product for auctioning*. For that purpose, he will choose a particular type of auction from the list of auction types available with the central auction system. After that he will specify the parameters associated with that auction type (this will include the product description and the quantity of product).
- He should be able to *place bid* on the type of auction which allows seller's bids if such an auction is going on.
- He should be able to *see results* of past auctions of his products. (What products were sold by him at what price, etc.)

Requirements for buyer

- The buyer should be able to *login* with his password.
- He should be able to *place buy bids* for an ongoing auction.
- He should be able to *view the list of ongoing auctions and their details.*
- He should be able to *view the details of each type of auction.*

- He should be able to *see the past results*.

Requirements for testing tool

- It should be possible to *create automatic bidders* whose functions are governed by some parameters specified before instantiating them.

- The tool should be able to *simulate auctions* with automatic bidders. The actions of bidders should have parameters that can be varied for each case to get different results in different cases.

- The simulation could be run over a number of parameters and the *result viewed*.

Chapter 5

High Level Design

This chapter describes the high-level design of the software. The first section gives the functions and parts of each module of the software and the next section specifies the mode of interaction between the modules. The last section discusses the implementation decisions taken.

5.1 Software Modules

This software consists of three modules and a database. Refer to fig.12.

5.1.1 Central System Module

This is the heart of the software and interacts with all the other modules and the database. This module is responsible for maintaining the accuracy of the information in the database by constant polling of the database. Moreover, all common tasks of the other modules (like bid comparison) are handed over to this module. The functions performed by the central system are:

- **Start an auction**: This requires detecting the start of an auction and then, on detection, performing the required initializations for that auction.
- **Process bids for active auctions**: This requires validating bids, accepting them and then comparing them to update records for the active auction.

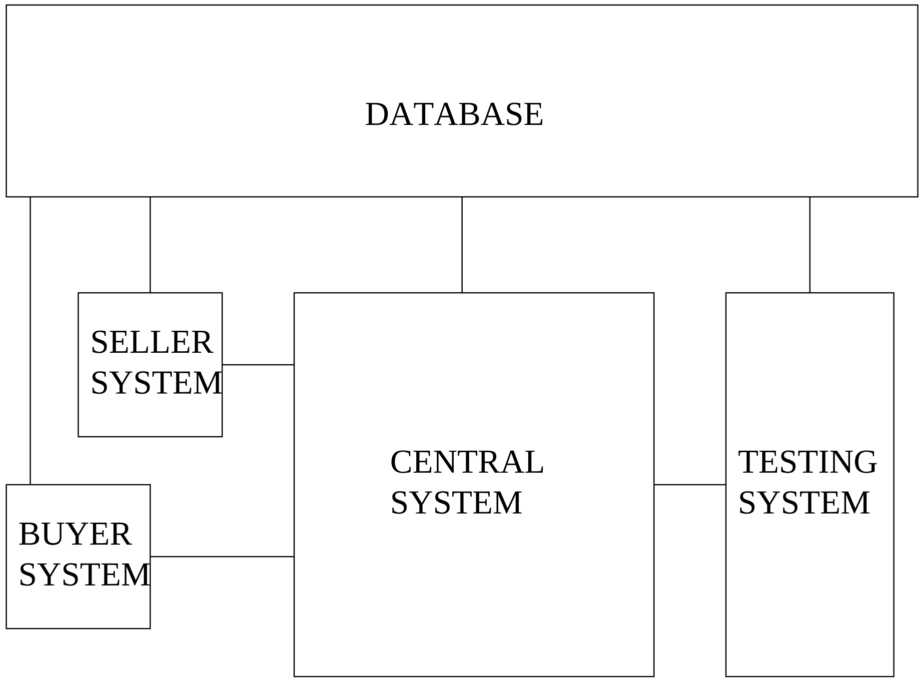

Figure 12: *High Level Design*

- **Stop auctions**: This requires checking each active auction periodically to find if it needs to be closed. In case an auction needs to be closed, the results of the auction need to be published and stored.
- **Send messages**: Some notifications need to be sent to the users about some important events. This may be start of an auction, close of an auction, or any any other event.

5.1.2 Auction System Module

This consists of a set of processes which do the processing on the information coming to them from the web interface. Two sets of sub-modules, the Seller System Sub-Module and the Buyer System Sub-Module, handle the task. The auction system also allows *creation of new auctioning model* by writing specifications and some code for that model.

▮ *Seller System Sub-Module*

The functions performed by this are:

- **Registration**: This involves storing (after validating) the registration data entered by the seller.
- **Login**: This involves validating the identity of the seller by matching his password from the registration information stored in the database.
- **Addition of auction**: This involves selecting the auction type and then supplying values for the parameters of the auction type selected.
- **Bidding**: There are some auction types in which even the seller is allowed to bid. In such auctions, the seller can enter the values for the parameters needed to make a bid. The bid is then sent to the central system.
- **View auction status**: Some parameters of the auction currently running, which are stored in the database, are displayed on selection of the auction.
- **View past results**: The results of the auctions are stored in the database as values of parameters. Some of these parameters are displayed when the seller wants to view results for auctions that he had added.
- **View messages**: This involves allowing the seller to view messages addressed to him and then deleting or saving these messages.

▮ *Buyer System Sub-Module*

This takes care of the actions of the buyer. The functions performed by the buyer system are:

- **Registration**: This is similar to registration of a seller.
- **Login**: This is similar to login of a seller.

- **Bidding**: The buyer specifies values for the bidding parameters for a particular running auction which need to be processed by the central system. The job of this sub-Module is to collect and send this data in an appropriate format to the central system.

- **View auction status**: A buyer can only view the values for the parameters made visible to the buyers.

- **View past results**: A buyer can only view the values for the parameters made visible to the buyers.

- **View messages**: This is similar to the message-box function in case of a seller.

5.1.3 Testing System Module

This is the system that handles the process of testing of auction strategies. The functions performed by this system are:

- **Creation of automatic bidders**: Some code can be added to the existing system to create automatic bidders.

- **Manage automatic bidders**: A number of automatic bidders simultaneously participate in a simulation of an auction. This module needs to allocate time to each of them from time-to-time.

- **Manage Simulation**: This involves starting a simulation, storing results and for each simulation computing the values of the parameters specified as functions.

- **Display Results**: This involves querying user for the parameters whose value they want displayed and then displaying their values.

5.1.4 Database

The functions of the database are:

- **Store all registration information**.
- **Store information for each auction type**.
- **Store information for each auction added**.
- **Store information about all the bids placed** in an auction that is currently being conducted.
- **Store results of auction** after the auction is completed.
- **Store simulation informations**.
- **Store all notification messages**.

5.2 Interaction Between Modules

All the modules interact with the database through *JDBC*, acting as a client to the SQL server of the database. In addition every module interacts with the Central System through *Client-Server* programs. The Seller System and the Buyer System need to inform the central system about the bids. For this purpose, the Central System opens a network connection and waits for bids. The Buyer System and the Seller System can send the bid by encoding it as a text and sending the text to the server waiting for bids on a fixed port. The automatic bidders similarly open a client connection to interact with the central system. The other modules or sub-Modules don't interact with each other.

5.3 Implementation Decisions

To make the system portable all parts are written in *Java*. Java servlets are used to handle web form requests. The GUI for the testing system is made in Java using *Swing*. The web pages and forms are all in plain *HTML*. The system has been developed on *Linux 2.0.36* platform. *Postgresql* database has been used as it is installed on linux machines and has a *JDBC* interface. However, the system can work on any other platform and with any other database. Only, the address of the database will have to be changed. Currently all servers and clients execute on one machine. However, that is not a restriction and the processes can be made to run on different machines. But, for that the IP addresses of the machines need to be changed in the code. A script to initialize the database has been written in *PERL* and a program in Java creates all the tables.

Chapter 6

Central System Module

The basic task of the Central System Module is to manage the different active auctions. This includes starting an auction, receiving and comparing bids for that auction, closing the auction and managing notifications about the auction. Each of these tasks is detailed in the following sections.

6.1 Start Of Auction

The auction starts at the time specified by the seller when listing his product for auction. So, this is not trigerred by any event. Hence detection of start of an auction must be decided by *polling* the database and looking for inactive auctions whose start time is before the present time. The polling rate determines the delay in the start of an auction. If the polling rate is high, it will hog the database resources making the system very slow, and if the polling rate is low, there will be visible delay in the start of auctions. In this system, the polling is done at every 30 seconds. A java thread manages this process.

Besides detecting the start of auctions, the central system is also responsible for activating the auction data in the database. This involves finding the auction type for that auction, changing the status of the auction and initializing the parameters required to hold parameters for the active auction. The *init()* method in each auction type class is responsible for carrying out this function. The central system just finds the auction type from the reference number of this auction and then calls the init() method of the appropriate auction type.

6.2 Bid Comparison

This involves receiving the bid, comparing this bid other bids (according to the rules for this auction type) and sending notifications (if needed). As the action to be taken is trigerred by an event (either a user sends a bid message through the web interface or an automatic bidder sends the bid message to the central system), no polling is required. There is a *client-server* protocol. The central system acts as a server and waits for a bid message on a socket. As this is implemented in java using pre-emptive threads, the resources are not continuously bogged. The client just sends the auction number and bid number of the bid (which helps in identification of bidder) along with the bid message to the server and the server is responsible for taking the appropriate action.

The central system, on receiving the bid, validates the bid, finds the correct auction type and calls the *bidCompare()* method of that auction type to perform bid comparison according to the rules of that particular auction type. Notifications can also be sent, within the method, to either the central system (acting as the auctioneer) or the bidder or the seller.

6.3 Close Auction

The closing time of an auction need not be a fixed predefined time. In many auctioning models, the closing time depends on the idle time (time for which no bid is received for that auction) or the quantity of objects left to be sold. Hence, it is not feasible to have an interrupt-driven implementation for this. From time-to-time, all active auctions need to be checked if they should be closed or not. The *polling* rate in this case is kept at once every 60 seconds. A java thread manages this process.

During the polling, all active auctions' auction type are detected and the corresponding *close()* method is called. This method checks whether that particular auction needs to be closed. This depends on the rules for that particular auction type. In case the auction is being closed, the status of the auction is changed and the results are stored in the database for further reference.

Chapter 7

Auction System Module

The Auction System Module consists of the **Buyer System Sub-Module**, the **Seller System Sub-Module** and the **Auction Packages** for each auction type. The two submodules consist of HTML files and java servlets to handle the requests sent through the web interface. The Auction Package is unique for each auctioning model and consists of a set of HTML, XML and java files. While some of the files in this package are automatically generated, the others need to be added for each auctioning model.

7.1 Buyer System Sub-Module

A list of the java servlets along with their functionalities is given below.

1. **BuyerRegister Servlet**: This handles the registration of the buyer. The registration data is stored in the database in case of correct data input and an error message is displayed in case of erroneous data input.

2. **BuyerLogin Servlet**: This gets as input the buyer login ID and the password and checks from the database for this buyer.

3. **BuyerSelectAuctionForViewingActiveStatus Servlet**: This selects the list of active auctions at this time from the database and displays it.

4. **BuyerBid Servlet**: This takes the values of the parameters needed for a buyer bid for the chosen auction and forms a buy bid and sends it to the Central System. The Central System returns either an acknowledgment or an error message. In either case, the result is displayed.

5. **BuyerActiveAuctionStatus Servlet**: This takes an active auction. Then, the status is displayed by calling the method for the appropriate auction type, as the method will be different for each auction type.

6. **BuyerViewPastResults Servlet**: This displays the result of a completed auction selected by the buyer.

An important thing to note here is that though the buyer's identity is not explicitly divulged to each servlet it is implied by the fact that the network session that he establishes with the central system has a unique ID which is stored during login and is later used when needed to find the identity of the buyer.

7.2 Seller System Sub-Module

A list of java servlets and their functionalities is given below.

1. **SellerRegister Servlet**: This handles the registration data of the seller. If the data does not have an error, the data is stored in the database. If there is an error, the error message is displayed.

2. **SellerLogin Servlet**: This gets as input the seller login ID and the password and checks from the database if a seller having this profile exists. If so, the login is successful and the welcome page for the seller is displayed. If not, the error message is displayed.

3. **SellerAddAuction Servlet**: This gets as input a set of values for parameters needed to add a new auction. This parses the input and calls the addauction() method of the appropriate auction type to add the auction. In case of unsuccessful addition, an error message is displayed.

4. **SellerSelectAuctionForViewingActiveStatus Servlet**: This takes the seller ID of the seller browsing the page as the input. Then it finds all the active auctions of this seller. This list of auctions is displayed to the seller for him to select one of the auctions from the list.

5. **SellerBid Servlet**: This takes the values of the parameters for seller bid and forms a bid and sends it to the central system. If the Central System replies that the bid has been processed, control is sent back with the message that the bid was successfully sent. If the Central System sends an error message to the servlet, the error message is displayed.

6. **SellerViewActiveAuctionStatus Servlet**: This takes as input an active auction. If the seller had added that auction, the values of the current parameters of the auction are derived from the database and displayed in a tabular form. The format of the output depends on the auction type. Hence, each auction type has a method that handles this task.

7. **SellerViewResults Servlet**: This asks the user to select an auction added by him and then displays results for that auction.

In this case also, the identity of the seller is maintained throughout the session after he logs in the system.

7.3 Auction Package

There is a package of files for each type of auction. The most important of these is the **XML specification file** for each auction. This file is used to automatically generate some of the other files in the package. This XML file follows the grammar of the DTD file mentioned in fig.13.

This specification file is used to generate the following files:

1. **AddAuction.html**: This is a form that is used to add an auction of this type. This specifies the parameters whose values need to be supplied for the addition of the auction. This is generated by extracting the list of auction

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT document (identity, body, rules)>
<!ELEMENT identity (name, version, authors, date)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT author EMPTY>
<!ATTLIST author name CDATA #REQUIRED address CDATA
#IMPLIED email CDATA #IMPLIED >
<!ELEMENT date (#PCDATA)>
<!ELEMENT body (auctionparameter, activeparameter,
resultparameter,sellbidparameter?, buybidparameter?)>
<!ELEMENT auctionparameter (parameter, globalconstraint*)>
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED datatype (INT |
NUMBER | SMALLCHAR | LARGECHAR | STRING) #REQUIRED
description CDATA #REQUIRED view (ALL|SELLER|BUYER|NONE)
#REQUIRED default CDATA #IMPLIED constraint CDATA #IMPLIED
>
<!ELEMENT globalconstraint (constraint+)>
<!ELEMENT constraint (#PCDATA)>
<!ELEMENT activeparameter (parameter*)>
<!ELEMENT resultparameter (parameter*)>
<!ELEMENT sellbidparameter (parameter*)>
<!ELEMENT buybidparameter (parameter*)>
<!ELEMENT rules (rule+)>
<!ELEMENT rule (#PCDATA)>
```

Figure 13: *DTD for Defining Auction Type*

parameters specified in the specification file and displaying them according to the data type of that parameter.

2. **SellerBid.html**: This is a form that is used to create a sell bid. The HTML form is constructed using the parameters defined as sellbidparameter in the specification file. The format of the form depends on the datatype.

3. **BuyerBid.html**: This is a form that is used to create a buy bid. The HTML form is constructed using the parameters defined as buybidparameters in the specification file. The format of the form depends on the datatype.

4. **<AuctionTypeName>.java**: This is a class that stores the parameters of an auction of this type and contains the java functions to manage the auction of this type. The methods included in this are:

   (a) *int addAuction(<parameter_list>)*: This will be called by the SellerAddAuction Servlet to add an auction of this type. The parameters are passed by the servlet. The method checks if the constraints are properly satisfied by the parameter values. If the constraints are satisfied, the auction reference number of the newly added auction is returned. If not, a negative number corresponding to an error code is returned.

   (b) *String viewOfActiveAuctionFromSeller(int <auction reference number>)*: This takes as parameter the auction reference number of an active auction and creates an HTML file that will be used to display the status of the auction to the seller who added the auction. The list of active parameters in the specification file have associated attributes to identify whether that parameter will be displayed to the seller or not. The data type of the parameters is used for formating.

   (c) *String viewOfActiveAuctionFromBuyer(int <auction reference number>, int <buyer reference number>)*: This takes as parameter the auction reference number of an active auction and the buyer reference number of a registered buyer and creates an HTML file that will be used to display the auction status for a buyer with the given buyer reference number. In the automatically generated code, the same information will be displayed to all buyers. In case some extra information pertaining to that buyer needs to be displayed the code for that should be added by editing this file. (eg. the latest bid of that buyer can be displayed.)

(d) *String viewOfResultOfPastAuctionFromSeller(int <auction reference number>)*: This takes as parameter the auction reference number of an auction and outputs an HTML file as a string to display the results of a past auction. The result parameters of the auction specified in the specification file are used to find which parameters to display and how.

(e) *String viewOfResultOfPastAuctionFromBuyer(int <auction reference number>)*: This takes the auction reference number as parameter and outputs the HTML file that should be displayed to show the result for that auction to a buyer. The result parameters of the auction specified in the specification file are used to find which parameters to display and how. There are attributes associated with the parameters to indicate whether it should be made visible to the buyer or the seller or both or none.

(f) *int setParameter(String <parameterName>, String <parameterValue>)*: The database does not have different tables for different auction types. For each auction, there are a set of columns of different data types in the table. Hence, a mapping of parameters to columns is needed. This mapping is automatically generated for each auction type using the specification file. The setParameter stores values of the auction parameter to the database.

(g) *String getParameter(String <parameterName>)*: This takes an auction parameter's name as input and extracts its value from the database. No changes should be made in the code of getParameter and setParameter methods. Both of these will be used a number of times by various functions.

Some additional code also needs to be added to handle the processing of bids, close of auction and initialization of auction. As this code cannot be generated automatically, the code for these methods need to be added by editing the <AuctionTypeName>.java file. These methods should use getParameter() and setParameter methods wherever there is a need to read or store auction parameters. These methods are:

1. *int init(int <auction reference number>)*: This changes the status of the auction reference passed to it as parameter to "Active" and set the initial values for active parameters as required. Some notification messages might also be added to the database.

2. *int bidComparison(Bid bid, int <auction reference number>)*: This checks if the bid is acceptable or not. If the bid is acceptable, it is added to the list of bids in the database. Then this bid is compared with the other bids in whichever manner necessary for this auction type. Some values of the parameters of the auction may be changed. Some notifications might also be sent.

3. *int closeAuction(int <auction reference number>)*: This returns a negative value if the auction need not be closed. If, however, the auction needs to be closed, it will change the status of this auction to "Closed" and store the result parameters. Some notifications will be sent.

The <**AuctionTypeName**> class will be a subclass of the **Auction** class. This Auction class contains some of the code that are generic for all auction types and, hence, need not be generated each time.

The complete process of addition of a new Auction Type consists of *writing an XML specification file*, using AutoGenerator class to *automatically generate some of the files* and lastly *modifying the <AuctionTypeName>.java file* to add some extra functions.

Chapter 8

Testing System Module

The Testing System Module allows the user to test auctioning strategies by creating bidding agents and then using them in simulations. This chapter describes *how an automatic bidding agent is created* and *how the agents can be used in simulations* to test auctioning strategies.

8.1 Automatic Bidding Agent

An automatic bidding agent for a particular auctioning module is a process thread that polls the database at regular intervals and depending on the current status of the auction makes a decision to bid. This agent has some parameters whose values are supplied when the agent is created.These parameters govern the action of the agents. An example of a bidding agent for English Auction is a 'Fixed Range Buyer'. This buyer has three parameters: Maximum bidding price, Minimum bidding price and the Increment value. Whenever the current winning price of the product being auctioned falls between the minimum bidding price and the maximum bidding price less the increment, the buyer makes a new bid which is greater than the current winning price by the increment value.

The steps involved in the creation of an automatic bidding agent are:

1. *Create a <AuctionTypeName><BidderName> class* that inherits the Buyer or Seller class depending on whether it is a buyer or a seller. In the case of

buyer, it has a login ID which is the sequence number of this buyer for this test run. In case of seller, the login ID is 'auto'. This class will contain the names of the parameters. The *run()* method in this class creates a bid and then calls the *bid()* method of its super class to send the bid to the central system. The decision on whether to bid and what to bid is based on the values of the parameters passed to this class through a GUI by the user and the auction status variables stored in the database.Some example bidding agents have been created in this system and the task of creating a new one requires just modifying some of the lines of code.

2. *Create a <AuctionTypeName><BidderName>Frame class* that is a GUI frame that consists of textfield components to get the values for the parameters for this bidder.Some example frames are created and new ones can be created by slight modifications.

3. *Update AddBuyer or AddSeller class*, as required, to include the name of this bidder in the list of bidders for this auction type.

8.2 Test Run

In a test run, a number of simulations of auctions are performed simultaneously and the user has the flexibility to specify the parameters as functions of simulation sequence number. The test run is implemented as a class having an array of automatic buyers, possibly an automatic seller, information about the auction being added and information about the test run itself (which consists of the number of simulations, the test run addition time, the name of the test run and the test run close time). This class is implemented as a *java thread* and when the thread is started, it starts adding all the auctions sequentially. A time gap of one minute is maintained between the additions and the initialization of bidders to allow the start of auction detection thread of the central system enough time to start the auction, i.e make the auction active, before the bidders start the process of bidding.

Before the auctions are added, the values for the parameters must be calculated by calculating the values of the functions for this specific simulation no. Take for example the function: N*N-3*N+2 for the quantities of product being auction in each simulation N. So, when the simulation number is 0, this function is evaluated to 2, For simulation number 1, the quantity of product is evaluated to 0. When simulation number is 2, the quantity of product is evaluated to 0. Suppose the number of simulations is specified to be 10. Then, for the last simulation having simulation sequence number 9, the quantity of product is evaluated to 56. Currently, the function is restricted to be polynomial function of N. In this simplistic case, the evaluation is done by replacing N by the sequence number and evaluating the arithmetic expression.

Chapter 9

Database

This chapter discusses the *database design* and the *table structure* of the tables used in the database.

9.1 Database Design

The basic purpose of the database is to *store information of the users* and the *information on the auctions and simulations*. The information of the two types of users are stored in the two tables *buyer_info* and *seller_info*. Additionally, the information on sessionID and user needs to be stored for authentication of the user. This information is stored in *session_info*. For the auctions, the auctioning parameters like quantity, name of product, type of auction, etc. need to be stored. This is stored in the *auctions* table. In addition, when the auction is in progress, the current parameters of the auction need to be stored, both for display and for calculation of results. This is stored in *active_auctions* table. All bids made for this auction are also stored. This is stored in *bid_info* table. When the auction is finally closed, the result of the auction also needs to be stored. The results are stored in the *auction_results* table. The rules and description about the various auctioning models supported by this auction system are stored in the *auction_types* table. The *messages* table is used to keep all messages. The *testrun* table stores the values of the test run which are later used to display the results when the test run is completed.

9.2 Database Tables

- **Session_Info**: *authenticates a network session for a user*
  - SessionId - Char(32) not null
  - Type - Char(1) not null ('S'/'B' to identity buyer or seller)
  - Login - Char(16) not null
  - Primary key: SessionId
- **Buyer_Info**: *keeps buyer information*
  - Login - Char(16) not null
  - Password - Char(16) not null
  - Name - Char(32) not null
  - MailAddress - VarChar(256) not null
  - PermanentAddress - VarChar(256) not null
  - EmailAddress - VarChar(256) not null
  - Primary Key: Login
- **Seller_Info**: *keeps seller information*
  - Login - Char(16) not null
  - Password - Char(16) not null
  - Name - Char(32) not null
  - MailAddress - VarChar(256) not null
  - PermanentAddress - VarChar(256) not null
  - EmailAddress - VarChar(256) not null
  - Primary Key: Login
- **Auction_Types**: *keeps information on the supported auction types*
  - TypeNo - Integer not null

- TypeName - Char(256) not null
- RuleDescription - VarChar(1024) not null
- SellerBidAllowed - Char(4) not null ('YES'/'NO')
- BuyerBidAllowed - Char(4) not null ('YES'/'NO')
- Primary Key: TypeNo

- **Auctions**: *keeps information of all auctions which are added*
  - AuctionNo - Integer not null
  - TypeNo - Integer not null (Foreign key of Auction_Types)
  - SellerLogin - Char(16) not null
  - Status - Char(8) ('INFANT'/'ADDED'/'ACTIVE'/'CLOSED')
  - ProductName - Char(32) not null
  - ProductDescription - VarChar(256) not null
  - ProductQuantityUnit - Char(32) not null ('UNITS'/'KILOMETERS'...)
  - ProductQuantity - Integer not null
  - Currency - Char(32)
  - StartTime - Timestamp not null
  - Param1-5 - Integer
  - Param6-10 - Float
  - Param11-15 - Char(32)
  - Param16-20 - VarChar(256)
  - Param21-25 - VarChar(1024)
  - Param26-39 - Timestamp
  - Primary Key: AuctionNo

- **Bid_Info**: *stores bid information*
  - AuctionNo - Integer not null (Foreign Key of Auctions)

- BidNo - Integer not null
- BidderType - Char(1) not null (S'/'B')
- Bidder - Char(16) not null
- BidStatus - Char(8) not null
- BidValue - VarChar(256) not null
- Primary Key: (AuctionNo,BidNo)

- **Active_Auctions**: *information on current auctions*
  - AuctionNo - Integer not null (Foreign Key of Auctions)
  - Param1-5 - Integer
  - Param6-10 - Float
  - Param11-15 - Char(32)
  - Param16-20 - VarChar(256)
  - Param21-25 - VarChar(1024)
  - Param26-30 - Timestamp
  - Primary Key: AuctionNo
- **Auction_Results**: *information on results of auctions*
  - AuctionNo - Integer not null (Foreign Key of Auctions)
  - Param1-5 - Integer not null
  - Param 6-10 - Float
  - Param 11-15 - Char(32)
  - Param 16-20 - VarChar(256)
  - Param 21-25 - VarChar(1024)
  - Param 26-30 - Timestamp
  - PrimaryKey: AuctionNo

- **Messages**: *stores all message information*
  - MessageNo - Integer not null
  - SenderType - Char(1) not null ('C'/'B'/'S')
  - SenderLogin - Char(16) not null
  - RecipientType - Char(1) not null ('C'/'B'/'S')
  - RecipientLogin - Char(16) not null
  - SendingTime - Timestamp not null
  - Status - Char(1) not null ('N'/'R'/'D')
  - Subject - VarChar(256) not null
  - Body1 - VarChar(4096) not null
  - Body2 - varChar(4096)
  - Primary Key: MessageNo
- **TestRun**: *stores test run info*
  - RunNo - Integer not null
  - Name - Char(32) not null
  - AuctionTypeName - Char(32) not null
  - NumOfBuyers - Integer not null
  - StartAuctionNo - Integer not null
  - StartTime - Timestamp not null
  - Primary Key: RunNo

Chapter 10

Conclusion

Almost all the objectives of the project have been achieved. The software is ready for use by a developer for testing of auction strategies. However, a few minor parts of the software could not be implemented due to insufficient time. Perhaps the most important of the left parts is the *notification system.* I hope that somebody completes this task. I have had some experience in the implementation of the notification system in an auction system and I assure anybody who completes this task that it will be an interesting and learning experience, taking into consideration the vast possibilities of the decision support system which can be incorporated into the system. Moreover, the testing system could not be tested completely as it was completed in the last month and since a typical set of auctions will run for a long duration, it requires some dedicated time. The testing system can be further developed by allowing the user to enter more general functions.

This project, being the first of its kind, has immense future research scope. The future work can be in the way of *using this software to test auction strategies* and developing some optimal strategies for some particular casees. Actual experiments can be carried out for theoretical algorithms to prove or disprove their utility in face of an actual real-life auction. The work of developing algorithms for auctions is a challenging and growing field and this tool could prove to be a big help in this field. The other direction of future work could be in the *development of an Internet Auction System* starting from this software. For that purpose, the payment procedure needs to be added to this basic framework. The new Internet Auction System will have the property that adding a new auction type to the system would be a matter of hours and would require no database additions or change in design. Even the Auction System does not need to be stopped for this change. Yet another direction of future work can be in the *development of this software* such that the it can cover more auctioning models. This means allowing more than one seller to sell identical products in a single auction and allowing more than one products being sold by a seller in a single auction and a combination of both. This would probably require some change in design, but would be a challenging task.

Bibliography

[1] *http://auctions.yahoo.com* Yahoo! Auctions Web Page.

[2] *http://www.ebay.com* The eBay Web Page.

[3] Kelly T. *Secure Anonymous Auctions Without Trusted Parties* Proc. IEEE Symposium on Security and Privacy, 1999.

[4] Kikuchi h., Harkavy M. and Tygar J.D. *Multi-round Anonymous Auction Protocols* Proc. First IEEE Workshop on Dependable and Real-Time E-Commerce Systems, 1998.

[5] Peng C.S., Pulido J.M., Lin K.J. and Blough D.M. *The Design of an Internet-based Real-Time Auction System* Proc. First IEEE Workshop on Dependable and Real-Time E-Commerce Systems, 1998.

[6] Wellman M.P. and Wurman P.R. *Real Time Issues for Internet Auctions* Proc. First IEEE Workshop on Dependable and Real-Time E-Commerce Systems, 1998.

[7] Wurman P.R. and Walsh W.E. *Flexible Double Auctions For Electronic Commerce: Theory and Implementation* Proc. Decision Support Systems, 1998.