INTELLIBID: AN EVENT-TRIGGER-RULE-BASED AUCTION SYSTEM OVER THE INTERNET

By

KUSHAL THAKORE

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002

Dedicated to:
My family

ACKNOWLEDGMENTS

I would like to take this opportunity to extend my sincere gratitude to Dr. Stanley Y.W. Su, chairman of my supervisory committee, for his continuous guidance and support during my thesis work and study. I shall be ever grateful to him for providing me with a challenging topic for my thesis and for his continuous feedback during the implementation and thesis writing. I also would like to thank Dr. Joachim Hammer and Dr. Douglas D. Dankel II for serving on my supervisory committee.

I would also like to thank Nicky Joshi, Seokwon Yang and Rakesh Lodha for their help during the implementation phase of my thesis. I would like to extend a sincere vote of thanks to Sharon Grant for her perennial and cheerful assistance.

Lastly but most importantly I would like to thank my family and friends for all the support and love without which I would not have been at this University in the first place.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

INTELLIBID: AN EVENT-TRIGGER-RULE-BASED AUCTION SYSTEM OVER THE INTERNET

By

Kushal Thakore

August 2002

Chair: Dr. Stanley Y. W. Su
Major Department: Computer and Information Science and Engineering

This thesis presents an advanced version of an Event-Trigger-Rule-Based auction system called IntelliBid. Nicky Joshi reported an earlier version of this auction system. IntelliBid is made up of a network of Knowledge Web Servers, each of which consists of a Web server, an ETR Server, an Event Engine, a Knowledge Profile Manager, a Metadata Manager, and Bid Servers and their proxies. IntelliBid provides full online auctioning functionality to the creator of an auction site, and the bidders and suppliers of various products. It offers a number of advantages over the existing Internet-based auction systems. IntelliBid offers flexibility to bidders for defining their own rules to control their bids in an automatic bidding process. By using different rules, the bidders can apply different bidding strategies. It also furnishes valuable statistical information about past auctions to both suppliers (or sellers) and bidders, which can assist a bidder in bidding and a seller in setting a reasonable base price and/or minimum incremental price. Moreover, since the rules controlling automatic bidding are installed and processed by

the ETR servers installed at bidders' individual sites, bidders' privacy and security are protected. IntelliBid has event, event filtering and event notification mechanisms that keep both bidders and suppliers better and timely informed of auction events so that they or their software systems can take the necessary actions in the auction process. Furthermore, any registered user of IntelliBid, bidder or supplier, can monitor the bids placed to any product being auctioned in IntelliBid. In IntelliBid, a bidder can place a bid either manually or can have a rule to do the bidding on the user's behalf. It also allows a bidder to participate in several auctions at the same time, in both manual and automated modes. The bidding of a product can depend on the result of the bidding of another product. Last, but not least, IntelliBid allows a user to play both the role of bidder and the role of supplier simultaneously as information about both are stored separately in the Profile Manager.

IntelliBid's architecture uses a parallel event management system to do event registration and notification. This thesis entails a detailed performance study on the implications of using such a system by considering the time taken to send out event notifications whenever an event is posted.

CHAPTER 1
INTRODUCTION

According to McAfee and McMillan [MCA87, p. 701], "auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants." Auctions have been present in our society since the dawn of civilization. Great changes have been observed in the procedures used to conduct auctions since then, and now Internet-based online auctions have become the most popular form of auction. The auctioneer usually starts the auction with an initial or base price, and then bidders submit their own bids in response to the initial price or submit bids on behalf of other bidders. The auction ends according to some established rules which makes an auction fundamentally a form of negotiation, in which a fixed auction protocol is followed, e.g., English auction, Dutch auction, Vickrey auction, etc. [KUM98, BEA96, JOS01].

Aucnet, a Japanese company, started the first online auction web site in the year 1995 closely followed by Onsale [ONS01] and eBay [EBA01] who started their respective web sites in May 1995 and in September 1995, respectively. Today eBay is considered to be the online auction leader, offering more than 4 million products daily [AUC01b].

Since 1996, an upward growth has been observed in the arena of online auctions, boosted by a positive market response, as well as the addition of Internet heavyweights like Yahoo [YAH01] and Amazon [AMA01]. Currently there are over 400 sites offering online auctions, providing buyers and sellers with a multitude of choices. A survey of

online auctions conducted by National Consumer League reveals that nearly one third of adults in the United States who go online have participated in an online auction—an estimated 35.6 million people [NAT01]. Observing such growth, analysts predict that by the end of 2002, the online auction market will be a thriving $15.5 billion industry [AUC01a].

Due to the tremendous popularity of online auctions, the online auction web sites continually add features and improvements but despite of all the face-lifts, they still have some limitations. One of the most critical limitations is the bidding process. Currently, to raise his/her bid, the bidder has to access the web site, find the product in question, and then register the desired bid. At most, the bidder can select a process on the auction site, which will mechanically raise the bid, limited by a fixed increment. There is no flexibility, privacy, or security in this process. Many auction sites do not even give its user the option to be notified when a particular product of interest is registered with their sites. Also, most sites do not allow users to observe the history of bidding or bidding statistics for a product in any form.

The rate of growth of Internet-based auctions, and the limitations observed in the existing auction sites have motivated our work on an event-trigger-rule-based auction system called IntelliBid. This system is implemented using the infrastructure provided by an Internet-based Knowledge Network (IKnet) developed at the Database Systems Research and Development Center of the University of Florida [LEE00, SU00, LEE01]. The IKnet infrastructure provides various services like the specification, subscription, filtering, and notification of events that are relevant to Internet auctions, the management and enforcement of rules that implement different bidding strategies, the management of

triggers that tie events to rules, and load-balancing the event servers, which can prove to be critical in real life Internet auctions. IntelliBid is formed by a network of replicated servers, which can conduct automated bidding on behalf of human bidders who want to participate in Internet-based English and/or Dutch auctions. It offers greater flexibility in applying different bidding strategies and a better protection of privacy and security in an automated auction process than the current auction sites [JOS01]. It also furnishes users with useful and much needed statistics of previously completed auctions. Furthermore, it allows bidders to participate in auctions for multiple products at the same time when the result of one auction may affect the action taken for the other. The auction site also offers a convenient feature for bidders as well as suppliers to obtain the bid history of products whose auctions have just concluded. IntelliBid's event registration and notification mechanisms are built on a parallel architecture of event servers developed by Agus Hendro [HEN01]. Event subscription information is distributed among these parallel servers, which carry out the event notification task in parallel. For example, when an OutBid event occurs, these servers can efficiently send out the event notifications to the bidders (via emails) to allow the bidders or their systems to promptly respond to the event.

This work is a continuation and extension of the work previously done by Nicky Joshi [JOS01] and Agus S. G. Hendro [HEN01]. The features that existed in the systems they developed are listed below:

- The basic auction mechanism–user registration (bidder/supplier), product registration, bid registration, and bid history.
- English Auction mechanism–The auctions allowed are typically single quantity auctions where the topmost bidder, at the end of auction, wins the auction.

- Events
  - AuctionProduct–This event is triggered whenever a new product is registered with IntelliBid. Notifications are sent to its subscribers about the new product.
  - OutBid–This event is triggered whenever any bidder is outbid. A bidder may subscribe to this event if he/she wishes to associate his/her own bidding rule to it.
  - AuctionClose–This event is triggered when any product's auction comes to an end. It then triggers the AuctionHistory rule and sends the product's auction history to all of its subscribers.
- Rules
  - IncreaseMyBid–This is a sample rule that can be associated with the OutBid event and be used to automatically increase a bid, thus providing an automatic bidding mechanism.
  - AuctionHistory–This rule is triggered by the AuctionClose event to send out the auction's bid history to its subscribers.
- Parallel Architecture of Event Servers for event subscription and notification–Only one Event Server was utilized under a Distributor for IntelliBid, which restricted the parallel processing of event subscriptions and event notifications.

OLD VERSION NEW VERSION

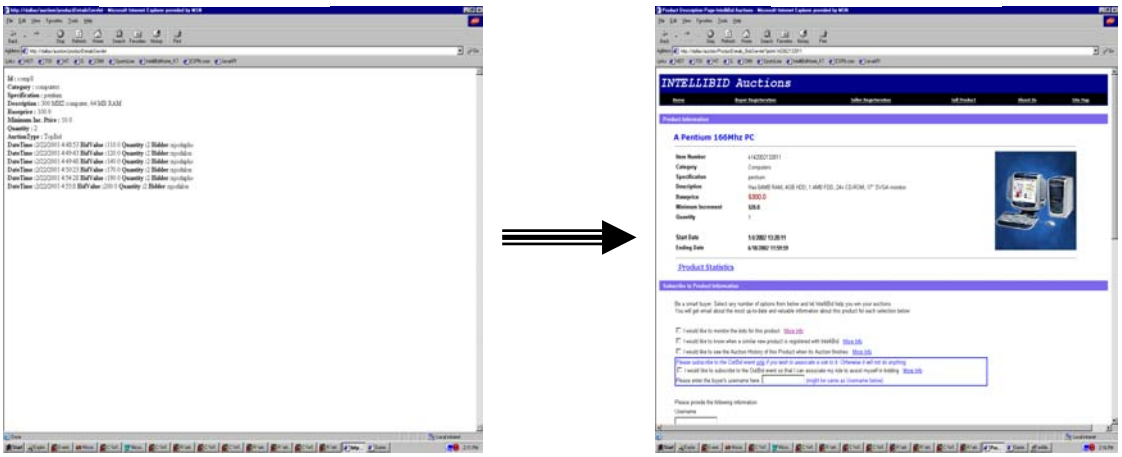

Figure 1.1 IntelliBid Web Site Transformation Example

The aim of this thesis, right from the onset, was not only to improve on what already existed but also to further enhance the system to better depict the features and use

of the IKnet framework. Keeping this in mind, some of the main additions and/or changes made to the system are listed below:

- Professional web sites look and feel with dynamic HTML, JavaScript and radiant colors. Figure 1.1 illustrates the change in the look of the IntelliBid web site.

- A Product table is introduced to replace the old AuctionProduct table to store product information. Product's auction starting date, an optional picture's URL, product's title, and most importantly a unique productid generated for each product have been included in the Product table over and above the fields of the original AuctionProduct table.

- IntelliBid now supports English and Dutch auction mechanisms. Dutch auctions are multiple quantity auctions where all the winners of the auction pay the same price, which is the lowest of all the winning bids.

- Events

    - Product–The old AuctionProduct event has been replaced by the Product event to accommodate the new Product table.

    - MonitorBids–A new MonitorBids event has been added which informs its subscribers whenever a new bid is placed for a particular product of interest.

- Rules

    - BidInfo_KT–This rule is triggered whenever the MonitorBids event is posted to gather information about the new bid and notify the subscribers of the event.

- Web site administration side–A whole new separate administration site has been added to assist IntelliBid's administrators to not only add a new Category and/or specification, but also to add a new administrator. Different levels of clearance can be assigned to different administrators.

- More dynamic data–Most of the data seen on the IntelliBid site now comes from the database. All the categories and its related specifications are also stored in the database and so as soon as a new category and/or specification is added to the site, it will show up right away.

- Product statistics–Any user on IntelliBid can view various statistics related to a specific product whose auction has ended. Such statistics can assist a bidder in specifying a bid and a supplier in specifying a base price and/or a minimum incremental price.

- Parallel Architecture of Event Servers for event subscription and notification–The architecture developed reported in [HEN01] has been modified in order to optimize the overall parallel processing of event subscriptions and event notifications and also to incorporate as many as three Event Servers under a single Distributor for IntelliBid. A detailed performance study related to this will be covered in Chapter 6.

The remainder of the thesis is organized as follows. In Chapter 2, the auction process and limitations of current online auction sites are presented. Chapter 3 describes the knowledge model used in the Web-based Knowledge Network and the architecture of IntelliBid. Chapter 4 describes the various business rules followed by IntelliBid Auctions along with a detailed explanation of English and Dutch auctions. It also includes the steps followed for the creation and execution of an auction in IntelliBid. Chapter 5 describes some implementation details of IntelliBid. Chapter 6 analyzes the performance gain obtained by using a parallel architecture of event servers for managing event subscription information and for performing event notification. Finally, Chapter 7 summarizes the work, and gives some suggestions for future work.

CHAPTER 2
PROCESS AND LIMITATIONS OF THE EXISTING INTERNET-BASED AUCTION SITES

This chapter explains the process and method used in the existing Internet-based auction sites, limitations observed in them, and the motivation behind this thesis.

2.1 Process of Internet-based Auctions

Whether an auction is conducted live or online, the key ingredient remains the same: eager people gathering to bid. However, the auction also has often sounded a keynote of any society--what do we find valuable? This holds true for the citizen of Ancient Greece just as it does for any eBay user.

2.1.1 Finding Products of Interest to a Particular Bidder at Auction Sites

By far, the most common procedure of actually finding a product that a person is interested in on an auction site involves the tedious process of visiting one or more of the popular auction sites. Few sites do allow for an email-based notification when a certain category of products is up for auction, but the category is generally very broadly based, such as “computers,” or “books and magazines.”

2.1.2 Actual Process of Auctions

In the following description of the auction process, we shall only consider the most popular form of auction today – namely, the English auction.

Users who wish to participate in auctions must first register with an auction site either as a seller or a bidder or both. Previously, auction sites used to provide all the products for auction to the bidders. However, now all sites have accepted an open policy

of having users supply products for auctioning. In this scenario, a registered supplier, along with a minimum incremental amount, specifies a minimum base price for each product he/she wishes to put on auction. Each bid that is placed by a registered bidder must be at least the minimum base price, plus the incremental amount. Up to a certain time limit, which is specified by the supplier, the product is open for auction and each registered user can keep on raising the bid. Each bid is considered by the system and accordingly the price of the product goes higher and higher. The system rejects bids that are not of a sufficient amount. Also, the bid amount is validated against a provided credit card, and only if the validation is successful, is the bid accepted. The latest bid information is displayed along the side of the product in real time on the auction site. For the latest information, the suppliers and buyers have to access the auction site. At the end of the time limit, the product is sold to the person with the highest bid and this person's credit card is charged. At this stage, the supplier is notified of the winning bidder and becomes responsible for shipping the product to the new owner [JOS01]. The above mentioned auction mechanism is called the conventional auctioning mechanism or the English Auction. All the major auction web sites follow this English Auction mechanism.

2.1.3 Outbid Notification

The crudest form of notification is for bidders to visit the web site every so often until the time limit of the product they are bidding for is expired, in order to verify if they have been outbid. This outbid information is displayed either in the form of only the top bid for a product, or in the form of the history of prices that have been bid for the product. Most sites provide e-mail notification when the bidders have been outbid so that they can decide if they want to raise their bids; however, with the email notification, they

have to be present to read the e-mails and then visit the web site to personally raise their bids.

A form of automatic bidding exists on some of the prominent auction sites; this works as follows:

When a bidder places his/her first bid, he/she may choose an option on the auction site to do automatic bidding for him/her. He/she is allowed to specify the maximum amount that he/she is willing to bid. Hence, when he/she is outbid, the program will automatically raise his/her bid, so that his/her bid will remain at the top. This process continues until the maximum specified price limit has been reached. For example, suppose a product is currently bidding at $100, with a minimum incremental bid of $5. Hence, a bidder can specify a maximum price limit of, say, $200 to the program. The program will not automatically raise his/her bid to $200, but rather to $105. Now, suppose another bidder raises the bid to $115. The program will automatically raise the bid of the first bidder to $120. This process will continue until the bidder's limit of $200 has been reached. One can see the main disadvantages of this scheme. There is no control over the rate of increase. The option to select such a process is on the auction site itself, leading to a possible breach of privacy. Also, once such a process is started, there is no option of stopping it [JOS01].

2.1.4 Winning Bid Notification

In the case of winning the auction on a particular product, a final notification is sent to the winning bidder in the form of an email. This is only to inform the bidder of this event, as the specified credit card number has already been charged. Whether the auction site or the supplier is hosting the product, the product will be shipped to the winning bidder.

2.2 Limitations of Current Auction Sites

On observing the process of the existing Internet-based auctions, several limitations and/or opportunities for improvements are easily seen.

2.1.1 Selection of Desired Products for Users

Most of the major auction web sites list the products on auction under broad categories. This makes the task of finding the desired product difficult for a user. Product specification should not be limited to merely a simple category of products. In our work, we provide the users with the flexibility of specifying or selecting a specification related to a category. Furthermore, the event publishing, and notification schemes of IntelliBid's architecture are used to inform users the availability of products of their interests.

2.2.2 Greater Flexibility and Privacy in Specifying Bids

The automatic bid increment mechanisms provided by most auction web sites do not allow bidders the freedom to vary the increments in a bidding process. For example, a bidder may want the system to automatically increase his/her bid by a larger amount than the pre-specified increment, if he/she would really like to have the product, and the other bidders are quite aggressive in their bids. The increments can be controlled by a sophisticated rule that takes into consideration many factors and uses a complex function to calculate the increments. In our work, different techniques or strategies of bidding can be expressed by rules and be carried out by an Event-Trigger-Rule (ETR) Server. Rules that guide the automatic bidding are private information, which are kept and used by the ETR Server installed at the bidder's site, instead of the auction site.

2.2.3 More than One Product

Currently, each product that is being bid for is mutually exclusive of every other product. However, it may be the case where someone sees two or more products that

he/she is interested in, but needs only one of them. For example, suppose that two computers are for sale, one 550 MHz Pentium III computer, and another 550 MHz AMD computer. The user's first preference is the 550 MHz Pentium Computer, but the bidder is willing only to spend $850 for it. In case of being outbid for the Pentium, the bidder's bid should be automatically placed for the AMD computer. As cancellation of a bid once placed is not allowed at any site, the bid for the second computer should only be placed when the bid for the first computer has been outbid. This can be achieved in our work by using bidder-side rules, which are processed by a rule server installed at the bidder's site.

2.2.4 Notification to More than One Person

It is possible that the product being purchased on the auction is for somebody else. In that case, if the bidder has secured the top bid and has been notified, the notification should be forwarded to the person whom the bidder represents. It is also possible that a bidder would like some other person to receive a notification when a particular product is put on auction. All this can be done by the notification mechanism provided by our system.

2.2.5 History of Bidding

The history of bidding of a product has gained importance over the years, due to the large increase in Internet-based auctions. This bidding history is valuable information for future auctions. The bidding history of a product usually includes the dates and times when bids were placed for the specific product, and the values of those bids. Both bidders and sellers can utilize this history. A bidder can observe the history of a specific product similar to the one that he/she is interested in, and may decide the start value of bidding, the rate of increment in bidding, and the maximal bid based on the historical information. For a supplier, this information can be of importance the next time he/she wishes to

supply a similar product for auction. The historical information can help him/her decide what the base price and/or the minimum incremental price of the product should be. In spite of the importance of the history of bidding, existing auction sites have neglected this feature. In our work, we use the auction side rule facility of the ETR Server to implement bid histories [JOS01].

2.3.6 Product Auction Statistics

A product's auction statistics is just as important as its history of bidding. Currently none of the online auction sites give any kind of statistical information about a product's auction which might include, the average of the base prices of previously completed auctions of similar products or the average of the winning bids of previously completed auctions of similar products or the average of all the minimum incremental values specified by sellers of already auctioned products. IntelliBid provides the facility of viewing such product auction statistics in the form of horizontal histograms thereby assisting the bidders in specifying appropriate bids as well as the sellers in setting the base price and the minimum incremental price for their products.

2.3 Survey of Existing Software Agent-based Auction Sites

The use of software agents for the process of online auctions is still primarily in the research stage. There are a few existing auction systems available today that provide a form of agent-based auctions. Online auction giant eBay does provide a "proxy bidding" mechanism, which can be viewed as a software agent that assists a user in automatically increasing his/her bids in a particular fashion till his/her maximum bid amount limit is met [EBA01]. The most prominent of agent supporting auction sites is AuctionBot [WUR98].

AuctionBot is an experimental multi-purpose Internet auction server developed at the University of Michigan. AuctionBot is a configurable, flexible and scalable server that supports both software and human agents in auctions. For creating software agents, AuctionBot provides code libraries, which are downloadable by bidders. These code libraries provide an API useful for implementing the low-level communication details in the software agents for communicating with the AuctionBot Server. However, the actual creation of the software agents to implement the bidder's policies is the responsibility of the bidder. These agents have to be implemented by the bidder in programming languages (C++, Java or Mathematica) using AuctionBot's API. Even if the bidder has access to the programming software but does not have the knowledge of these programming languages, he/she will not be able to take advantage of the software agent approach offered by AuctionBot. Moreover, AuctionBot is based on the pull model, rather than the more efficient push model [JOS01].

The Fishmarket project [FIS01] and MAGMA [TSV97] are also examples of agent-mediated electronic auction houses. Fishmarket is a project concerned with communicational aspects of multiagent systems. It attempts to map the traditional fish market industry, and uses the Dutch auction mechanism. Similar to AuctionBot, it allows the participation of both human as well as software agents in the auction. Fishmarket provides a downloadable library of agent templates written in Java, C and Lisp to assist programmers in building their agents. This allows the bidders to concentrate their efforts on developing auction strategies. Fishmarket also provides a built-in agent builder facility for automatic generation of agents. These agents can be customized to use different auction strategies also [JOS01].

MAGMA (**M**innesota **AG**ent **M**arketplace **A**rchitecture) is architecture for an agent-based virtual market that includes all elements required for simulating a real market. It is designed so that agents can trade several kinds of goods using the Vickrey auction mechanism. MAGMA attempts to create the concepts observed in a real market like banking, advertising, transfer and storage of goods, etc. For this purpose the system architecture consists of components like an advertising server, a bank, a relay server and trader agents. All the components communicate through the relay server, which acts as a central hub. The relay server is implemented in Allegro Common Lisp and the trading agents are written in Java.

However, the main aims of the Fishmarket project and MAGMA are the communicational aspects of multi-agent systems. They use the auction protocol to study different techniques for allowing agents to communicate with each other. IntelliBid's focus is more on implementing an auction service over the Internet using events, triggers and rules [JOS01]. This can also be viewed as a software agent doing work for its subscribers. The subscribers might themselves be the creators of such software agents defined at a high-level by events, triggers and rules.

CHAPTER 3
KNOWLEDGE MODEL AND ARCHITECTURE OF THE INTELLIBID AUCTION SYSTEM

This chapter describes a knowledge model and the architecture of an event-trigger-rule-based auction system named IntelliBid. The knowledge model is used for the specification and implementation of the Web-based Knowledge Network, which provides the information infrastructure and services needed to implement IntelliBid.

3.1 Knowledge Model

The knowledge model used in the development of the Internet-based knowledge network is an active object model (AOM). In this model, all things of interest are modeled as active objects. In addition to the traditional way of defining an object class in terms of attributes (or properties) and methods, AOM allows the inclusion of events, triggers, and rules in an object class definition.

3.1.1 Events

Generally speaking, events are things of interest that have happened or can happen on the Internet. For example, new data is made available on a site, a Web page has been modified, an application system is about to be invoked, a user strikes a key on the keyboard, a signal is received from an external device, etc. Users or software systems can subscribe to events and be notified when the events occur. Events can have parameters (i.e., parameterized events), which are data associated with the events that are to be transmitted in event notifications. In the context of an auction, events can be used to represent things or points of interest during the auctioning process. For example, a bidder

would like to be notified when a product of interest is registered with the auction site so that he/she can participate in the auction of that particular product. The bidder can specify a particular product of interest by providing values to some attributes, which are provided in an event filter template. Similarly, the bidder would be interested in being notified when another bidder has outbid his/her bid in a bidding process. Another point of interest would be when the auction on a particular product has come to an end [JOS01]. A bidder and/or a supplier might even be interested in monitoring all the bids being placed for a particular product. Table 3.1 lists the events implemented in IntelliBid, their parameters, and the attributes associated with event filters.

Table 3.1 Implemented Events in IntelliBid

Event Name	Parameters	Parameter Type	Filter	Description
Product	ProductId SupplierId ProductTitle ProductCategory ProductSpecification ProductDescription PictureUrl BasePrice MinIncrementalPrice StartingDate EndingPeriod Quantity AuctionType Status	String String String String String String String FLOAT Float String String Integer String String	On ProductSpecification and ProductCategory	Event posted when a product is registered with IntelliBid
OutBid	ProductId BidderId AuctionType BidValue MinIncrementalPrice	String String String Float Float	On ProductId and BidderId	Event posted when a bidder is outbid on a product
MonitorBids	ProductId	String	On ProductId	Event posted when a new bid is placed for a product
AuctionClose	ProductId	String	On ProductId	Event posted when a product's auction time limit has expired.

3.1.2 Rules

In the knowledge model, rules are Condition-Action-Altaction (CAA) rules, which can be used to define business constraints, policies, regulations, integrity, and security constraints. Each CAA rule represents a small granule of control and logic. A number of related rules can form a rule structure to express a larger granule of control

and logic for modeling a more complex policy, regulation, etc. A rule can participate in multiple rule structures, thus, making each rule reusable. A rule has a rule name and can have parameters. When the rule is invoked upon the occurrence of some event, it evaluates the CONDITION clause of the rule. If the result is true, the operations specified in the ACTION clause are executed. Otherwise, the operations specified in the ALTACTION clause are executed [JOS01].

A rule also has a RULEVAR clause, which allows variables to be defined in a rule. The variables can be persistent or temporary. It is also possible to define customizable rule variables, which can be assigned different values for different users, making the rule a "parameterized rule."

Different from the Event-Condition-Action rules used in some active database management systems [DAY88, STO88, CHA94, SU97, WID96], events and rules in the knowledge network are separately defined by users or business organizations; and events are tied to rules by trigger specifications (to be described next). This separation is important in a distributed environment in which software systems are loosely coupled. In an automated auction system, an auction site may define and post events, which are subscribed by many bidders who may define different rules for implementing different bidding strategies in a bidding process. These rules are managed and used by event and rule servers installed at the bidders' sites, thus bidders' privacy will not be compromised. Rules can also be defined and processed at the auction site and be triggered for processing by the occurrence of the same event. For example, every time the auction site receives a bid (an event), a rule may be triggered to record some information in a bid

history file for a product [JOS01]. Table 3.2 shows three examples of rules currently implemented in IntelliBid.

Table 3.2 Example Rules in IntelliBid

Rule Name	Side	Description
AuctionHistory	IntelliBid (Auction)	Used to gather information about a product's bid history
SendBidInfo	IntelliBid (Auction)	Used to inform the user about the latest bid placed for a product along with all the previous bids, if any
IncreaseMyBid	Bidder	Used to increase bidder's bid on a specific product

A simple bidder side rule that defines the bidder's bidding strategy is shown in Figure 3.1. It should be pointed out that a more complex rule, that makes use of a more sophisticated formula to calculate a new bid, could be likewise defined.

RULE	IncreaseMyBid (String buyerid,String productid,String auctiontype,Float oldbid, Float Minincrementalprice)
RETURNS	void
DESCRIPTION	"Check to see if my new bid is within my upper limit, and then increase bid according to a formula"
RULEVAR	float newbid
CONDITION	(oldbid + 2*minincrementalprice) < RuleLib.NewLimit.getUpperLimit()
ACTION	newbid = oldbid + 2 * minincrementalprice RuleLib.EvaluateBidClientXML.SendBid(productid, newbid, quantity, creditcard, exp_date) RuleLib.Screen1.screenIncBid(productid, newbid)
ALTACTION	RuleLib.Screen1.screenNoBid(productid)

Figure 3.1 Description of the IncreaseMyBid Rule

The IncreaseMyBid rule's main purpose is to increase the bidder's bid to a specified amount, if it satisfies the condition of his/her upper limit.

The rule has five input parameters--BidderId, ProductId, AuctionType, BidValue and MinIncrementalPrice. These parameters correspond to the parameters of the event OutBid. These five parameters are necessary because they are used in the rule body to calculate the value of the new bid.

As the rule does not return any value, the RETURNS clause of the rule is set to void. The description of the rule is a statement describing the function of the rule.

A rule variable newbid has been defined in the RULEVAR clause. Its type has been defined as float. Its purpose is to temporarily store the value of the calculated new bid.

The CONDITION clause is used to evaluate whether the new bid placed will satisfy a certain upper limit. In our example, we choose to vary this upper limit dynamically in the rule. Currently, to change the limit dynamically, we make use of a class defined in the bidder's rule library. In the bidder's rule library, we have created a class called NewLimit. This program reads from a text file the value of the limit specified. This value in the text file can be changed and saved back to the file. Hence, every time this rule is executed, it picks up the newest value from the file. Thus, changes in the value of the upper limit are reflected in the rule. Using the upper limit, the CONDITION is evaluated. Depending on the result of the evaluation, either ACTION or ALTACTION is carried out.

The ACTION section is executed when the current top bid value is less than that of the upper limit. Here, any mathematical formula can be used for calculating the value of the new bid. In our example, we chose a simple formula having the general form as shown below:

*Value of new bid = Value of old bid + K * minimum incremental price*

The value of K will decide what the increment of the bid should be. If K is less than 1, then the bid will be less than that of the acceptable minimum incremental value, and should not be used. Such a bid would only result in being rejected. In our example, we have chosen the value of K to be 2. The new bid is sent to the auction site in an XML message.

3.1.3 Triggers

A trigger relates a structure of events with a structure of rules. An event structure has two parts, namely: a TRIGGEREVENT part and an EVENTHISTORY part. The TRIGGEREVENT part specifies a number of events. The occurrence (or posting) of any one of these events would initiate the evaluation of the EVENTHISTORY part. The EVENTHISTORY part can be a complex event expression, which makes reference to some events that have already occurred. "E1 follows E2 follows E8," and "E4 and E7 occurred within a certain time window" are examples of event history expressions. Thus, upon the occurrence of an event specified in TRIGGEREVENT expression, the EVENTHISTORY expression is evaluated. If the result is true, then the rule structure given in the trigger specification is processed. In an automated auction application, triggers can be specified both at the auction site and the bidders' sites to trigger different rules upon the occurrence of the same event. Table 3.3 shows two example triggers.

Table 3.3 Example Triggers in IntelliBid

Trigger Name	Event	Rule	Side
OutBidIncreaseMyBid	OutBid	IncreaseMyBid	Bidder
MonitorBidsSendBidInfo	MonitorBids	SendBidInfo	IntelliBid
AuctionCloseSendAuctionHistory	AuctionClose	SendAuctionHistory	IntelliBid

3.2 Architecture of IntelliBid

The architecture of IntelliBid is shown in Figure 3.2. IntelliBid is a network of Knowledge Web Servers, each of which consists of a Web server and a number of additional components that provide auction-related services. These components are a Knowledge Profile Manager, an Event-Trigger-Rule (or ETR) Server, an Event Engine, a Persistent Object Manager [SHE01], and a Metadata Manager. The Knowledge Web Server is replicated and installed at multiple sites. A Knowledge Web Server installed at an auction site would provide the services needed to maintain all the auction related information. It has an additional component called Bid Server for administrating the auctions of products. Knowledge Web Servers installed at the bidders and suppliers' sites would provide the services needed for playing the roles as bidders and suppliers, respectively. Any person or organization at a single site can be both a bidder and a supplier. The components installed at each site can support the operations and maintain the information for both roles.

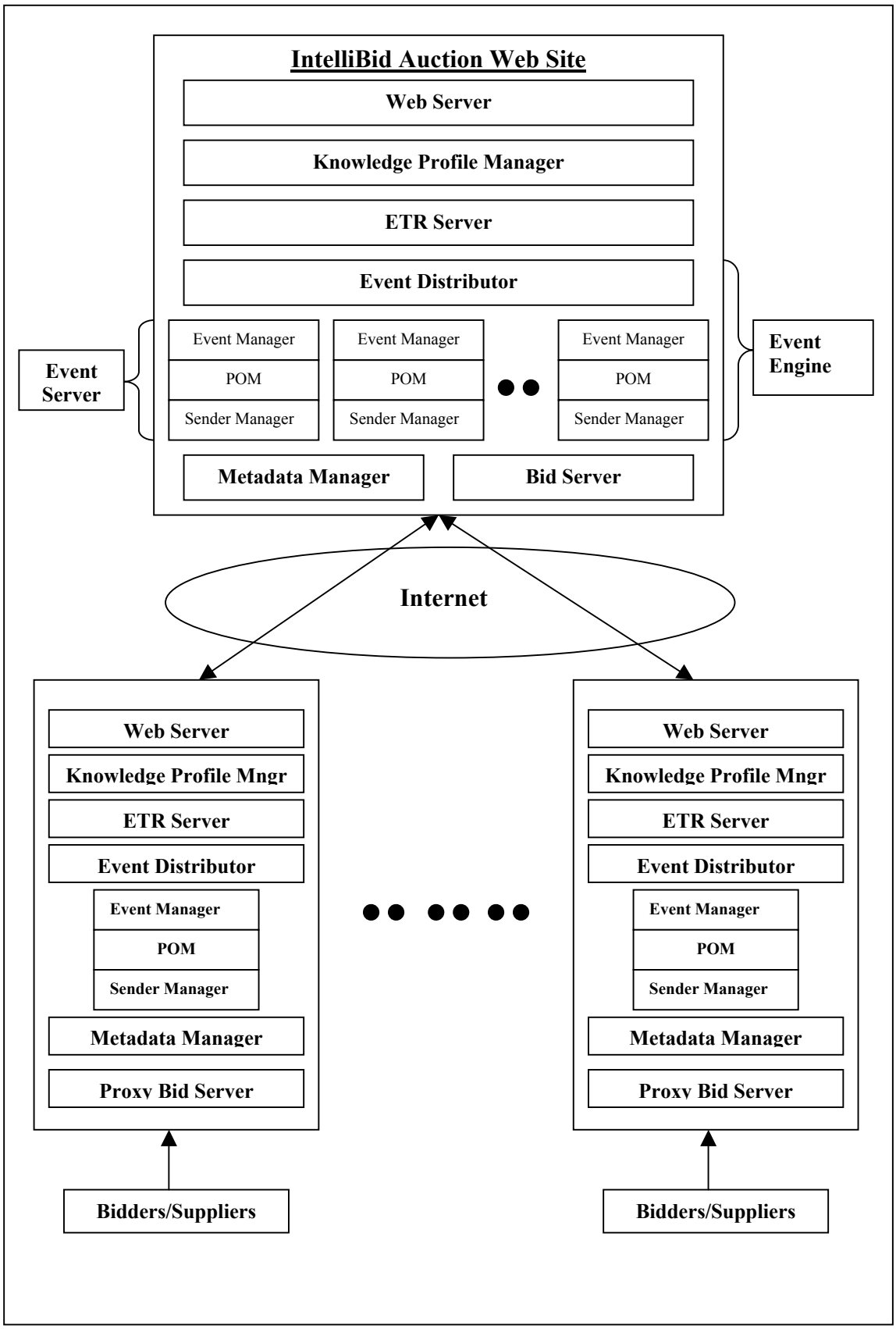

Figure 3.2 Overall Architecture of the IntelliBid Auction System

3.2.1 Knowledge Profile Manager

The Knowledge Profile Manager (KPM) is responsible for maintaining the knowledge profile of either the auction site or the bidder/supplier site that participates in an auction. The knowledge profile of the auction site contains the events, triggers, and rules that are published by the creator of the auction site. At a bidder/supplier site, the knowledge profile contains the events that the bidder/supplier has subscribed to, and the rules and triggers defined at that site.

The Knowledge Profile Manger provides GUI interfaces for defining events, triggers, and rules [PAR99]. A bidder/supplier with no programming experience can use the GUIs to define events, triggers, and rules by using forms that contain text boxes and drop-down menus. KPM maintains separate profiles for a user or organization that plays both roles as a supplier and a bidder.

3.2.2 ETR Server

The ETR Server performs the installation and processing of triggers and rules defined for each site that participates in an auction. The ETR Server interacts with the Knowledge Profile Manager and the Event Manager. It receives the definitions of rules and triggers entered at the auction site or a bidder/supplier site using the graphical user interface provided by the Knowledge Profile Manager. These triggers and rules are translated into an internal data structure and program code for run-time processing, respectively. The ETR Server also receives event notifications from the Event Manager when events occur. It would look up the internal data structure to identify the proper triggers and schedule the appropriate rules for execution.

3.2.3 Event Engine

As shown in Figure 3.2, the IntelliBid's Event Engine consists of an Event Distributor, and one or more Event Servers.

3.2.3.1 Event distributor

The function of the Event Distributor varies from one phase to another of the Event Engine. In the event subscription phase its function is to effectively distribute event subscriptions from subscribers to the Event Servers in the Event Engine. During the notification phase, the function of the Distributor is to forward or distribute the occurring event to all of the Event Servers inside the Event Engine.

The Distributor uses a distribution table to manage the Event Servers under its control. When the Distributor starts up, the list of Event Servers is empty. It will remain empty until one of the Event Servers registers to the Distributor to participate in the Event Service. During registration, the Distributor will create an Event Server entry and store it in the list of Event Server. However, the event list within the Event Server entry is initially empty. It will be filled later when event subscriptions come.

Each Event Server entry in the distribution table contains a list of pairs (events and subscribers count). When an inbound subscription comes, the Distributor will use a specified algorithm (either round robin or event based) to choose one Event Server from the pool of Event Servers. If the newly subscribed event is already in that Event Server list, then the Distributor increments the subscribers count. Otherwise, the Distributor will create a new entry for that event. When a subscriber deletes the subscription of an event, the corresponding subscribers count is decremented [HEN01].

The Distributor also provides reliability in the Event Engine by providing a recovery mechanism. The Distributor keeps track of all the Event Servers under its

control and as soon as any of them becomes dead, it removes it from the distribution table and saves the data in the database. As soon as that Event Server is up and running again, the Distributor adds it back to the distribution table. As all the data of the Event Server is stored in the database, it can interrogate the database via the POM library and participate again in the system.

3.2.3.2 Event server

The Event Server of the Event Engine consists of three main components: an Event Manager, a Persistent Object Manager and a Sender Manager. IntelliBid Auction web site can have one more Event Servers running and hence registered with the Distributor, while on the Bidder/Supplier side, only one Event Server needs to be running. The idea behind having multiple Event Servers on the IntelliBid side is that it helps in concurrent event subscription and event notification as there can be several events being subscribed to or posted at any time.

3.2.3.2.1 Event Manager

The auction site's Event Manager is responsible for accepting and completing any event subscriptions or event postings forwarded to it by the Distributor. The Event Manager has its API (Application Programmers Interface), which allows the auction programs executing on the auction site to be able to connect to it and post the auction events.

During the event registration, the bidders/suppliers can define their own event filters by selecting or providing values to attributes that are provided in the pre-defined event filter templates. These filters specify the data conditions under which the bidders/suppliers and the Event Managers of their corresponding Knowledge Web

Servers would be notified. They allow bidders and suppliers to have selective subscriptions to auction events.

The Event Manager on the bidder/supplier side performs the task of listening to events from the Event Manager of the auction site and forwards these events to its corresponding ETR Server to activate rules.

Following is a description of the filters defined on the IntelliBid events:

1) AuctionClose: The bidders/suppliers that subscribe to this event will be interested in being notified when the auction on a particular product finishes. Hence, the attribute productid is provided in a filter template in XML format as shown below in Figure 3.3. A bidder will specify a unique product identifier for this attribute [JOS01].

```
<EVENTFILTER>

 <DESCRIPTOR>This event is posted when a specific
product's auction is finished</DESCRIPTOR>

 <EVENTNAME>AuctionClose</EVENTNAME>


 <ATTRFILTER>
  <FILTERNUM>1</FILTERNUM>
  <ATTRNAME>productid</ATTRNAME>
  <TYPE>String</TYPE>
  <OPERATOR>EQUAL</OPERATOR>
 </ATTRFILTER>


</EVENTFILTER>
```

Figure 3.3 Filter Template in XML Format Defined for the AuctionClose Event

2) Product: A bidder interested in obtaining information about some registered auction products would specify his/her interest in terms of the category and the specification of a specific product. Hence, two attributes are provided in the event filter template defined for the event Product: ProductCategory and ProductSpecification. For ProductCategory, a bidder can select a single value from a list of values, such as Automotive, Computers, Electronics, etc. As soon as the bidder selects a category, all the specifications related to that category will show up in the form of checkboxes. A

bidder can select multiple specifications under a single category to subscribe to the Product event. For example, say the bidder selects the category: "Automotive" and he would like to be informed when a "Honda" or a "Toyota" is put up on IntelliBid. So he simply check marks "Honda" and "Toyota" from the specification list and presses the Subscribe button. The bidder would be notified as soon as either a Toyota or a Honda is put up on IntelliBid.

3) Outbid: A subscriber to the event of outbid will most likely be a bidder in an auction, and hence will be interested in being informed if he/she is outbid during the auction process. The combination of the bidder's username and productid will uniquely identify the bidder on the product. Hence, the Event Manager will send out event notification to only this specific bidder of the specific product. Therefore, the attributes' username and productid are provided in the filter template defined for the Outbid event. The bidder will provide a single value for each of these two attributes [JOS01].

4) MonitorBids: A bidder/supplier interested in following each and every bid placed for a particular product can subscribe to this event. As it is related to a particular product, again the attribute productid is provided in a filter template.

There are two ways a user can subscribe to any of the above-mentioned events. The first way is by filling out and submitting the event subscription form generated by the respective event filter template; and the second way is by simply check-marking any/all events displayed on the product description page of the product of interest, which is currently being auctioned on IntelliBid.

3.2.3.2.2 Persistent Object Manager

The Persistent Object Manager (POM) consists of two main components: 1) Object-Relational Mapping Engine and 2) XML-Relational mapping engine.

The general-purpose Object-Relational Mapping Engine provides a persistent storage facility for storing information related to auctions. It also provides a high-level mechanism, in the form of APIs, for auction programs to store, retrieve, update and delete auction objects without having to know the internal data structures of the auction objects. The auction programs executing on the auction site use these API's to perform their database-related operations. The XML-Relational Mapping Engine provides the

persistence capability and a filtering mechanism to the Event Manager of the auction site. It includes a parser, which maps filter definitions of the auction events stored in the form of XML files to a relational form, and a set of Application Program Interfaces (APIs). POM is implemented on top of an Object-Relational database system called Cloudscape. To utilize POM for persistent storage at the auction site, a configuration file for the auction database must be created and stored in the directory of the program, which invokes the APIs of POM. This database is then used for storing the auction-related information such as supplier information, bidder information, product's bid information, etc. Figure 3.4 is a sample of a configuration file.

```
Host        localhost
Port        9099
Database    auctionDB
```

Figure 3.4 Sample of a Configuration File for the IntelliBid Auction Database

The first parameter specified is the host on which the database is located. In the IntelliBid auction system, the entire auction related database operations are performed on the IntelliBid server side. Hence, the value of the host parameter is specified as "localhost" since the local applications are the ones that issue the database operations. This parameter can also be the machine name or IP address of a remote server that hosts the auction.

The second parameter is the connection port number. POM uses this port number for connection. No other program can be allowed to use this port number. In the IntelliBid auction system, 9099 was available, and hence was chosen to be the port number.

The final parameter specified is the name of the database. The database is created in the default path specified in the Persistent Object Manager's settings.

Once a configuration file is specified in the directory of the auction programs utilizing the APIs of POM, no additional measures need to be taken to create the database. The first time that the auction database is used, POM checks to see if the specified database in the configuration file exists. If it does not, it creates the database in the specified default path; its log files, and related information. If the database already exists, then POM directly proceeds to perform the requested operation on the auction database [JOS01].

3.2.3.2.3 Sender Manager

The Sender Manager is a stand-alone server that is implemented as a thread and hence can be replicated to improve performance and scalability of a typical Knowledge Network framework. It acts as a sender on the provider side and sends event notifications and it acts as a receiver on the event subscriber or client side to receive the notification.

As a sender, it waits for a list of subscribers from the Event Manager that matched the occurring event in the system. Once the list is received, The Sender Manager will notify the default local ETR Server to trigger any provider side rules. Then it will iterate through the list and notify the subscribers through the corresponding Sender Manager on the subscriber site.

As a receiver on the subscriber side, The Sender Manager awaits notification from the Sender Manager of the provider side. Once it receives a notification, it will notify the local ETR Server [HEN01].

3.2.4 Metadata Manager

The creator of the auction site decides the appropriate events, triggers, and rules that should be created for controlling and managing auctions. Once created using the Knowledge Profile Manager, these events, triggers and rules information are stored and maintained by the auction site's Metadata Manager. Similarly, the triggers and rules that the bidders/suppliers define for execution on the auction events are stored and maintained by their respective Metadata Managers [JOS01].

3.2.5 Bid Server

The Event-Trigger-Rule-Based auction concept allows for bidders to define rules, which implement their bidding strategies locally on their respective sites. A rule may calculate the value of the bid amount, which is to be placed for a particular product. This bid amount needs to be transferred to the auction site for it to be considered. For this purpose, the auction site needs to have a Bid Server executing and expecting bids from bidders generated by bidder side rules.

For a bidder side rule to communicate with the Bid Server on the auction site, a Proxy Bid Server is provided by the auction site and included in the rule library of the bidder. The proxy is programmed with the knowledge of the IP address and port number of the Bid Server. Invoking the proper method of the proxy through the rule will transfer the new bid to the auction site.

Depending on the particular implementation of the auction, the Bid Server can be implemented using a variety of communication technologies. Also the auction site can provide more than one Bid Server, implemented with different technologies for the bid transfer purpose. IntelliBid has been implemented with two Bid Servers, one using RMI [JAV01] technology and the other using XML-RPC [USE01]. Each bidder is provided

with both proxies for communication with the auction site in their respective rule libraries. The bidder is free to choose one from these two alternatives. The purpose of implementing more than one Bid Server is mainly for security reasons. Certain implementations may be more suitable in one environment over another [JOS01].

CHAPTER 4
INTELLIBID AUCTION RULES AND STEPS FOR THE CREATION AND EXECUTION OF AN AUCTION USING INTELLIBID

This chapter first of all lists the various business rules followed to do auctioning on IntelliBid for both the supplier and the bidder of a product. The later half of the chapter describes in detail the steps that need to be followed to successfully create and execute an auction on IntelliBid.

4.1 IntelliBid Auction Rules

IntelliBid has different policies for sellers and bidders as per their respective functions. Even depending on the type of auction, English or Dutch, there are different bidding rules.

4.1.1 Business Rules related to a Seller

Below is the list of business rules defined for a typical seller to supply a product on IntelliBid:

- To be able to put up a product for auctioning, a user needs to be properly registered, with valid contact information, as a Seller on IntelliBid.
- During Seller Registration, a valid credit card needs to be provided so that the product insertion fee can be charged to it. All major credit cards are accepted.
- Every product to be put up for auctioning should be put under one Product Category and one Product Specification.
- The seller is solely responsible for all the information and pictures put up for their product(s).
- On the completion of the auction for a product, the seller is responsible for shipping the product to the winner of the auction.

4.1.2 Business Rules related to a Bidder

There is a separate set of business rules defined for a typical bidder on IntelliBid:

- To be able to bid for a product, a user needs to be properly registered, with valid contact information, as a Buyer on IntelliBid.
- During Buyer Registration, a valid credit card needs to be provided so that only serious buyers would register. This helps in keeping the number of garbage information to the minimum in the database.
- A registered buyer needs to supply his/her username, password and valid credit card information every time he/she wishes to place a bid for security purposes. The credit card is not charged at this point; it is just used to verify if the bidder's credit limit would cover the bid amount.
- On the completion of an auction, the winning bidder's credit card is charged with the winning bid amount. No additional fees are charged to the winning bidder.

4.1.3 IntelliBid Bidding Rules

IntelliBid has a different set of bidding rules depending on the type of auction the product is under. Currently IntelliBid supports *English* and *Dutch* Auctions.

4.1.3.1 English auctions

English auctions are the conventional top-bid auctions. They are always single quantity auctions. Following are the bidding rules for a typical English Auction on IntelliBid:

- All bidders need to specify a bid amount to place a bid.
- Every new bid amount must be at least equal to the sum of the last bid placed and the minimum incremental value defined by the seller of the product. For the first bid, the bid amount should be at least equal to the sum of the base price and the minimum incremental price defined by the seller.
- All bids placed for any product on IntelliBid are non-cancelable.
- Bidders may rebid after being outbid.
- The bidder whose bid is the highest priced bid at the time of auction close is the winner of the auction for that product.

4.1.3.2 Dutch auctions

All multiple quantity auctions are by default Dutch Auctions. A box of 50 sweets if put on auction as a whole box, represents an English Auction while if all 50 sweets are sold separately as 50 units of the same sweet, then it represents a Dutch Auction.

Following are the bidding rules for a typical Dutch auction on IntelliBid:

- Bidders have to specify both a bid price and the quantity they want to buy.
- Every new bid amount must be at least equal to the sum of the last bid placed and the minimum incremental value defined by the seller of the product. For the first bid, the bid amount should be at least equal to the sum of the base price and the minimum incremental price defined by the seller.
- All bids placed for any product on IntelliBid are non-cancelable.
- All winning bidders pay the same price per item, which is the lowest successful bid. This amount might be less than the highest successful bid amount.
- When there are two bids at different prices, the higher value bid always wins.
- When there are bids at the same price but different in quantities, the higher quantity bid wins.
- When there are bids at the same price and equal in quantities, the bidder who placed his/her bid first wins.
- Bidders may rebid after being outbid.

To further clarify the bidding rules, let us assume that there are 6 pens for auction and the bids placed look something like shown in the figure below. Assume the minimum possible bid amount is $1 and the minimum incremental value is $0.25.

USERNAME	BID VALUE	QUANTITY	DATE OF BID
Kushal **(W)**	$1	1	3/13/2002 5:7:15
Payal **(W)**	$1	1	3/13/2002 10:7:15
Hendro	$1	1	3/13/2002 15:0:15
Sharon	$1	1	3/13/2002 15:7:15
Anu	$1	1	3/14/2002 12:0:0
Nicky	$1	1	3/15/2002 10:10:10
Jack **(W)**	$1.25	2	3/15/2002 15:7:15
Jill **(W)**	$1.25	1	3/16/2002 20:0:0
Hill **(W)**	$1.25	1	3/16/2002 23:59:50

Figure 4.1 Example of Dutch auction Bidding Rules

As shown in Figure 4.1, all the bidders with a bid amount of $1.25 are guaranteed their pen(s). Now, as in this case, we have six pens and the total quantity bid for by the $1.25 bidders is four. On close of auction, two of the $1 bidders, whose bids came before the other $1 bidders, will be awarded a pen each. The "**(W)**" by the side of the username in Figure 5 represent winners of this particular Dutch Auction [EBA01, CNE02].

4.2 Steps for the Creation and Execution of an Auction on IntelliBid

To construct an auction service in IntelliBid, the following steps can be followed.

1) Publishing and installation of auction events, triggers, and rules on IntelliBid: The auction creator decides which events, parameterized event filter templates and auction side rules would be suitable for the auction that he/she wishes to deploy. They are defined by using the GUIs of the auction site's Knowledge Profile Manager. The Knowledge Profile Manager gives the event and filter template definitions to its corresponding Event Manger. Auction-side customizable rules are given to the auction site's ETR Server [JOS01].

2) General registration and Product event registration by a bidder/supplier: The bidders and suppliers begin registration with IntelliBid by accessing the IntelliBid web site at a known URL. By filling out their respective registration forms, the necessary information required for bidding or supplying is obtained by IntelliBid. After registration, the bidder/suppliers can subscribe to IntelliBid events. To subscribe to an event, a bidder/supplier can either simply select the name of the event from the

product's description page in which the user has interest or can go to a separate event registration page generated based on a pre-defined event filter template. During event registration, the values for "installing" a filter based on the event filter template are also provided by the bidder/supplier. If this registration is successful, the event subscription is forwarded to the Event Manager of the bidder's/supplier's site. Consider the following as an example scenario: A supplier S who is interested in supplying auction products registers with IntelliBid. Two bidders, A and B, interested in participating in these auctions also register with IntelliBid. Bidder A is interested in a particular product and subscribes to the "Product" event. He specifies his interests by installing a filter for the "Product" event by selecting say "Electronics" as a category and "TV" as a specification [JOS01].

3) Product event generation and filter processing on IntelliBid: Suppose that, in our scenario, supplier S submits a product under category "Electronics" and specification "TV" for auctioning after registration. Assume that the supplier decides to start off with a base price of $100, with a $10 minimum incremental price on the product. This registration of the product results in the posting of the Product event [JOS01]. The steps in Figure 4.2 describe the activities of submitting a product for auction by a supplier. Once the Product event is generated, it is given to IntelliBid's Event Manager via the Distributor. The Event Manager processes the event filters that have been defined by the bidders/suppliers to determine if the data conditions specified in these filters have been met. For those filters that pass the filtering process, their corresponding bidders/suppliers are notified by IntelliBid's event notification mechanism.

4) Product event notification to the bidders/suppliers: The bidders/suppliers that had subscribed to the Product event and whose filters have passed the filter-processing are notified in our scenario, only bidder A passes the filter processing and is notified. This is done by sending a notification to the Event Manager of the bidder A (i.e., push notification [POI01]) and/or an e-mail notification to bidder A himself.

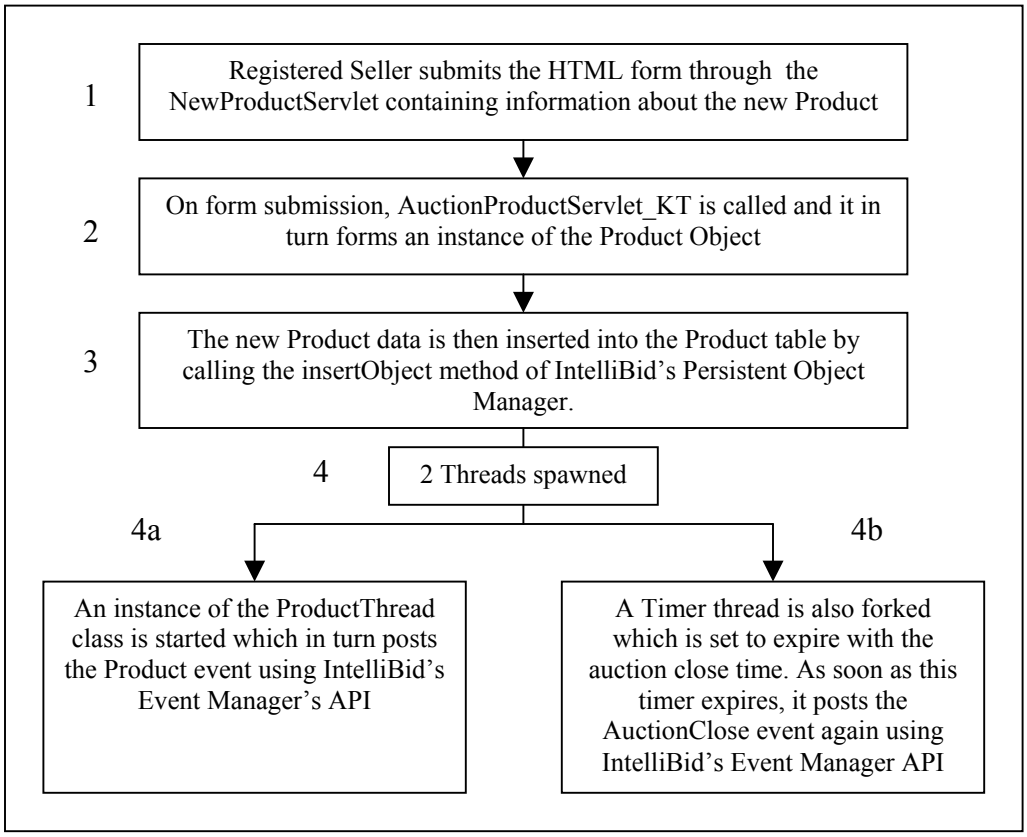

Figure 4.2 Steps Involved in the Process of a Supplier Submitting a Product for Auction in IntelliBid

5) Initiate bidding and subscribe to the outbid event: Once a product is registered, bidding can begin. As described earlier, IntelliBid has two types of Auctions implemented: *English* and *Dutch* each of whose bidding rules has been described earlier. IntelliBid provides two methods for placing bids on the auction product for either auction type:

    - Manual (on-line) bidding: Use a Web browser to access the IntelliBid web site. Submit an HTML form through the ProductDetails_BidsServlet servlet with the necessary bid information.

    - Automatic bidding: Subscribe to the OutBid event, and then create a bidder side rule that will define a bidding strategy. Use the proxy bid server on the bidder's site to place a bid [JOS01].

6) On receiving the Product event notification, bidder A visits the IntelliBid web site and verifies the detailed product description to see if it matches his choice. He is satisfied with the product, and decides to participate in bidding. He submits an HTML form to IntelliBid with the appropriate bid information for the product through the ProductDetails_BidsServlet servlet. He is also interested in using a bidder side rule for bidding, and hence subscribes to the "OutBid" event.

7) Following this, bidder B accesses the IntelliBid site, and also decides to participate in the auction of the same product. She places her bid at the IntelliBid web site and decides to use her own rule to do automatic bidding, and hence subscribes to the "OutBid" event. She is also interested in the history of the bidding of this product, and hence subscribes to the "AuctionClose" event, linking this subscription to the auction side rule "AuctionHistory" [JOS01].

8) The suppler S is interested in monitoring the bids being placed for the product he supplied and thus goes ahead and subscribes to the "MonitorBids" event through the ProductDetails_BidsServlet servlet generated for his product. The "MonitorBids" event is in turn linked with the "BidInfo_KT" rule which notifies S as soon as a new bid is placed for his product.

9) Bidder/Supplier side rules definition and installation: After the bidder/supplier has successfully registered for an event, he/she can access his/her Knowledge Profile using the local Knowledge Profile Manager and then define triggers and rules related to the subscribed events that are to be processed at his/her site. For automated bidding, Bidder A creates a bidder side rule (similar to IncreaseMyBid in Figure 3.1), which will increment his bid on the outbid event, until an upper limit of, say, $300. Bidder B also creates a similar rule using her Knowledge Profile Manager. She sets her upper limit in her bidder side rule to $350. The bidder/supplier's Knowledge Profile Manager will then install the triggers and rules in the ETR Server of the bidder/supplier's site.

10) OutBid event notification and Automatic bidding: As bidder B places her bid, bidder A is outbid. Hence the OutBid event is posted, with event notification being sent to bidder A's knowledge web server. This notification will trigger the rule that he defined on the OutBid event. The rule will be activated and will calculate a new bid for the product. The proxy bid server in the rule library of bidder A is then invoked to transfer the new bid amount to the IntelliBid auction site. This action will result in bidder B being outbid. Once again the OutBid event is posted, but this time, event notification is sent only to bidder B, as only she passes the event filter. Her new bid is similarly placed since she also has defined a rule to implement her bidding strategy. For every new bid being placed, the supplier S is being notified as he is subscribed to the "MonitorBids" event. Due to this action of bidding, the bids of these two bidders will be raised until bidder A is finally outbid due to a lower limit. Once his upper limit of $300 has been reached, bidder A decides to now increase his upper limit to $400. The auction once again proceeds; and after $350 has been reached, bidder B's bids will not be placed, as further bids will be greater than her limit. She decides not to place any more bids.

11) End of Auction and posting of AuctionClose event: When submitting the product for auction, the supplier decides the time limit of the auction. IntelliBid notes this time, and on expiration, posts the "AuctionClose" event. This event marks the end of the auction on the specific product. In the above scenario, bidder A will be declared to be the winner. E-mail will be sent to bidder A informing him of winning the auction on the product. Similarly, e-mail will be sent to supplier S, informing him of the winner

of his auction. Also, since the “AuctionClose” event has been posted, checking is done if a bidder/supplier had customized the AuctionHistory rule. In our scenario, bidder B had customized the AuctionHistory rule. Hence, the customized rule is executed on IntelliBid’s ETR Server. The rule collects the history of bids for the product, and the history will be sent by e-mail to bidder B [JOS01].

CHAPTER 5
IMPLEMENTATION

IntelliBid has been implemented using JDK 1.3, on a Windows NT platform. The Web Server used is Jakarta-Tomcat 3.2. Internet Explorer 5.5 and 6.0 are used as browsers. The Knowledge network components are installed and used for event subscription, event notification, and trigger and rule definition [JOS01]. A distributor system is used to do load balancing in terms of event posting and event notification.

All the forms used for user interface have been implemented using HTML and JavaScript. The dynamic web programming has been performed using Java Servlets. The Java Servlets use the servlet.jar component of Jakarta-Tomcat 3.2. The transfer of bids from the bidder to the IntelliBid auction site is done using XML-RPC as well as RMI. For XML-RPC to be used, the required XML parser used is the SAX parser. The pop-up windows on the bidder's side are implemented using Java's Abstract Window Toolkit (AWT) [JOS01].

5.1 Tables in the IntelliBid Database

The IntelliBid auction system uses the Object-Relational Mapping Engine of the Persistent Object Manager to perform database operations. Therefore, it is necessary to have objects corresponding to relational tables in IntelliBid. To contain the necessary information for the basic auction process, we have developed seven objects, each of which corresponds to a relational table in the auction database. The following table shows the data members in these objects and a brief description of their purposes.

Table 5.1 Description of Objects in IntelliBid and their Attributes

NEWBUYER TABLE		
Data Members	Type	Description
b_fname	String	First Name
b_lname	String	Last Name
b_email	String	Email Address
b_comp_name	String	Name of Employing Company if any
b_phone_acode	String	Phone Area Code
b_phone_1	String	Phone Number--First 3 digits
b_phone_2	String	Phone Number--Last 4 digits
b_bill1	String	Billing Address Street 1
b_bill2	String	Billing Address Street 2
b_bcity	String	Billing Address City
b_bstate	String	Billing Address State
b_bzip	String	Billing Address Zip
b_bcountry	String	Billing Address Country
b_ship1	String	Shipping Address Street 1
b_ship2	String	Shipping Address Street 2
b_scity	String	Shipping Address City
b_state	String	Shipping Address State
b_szip	String	Shipping Address Zip
b_scountry	String	Shipping Address Country
b_cc_type	String	Credit Card Type: MC/AMEX/VISA/DISC
b_cc_num	String	Credit Card Number
b_cc_name	String	Name as appears on the Credit Card
b_cc_exp1	String	Credit Card Expiry Month
b_cc_exp2	String	Credit Card Expiry Year
b_userid	String	Username
b_passwd	String	Password

Table 5.1 Continued

NEWSUPPLIER TABLE		
Data Members	Type	Description
s_fname	String	First Name
s_lname	String	Last Name
s_email	String	Email Address
s_comp_name	String	Name of Employing Company if any
s_phone_acode	String	Phone Area Code
s_phone_1	String	Phone Number--First 3 digits
s_phone_2	String	Phone Number--Last 4 digits
s_bill1	String	Billing Address Street 1
s_bill2	String	Billing Address Street 2
s_bcity	String	Billing Address City
s_bstate	String	Billing Address State
s_bzip	String	Billing Address Zip
s_bcountry	String	Billing Address Country
s_ship1	String	Shipping Address Street 1
s_ship2	String	Shipping Address Street 2
s_scity	String	Shipping Address City
s_state	String	Shipping Address State
s_szip	String	Shipping Address Zip
s_scountry	String	Shipping Address Country
s_cc_type	String	Credit Card Type: MC/AMEX/VISA/DISC
s_cc_num	String	Credit Card Number
s_cc_name	String	Name as appears on the Credit Card
s_cc_exp1	String	Credit Card Expiry Month
s_cc_exp2	String	Credit Card Expiry Year
s_userid	String	Username
s_passwd	String	Password

Table 5.1 Continued

PRODUCT TABLE		
Data Members	Type	Description
Supplierid	String	Supplier's ID registered with IntelliBid
Productid	String	Unique Product ID formed by using Supplier ID's first letter and the date-time stamp
Producttitle	String	Product Title
Productcategory	String	Product Category
Productspecification	String	Product Specification/Classification
Productdescription	String	Product's Description in Detail
Pictureurl	String	Product's Picture URL (optional)
Baseprice	Float	Product Selling Base Price
Minincrementalprice	Float	Minimum Bid Increment Value
Startingdate	String	Auction Start Date and Time
Endingperiod	String	Auction Closing Date and Time
quantity	Integer	Quantity of a Product put on Auction
auctiontype	String	Type of Auction (English/Dutch)
status	String	Product Status: "A" if Product's Auction is still going on "I" if Product's Auction has closed

Table 5.1 Continued

BIDS TABLE		
Data Members	Type	Description
productid	String	Product ID for which bid placed
buyerid	String	ID of the bidder
datetime	String	Date and Time when bid was placed
status	String	Indicates the status of the bid: “A” if bid is currently winning the product “I” if bid has been outbid “W” if bid is the winning bid
Qty	Integer	Quantity of Product desired
Bidvalue	Float	Value of the Bid placed
Comments	String	Any special comments regarding the Bid. E.g. 1 of 2 quantity in Dutch Auctions. (optional)

Table 5.1 Continued

CATEGORY TABLE		
Data Members	Type	Description
Category	String	Name of a Product Category on IntelliBid. All Products have to be put under a specific category.

SPECIFICATION TABLE		
Data Members	Type	Description
Name	String	Name of a Product Specification/Classification available on IntelliBid. All Products have to be put under a particular Category and Specification.
Category	String	Name of the Category the particular Specification refers.

ADMIN TABLE		
Data Members	Type	Description
Username	String	Unique Administrator's username
Password	String	Password
FirstName	String	First Name of the Administrator
LastName	String	Last Name of the Administrator
Clearance	String	Clearance level given: Full--full access to the IntelliBid System Category--Access restricted till the ability of adding new Categories and Specifications.

5.2 Implementation of Auction Logic

An important component in any auction is the logic to implement the bidding process. Chapter 2 mentions the different types of Internet-based auctions. By far, the most common one is the English auction in which the winner is the person with the highest bid. IntelliBid's implementation currently supports the English and the Dutch auction, and it is easily expandable to other types of auctions.

For either type of auctions, the bidding logic in IntelliBid is implemented in a class included in the auction site's rule library, called EvaluateBid_KT. This class contains three principal methods of the auction process including separate methods to evaluate bids involving English and Dutch auctions:

1) evaluateBuyer()–This method contains the logic to verify the credit card number provided with the bid amount. It is necessary to verify that the credit card provided is valid with a sufficient credit limit before any bid can be accepted. This prevents fraudulent transactions and guarantees the suppliers with the bid amount. This is a dummy method, as currently IntelliBid does not have the capability to actually verify a credit card.

2) checkIfAcceptedBid()–Once the buyer's credentials are evaluated and cleared, this method is executed. It checks if the bid is of a sufficient amount and if the quantity of product desired meets the available quantity. This method is called whenever a bid is placed for an *English* auction, which in turn is always a single quantity auction and hence the quantity desired can never be more than or less than one. This method follows the following algorithm:

    - Obtain the total quantity of the bid-for product. If the quantity requested is not equal to one, then the bid is rejected and the bidder can correct the bid and try again. This prevents acceptance of a bid, which is greater or lesser than one.

    - Obtain the number of bids that have previously been placed on the product being bid for. If this number is zero, then this is the first bid obtained on the product. Check the base price and the minimal incremental price of the product, and obtain their sum. If the value of the bid placed is greater than the summed amount, it is a legitimate bid. Accept the bid and return. If the value of the bid is less than the summed amount, it is an unacceptable bid.

    - If the number of bids previously placed on the product is not zero, then this implies that some earlier bids have been received. Obtain the bid amount of

the current top bid. Verify that the amount of the new bid is greater than or equal to the sum of the current top bid and the minimum incremental price. If so, insert into the bid table the new bid that has been placed. Also, update the status of the old top bid in the bid table by setting it to be outbid. If the new bid is of insufficient value, it is not accepted.

The above steps perform the process of bidding. During this process (step c), if bidders have been outbid, the checkIfAcceptedBid() method forks a thread: OutBidThread, which is responsible for posting the OutBid event. It is necessary to separate the validation of the new bid from the actual posting of the OutBid event, because of the independence of these two actions. Once a bidder places a bid, this action is independent of whether or not another bidder is outbid. If these actions are not separated, then the bidder's browser or ETR server will be waiting for a response from IntelliBid's Event Manager until it has finished posting the OutBid event. This is an unnecessary penalty of time, which may be excessive in the case of a large number of the OutBid events. Hence, it is a necessity to have a multi-threaded solution for the bidding process.

Soon after the OutBidThread, the MonitorBidsThread is forked which in turn posts the MonitorBids event. The MonitorBids event notifies all of its subscribers by email about the new bid placed for the product.

3) checkBid_Dutch()–This method is called whenever a bid is placed for a *Dutch* auction, which is always a multiple quantity auction. The method's behavior is similar to checkIfAcceptedBid() method.

    - Again obtain the total quantity of the bid-for product. If the quantity requested is not lesser than the quantity available or is not greater than zero, then the bid is rejected and the bidder can correct the bid and try again. This prevents the acceptance of a bid, with improper quantity.

    - Check the base price and the minimal incremental price of the product, and obtain their sum. If the value of the bid placed is greater than the summed amount, it can be a legitimate bid. Then check and see if there is any quantity of left unclaimed for. If there is then accept the bid and return; else either the

quantity desired have to be increased so that it is greater than at least of the previous bidder's quantity or the bid amount needs to be incremented with the minimum incremental value.

- Now while performing b), any previous bids that have been outbid by the latest bid are marked. Email notifications are sent to their bidders.

Then finally, just like checkIfAcceptedBid(), an OutBidThread and a MonitorBidsThread are spawn and all the subscribers to the OutBid and MonitorBids events are notified. Hence this method takes care of the Dutch auction's logic.

5.3 Posting of IntelliBid Events

IntelliBid currently has four events implemented. The method of posting each of these events is described below.

5.3.1 Product

This event is posted when a supplier registers a new product with the IntelliBid system, for the purpose of auction. For supplying the information of the product to IntelliBid, the supplier submits the NewProductServlet servlet. The servlet form's action is set to point to another servlet: AuctionProduct_KT, which collects the information of the product and posts the Product event.

This event also corresponds to a table called Product in the database called AuctionDB and hence is extended with Get and Set methods for the Product object. The Product table is used for the persistent storage of the product information.

5.3.2 OutBid

This event is posted when a bidder involved in the auction process is outbid regardless of the type of auction.

This event does not correspond to a table in the auction database and is therefore not extended with Get and Set methods. Figure 5.1 shows the steps involved in the

posting of the OutBid event. This event can be posted by two different methods. The first method is by a server side program (i.e., a Java Servlet), which is used to accept the bid information through the web browser. This program calls the three methods of the EvaluateBid_KT class to validate the bid and then complete the bidding process, as previously explained in Section 5.2. The second method is by the Bid Server, which also calls the same three methods of the EvaluateBid_KT class, to evaluate bids placed through bidder side rules [JOS01].

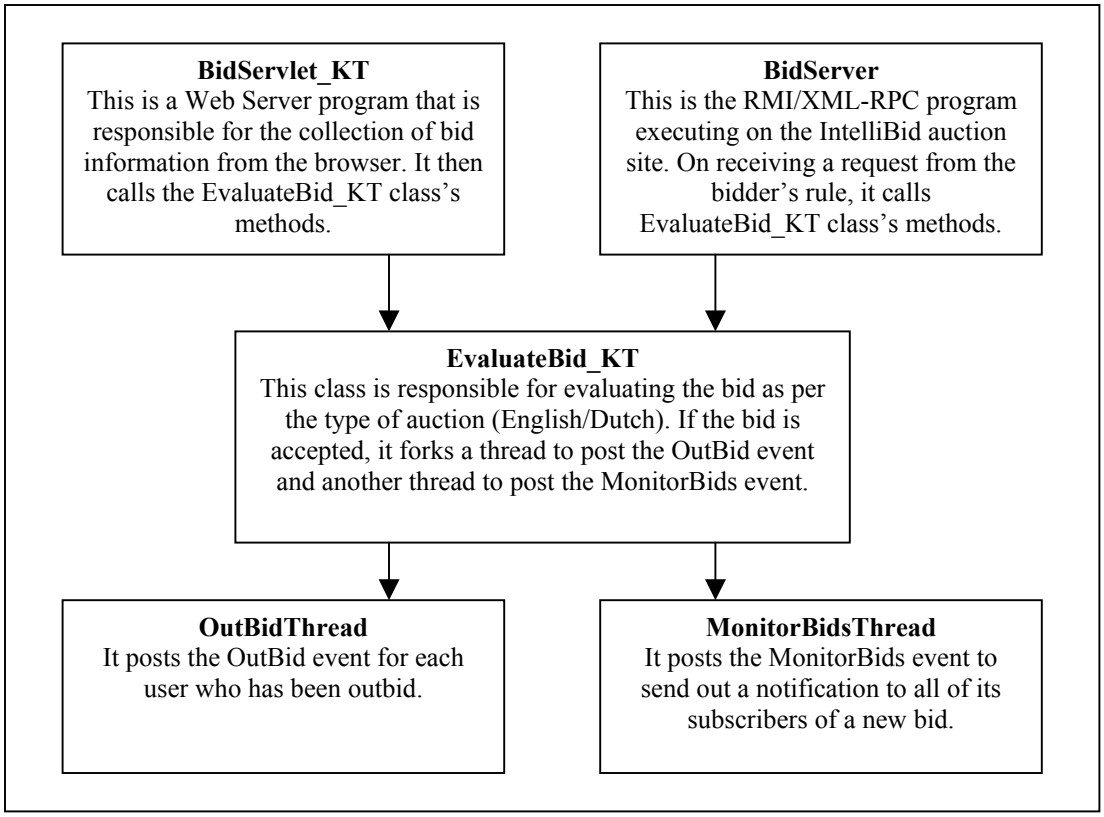

Figure 5.1 Steps Involved in the Posting of the OutBid and MonitorBids Events

5.3.3 AuctionClose

This event is posted when the time limit of the auction on a specific product has expired.

The supplier, who registers the product for auctioning, decides the actual amount of time that his/her product will be active for bidding. On receiving this time from the HTML form, the server program calculates the difference between the current time and the specified time limit. As indicated in the event Product (Figure 4.2), a separate thread is started when a product is accepted for bidding. This is a timer-based thread and its timer is set to the value calculated for the specific time limit. The timer counts down till the zero count is reached, and then the thread posts the AuctionClose event for the specific product [JOS01].

5.3.4 MonitorBids

As shown in figure 5.1, this event is posted whenever a new bid is placed for any product currently on auction.

Similar to the OutBid event, this event is posted through the EvaluateBid_KT class soon after the placed new bid is accepted. EvaluateBid_KT spawns a thread implemented by the class MonitorBidsThread to post the event.

MonitorBids is linked with a provider side rule BidInfo_KT. This rule gathers the new bid information along with information about all the previous bids and notifies the event's subscribers of the new bid.

5.4 Proxy Bid Server and Classes in a Bidder's Rule Library

As described previously, IntelliBid is designed to allow each bidder to define rules on his/her own site to implement his/her bidding strategies. An essential component for this distributed bidding model is a proxy bid server executing at the bidder's site to transfer the newly calculated bid back to the Bid Server executing at the IntelliBid site. Thus, the actual transfer is transparent to the bidder side program that generates the new

bid. The transfer can be done either through Java's Remote Method Invocation (RMI) or XML-RPC over the Internet.

RMI allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine, providing a technique to build distributed applications. There are some security concerns using RMI, because of the distribution of a stub file on to a client machine. The stub file represents the server interface, and if modified may cause a breach in security.

XML-RPC is an extremely simple protocol. It uses XML to encode function calls and responses, and sends these messages over HTTP. XML-RPC leverages existing standards to create a basic remote procedure call protocol. This makes XML-RPC simple and appealing; however, there are some shortcomings in using XML-RPC. One limitation is that XML-RPC's marshalling is limited in the kinds of objects that it can pass to and from methods. In our example, the bid value calculated is actually of data type Float; but to enable the use of XML-RPC on the bidder side, we convert the value to a String, transport it as a String, and then on the IntelliBid auction site, we convert the String back to Float. The same technique is used to transfer the quantity of the product bid for, which is of type Integer, back to the IntelliBid server side. Following is an explanation of the different components in the bidder's rule library [JOS01].

5.4.1 EvaluateBidClientXML

This component consists of a method SendBid (productid, newbid, quantity, creditcard, exp_date), which is invoked in the sample IncreaseMyBid rule (Figure 3.1). This component contacts IntelliBid's Bid Server and transfers the bid back to the auction site.

The IntelliBid Bid Server's URL address is specified in a configuration file. The configuration file is a text file. In case of a change in the URL, it is sufficient to simply change the text file. This prevents the change of the EvaluateBidClientXML component and hence avoids recompilation [JOS01]. The protocol that this component utilizes is XML-RPC. To utilize XML-RPC, it is necessary to have an XML parser on both the auction site as well as the bidder side. This is also provided in the bidder's rule library.

5.4.2 EvaluateBidClient

This component also contacts the IntelliBid's Bid Server and transfers the bidder's bid to the IntelliBid auction site. It presents a similar SendBid (productid, newbid, quantity, creditcard, exp_date) method to the bidder, but the underlying transfer protocol for the process is Java's RMI. Also, for this component to work successfully, the necessary stub file – EvaluateBid_stub.class must be present in the bidder's rule library.

5.4.3 NewLimit

The purpose of this class is to dynamically change the upper limit of the bidder without stopping and restarting the bidder's ETR Server. Currently, the bidder's ETR Server does not support dynamic rule loading, unless the dynamic API call is used. Once it loads a copy of a rule, it uses that copy regardless of whether the rule has been changed and recompiled. This presents a problem of dynamically changing the bidding rule without restarting the ETR Server. It is not prudent to restart the ETR Server during the process of an auction since the bidder may have subscribed to different events. If the ETR Server is not active, then the bidder will not receive the event notifications.

To avoid these problems, a class has been provided in the bidder's rule library to enable dynamic changing of the bidder's upper limit without restarting the ETR Server. This is done by providing a limit in the form of a value in a text file. This class in the rule

library reads the value from the text file every time it is invoked. Changing the value in the text file dynamically changes the upper limit specified in the rule [JOS01].

This technique can also be used to dynamically change other parameters in the rule such as variables in the bid calculating formulas.

5.4.4 Screen1

This is a class provided on the bidder's side rule library to provide an additional facility for visual notification when a subscribed event has occurred. An example of the use of this class is in the rule IncreaseMyBid (figure 3.1). Even though e-mail is sent out to the bidder, the bidder is responsible for checking his/her e-mail account for this occurrence. A visual notification such as a pop up window would prevent such an effort on the part of the bidder. He/she can simply observe the window appearing on the computer monitor and obtain the necessary information about the bid status. The only requirement is that the bidder/supplier has to invoke the appropriate method from the class to observe the necessary information.

The window has also been provided a facility to interact with the text file referred to by the NewLimit class. Each window has been provided with a text field, which contains the value of the current upper limit. This value can be changed to any desired value, by entering the new value in the text field. By clicking the Change button, the value of the limit in the file will be replaced by the new specified value. From this point onwards, until a next change, the rule will utilize the new value of the upper limit.

The class has currently five methods, which can be invoked from the rule. These can be classified as below

1) IncBid–Pops up a window if the bid has been incremented for the bidder. This is useful if the bidder is currently bidding for only one product.

2) IncBidProductId (String productid)–From the rule, the productid has to be specified, as the window will report the result of incrementing a bid for the specific productid. This is useful if the bidder is bidding for more than one product simultaneously.

3) IncBidProductId (String productId, float newBidValue)–Specifying the value of the productid and the newBidValue from the rule will display these values in the window. This is useful if the bidder is bidding for more than one product simultaneously, and wants to keep in track of the new bid value that has been placed. This value will also help the bidder in deciding whether or not to increase his or her upper limit.

4) NoIncProductid()–Pops up a window if the bidder has been outbid, but the specified upper limit has been exceeded, and no higher bid was placed. This method is useful if the bidder is bidding for only one product, as no productid information is displayed. It should be invoked from the ALTACTION section of a rule, in case the next bid to be placed exceeds the bidder's upper limit.

5) NoIncProductId(String productid)–This method pops up a window if the bidder has been outbid, but once again the upper limit has been exceeded, and no higher bid was placed. But as the productid is displayed, it is useful if more than one product is being bid for simultaneously.

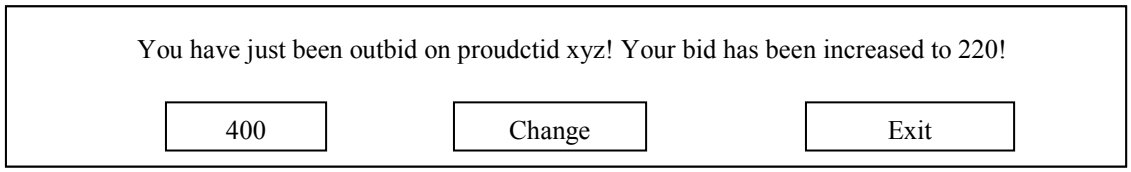

Figure 5.2 The Appearance of the Screen Class on a Bidder's Monitor

All the windows have their own separate exit button, and can be closed independently of each other. Figure 5.2 indicates the appearance of the window on the bidder's computer monitor. This window is observed on invoking the IncBidProductId (String productId, float newBidValue) method from the rule IncreaseMyBid. Even though all the five methods presented here are involved in the bidding process, the utility of a screen class is not limited to bidding. Additional methods can be added to this class to be utilized with other events. For example, it can provide a form of visual notification when a product of interest is registered with IntelliBid [JOS01].

5.5 HTML Files

These files provide the Web interface of the IntelliBid auction system. They basically are written in HTML, but some have been augmented with JavaScript. The following is a list of these HTML files, their purpose and details.

5.5.1 BuyerRegNew.html

It is an HTML form, which accepts information about a new bidder registering with IntelliBid. On submitting this form, the action goes to the NewBuyerServlet. This HTML page also provides a link for registering for the Product event, allowing for a new bidder to register for a product of interest right after the bidder registration.

5.5.2 SellerRegNew.html

It is an HTML form, which accepts information about a new supplier registering with IntelliBid. On submitting this form, the action is set to NewSellerServlet servlet.

5.6 Java Servlets

For the dynamic web server programming needs, we have chosen the Java Servlet architecture. Because servlets are written in Java, they can take advantage of Java's memory management and rich set of API's. Also, they allow the creation of the dynamic web server programs on platform independent systems. Servlets also solve the major problem of the CGI architecture: a servlet request spawns a lightweight Java thread, whereas a CGI request will spawn a complete process.

The following is a list of servlets used in the IntelliBid auction site, their purpose and details.

5.6.1 AdminMain_KT.java

This servlet represents a login screen for IntelliBid’s administrators to log into the administrator’s side of the web site and perform administrative tasks.

The administrator needs to enter his/her designated username and password to log on to the administrator's home page represented by the AdminHome_KT servlet.

- void doPost(HttpServletRequest request, HttpServletResponse response)--The form's action calls the AdminHome_KT servlet.

5.6.2 AdminHome_KT.java

This servlet receives the form submitted by the AdminMain_KT servlet and validates the username and password. If everything is fine, it provides a list of administrative tasks that the logged in administrator can perform as per his/her clearance level. Currently there are two designated clearance levels: Full and Category. The Full level allows the administrator not only to add new Categories and/or Specifications on the site but also allows him/her to add new administrators for the web site. The Category level on the other hand allows its holders only the access to the Category and Specification tasks.

- void doPost(HttpServletRequest request, HttpServletResponse response)--It sends a POM query to the Admin table and validates the user and if everything fine then displays the list of tasks; else it shows an error message so that the user can try again.

5.6.3 AdminAction_KT.java

This servlet is called when an administrator tries to perform any admin task that he/she is authorized to perform.

- void doPost(HttpServletRequest request, HttpServletResponse response)--It sends a POM query to the table related to the task to be performed and prints either a success or failure message once the query returns.

5.6.4 AuctionProductServlet_KT.java

This servlet is executed from the servlet NewProductServlet.java. Its function is to gather the information about a new product, which has been registered, post the Product event, and insert the Product object into the database. It is also responsible for

starting a timer until the product's time limit is finished. At that point, it posts the AuctionClose event for that product.

There is a class inside the AuctionProductServlet class, which is the timer thread.

- class RemindTask–This class has the following two methods.
    - RemindTask (String productid, Timer timer)--This method is a Constructor, which notes the productid of the specific product. It also takes as input the timer, which is an instance of the Timer object. This instance has been set to the required time limit, by using the Schedule method of the Timer class. Within the constructor, the IP address of IntelliBid is obtained. The Event Manager uses this IP address for posting an event [JOS01].
    - run()--When the timer has reached its zero count, this method is executed. Using the information obtained in the constructor of the RemindTask class, this thread posts the AuctionClose event using the Event manager's API. As this also marks the end of the auction, it uses the POM to update the product's status to "I" (Inactive). This prevents any more bids from being accepted. This method also identifies the supplier of the product, and obtains his/her e-mail (by querying the NewSupplier table). This information about the supplier of the product is presented in text format, and is sent by e-mail to the winner of the auction. Similarly, the e-mail of the winner(s) of the auction is/are obtained, and this information is e-mailed to the supplier of the product. Finally, this instance of the timer is canceled, to prevent any memory leak [JOS01].

The main method of this servlet is the doPost method described below.

- void doPost(HttpServlet request, HttpServletResponse response)--This method is executed when the HTML form of the NewProductServlet is submitted. It collects the supplied information by the supplier from the HTML form. It then obtains the current date and time, to note when the auction started. Also, from the form, the date and time when the auction will end is noted. The difference between these two times is calculated in milliseconds, and a timer is set to this value. It then creates a unique product id using the first letter of the supplier's username and appending the date-timestamp to it. It then checks the quantity of product submitted for auctioning and accordingly sets the auction type to *English* if quantity is "1" else *Dutch* if quantity is more than "1."

Now before inserting the data into the database using the POM, it checks to see if all the necessary fields have been filled up. If not, it prints an error and allows the user to fill all the required fields and try again. Once the information is inserted into the Product

table, the Event Manager's API is used to post the Product event by forking a thread by calling the class ProductThread.

5.6.5 BidServlet_KT.java

This servlet is called when a new bid is placed or submitted using the ProductDetails_BidsServlet servlet. It has one public method.

- void doPost (HttpServletRequest request, HttpServletResponse response)--This method is executed when the HTML form related to placing bids is submitted. The method's primary function is to gather the data submitted by the form, and register the bid for the product [JOS01]. If any of the form values are not blank, then these values are used to form an instance of the EvaluateBid_KT object, found in the IntelliBid Server's RuleLib package which in turn validates the values submitted as explained in Section 5.2 earlier. It then returns a string indicating whether or not the bid has been accepted. This result is then displayed on the browser. In case any of the fields of the submitted form are empty, a message is displayed regarding this matter.

5.6.6 CategoryServlet.java

This servlet is called through the IntelliBidHome_KT servlet when a web surfer wants to view the items on auction under a particular category.

- void doPost (HttpServletRequest request, HttpServletResponse response)--This method sends a POM query to the Product table and displays all the products currently on auction under the category selected.

5.6.7 IntelliBidHome_KT.java

This servlet represents the homepage of the IntelliBid Auction web site. It displays the list of all the categories in the form a side menu bar, displays at the maximum ten products currently on auction in the middle and displays a list of useful links and user login option on the other side.

- void doPost (HttpServletRequest request, HttpServletResponse response)--This method sends a POM query to the Product table and displays at the maximum 10 latest products currently on auction with a short description and a short image beside each. To view more details, anybody can click on the image and the ProductDetails_BidsServlet is called which provides detailed description of the product.

In case any of the fields are left blank, the above steps are not followed, but a message of blank fields is displayed on the supplier's browser [JOS01].

5.6.10 ProductDetails_BidsServlet

This servlet is called from several places throughout the web site wherever the name of a product appears. One principal place from where it would be called the most would be the IntelliBidHome_KT servlet to view detailed description of the product on auction. The servlet has one public method:

- void doPost (HttpServletRequest request, HttpServletResponse response) – This method's main function is to display the information about the specified product in HTML form. This information is spread out in two tables in the IntelliBid database - Product and Bids. Hence two Select queries are necessary to obtain the required information. The first query to Product obtains information about the product such as the product's specifications, detailed description, start and end date, bigger picture, if available, and minimum incremental price. This information is then formatted in HTML form and added to the output buffer. The second query to the Bids table is to obtain any information regarding bids that have been placed. If the select query returns no result, then a message that no bids have been placed is added to the output buffer and then displayed on the browser. If bids have been placed, then a list of all the bids is displayed in the form of an HTML table.

This servlet also displays a section that can assist bidders and/or sellers to subscribe to some events like the Product event that notifies its subscribers when a similar product registers on IntelliBid. Users can subscribe to events provided by IntelliBid by check marking them from the list and providing their own username, password and email address. On submitting the event subscription form, CheckOptions_SubscribeEvents_KT servlet is called, which in turn looks for the check marks passed to it and inserts the subscriber's and event's data in a table into the database by issuing an insertion command to POM.

5.6.11 ProductEventForm.java

This servlet is called either from UserHome_KT or from IntelliBidHome_KT servlet. It provides a form to a user to subscribe to the Product Event by selecting a particular category and multiple corresponding specifications. It has one public method:

- void doPost (HttpServletRequest request, HttpServletResponse response) – This method displays the form and when the form is submitted, calls CheckOptions_SubscribeEvents_KT servlet to subscribe to the Product event. It also allows a user to select multiple specifications under a particular category. For example, a user may want to get notified when a new product is put up under the category: "Automotive" with specification: "Honda" or specification: "Toyota". A user can do that by marking the checkboxes for "Honda" and "Toyota" under the "Automotive" category through this servlet.

5.6.12 ProductStatsServlet.java

This servlet is called either from the IntelliBidHome_KT servlet or ProductDetails_BidsServlet servlet. It provides graphical statistical representation of the bidding history for a category and/or specification. The servlet has one public method:

- void doPost (HttpServletRequest request, HttpServletResponse response) – This method displays two drop-down boxes, one to select a product category and other to select a product specification. It also provides a text box to enter a new specification of a product category that is not present in the drop-down for specifications. By choosing a specific category, the specifications drop-down will populate automatically to show the specifications related to that category. In order to view any statistical information related to a particular product, user may choose any category and specification or may just choose a category. Then when he/she submits the selection, this same method gets called again but now below the drop-down boxes the product's statistics are shown. Product statistics like average of the winning bids, averages of the base prices, average of the minimum incremental prices of similar products whose auctions have been completed are calculated and shown. Furthermore, as any of the winning bid or base price or minimum incremental price might have been same for many products, such information is depicted graphically in the form of horizontal histograms of "*". Such information can help in understanding which values were used more frequently by various bidders and sellers. Depending on the minimum and maximum values from a set of values got back from the database, an appropriate value for each "*" is computed using the mechanism shown in Figure 5.3 below. This value is shown below each graph for a better understanding.

```
// myMax is the max value from the set of values obtained from the database
// max = 100.0, min = 1.0 and barWidth = 10.0
// duplicateCount represents the number of duplicates present for a value within
// the set of values

mappedValue = (duplicateCount / myMax) * 100

tempVar = ( (max - min) / barWidth );

starValue = new Float(duplicateCount /(mappedValue / tempVar))
```

Figure 5.3 Formula to calculate the value of each "*" to plot a graph of Product Statistics

The values "max", "min", and "barWidth" are constant as shown. All the values are eventually mapped to the values of "min", "max" and "barWidth" by first mapping them to their respective "myMax" to get "mappedValue". The above formula is then utilized to calculate the value each "*" will represent and then the graph is plotted accordingly with minimum one "*" (as min = 1.0) and maximum ten "*" (as barWidth = 10.0).

A detailed explanation of the graphs and statistics is also presented in HTML form on the page itself so that users can easily understand what they are presented with and make the best use of it. Such information can prove to be extremely useful not only for bidders in bidding but also for sellers who want to know what kind of base price and/or minimum incremental price to set for their new product.

5.6.13 UserHome_KT.java

This servlet is executed when a supplier or a bidder is interested in viewing his/her specific information or his/her homepage. For a supplier, this information would include products submitted for auctioning. For a bidder, this information would include

bids that have been placed on specific products. It is called from the IntelliBidHome_KT servlet when a user fills out his/her username and password and tries to log in.

This class also has one public method:

- void doPost (HttpServletRequest request, HttpServletResponse response) - This method first queries the NewSupplier and NewBuyer table to set the respective newSupplier and newBuyer flags accordingly. It is possible that a person is registered both as a supplier and a buyer. In this case, both flags will be set to true. Once the flags are set, the queries are performed on the appropriate table to obtain the required information. For a bidder, a query is made to the Bids table, by setting the bids status to "A" (active) once, then the query is repeated with status set to "I" (inactive/outbid) and repeated again with bids status set to "W" (winning). This allows a separation of bids into active (ongoing), inactive (previously submitted bids and now outbid), and winning (bids that won auctions) bids on auction products [JOS01]. This information is then presented in the form of three different HTML tables. Similarly for a supplier, a query is performed on the Product table, by setting the auction product status to "A" and then repeating with the status set to "I." This separates the products into active (auction ongoing) and inactive (completed auction) products [JOS01]. This information is then presented in the form of two different HTML tables.

5.7 IntelliBid Server Rule Library

The Rule library on the server side of IntelliBid contains some classes, which augment the auction process by providing additional facilities like auction history and e-mail notification.

5.7.1 AuctionHistory.java

This class is executed from the auction side rule - AuctionHistory. Its primary function is to gather the bid history of a particular product [JOS01]. It has the following methods:

- public AuctionHistory (String productid, String email) : This method is a constructor, which will obtain the productid and the email of the product and the subscriber of the rule AuctionHistory. It then calls the two methods GetHistory() and SendMail() of the AuctionHistory class.

- private void GetHistory() : This method performs queries on the bid table to obtain the bid history of the product in the form of a vector of bids.

- private void SendMail() : This method uses the previously obtained vector in GetHistory() to form a text message. It then sends the text message by e-mail to the subscribers, using the NotifyAuctionMail class.

5.7.2 BidInfo_KT.java

This class is executed from the auction side rule – SendBidInfo. Its primary function is to notify all the subscribers of the MonitorBids event about a new bid placed for the specified product. It also provides a list of all the previously placed bids for the product. It has the following methods:

- public BidInfo_KT (String productid, String email) : This method is a constructor, which will obtain the productid and the email of the product and the subscriber of the rule SendBidInfo. It then calls the two methods GetBidInfo() and SendMail() of the BidInfo_KT class.

- private void GetBidInfo() : This method performs queries on the Bids table to obtain all the previous bids placed for the product and stores them into a vector.

- private void SendMail() : This method uses the previously obtained vector in GetBidInfo() to form a text message. It then sends the text message by e-mail to the subscribers, using the NotifyAuctionMail class.

5.7.3 MonitorBidsThread

This class is a thread, which is forked either from the EvaluateBid_KT.java's checkifOutBid() or CheckBid_Dutch() method, in case a new bid is placed for a product. Its primary function is to post the MonitorBids event for the produt on which the bid is placed. It has two methods:

- MonitorBidsThread (String pid, String s): A constructor, which obtains the productid of the product on which a new bid is placed and another string s which is the IP address of IntelliBid.

- public void run() : This method utilizes the API of the Event Manager for posting the MonitorBids event.

5.7.4 NotifyAuctionMail.java

All the e-mails that are sent in the process of IntelliBid auctions are sent using the Send method of this object. It has one method:

- public static void send (String email, String userid, String text, String mode) - This method consists of a case statement. Based on the mode passed into the method, a text message is prepared and an e-mail is sent using the SendAuctionMail class [JOS01].

5.7.5 OutBidThread.java

This class is a thread, which is forked either from the EvaluateBid_KT.java's checkifOutBid() or CheckBid_Dutch() method, in the case of a bidder being outbid on a product. Its primary function is to post the outbid event for the bid objects that have been passed into the thread in the form of a vector [JOS01]. It has two methods:

- OutBidThread(Vector o, String s) : A constructor, which obtains the vector of the bid objects to be outbid and a String, which is the IP address of IntelliBid.
- public void run() : This is a method which will iterate through the outbid vector and post the Outbid event for the bids in the vector. It utilizes the API of the Event Manager for posting the event.

5.7.6 SendAuctionMail

Using the mail server mail.cise.ufl.edu, e-mails are sent in this class. The administrator for the e-mails in this class is set to be IntelliBid, as the auction site is sending out the e-mails [JOS01].

5.8 Miscellaneous Classes

There are a few Java classes whose methods are called by several different Java Servlets and there are some classes that are instantiated in one or more Servlets. We shall look at those classes here.

5.8.1 HeaderClass.java

This class consists of some public methods that are called by most of the servlets described above. This class contains methods used for formatting the HTML content of the servlets such as the main header of IntelliBid and the title bar.

5.8.2 ProductThread.java

This class is a thread that is forked from the AuctionProductServlet_KT's doPost method. This class posts the Product event and notifies all of its subscribers about the registration of a new product. It has the following two methods:

- public ProductThread(String supplierid, String productid, String producttitle, String productcategory, String productspecification, String productdescription, String pictureurl, Float baseprice, Float minincrementalprice, String startingdate, String endingperiod, int quantity, String auctiontype, String status, String localIP): This is a constructor that obtains all the different values required for the Product event.
- public void run () : This method posts the Product event with all the values passed to the ProductThread. It uses the Event Manager API to post the event.

CHAPTER 6
PERFORMANCE ANALYSIS OF DEPLOYING MULTIPLE EVENT SERVERS FOR EVENT NOTIFICATION

As described in Chapter 3, a Distributor is responsible for forwarding the necessary information to all the Event Servers registered with it whenever an event is posted on the IntelliBid system. The Event Servers in turn send a query to their respective databases to check if that particular event has any registered subscribers. If subscribers do exist, then the Event Servers send out event notifications to those subscribers [HEN01]. The event notification process, not only has to send out emails to all its subscribers, which may be a large number, but it sometimes also has to gather and process a lot of information from the database. Because of this, the event notification process can take a lot of time compared to other activities on IntelliBid. Hence this chapter focuses on the performance implications of using multiple Event Servers under a single Distributor for event notification.

6.1 Performance Analysis using One, Two and Three Event Servers

To perform the performance analysis, we created separate test cases to do event notification using first one, then two and finally three Event Servers. Instances of the Product event have been used to gather the readings.

The steps followed for creating and running a typical test case for performance analysis are listed below:

1. Start the desired number of Event Servers (1, 2 or 3) under the same distributor.

2. Register a set of subscribers (100, 500 and 1000) to the same instance of the Product Event with a specific *productcategory* (e.g. “Computers”) and *productspecification* (e.g. “Keyboard”). All the event subscriptions get evenly distributed among various Event Servers that are registered under the Distributor.

3. Once the subscription to the event is completed, post the instance of the Product event with the product category and product specification previously used for subscription. The Distributor will notify all the Event Servers registered under it to notify the subscribers of this particular Product Event.

4. Note the total time taken from the point the Distributor starts notifying the Event Servers till all the Event Servers finish sending out notifications to all of the subscribers simultaneously. The time obtained in seconds would be the maximal time taken by all the Event Servers to send out the event notifications.

By following the steps 1 to 4 mentioned above, different sets of readings were taken with 100, 500 and 1000 subscribers of one instance of the Product event.

6.1.1 Performance Readings Taken for Different Sequential Runs

In this section, we shall explain the graphs plotted for various readings taken by posting an instance of the Product event sequentially, without any overlap between two consecutive posting of the same event, for 100, 500 and 1000 subscribers each.

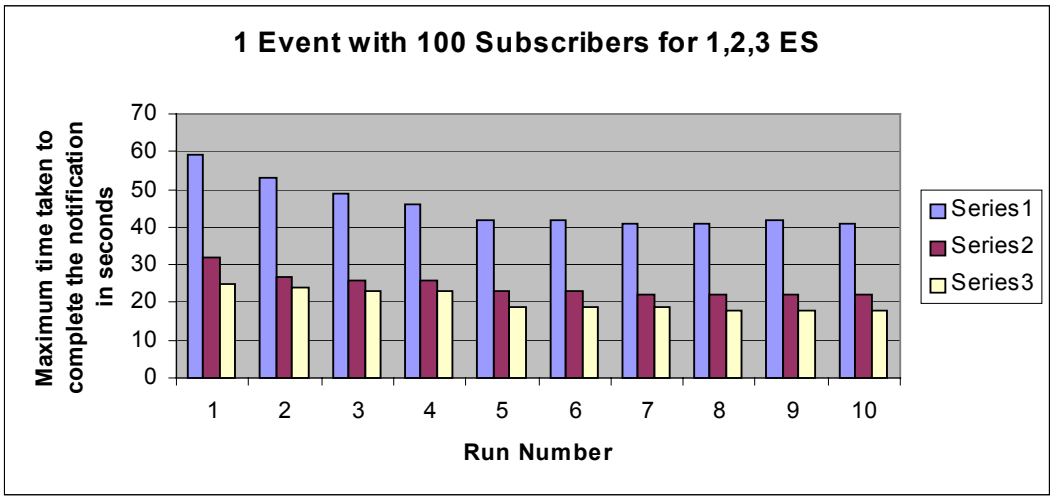

Figure 6.1 One instance of the Product event with 100 subscribers for 1, 2 and 3 Event Servers

The Figure 6.1 above shows the performance analysis of posting an instance of the Product Event with 100 subscribers each for 1, 2 and 3 Event Servers (ES) respectively. Due to the overhead generated by the distribution mechanism, the performance gain obtained by shifting from 1 ES to 2 ES is a little less than two times and the performance gain obtained by shifting from 1 ES to 3 ES is even lesser than three times. But as Figure 6.1 rightly depicts, for all the ten times that the event was posted, the ratio of performance gain obtained by adding more ESs remains constant which justifies the usage of more ESs.

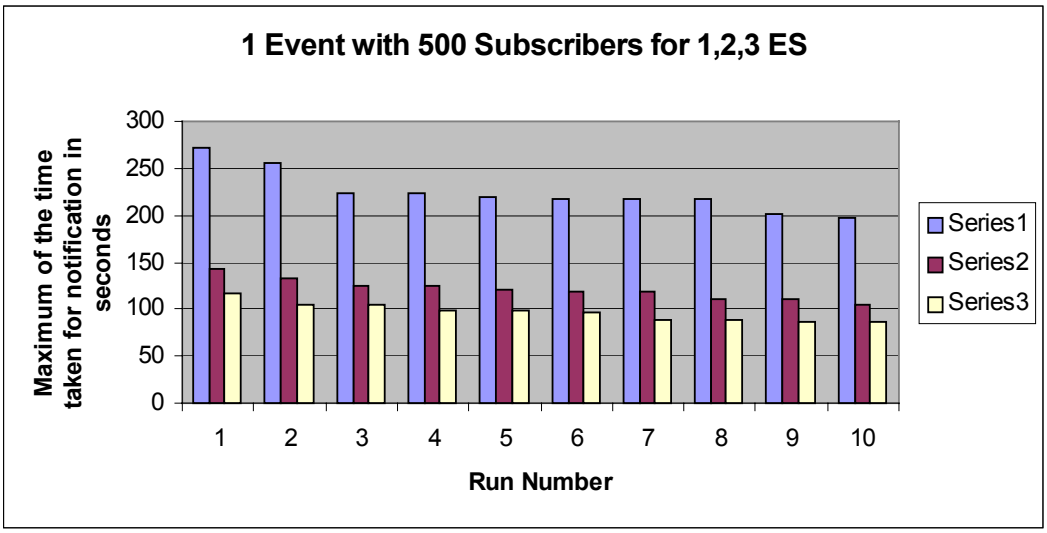

Figure 6.2 One instance of the Product event with 500 subscribers for 1, 2 and 3 ESs

The Figure 6.2 above shows the performance analysis of posting an instance of the Product Event with 500 subscribers each for 1, 2 and 3 ESs respectively. Comparing Figure 6.1 and Figure 6.2, one can see that the performance gain obtained by adding one more ES is slightly more in Figure 6.2 than in Figure 6.1. This difference in performance gain can be attributed to the increase in the load being dispersed to each ES. For Figure 6.1, there were 100 subscribers for 1 instance of the Product event, which resulted in

having approximately 33 subscribers per ES if 3 ESs were used. On the other hand for Figure 6.2, there were 500 subscribers and hence each ES got approximately 166 subscribers. This increase in the load on each ES helps reduce the negative effect produced by the overhead of having more ESs. This overhead cannot be completely negated and hence the performance gain obtained by shifting from 1 to 2 ESs is still not 2 times nor is the performance gain obtained by shifting from 1 to 3 ESs 3 times. But the performance gain has surely increased and is better than that obtained by having 100 subscribers.

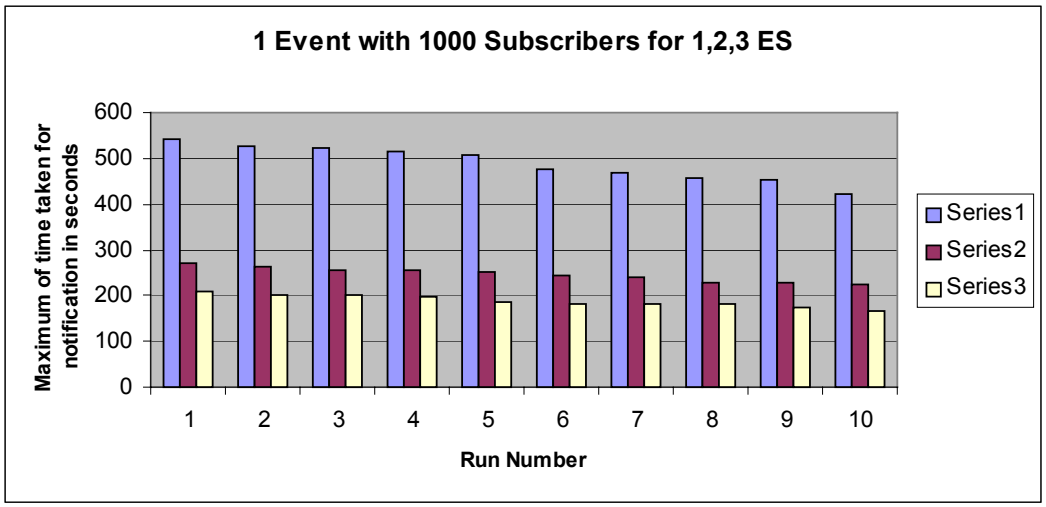

Figure 6.3 One instance of the Product event with 1000 subscribers for 1, 2 and 3 ESs

The Figure 6.3 above shows the performance analysis of posting an instance of the Product Event with 1000 subscribers each for 1, 2 and 3 ESs respectively. As in this case we have 1000 subscribers, each ES shared the load of approximately 333 subscribers and, once again, the increase in the load reduces the negative effect of the overhead generated by having 3 ESs. Hence, in this case, the performance gain obtained by shifting from 1 to 2 ESs is very close to 2 times and that obtained by shifting from 1 to 3 ESs is very close to 3 times higher than the performance obtained by using just 1 ES.

6.1.2 Performance Readings Taken for Different Parallel Runs

In this section, we shall first look at the graphs plotted for various readings taken by posting two instances of the Product event, with some degree of parallelism, for 1000 subscribers each. We shall then analyze the readings obtained by steadily increasing the parallelism between two instances of the same event to get a better picture of the performance gain obtained by using multiple ESs.

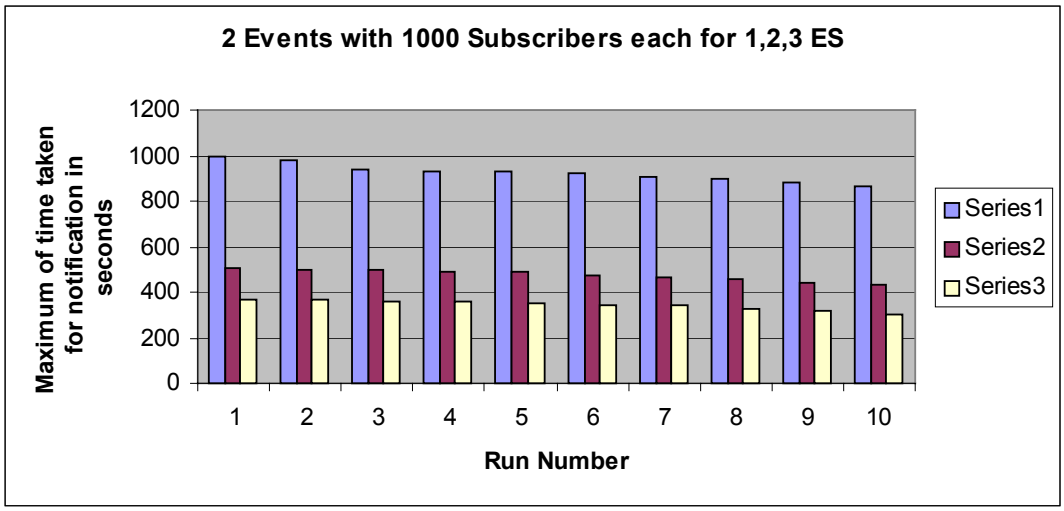

Figure 6.4 Two instances of the Product event with 1000 subscribers each for 1, 2, 3 ES

To obtain the readings for Figure 6.4 above, two instances of the same Product event, with 1000 subscribers each, were posted approximately at the same time by spawning separate threads. This doubles the burden on each ES, as now they have to notify all the subscribers twice. Again due to the increase in the load on each ES, the performance gain obtained by adding more ESs is apparent as seen from Figure 6.4.

To obtain a different type of performance we continually increase the degree of parallelism between posting of two instances of the same Product event for 1, 2 and 3 ESs respectively. Figure 6.5 below depicts the result of the performance analysis in such an experiment.

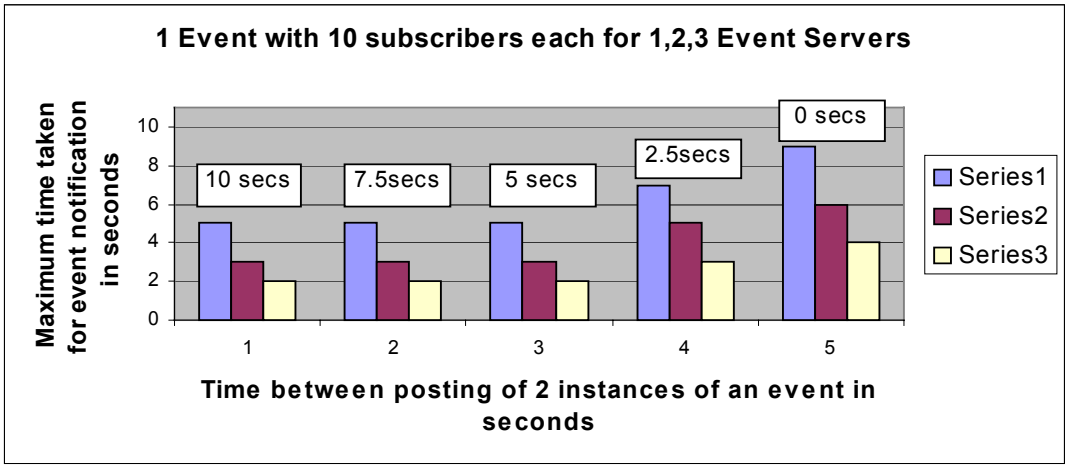

Figure 6.5 Two instances of the Product event posted with varying degree of parallelism

Figure 6.5 shows the comparison chart obtained by posting two instances of the same Product event, with 10 subscribers each, 10 seconds, 7.5 seconds, 5 seconds, 2.5 seconds and 0 seconds apart. A typical Product event when posted with 10 subscribers takes approximately 5 seconds with 1 ES, 3 seconds with 2 ESs and approximately 2 seconds with 3 ESs. Hence by decreasing the time between the postings of two instances of the same event, we increase the degree of parallelism to perform this experiment.

It becomes quite apparent from Figure 6.5 that as the degree of parallelism increases, the time taken to complete an event notification also increases. This can be attributed to the fact that as the degree of parallelism increases, the load on each ES also increases sooner than it would otherwise. For example, when we post two instances of the same event with 0 seconds apart or rather at the same time, each ES would have to execute multiple select queries to their respective databases approximately at the same time to get the subscribers for both posted event instances. This slows down the process of event notification. But once again as seen from Figure 6.5, this increase in time is seen less with 2 and 3 ESs than with 1 ES. Hence this again would justify having multiple ESs registered under a Distributor for faster event notification.

In all the figures of this chapter, the initial readings taken for 1, 2 or 3 ESs are, more often than not, higher than the later ones. This is due to the fact that, after the event posting code has executed once, the list of subscribers obtained from the database is brought into the memory from secondary storage and is buffered. Hence all the future postings of the same instance of the event execute faster as they do not have to fetch the information again from secondary storage. Moreover, during the performance analysis, issues like network bandwidth variation and load on the email server were taken into account and hence the occasional out-of-the-ordinary reading has been omitted in the graphs of the above figures to avoid ambiguity.

The advantages of using multiple ESs for event notification become obvious from the above figures and discussion. In spite of the overhead generated by having more ESs registered with the Distributor, they render a significant performance gain. Moreover, as this advantage becomes more apparent with higher load on each ES, for a system like IntelliBid Auctions where the amount of events being posted can be a large number, multiple ESs can help speed up the event notification process considerably.

CHAPTER 7
SUMMARY AND FUTURE WORK

In this thesis, we have presented the concept and the technique of conducting event-trigger-rule-based auctions over the Internet. IntelliBid, an Internet-based auction system, has been implemented to test and demonstrate the concept and technique. A network of Knowledge Web Servers, each consisting of a Web server, an ETR Server, an Event Engine, a Knowledge Profile Manager, a Metadata Manager, and Bid Servers and their proxies, constitutes IntelliBid. By replicating the Knowledge Web Server at multiple sites, its components provide various services to the creator of an auction site, the bidders, and the suppliers of products. IntelliBid also offers a number of advantages over the existing Internet-based auction systems. First and foremost is the flexibility offered to bidders for defining their own rules to control their bids in an automatic bidding process, which frees the bidders from having to be on-line to place bids. By using different rules, the bidders can apply different bidding strategies. These rules can be constructed using a product's previous auction history statistics provided by IntelliBid. Second, since rules that control the automatic bidding are installed and processed at bidders' individual sites, bidders' privacy and security are safeguarded. Third, IntelliBid's event, event filtering, and event notification mechanisms keep both bidders and suppliers better and more timely informed of events of interest so that they or their software system can take the proper actions in an auction process. Fourth, both bidders and suppliers can have access to the bidding history of a product maintained at the auction site. The information is useful, for example, to a supplier in setting his/her minimal price of a product and to a

bidder in deciding his/her maximal bid or the rate of bid increment. Fifth, IntelliBid allows bidders to do both on-line (or manual) bidding and automatic bidding. It also allows a bidder to participate in several auctions at the same time in both manual and automated modes. The bidding of a product can depend on the result of the bidding of another product. Sixth, IntelliBid also allows both bidders and suppliers to be notified whenever a new bid is placed for a product registered for auctioning on IntelliBid. Last, but not least, IntelliBid allows a person or organization to play both the role of bidder and the role of supplier simultaneously.

This work is a continuation and extension of the work previously done by Nicky Joshi [JOS01] and Agus S. G. Hendro [HEN01]. The features added as a result of this thesis work are listed below:

- Professional web sites look and feel with dynamic HTML, JavaScript and radiant colors.
- A Product table is introduced to replace the old AuctionProduct table to store product information.
- IntelliBid now supports English as well as Dutch auction mechanisms.
- Events
    - Product – The old AuctionProduct event has been replaced by the Product event to accommodate the new Product table.
    - MonitorBids – A new MonitorBids event has been added which informs its subscribers whenever a new bid is placed for a particular product of interest.
- Rules
    - BidInfo_KT – This rule is triggered whenever the MonitorBids event is posted to gather information about the new bid and notify the subscribers of the event.
- Web site administration side – A whole new separate administration site has been added to assist IntelliBid's administrators to not only add a new Category and/or specification, but also to add a new administrator.

- More dynamic data – Most of the data seen on the IntelliBid site now comes from the database.
- Product statistics – Any user on IntelliBid can view various statistics related to a specific product whose auction has ended.
- Parallel Architecture of Event Servers for event subscription and notification – The architecture developed reported in [HEN01] has been modified in order to optimize the overall parallel processing of event subscriptions and event notifications and also to incorporate as many as three Event Servers under a single Distributor for IntelliBid.

In spite of the above-mentioned features and advantages of IntelliBid, some future issues can be addressed. Currently when a supplier wants to put up a product on IntelliBid, he/she can put it under a category and a specification (subcategory). For example if say some supplier S wants to put up his Honda on IntelliBid, he can select the category: Automotive and the specification: Honda while registering the product on IntelliBid. But other than that, the supplier is not required to provide any specific information related to the product, as in this example, the car's type (Accord/Civic etc.) or the car's color (Red/Blue/Black etc.) etc. Future work should be concentrated on extending the system to support various attributes related to various categories on IntelliBid. This can help the suppliers to provide more specific information about their product and also the bidders to locate their specific product more accurately. IntelliBid currently supports the English and the Dutch auction mechanisms only. IntelliBid is designed using the classic object-oriented programming technique. Hence simply by augmenting the class which implements the bidding rules for the Dutch and English auctions, other auction mechanisms like the Reserve Price and Private Auction can be easily added and should be considered for the future. Moreover, currently IntelliBid allows a product's supplier to specify URL for an optional image of their product. Possible ways of developing a component to upload an image file should be explored. It

would allow suppliers to simply upload the image file related to their product to IntelliBid so that they themselves do not have to worry about hosting the image. Moreover, currently the user interface provided by the Knowledge Profile Manager to define a custom rule, requires its users to posses some working knowledge of Java. Some future work can be targeted to provide IntelliBid's users with some simpler ways of defining at least some basic rules so that users without any knowledge of Java can also define a rule to automatically increase their bid by just checking some options and/or clicking some buttons.

LIST OF REFERENCES

[AMA01] Amazon, "Amazon.com – Earth's Biggest Selection," http://www.amazon.com, 10/15/01

[AUC01a] AuctionWatch, "AuctionWatch.com," http://www.auctionwatch.com, 08/26/01

[AUC01b] AuctionWatch, "History of the Auction," http://www.auctionwatch.com/awdaily/features/history/7.html, 06/10/01

[BEA96] Beam, C., Segev, A., and Shanthikumar, J., "Electronic Negotiation through Internet-based Auctions," Technical Report, University of California at Berkeley, 1996

[CHA94] Chakravarthy, S., Anwar, E., Maugis, L., and Mishra, D., "Design of Sentinel: An Object-Oriented DEBMS with Event-based Rules," Information and Software Technology, vol. 39, no. 9, pp. 555-568, London, Sept. 1994

[CNE02] CNET Auctions, "CNET Auctions: Buy and sell online," http://www.auctions.com, 02/02/02

[DAY88] Dayal, U., "The HiPAC Project: Combining Active Databases and Timing Constraints," ACM SIGMOD Record, vol. 17, no. 1, March 1998

[EBA01] eBay, "eBay – The World's Online Marketplace," http://www.ebay.com, 08/15/01

[FIS01] FishMarket, "The Fishmarket Project," http://www.iiia.csic.es/Projects/fishmarket/newindex.html, 08/10/01

[HEN01] Hendro, A. S. G., "A Scalable Event Service to support Internet-based E-Business Enterprises," Master's Thesis, University of Florida, 2001

[JAV01] Java, "Remote Method Invocation," http://java.sun.com/products/jdk/rmi/, 07/10/01

[JOS01] Joshi, N., "IntelliBid: An Event-Trigger-Rule based Auction system over the Internet," Master's Thesis, University of Florida, 2001

[KUM98] Kumar, M., and Feldman, S., "Business Negotiation on the Internet." IBM Institute for Advanced Commerce (IAC) Report, 1998. http://www.ibm.com/iac/reports-technical/reports-bus-neg-internet.html

[LEE00] Lee, M., "Event and Rule Services for Achieving a Web-based Knowledge Network," PhD dissertation, University of Florida, 2000. http://www.cise.ufl.edu/tech-reports/tech-reports/tr00-abstracts.shtml, TR 002.

[LEE01] Lee, M., Su, S. Y. W., and Lam, H., "Event and Rule Services for Achieving a Web-based Knowledge Network," to appear in the Proceedings of the First Asia-Pacific Conference on Web Intelligence (WI-2001), Maebashi City, Japan, Oct. 23-26, 2001.

[MCA87] McAfee, R. P., and McMillan, J., "Auctions and Bidding," Journal of Economic Literature, vol. 25, no. 2, pp. 699-738, June, 1987

[NAT01] National Consumers League, "Welcome to the National Consumers League," http://nclnet.org/onlineauctions/auctionsurvey2001.htm, 01/15/02

[ONS01] Onsale, Egghead.com – Homepage, http://www.onsale.com, 03/15/01

[PAR99] Parui, U., "Knowledge Profile Manger for Supporting Event-trigger-rule services on the Internet," Master's Thesis, University of Florida, 1999

[POI01] PointCast, "Infogate," http://www.pointcast.com, 10/26/01

[SHE01] Shenoy, A., "Persistent Object Manager," Master's thesis, University of Florida, 2001

[STO88] Stonebraker, M., Hanson, E.N. and Potamianos, S., "The POSTGRES Rule Manager," IEEE Transactions on Software Engineering, vol. 14, no, 7, pp. 897-907, July 1988

[SU97] Su, S. Y. W. and Yu, T. F., "Distributed Information Mediation and Query Processing in a CORBA Environment," International Symposium on Digital Media Information Base, Nara, Japan, pp. 120-131, Nov. 26-28, 1997

[SU00] Su, S. Y. W. and Lam, H., "Iknet: Scalable Infrastructure for Achieving Internet-based Knowledge Network," Invited paper, Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, l'Aquila, Rome, Italy, July 31-Aug. 6, 2000

[TSV97] Tsvetovatyy, M., Gini, M., Mobasher, B., Wieckowski, Z., "MAGMA: An Agent-Based Virtual Market for Electronic Commerce," Journal of Applied Artifical Intelligence, special issue of Intelligent Agents, vol. 11, no. 6, pp. 501-524, September, 1997

[USE01] UserLand Software, Inc., XML-RPC home page, http://www.xml-rpc.org/, 04/10/01

[WID96] Widom, J., "Active Database Systems: Triggers and Rules for Advanced Database Processing," San Francisco, CA. Morgan Kaufmann, 1996

[WUR98] Wurman, P., Wellman, M., and Walsh, W., "The Michigan Internet AuctionBot: A Configurable Auction Server for Human Software Agents," Proceedings of the Second International Conference on Autonomous Agents, pp. 301-308, Minneapolis, MN, May 1998

[YAH01] Yahoo!, Yahoo! Auctions: Auctions, http://auctions.yahoo.com, 08/15/01

BIOGRAPHICAL SKETCH

Kushal Thakore was born on November 2nd, 1977 in Ahmedabad, Gujarat, India. He received a bachelor's degree in computer science (cum laude) from Florida Atlantic University, Boca Raton, Florida, USA, in May 2000.

He worked full-time as a programmer with e-Integrators, Inc, Boca Raton, Florida, USA, from May 2000 till August 2000.

He joined the University of Florida in August 2000 to pursue a master's degree in computer engineering. He has been working part-time as a programmer with e-Integrators, Inc., during his study in the University of Florida. He was married to Payal Parikh in February 2002 while working on his thesis at the University of Florida.

His research interests include the fields of e-commerce, wireless networks and databases.