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

By

NICKY JOSHI

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

2001

Copyright 2001

by

Nicky Joshi

Dedicated to:
My dearest father and mother

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. Stanley Y.W. Su, chairman of my supervisory committee, for his continuous guidance and support throughout my study. I am very thankful to him for providing me with a challenging topic for my thesis and for his continuous feedback and support during the implementation and thesis writing. I also would like to thank Dr. Herman Lam and Dr. Abdelsalam Helal for serving on my supervisory committee and Dr. Joachim Hammer for his precious time.

I would like to thank our secretary Sharon Grant for maintaining an excellent work environment in the Database Center and for her help in correcting my thesis. I also thank Seokwon Yang, Anuradha Shenoy and Kushal Thakore for their help in implementing my thesis.

On a more personal note, I have to thank my family, without whose love and support, none of this would have been possible.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Figure Page

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

Nicky Joshi

August 2001

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

This thesis presents the design and implementation of an Event-Trigger-Rule-Based auction system called IntelliBid. IntelliBid is constructed by a network of Knowledge Web Servers, each of which consists of a Web server, an Event Manager, an ETR Server, a Knowledge Profile Manager, a Persistent Object Manager, a Metadata Manager, and Bid Servers and their proxies. Together, they provide auction-related services to the creator of an auction site and the bidders and suppliers of products. IntelliBid 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. Second, since the rules that control the automatic bidding are installed and processed by the ETR servers installed at bidders' individual sites, bidders' privacy and security are safe-guarded. Third, IntelliBid's event, event filtering and event notification

mechanisms keep both bidders and suppliers better and more timely informed of auction events so that they or their software system can take the proper actions in the auction process. Fourth, both bidders and suppliers can have access to the bidding history of a product maintained at the auction site. This 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. Last, but not least, IntelliBid allows a person or organization to play both the role of bidder and the role of supplier simultaneously. The Profile Manager keeps the information as a bidder and information as a supplier separately.

# CHAPTER 1
# INTRODUCTION

Auctions have been present in human society since the dawn of civilization. We have observed a great change in the procedures used to conduct auctions since then; and now Internet-based online auctions have become a very popular form of auction. In early 1995, the Japanese company Aucnet inaugurated the world of auctions on the Web, starting with online auctions of automobiles. Since then, Onsale [ON01] and eBay [EB01] followed with their online auctions 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 [AWH01].

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 [YH01] and Amazon [AZ01]. As of late 2000, there were over 400 sites offering online auctions, providing buyers and sellers with a multitude of choices. On observing all this growth, analysts predict that by the end of 2001, the online auction market will be a thriving $15.5 billion industry. This is not surprising when one considers the statistics of the large number of visitors to some of the popular auction sites. For example, in C2C auctions conducted by eBay, there were 8.5 million visitors to their auction website in November 1999. This number had increased to 12.5 million in November 2000, in just one year's time [AW01].

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." The auctioneer usually starts the auction with an initial offer, and then bidders submit their bids in response to the initial offer or bids from other bidders. The auction ends according to some established rules. Thus, auction is basically a form of negotiation, in which a fixed auction protocol is followed, e.g., English auction, Dutch auction, Vickrey auction, etc. [KUM98, BEA96].

We have observed some limitations in the latest form of Internet auctions. First and foremost is the question of the flexibility in specifying bids. Currently, to raise his/her bid, the bidder has to access the web site, access the web page of 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 notify the potential bidders when a particular product of interest is registered with their sites. For those that do, their notification is based on bidders' specifications of product categories rather than detailed descriptions of products. Also, most sites do not allow users to observe the history of bidding 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 information infrastructure provided by a Web-based Knowledge Network developed at the Database Systems R&D Center of the University of Florida [LEE00, SU00, LEE01]. The infrastructure provides services for 1) the specification, subscription, filtering, and notification of events that are relevant to Internet auctions, 2) the management and enforcement of rules that implement different bidding strategies, and 3) the management of triggers that tie events to rules.

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 auctions. It offers a much 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. 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. IntelliBid offers a capability for the bidders to specify their interests in products in greater details. This interest specification is not limited by the product category, but allows a more detailed specification. This site also offers a convenient feature for bidders as well as suppliers to obtain the bid history of products whose auctions have just concluded.

The remainder of the thesis is organized as follows. In Chapter 2, the auction process and limitations of existing online auction sites are presented. Chapter 3 describes the knowledge model used in the Web-based Knowledge Network and the architecture of IntelliBid. It also includes the steps followed for the creation and execution of an auction on IntelliBid. Chapter 4 describes some implementation details of IntelliBid. Finally, chapter 5 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. Some observed limitations of these auction sites are also explained.

## 2.1 Process of Internet-based Auctions

### 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, to find the desired product to bid on. Certain 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." Some sites also allow a keyword specification of a certain product that a person is interested in. The more keywords specified, the narrower the search is for an interested product, i.e., the keywords form a Boolean expression with logical AND operators.

### 2.1.2 Actual Process of Auctions

In the following description of the auction process, we shall only consider the most popular form of auctions today – namely, the English auction. Other forms of auctions, such as Dutch Auctions, are not very prevalent and are not considered here.

Users who wish to participate in auctions must first register with an auction site. 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, the users register as either suppliers or buyers with the auction site. The only difference is the validation of the person registering with the auction site. A supplier must provide a credit card, which must be validated; and only on approval, is the person a registered supplier with the auction site. The supplier then registers products for auctioning. He/she, along with a minimum incremental amount, specifies a minimum base price for each product. Each bid that is placed by a registered user must be at least the minimum base price, plus the incremental amount. Up to a certain time limit, which is specified by the supplier when the product is open for auction, each registered user can go 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 is 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; 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. Some auction sites, having an open suppliers policy, also take the responsibility for auctioning a variety of products. In this case, each site acts as the supplier; and, on the close of an auction, the site is responsible for shipping the product to the winning bidder.

### 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 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. then 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.

### 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, places where some form of knowledge can be applied to alleviate the limitations can be easily seen:

### 2.1.1 Selection of Desired Products for Users

The users should be allowed to more explicitly state the product that they are interested in. Product specification should not be limited to merely a simple category of products. It should include a detailed specification of the features of a product. For example, a user should be able to specify not just an interest in computers, but an interest in a Pentium III Processor 550MHz computer. He/she should be notified if such a product becomes available for auction. In our work, we use an object-oriented model for the specifications of products and event publishing, filtering, and notification schemes to inform users the availability of products of their interests.

### 2.2.2 Greater Flexibility and Privacy in Specifying Bids

Instead of using a constant increment in an automatic bidding process as supported by some auction sites, a bidder should have a greater flexibility in terms of the amount to increase in the process. For example, a bidder may want the system to automatically increase his/her bid by a larger amount than the minimum allowed, 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 More than One Site

The same logic for bidding for multiple products at one site can be applied for bidding for multiple products at multiple sites. This is possible if the rules for reacting to the outbid cases are maintained and enforced by an ETR Server installed at the bidder's site. The activation of a rule will automatically place a bid at another auction site.

### 2.2.5 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. This can also be done by the notification mechanism provided by our system.

### 2.2.6 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. The names of the bidders who placed those bids are omitted for privacy reasons. 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.

## 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. The prominent existing auction sites like e-bay and Yahoo do not provide a facility of using software agents for bidding in their auctions. However, there are a few existing auction systems available today that support a form of agent-based auctions. The most prominent of agent supporting auction sites is AuctionBot [WUR98].

AuctionBot is a 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 that are responsible for implementing the bidder's policies have to be implemented by the bidders themselves in programming languages (C++, Java or Mathematica) using AuctionBot's API. If the bidder does not have the knowledge of these programming languages or even have access to the programming software, then this technique proves to be a limitation to such bidders wishing to use software agents in auctions. Also, AuctionBot is based on the pull model, rather than the more efficient push model.

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.

MAGMA is an 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 multiagent 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.

# 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 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. Table 1 lists the events implemented in IntelliBid, their parameters, and the attributes associated with event filters.

Table 1. Implemented Events in IntelliBid

| Event Name | Parameters | Parameter Type | Filter | Description |
|---|---|---|---|---|
| AuctionProduct | ProductId<br>SupplierId<br>ProductCategory<br>ProductSpecification<br>ProductDescription<br>BasePrice<br>MinIncrementalPrice<br>Quantity<br>AuctionType<br>Status | String<br>String<br>String<br>String<br>String<br>Float<br>Float<br>Integer<br>String<br>String | On ProductSpecification and ProductCategory | Event posted when a product is registered with IntelliBid |
| OutBid | ProductId<br>BidderId<br>AuctionType<br>BidValue<br>MinIncrementalPrice | String<br>String<br>String<br>Float<br>Float | On ProductId and BidderId | Event posted when a bidder is outbid on 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.

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 [DA88, ST88, CHA94, SU97, WD96], 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. Table 2 shows two examples of rules implemented in IntelliBid.

Table 2. Example Rules in IntelliBid

| Rule Name | Side | Description |
|---|---|---|
| AuctionHistory | IntelliBid (Auction) | Used to gather information about a product's bid history |
| 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 1. It should be pointed out that a more complex rule, that makes use of a more sophisticated formula to calculate a new bid, can 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 < RuleLib.NewLimit.limit() |
| **ACTION** | newbid = oldbid + 2 * minincrementalprice<br>RuleLib.EvaluateBidClientXML.sendbid(productid, newbid, quantity, creditcard, exp_date)<br>RuleLib.Screen.IncBid(productid) |
| **ALTACTION** | RuleLib.Screen.NoIncBid(productid) |

Figure 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 shows two example triggers.

Table 3. Example Triggers in IntelliBid

| Trigger Name | Event | Rule | Side |
|---|---|---|---|
| OutBidIncreaseMyBid | OutBid | IncreaseMyBid | Bidder |
| AuctionCloseSendAuctionHistory | AuctionClose | SendAuctionHistory | IntelliBid |

## 3.2 Architecture of IntelliBid

The architecture of IntelliBid is shown in Figure 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: an Event Manager, an Event-Trigger-Rule (or ETR) Server, a Knowledge Profile Manager, 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. Although we separate bidders from suppliers in Figure 2, a 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.

### 3.2.1 Event Manager

The auction site's Event Manager is responsible for accepting event registrations from bidders/suppliers and delivering the events (i.e., event notification) to them. The Event Manager provides an event registration facility for bidders/suppliers to subscribe to the auction related 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.

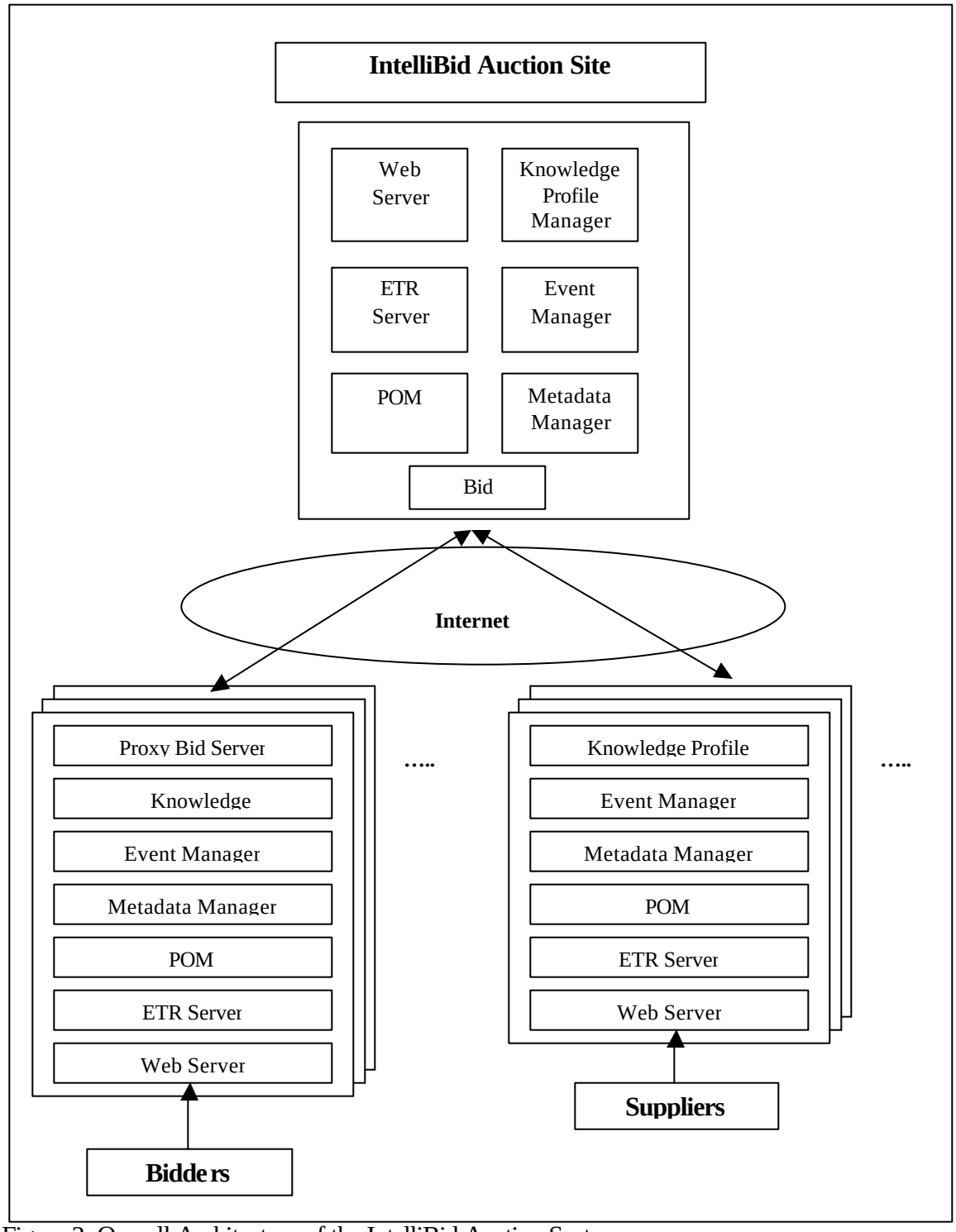


Figure 2: Overall Architecture of the IntelliBid Auction System

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 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. It also performs event filtering before notifying the bidders and/or suppliers.

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) AuctionProduct: 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 AuctionProduct: ProductCategory and ProductSpecification. For ProductCategory, a bidder can select a single value from a list of values, such as computers, electronics, cars, etc., provided in a form which is generated from a filter template in XML format, as shown in Figure 3.

For productspecification, the bidder can provide values for multiple attributes that specify a product. Some attributes may have multiple values or ranges of values. For example, a bidder may consider to bid for different models and makes of TVs within the price range of $1,000 to $2,000.

```
<EVENTFILTER>
<DESCRIPTOR>Event posted when an auction product is
registered with the system</DESCRIPTOR>
<EVENTNAME>AuctionProduct</EVENTNAME>
<SINGLESELECTIONLIST>
  <ATTRNAME>productcategory</ATTRNAME>
  <TYPE>String</TYPE>
  <LISTVALUES>computers, electronics, cars </LISTVALUES>
</SINGLESELECTIONLIST>
<MULTIPLESELECTIONLIST>
  <ATTRNAME>productspecification</ATTRNAME>
  <TYPE>String</TYPE>
   <LISTVALUES>pentium, pentium2, pentium3, oracle, Sybase,
DB/2,Honda, Toyota, stereo, TV, CDPlayer</LISTVALUES>
 </MULTIPLESELECTIONLIST>
</EVENTFILTER>
```

Figure 3: Filter Template in XML Format Defined for the AuctionProduct Event

2) 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.

3) 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. A bidder will specify a unique product identifier for this attribute.

### 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 inputted 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 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.

### 3.2.4 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 4 is a sample of a configuration file.

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

Figure 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, all of the 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 empty, and 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 are needed 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; it's log files, and related information. If the database already exists, then POM directly proceeds to perform the requested operation on the auction database.

### 3.2.5 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 Manger. 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.

### 3.2.6 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 [RMI01] technology and the other using XML-RPC [XML01]. 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.

### 3.3 Steps for the Creation and Execution of an Auction Service in IntelliBid

Before describing the steps for the creation and execution of an auction, we first discuss IntelliBid's policy of Internet users as suppliers. By registering as a supplier with the auction site, the Internet users are capable of auctioning products that they are interested in selling. They become responsible for the product, its description and information. Once the auction site has completed the auction on the supplier's product, it is the supplier's responsibility for shipping the product to the winner of an auction. The

only risk in this policy is that the supplier is trusted to be genuine. Precautions are taken in the form of credit card validation of the supplier who registers with the auction system. Also, the supplier is responsible for providing the charges related to the auction. IntelliBid has been constructed based on this model of open suppliers.

To construct an auction service in IntelliBid, the following steps can be followed. Figure 5 gives a pictorial view of the steps involved in the process.

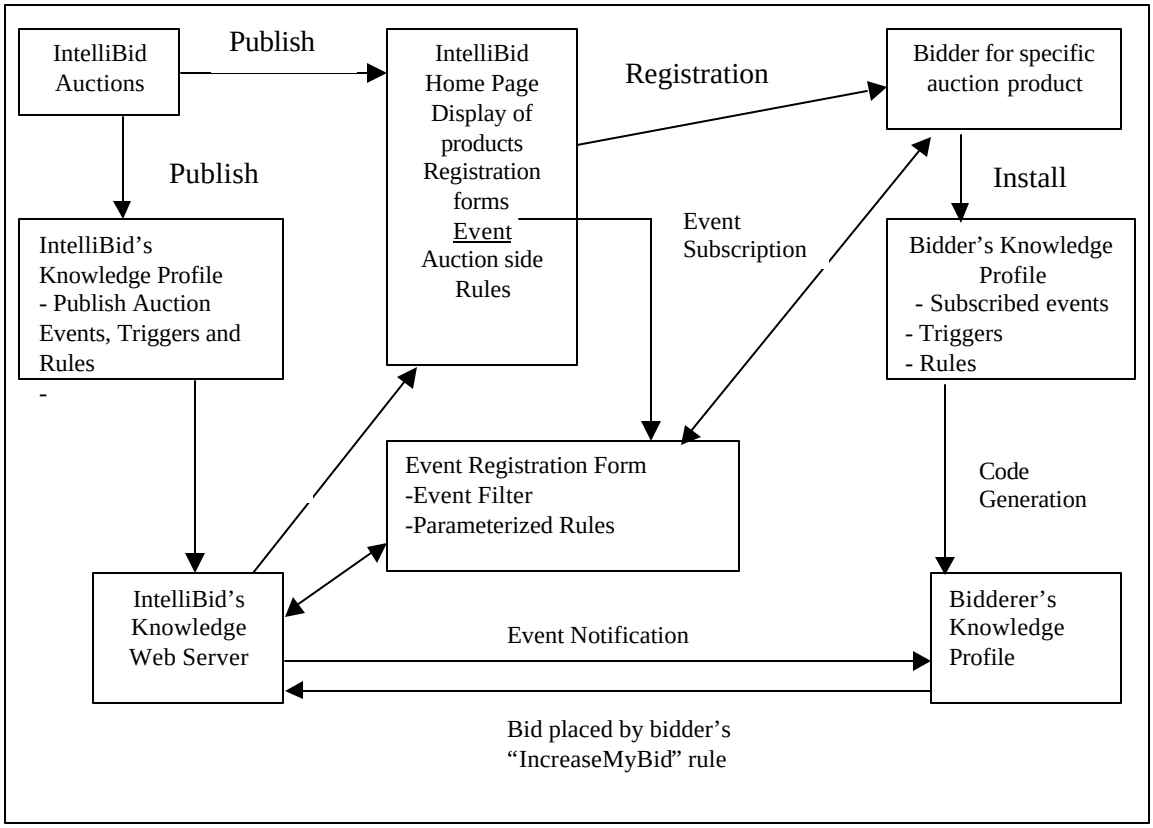


Figure 5: Steps in Creation and Execution of IntelliBid

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.

2) General registration and AuctionProduct event registration by a bidder/supplier: Before any person can take part in the auctions, he/she must be appropriately registered with IntelliBid; i.e., a person who is interested in submitting their own products for auctioning must register as a supplier with IntelliBid. Similarly, before anybody can bid for a product, they must register as a bidder.

The process of general registration is as follows: 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. A credit check is performed on the suppliers during registration with IntelliBid. Only on completing the credit check successfully is a supplier allowed to supply products. A credit check is not performed on bidders during registration. After registration, the bidder/suppliers can subscribe to IntelliBid events. During event registration, the values for "installing" a filter based on a pre-defined 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 "AuctionProduct" event. He specifies his interests by installing a filter for the "AuctionProduct" event.

3) AuctionProduct event generation and filter processing on IntelliBid: Suppose that, in our scenario, supplier S submits a product 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. The supplier accesses the IntelliBid web site and submits their HTML form for an auction product. In the form, the supplier provides information about the product: its description, base price, minimum incremental price, and time limit of the auction. This registration of the product results in the posting of the AuctionProduct event. The steps in Figure 6 describe the activities of submitting a product for auction by a supplier. Once the AuctionProduct event is generated, it is given to IntelliBid's Event Manager. 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) AuctionProduct event notification to the bidders/suppliers: The bidders/suppliers that had subscribed to the AuctionProduct event and whose filters have passed the filter-processing step in step 4 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 [PC01]) and/or an e-mail notification to bidder A.

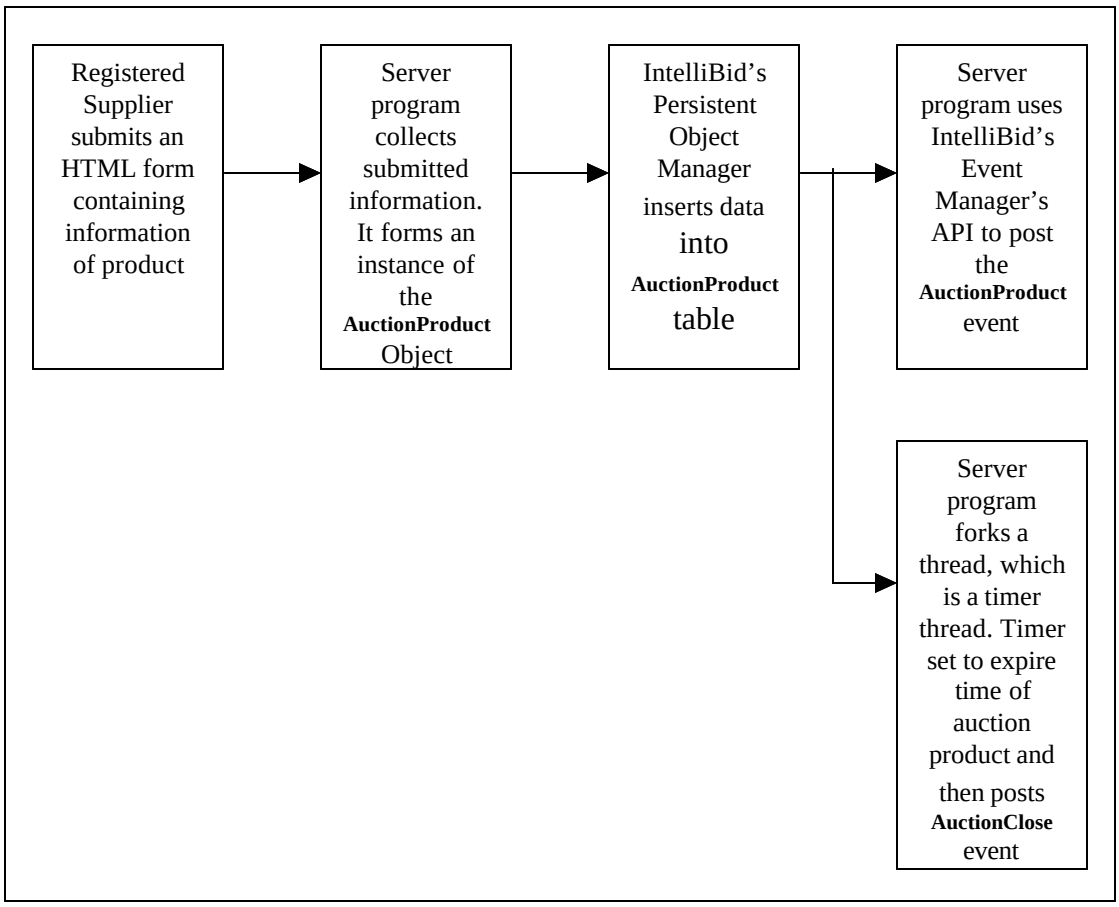


Figure 6: Steps Involved in the Process of a Supplier Submitting a Product for Auction in IntelliBid

5) Initiate bidding and subscribe to the outbid event: Once a product is registered, bidding can begin. IntelliBid provides two methods for placing bids on the auction product:

1) Manual (on-line) bidding: Use a Web browser to access the IntelliBid web site. Submit an HTML form, which contains the required bid information for a product.
2) 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.

On receiving the AuctionProduct 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. He is also interested in using bidder side rules for bidding, and hence subscribes to the "OutBid" event.

Following this, bidder B accesses the IntelliBid site, and also decides to participate in the auction of the same product. She places her bid by submitting the HTML form with appropriate information for bidding, at the IntelliBid web site. She also 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".

6) 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 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.

7) 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 filters. Her new bid is similarly placed since she also has defined a rule to implement her bidding strategy.

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.

8) 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.

# CHAPTER 4
# 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 browser 5.0 is also used. The Knowledge network components were installed and used for event subscription, event notification, and trigger and rule definition.

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 is the SAX parser. The pop-up windows on the bidder's side are implemented using Java's Abstract Window Toolkit (AWT).

## 4.1 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 four 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 4: Description of Objects in IntelliBid and their Attributes

| Object/Table Description of Object | Data Members | Type | Description |
|---|---|---|---|
| NewBuyer<br><br>This object/table contains the information provided by a new bidder when registering with IntelliBid. | b_fname | String | First Name of Bidder |
| | b_lname | String | Last Name of Bidder |
| | b_email | String | Email of Bidder |
| | b_comp_name | String | Company Name of Bidder |
| | b_phone_acode | String | Phone area code of Bidder |
| | b_phone_1 | String | First 3 digits of Bidder's phone |
| | b_phone_2 | String | Last 4 digits of Bidder's phone |
| | b_bill1 | String | Billing address 1 of Bidder |
| | b_bill2 | String | Billing address 2 of Bidder |
| | b_bcity | String | Billing city of Bidder |
| | b_bstate | String | Billing state of Bidder |
| | b_bzip | String | Billing zip code of Bidder |
| | b_bcountry | String | Billing country of Bidder |
| | b_ship1 | String | Shipping address 1 of Bidder |
| | b_ship2 | String | Shipping address 2 of Bidder |
| | b_scity | String | Shipping city of Bidder |
| | b_state | String | Shipping state of Bidder |
| | b_szip | String | Shipping zip of Bidder |
| | b_scountry | String | Shipping country of Bidder |
| | b_cc_type | String | Credit card type of Bidder |
| | b_cc_num | String | Credit card number of Bidder |
| | b_cc_name | String | Credit card name of Bidder |
| | b_cc_exp1 | String | Credit card expiry month of Bidder |
| | b_cc_exp2 | String | Credit card expiry year of Bidder |
| | b_userid | String | User ID of Bidder |
| | b_passwd | String | Password of Bidder |
| NewSupplier<br><br>This object/table contains the information provided by a New Supplier where registering with IntelliBid | s_fname | String | First Name of Supplier |
| | s_lname | String | Last Name of Supplier |
| | s_email | String | Email of Supplier |
| | s_comp_name | String | Company Name of Supplier |
| | s_phone_acode | String | Phone area code of Supplier |
| | s_phone_1 | String | First 3 digits of Supplier's phone |
| | s_phone_2 | String | Last 4 digits of Supplier's phone |
| | s_bill1 | String | Billing address 1 of Supplier |
| | s_bill2 | String | Billing address 2 of Supplier |
| | s_bcity | String | Billing city of Supplier |
| | s_bstate | String | Billing state of Supplier |
| | s_bzip | String | Billing zip code of Supplier |
| | s_bcountry | String | Billing country of Supplier |
| | s_ship1 | String | Shipping address 1 of Supplier |
| | s_ship2 | String | Shipping address 2 of Supplier |
| | s_scity | String | Shipping city of Supplier |
| | s_state | String | Shipping state of Supplier |
| | s_szip | String | Shipping zip of Supplier |
| | s_scountry | String | Shipping country of Supplier |
| | s_cc_type | String | Credit card type of Supplier |
| | s_cc_num | String | Credit card number of Supplier |
| | s_cc_name | String | Credit card name of Supplier |
| | s_cc_exp1 | String | Credit card expiry month of |

Table 4 continued

| | | | |
|---|---|---|---|
| | | | Supplier |
| | s_cc_exp2 | String | Credit card expiry year of Supplier |
| | s_userid | String | User ID of Supplier |
| | s_passwd | String | Password of Supplier |
| AuctionProduct<br><br>This Object/Table contains the information about a product open for bidding by a supplier. | Supplierid | String | ID of the supplier who has put up this particular product for auction |
| | Productcategory | String | Category of product to which it belongs – categories defined by IntelliBid and chosen by supplier |
| | Productspecification | String | Specification of product to which it belongs – categories defined by IntelliBid and chosen by supplier |
| | Productdescription | String | Textual description of product – description provided by supplier |
| | Baseprice | Float | Price at which supplier wants auction to begin |
| | Minincrementalprice | Float | A minimal increment for each bid – a supplier provided value |
| | Endingperiod | String | Time and date at which the product's auction should end. Specified by supplier |
| | Productid | String | System generated unique ID for each product |
| | Quantity | Integer | Quantity of product supplier is interested in putting up for auction |
| | Auctiontype | String | Type of auction desired. Values provided by IntelliBid, chosen by supplier |
| | Status | String | Value to indicate whether product is active or not |
| Bids<br><br>This Object/Table contains information about | Productid | String | ID of product being bid for |
| | Buyerid | String | ID of buyer placing the bid |
| | Datetime | String | Date and time when bid was placed |
| | Status | String | "A" if product is currently winning bid, "I" if product is a bid which has been outbid by another one |
| | Qty | Integer | Quantity that bidder has bid for |
| | Bidvalue | Float | Amount of bid |

## 4.2 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 only the English auction, but it is easily expandable to other types of auctions. For the English auction in IntelliBid, the auction logic is implemented in a class included in the auction site's rule library, called EvaluateBid. This class contains two methods in the auction process:

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.
2) CheckifOutbid() – After the bid is evaluated and has been validated as a legitimate bid, this method is executed. It checks if the bid is of a sufficient amount and quantity. This method follows the following algorithm:

    a) Obtain the total quantity of the bid-for product. If the quantity requested in the bid is greater than that of the maximum available quantity of the product, then reject the bid. The bidder can replace a bid with the correct quantity. This prevents acceptance of a bid, which is greater than the available quantity.

    b) 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.

c) 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 become outbid, the checkIfOutBid() method forks a thread, 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 events. 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.

## 4.3 Posting of IntelliBid Events

IntelliBid has three events in the present implementation. The method of the posting of these events is described below:

1) AuctionProduct – This event is posted when a supplier registers a product with the IntelliBid system, for the purpose of auction. For supplying the information of the product to IntelliBid, the supplier submits an HTML form. The HTML form's action is set to a Java Servlet, which collects the information of the product. This servlet posts the AuctionProduct event.

This event was extended with Get and Set methods for the AuctionProduct object. This is because this event also corresponds to a table in the auction database. This table is used for the persistent store of the product information.

2) OutBid event – This event is posted when a bidder involved in the auction process is outbid.

This event does not correspond to a table in the auction database and is therefore not extended with Get and Set methods. Figure 7 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 two methods of the EvaluateBid class to validate the bid and then complete the bidding process, as previously explained in Section 4.2. The second method is by the Bid Server, which also calls the same two methods of the EvaluateBid class, to evaluate bids placed through bidder side rules.

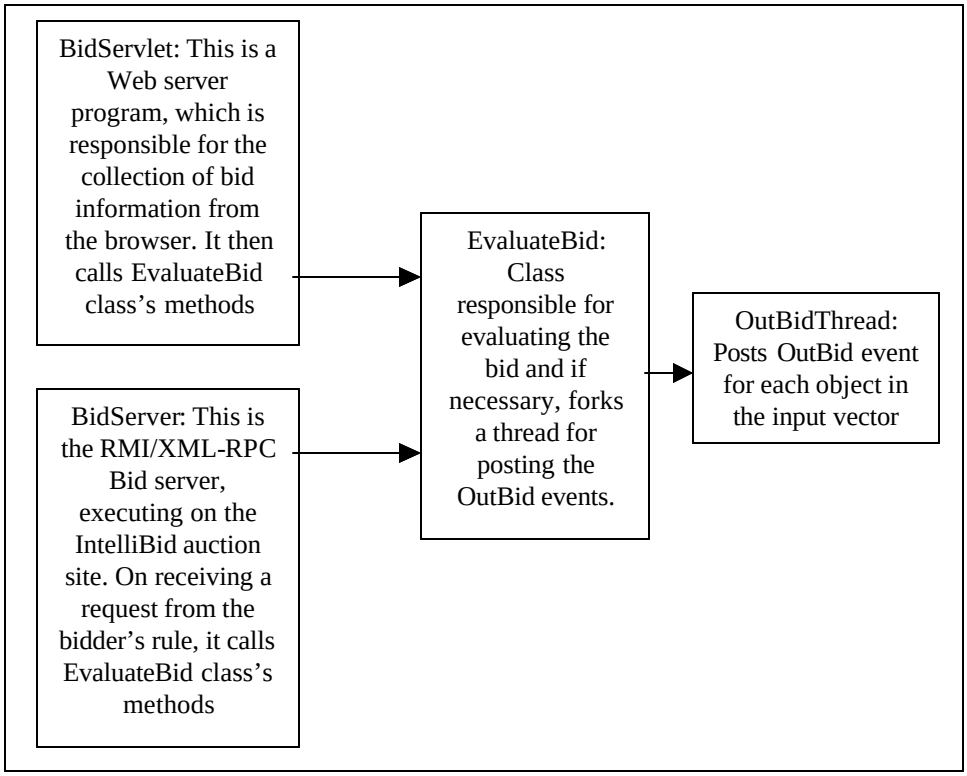


Figure 7: Steps Involved in the Posting of the OutBid Event

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

The supplier, which 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 AuctionProduct (Figure 6), 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.

## 4.4 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.

1) EvaluateBidClientXML – This component consists of a method SendBid(productid, newbid, quantity, creditcard, exp_date), which is invoked in the sample IncreaseMyBid rule (Figure 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. 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.

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.

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.

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

4) Screen – 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 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

a) 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.

b) 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.

c) 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.

d) 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.

e) 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.

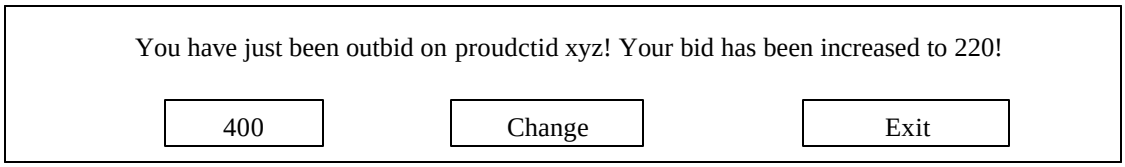


Figure 8: The Appearance of the Screen Class on a Bidder's Monitor

All the windows have their own separate exit button, and can be closed independently of each other. Figure 8 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.

## 4.5 HTML Files

These files provide the Web interface of IntelliBid auction systems. 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.

- IntelliBidHome.html: This is the home page of IntelliBid auctions. Its basic function is to provide links to other HTML forms, which are used to obtain information necessary to conduct the auction. All the following HTML pages have a link from this web page.

- BuyerRegNew.html: A 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 AuctionProduct event, allowing for a new bidder to register for a product of interest right after registering.
- SellerRegNew.html: A form, which accepts information about a new supplier registering with IntelliBid. On submitting this form, the action is set to NewSellerServlet.java.
- auctionproduct.html: A form to accept information about a product which is submitted by a supplier interested in auctioning a product. The action of this form is set to the AuctionProductServlet.java
- bidplacement.html: A form to accept bids from the bidders interested in bidding for auction products. The action of this form is set to BidServlet.java. This HTML page also provides two links for registering for events. The first event is OutBid while the second event is AuctionClose.
- productdetails.html: This pages basic function is to provide details about a specified or general product in the auction. It consists of two forms. The form on the top of the page is for providing a specific productid, and the action of the form is set to productDetailsServlet. The lower form is for all ongoing auction products, and the action of this form is set to activeProduct.java.

## 4.6 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.

### 4.6.1 ActiveProducts.java

This servlet is executed from the HTML form ActiveProducts.htm. It’s function is to provide a list of all ongoing (active) products in the auction.

Each product has a status attribute, which is “A” for active and “I” for inactive. When the product's auction is ongoing, it is set to “A” and once the auction product’s time limit is completed, the status attribute is switched to “I.” Hence, performing a search for auction products, which have their status set to “A,” will yield all the ongoing auction products.

- void doPost(HttpServletRequest request, HttpServletResponse response)

Using the POM, we perform a select query on the AuctionProduct table, for all products with status “A.” The returned result is stored in a vector. The products are displayed on the web browser, by iterating through the vector.

4.6.2 AuctionProductServlet.java

This servlet is executed from the HTML form AuctionProduct.htm. Its function is to gather the information about a new product, which has been registered, post the AuctionProduct event, and insert the AuctionProduct object into the database. It also responsible for starting a timer until the product's time limit is finished, and at that point posting the AuctionClose event for that product.

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

- class RemindTask
  - 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.

  - 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.

- doPost(HttpServlet request, HttpServletResponse response)

This method is executed when the HTML form of auctionproduct.htm is submitted. It collects the supplied information by the supplier from the auctionproduct 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. Also, using the POM, the product's information is inserted into the database. The Event Manager's API is then used to post the AuctionProduct event.

4.6.3 BidServlet.java

On submitting the bidplacement.html form, this servlet is executed. It has one public method.

- void doPost(HttpServletRequest request, HttpServletResponse response)

This method is executed when the HTML form of bidplacment.html is submitted. The method's primary function is to gather the data submitted by the form, and register the bid for the product.

Hence, the method gathers the data from the form using the request.getParameter() method. Also, it notes the time when the bid was submitted.

If any of the form values are not blank, then these values are used to form an instance of the EvaluateBid object, found in the IntelliBid Server's RuleLib package. Then the two methods are called on this object (this was explained in section 4.2): -

evaluateBuyer(), which verifies the validity of the bid placed with the submitted credit card. Following this, the method checkIfAcceptedBid() is executed. This method returns a result in the form of a String - indicating whether or not the bid has been accepted. This result is then displayed on the browser. In case the fields of the submitted form are empty, then a message is displayed regarding this matter.

4.6.4 NewBuyerServlet.java

On submitting the BuyerRegNew.html form, this servlet is executed. It has one public method.

- void doPost(HttpServletRequest request, HttpServletResponse response)

This method is executed when a new bidder who is interested in bidding in auctions is going to register with IntelliBid. In the method the following steps are followed

1) Persistent Object Manager is instantiated.

2) Object of a NewBuyer is formed from the information supplied by the BuyerRegNew HTML form.

3) This object is inserted into the IntelliBid database in the NewBuyer table.

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 bidder's browser.

4.6.5 NewSellerServlet.java

On submitting the SellerRegNew.html form, this servlet is executed. It has one public method.

- void doPost(HttpServletRequest request, HttpServletResponse response)

This method is executed when a new supplier who is interested in supplying products for auctions is going to register with IntelliBid. In the method the following steps are followed:

1) Persistent Object Manager is instantiated.

2) Object of a NewSupplier is formed from the information supplied from the SellerRegNew HTML form.

3) This object is inserted into the IntelliBid database in the NewSupplier table.

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.

4.6.6 productDetailsServlet.java

This servlet is called from the upper part of the HTML form productdetails.html. 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 - AuctionProduct and Bids. Hence two select queries are necessary to obtain the required information.

The first query to AuctionProduct obtains information about the product such as it's specifications, descriptions and minimum incremental price. This information is then formatted in HTML form and added to the output buffer.

The second query to the Bids object 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 this information is formatted in HTML form and added to the output buffer.

4.6.7 activeProducts.java

This servlet is called from the lower part of the HTML form productdetails.html. The servlet has one public method:

- void doPost(HttpServletRequest request, HttpServletResponse response)

This method is invoked when the request for viewing all the active products in the auction is submitted. A select query on the Auctionproduct table is executed, to list all the products with status "A." The returned vector is then iterated through, and the information is formatted in HTML. This is displayed on the browser. Note that here, only the product information is displayed and not the actual bids, because this may result in an unnecessary large output.

4.6.8 userDetails.java

This servlet is executed when a supplier or a bidder is interested in viewing his/her specific information. 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.

This class also has one public method

- void doPost(HttpServletRequest request, HttpServletResponse response)

The HTML form, which invokes this method, only submits the username and password of a person. Hence, it becomes the task of the servlet to decide whether the person submitting the information is a supplier or a bidder. Initially the method performs this checking. The Servlet 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," and then the query is repeated with status set to "I." This allows a separation of bids into active (ongoing) and inactive (previously submitted bids and now outbid) bids on auction products. This information is then formatted to HTML, and added to the output buffer.

Similarly for a supplier, a query is performed on the auctionProduct 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 auctions) products. This information is then formatted into HTML and also added to the output buffer. When the querying is completed, this information is displayed on the bidders/suppliers' web browser.

## 4.7 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.

### 4.7.1 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. 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.

4.7.2 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.

4.7.3 OutBidThread.java

This class is a thread, which is forked from the EvaluateBid.java's checkifOutBid() 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. 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 InteliBid.

- 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.

4.7.4 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.

# CHAPTER 5
# CONCLUSION AND FUTURE WORK

In this thesis, we have presented the concept and the technique of conducting event-trigger-rule-based auctions over the Internet. An Internet-based auction system, IntelliBid, has been implemented to test and demonstrate the concept and technique. IntelliBid is constructed by a network of Knowledge Web Servers, each of which consists of a Web server, an Event Manager, an ETR Server, a Knowledge Profile Manager, a Persistent Object Manager, a Metadata Manager, and Bid Servers and their proxies. 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 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. Second, since rules that control the automatic bidding are installed and processed at bidders' individual sites, bidders' privacy and security are safe-guarded. 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. Last, but not least, IntelliBid allows a person or organization to play both the role of bidder and the role of supplier simultaneously. The Profile Manager keeps his/her/its information as a bidder and information as a supplier separately.

In spite of the above desired features of IntelliBid, some future issues can be addressed. Currently, the bids are being specified in terms of simple rules and formulas. More complex rules and formulas for calculating the bid increments can be defined and applied to take more factors into consideration. For example, a new bid may depend on the rates of bid increments given by other bidders, the bid history of a similar product, the dependency relationship with the other auctions that a bidder is currently participating in, etc. Also, rules for controlling bids should be allowed to be changed dynamically, either manually by an on-line bidder or automatically by another rule, to account for some dynamic situation during an auction process. The present system supports one form of "dynamic rules" by changing the values stored in a file read by rules. Another technique for implementing "dynamic rules" is available. Since rules are translated into Java classes in our implementation, Java's class reloading facility can be used to dynamically replace the Java classes of old rules by those of the new rules. Currently, IntelliBid supports only the English auction. The software architecture of IntelliBid is designed in such a way that other types of auctions can be supported by replacing the program that implements

the English auction rules by programs that implement other types of auctions, such as the Dutch auction.

LIST OF REFERENCES


[AW01] AuctionWatch, "AuctionWatch.com," **http://www.auctionwatch.com**, 02/26/01

[AWH01] AuctionWatch, "History of the Auction," **http://www.auctionwatch.com/awdaily/features/history/7.html**, 03/10/01

[AZ01] Amazon, "Amazon.com – Earth's Biggest Selection," **http://www.amazon.com**, 03/15/01

[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

[DA88] Dayal, U., "The HiPAC Project: Combining Active Databases and Timing Constraints," ACM SIGMOD Record, vol. 17, no. 1, March 1998

[EB01] e-bay, "eBay – The World's Online Marketplace," **http://www.ebay.com**, 03/15/01

[FIS01] FishMarket, "The Fishmarket Project," **http://www.iiis.csic.es/Projects/fishmarket/newindex.html**, 6/15/01

[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, Department of Computer and Information Science and Engineering, 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

[ON01] 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

[PC01] PointCast, "Infogate," **http://www.pointcast.com**, 04/26/01

[RMI01] Java, "Remote Method Invocation," **http://java.sun.com/products/jdk/rmi/**, 04/10/01

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

[ST88] 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

[WD96] 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

[XML01] UserLand Software, Inc., XML-RPC home page, **http://www.xml-rpc.org/**, 04/10/01

[YH01] Yahoo!, Yahoo! Auctions: Auctions, **http://auctions.yahoo.com**, 03/15/01

## BIOGRAPHICAL SKETCH

Nicky Joshi was born on April 3rd, 1977 in St. Louis, Missouri, USA. He received a bachelor's degree in electronics engineering, securing a First Class with Distinction from Maharaja Sayajirao University, Baroda, India, in May 1999.

He joined the University of Florida in August 1999 to pursue a master's degree in computer science and engineering. He has been a teaching assistant for database related courses during his study in the University of Florida.

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