Requirements Analysis Document

Jason Venema, Steve Beckman, Bryan DeZeeuw, David Korthuis

Automated Bidding System
Team 2

April 26, 2002
## Contents

1. **Introduction**  
   1.1 Problem Description .................................................. 3  
   1.2 Motivation ................................................................. 3  

2. **Overview**  

3. **Requirements**  
   3.1 Auction Requirements ................................................... 4  
   3.2 User Requirements ....................................................... 5  

4. **UML Analysis**  
   4.1 Use Cases ................................................................. 6  
   4.2 Object Model ............................................................. 11  
   4.3 Data Dictionary ........................................................... 12  
   4.4 Dynamic Model ............................................................ 17  
   4.5 State Diagrams ............................................................ 17  
   4.5.1 Auction Site State Diagram ........................................ 17  
   4.5.2 User State Diagram .................................................. 19  
   4.5.3 Auctioneer State Diagram ......................................... 19  
   4.5.4 Auction State Diagram ............................................. 21  
   4.6 Sequence Diagrams ..................................................... 22  
   4.6.1 Create Auction and View Results ................................. 22  
   4.6.2 Winning an Auction ................................................ 23  
   4.6.3 Remove User and Auction ......................................... 24  

5. **Promela and Spin Analysis**  
   5.1 Simulation Results ....................................................... 26  
   5.2 LTL Results ............................................................... 27  

6. **Appendices**  
   6.1 Appendix A (State Machine Code Without Init) .................... 28  
   6.2 Appendix B (Init Code for LTL Analysis) ........................... 33  
   6.3 Appendix C (Init Code for Simulation) .............................. 34
1 Introduction

The online world is a booming world. Every day the newspapers are filled with more and more E-business announcements and advertisements. Everyone wants to get into the enormous market the Internet has created. There is a huge possibility of trade in this market but right now the traders are scattered throughout the online world trying to find each other. What is needed is something to bring the traders together.

This document describes the requirements of a realtime commerical bidding system. It is separated into four general sections: the introduction, the overview, specific requirements for the system, and UML analysis of the system. The introduction is first and contains this brief introduction as well as the problem description and motivation. The next section in the document is the overview. The requirements section follows the overview and is subdivided into auction requirements and user requirements. Finally the document has the UML analysis. This section contains multiple parts, starting with use cases. From there it goes into the object model and data dictionary. Following that is the dynamic model with state diagrams. The modeling analysis on the model the customer provided is next in the document. Finally, the document ends with the appendices, which are the pieces of promela code for the model checking.

1.1 Problem Description

The Internet can be a large capitalistic market, and as such it is driven by the laws of supply and demand. There are a lot of goods to be supplied on the Internet, and a lot of goods demanded. The online world is a huge world, however. It is not all that easy for the people doing the demanding to find those doing the supplying. The problem we face, then, is bringing these people together into one centralized location so they can trade without having to search to find each other.

1.2 Motivation

The system proposed in this document would provide that centralized location. With all the trading that occurs in the online world, all that would be needed is to manage to attract a small portion of the trading and the result would be a lot of money flowing through the system. There would, of course, be a small fee for using our system, but that fee would easily be worth it to the people selling and buying as it will help them to get the best deals available in the market.

The primary challenge which would be faced is to get word out to the market that this centralized place of trade exists. It is hard for the buyers and sellers to find each other scattered across the vast online world, and that same challenge will be presented to anyone setting up a system such as this. Somehow the buyers and sellers must be reached and informed of this system’s existence.

2 Overview

The Realtime Commercial Bidding System is designed to facilitate the brokering of various kinds of commercial goods and commodities. This is facilitated through an auction site consisting of auctions and reverse-auctions. The auction process is conducted by an automated auctioneer. When an auction or reverse-auction is created the auctioneer is given the auction details and creates the auction in accordance with these specifications. If a user would like to place a bid on a current auction or create an auction of their own, they must first register and login to the site. After successfully logging in, a user can place a bid by joining their desired auction and then submitting their bid to the auctioneer. The auctioneer receives the bid and initially checks to see
if the bid is valid. If the bid is valid, the auctioneer informs the auction creator and all current 
auction participants that there is a new highest bid and reports the value of this bid. If the bid 
is valid the bidder is informed that they have the highest current bid, but if the bid is invalid 
only the bidder is informed. The auctioneer continues to moderate an auction until the time limit 
is reached. When an auction ends, the auctioneer contacts the auction creator with the winning 
bidder’s contact information and informs the winning bidder to contact the creator.

3 Requirements

3.1 Auction Requirements

1. An auction can be either "reserved" or "no reserved". In a "reserved" auction, a minimum 
price must be met to complete a sale in a standard auction and a maximum must be met in 
a reverse auction. If the price is not met, the auction creator will not sell their item(s). In 
a "no reserve" auction, the high bidder in a standard auction wins regardless of price, and a 
lowest bidder wins in a reverse auction.

2. Auctions begin at the time specified by the creator. The default start time is the time of 
auction submission, but it can begin in the future if the creator so specifies. If the auction 
is scheduled for a time other than submission time, the auctioneer will begin the auction 
automatically.

3. A user can create an auction in order to sell their commercial goods or services. The auction 
can take two forms, standard or reverse. The highest bid from a group of bidders determines 
the winner of a standard auction. A reverse auction consists of a group of clients competing 
to sell their goods or services, in this case the lowest bid wins the good or service. Both 
auction forms are moderated by the automated auctioneer. The auction creator submits to 
the auctioneer the auction’s description, start date and time, end date and time, an optional 
minimum price, and an optional bid increment. An auction ends after ten minutes of dead 
time by default if an end date and time are not specified.

4. An auction must have the ability to be deleted. If an auction creator wishes to cancel their 
auction they must contact a site administrator who is able to delete the auction. If the auction 
has already started, all current participants must be informed if the auction is to be deleted.

5. Bids are placed at the time they are received by the auctioneer. When an auctioneer receives 
a bid, the time of the bid is the local time of the auctioneer, not the local time of the bidder 
when they submit their bid. Concurrent bids are not allowed, and if they were to occur, both 
bids would be nullified and the clients informed via a popup window.

6. All current auction participants must be informed when a new winning bid is accepted. When 
an auctioneer receives a new winning bid it will inform all current auction participants of the 
new bid and its value via the auction window.

7. Every auction has its own auctioneer. When an auction is created, an auctioneer with the 
corresponding auction information is created. The auctioneer will start the auction, manage 
bids, notify the bidders of all changes to the auction via the auction window, end the auction, 
notify the creator with the contact information of the winning bidder, and inform the winning 
bidder of their winning status and any pertinent information about the creator.
8. Bid histories must be archived. The auction site will maintain a database of all successful bids. The user must be able to view their past bid amounts for every auction they have participated in, including any active auctions. The bid history of every auction will be archived on the server and will be accessible to auction creators if the information is requested through a site administrator.

9. When an auction ends, all auction participants, the winning bidder, and the auction creator must be notified. All auction participants will be notified of an auction’s end when the auctioneer reports the end in the auction’s window. The winning bidder will be notified via their preferred contact method of their win as well as contact information for the auction creator. The auction creator will be notified of the winning bid amount and the winning bidder’s contact information.

10. If a reserve price is not met or no bids are placed on an auction before the end time there is no winner. When there is no winner, any auction participants and the auction creator are informed that no winner has been determined.

11. Dates and times must be in an explicit format in order to designate whether the time is in ante meridiem or post meridiem time.

12. All auctions must have a "Terms and conditions" click-through screen before a user is allowed to enter. This is in order to prevent any legal issues from arising. The text of this message will be provided by our attorney before production.

### 3.2 User Requirements

1. A user must be able to initially sign up with the auction site. A user is required to submit their name, email address, phone number, mailing address, and method of payment. The site will verify this information before a user can place a bid.

2. A user must join an auction in order to place bids on the goods or services. This can occur before or after the auction has started. A user can view the list of standard or reverse auctions that are both active and inactive via a menu after logging in.

3. A user has the ability to withdraw from an auction regardless if they are the current winning bidder or not. When a winning bidder wishes to withdraw from an auction, they will be notified via a pop-up window that leaving an auction does not negate their responsibility to perform the auction contract if they remain the current winning bidder. Actual text of this message will be provided by our attorney before production.

4. Bidders must be able to view the history of their bids. Bid history is stored on the auction site and past bids can be retrieved upon request of the user via the "Bid History" button. Recent auctions the user has participated in, the auction results, and the string of bids will be listed in this history.

5. In order to submit a bid a user must be logged into the system and have joined the auction. Anonymous or unvalidated users are not allowed in the system.

6. A user can be removed or suspended if they submit invalid registration information, do not pass the credit check, or do not honor their winning bids. Only a site administrator can remove or suspend a user.
7. A winning bidder must be informed with their winning bid amount and the contact information of the auction creator. This will be facilitated through the user’s desired method of communication. The auctioneer will notify active bidders of an auction’s end by updating the auction screen with the winning bid value and username of the winning bidder.

4 UML Analysis

4.1 Use Cases

Figure 1 is the use case diagram. The use case diagram shows how the system should function from the viewpoint of external elements. The external elements are called Actors and are represented by stick figures. Actions that occur within the system are called use cases and are represented by ovals in the use case diagram. An association line is drawn between Actors and the use cases they interact with. Arrows are used to show relation between use cases. If the related use case is sometimes used by the other use case, the exceptional use case “extends” the other use case. If the related use case is always used by the other use case, the other use case “includes” the included use case.

<table>
<thead>
<tr>
<th>Use case: Make Bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors: Bidder</td>
</tr>
<tr>
<td>Type: Primary</td>
</tr>
<tr>
<td>Description: Submits a bid from a user, in an auction, to the auction.</td>
</tr>
<tr>
<td>Cross-reference: Auction Requirement 5, User Requirement 5</td>
</tr>
<tr>
<td>Includes:</td>
</tr>
<tr>
<td>Extended by:</td>
</tr>
<tr>
<td>Extends:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case: View Bid History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors: Bidder</td>
</tr>
<tr>
<td>Type: Primary</td>
</tr>
<tr>
<td>Description: Displays to a user, in an auction, their bid history for that auction.</td>
</tr>
<tr>
<td>Cross-reference: Auction Requirement 8, User Requirement 4</td>
</tr>
<tr>
<td>Includes:</td>
</tr>
<tr>
<td>Extended by:</td>
</tr>
<tr>
<td>Extends:</td>
</tr>
</tbody>
</table>
Figure 1: Use Case Diagram

**Use case:** Withdraw From Auction  

**Actors:** Bidder  

**Type:** Primary  

**Description:** Exits a Bidder, in an auction, from the auction. However, their bid will still be valid.

**Cross-reference:** User Requirement 3  

**Includes:**  

**Extended by:**  

**Extends:**
<table>
<thead>
<tr>
<th>Use case</th>
<th>Description</th>
<th>Cross-reference</th>
<th>Includes</th>
<th>Extended by</th>
<th>Extends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Auction</td>
<td>Joins a User, whose status is logged in, to an auction the user selected.</td>
<td>User Requirement 2</td>
<td></td>
<td></td>
<td>View Auctions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View Auction</td>
<td>Displays auctions.</td>
<td></td>
<td></td>
<td></td>
<td>Join Auction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Collects information from the user and chooses to register them, based on information submitted.</td>
<td>User Requirement 1</td>
<td></td>
<td></td>
<td>Login</td>
</tr>
</tbody>
</table>

1. Use case: Join Auction
   - Actors: Bidder, Auction Creator, Administrator
   - Type: Primary
   - Description: Joins a User, whose status is logged in, to an auction the user selected.
   - Cross-reference: User Requirement 2
   - Includes:
   - Extended by:
   - Extends: View Auctions

2. Use case: View Auction
   - Actors: Bidder, Auction Creator, Administrator
   - Type: Primary
   - Description: Displays auctions.
   - Cross-reference:
   - Includes:
   - Extended by: Join Auction
   - Extends: Join Auction

3. Use case: Register
   - Actors: Bidder, Auction Creator, Administrator
   - Type: Primary
   - Description: Collects information from the user and chooses to register them, based on information submitted.
   - Cross-reference: User Requirement 1
   - Includes:
   - Extended by:
   - Extends: Login
<table>
<thead>
<tr>
<th>Use case</th>
<th>Login</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
<td>Bidder, Auction Creator, Administrator</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>Primary</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Chooses to log a user in based on identification provided by the user.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong></td>
<td>User Requirement 1</td>
</tr>
<tr>
<td><strong>Includes:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
<td>Register</td>
</tr>
<tr>
<td><strong>Extends:</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case</th>
<th>Logout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
<td>Bidder, Auction Creator, Administrator</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>Primary</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Removes logged in status from a user who is logged in and exits the user.</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Includes:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extends:</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case</th>
<th>Create auction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
<td>User</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>Primary</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Creates a forward auction based on the auction creator’s parameters</td>
</tr>
<tr>
<td><strong>Cross-reference:</strong></td>
<td>Auction Requirements 1 and 3</td>
</tr>
<tr>
<td><strong>Includes:</strong></td>
<td>Set Auction Properties</td>
</tr>
<tr>
<td><strong>Extended by:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Extends:</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Use case: Create Reverse Auction

**Actors:** User  

**Type:** Primary  

**Description:** Creates a reverse auction based on information submitted by the user.  

**Cross-reference:** Auction Requirements 1 and 3  

**Includes:** Set Auction Properties  

**Extended by:**  

**Extends:**

### Use case: Set Auction Properties

**Actors:** User  

**Type:** Primary  

**Description:** Collects information from the user that is used to set auction properties when creating an auction.  

**Cross-reference:** Auction Requirements 2 and 3  

**Includes:**  

**Extended by:**  

**Extends:**

### Use case: Delete Auction

**Actors:** Administrator, Auction Creator  

**Type:** Secondary  

**Description:** Removes an auction from the system.  

**Cross-reference:** Auction Requirement 4  

**Includes:**  

**Extended by:**  

**Extends:**
<table>
<thead>
<tr>
<th>Use case:</th>
<th>Register Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Type:</td>
<td>Secondary</td>
</tr>
<tr>
<td>Description:</td>
<td>Registers another administrator with the system.</td>
</tr>
<tr>
<td>Cross-reference:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Includes:</td>
</tr>
<tr>
<td></td>
<td>Extended by:</td>
</tr>
<tr>
<td></td>
<td>Extends:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case:</th>
<th>Delete Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Type:</td>
<td>Secondary</td>
</tr>
<tr>
<td>Description:</td>
<td>Removes another administrator from the system.</td>
</tr>
<tr>
<td>Cross-reference:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Includes:</td>
</tr>
<tr>
<td></td>
<td>Extended by:</td>
</tr>
<tr>
<td></td>
<td>Extends:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case:</th>
<th>Delete User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Type:</td>
<td>Secondary</td>
</tr>
<tr>
<td>Description:</td>
<td>Removes a user from the system.</td>
</tr>
<tr>
<td>Cross-reference:</td>
<td>User Requirement 7</td>
</tr>
<tr>
<td>Includes:</td>
<td></td>
</tr>
<tr>
<td>Extended by:</td>
<td></td>
</tr>
<tr>
<td>Extends:</td>
<td></td>
</tr>
</tbody>
</table>

4.2 Object Model

This section describes the classes used in the implementation of the real-time auction site, and the ways in which they interact with one another. A class diagram is used to model these interactions, as well as the operations that each object is capable of performing. Each class is represented by a rectangle, which is divided into three constituent partitions. The topmost partition is the class name, the middle partition contains a listing of all class attributes, and the bottom partition contains all of the operations that the class can perform. Attributes are data members maintained
by the class. Figure 2 is the class model for this system.

Figure 2: Class Diagram

4.3 Data Dictionary

This section contains definitions of all objects, attributes and operations contained in the class diagram.

- **AuctionSite** - This class is comprised of most of the administrative functions that the system and system administrators can perform, such as maintaining user accounts and user auctions.
- **Operations**

  * **ContactUser(UserID, Message, ContactMethod)** - The purpose of this function is to provide a means by which system administrators, as well as the system automated auctioneers, can contact users. The parameters indicate the user to contact, the message to send, and the method of contact the system should use.

  * **CreateAuction(Item, isReverse, CreatorID)** - Any user has the ability to create an auction. An auction object is instantiated and a list of auctions is maintained by the auction site. The parameters indicate the item up for bid, the type of auction and the auction's creator.

  * **EmailPasswd(UserID)** - If a user has forgotten his/her password, they can request to have it e-mailed to them. This is managed by the auction site.

  * **Login(Name, Passwd)** - This function authorizes users to enter the auction site. Users have no access to any of the auction site’s functionality until they have logged in via this operation. Logging in requires a user name and password.

  * **Logout(Name)** - If a user can log in to the system, they must also be able to log out. Logging out simply disables that user’s ability to navigate through the auction system.

  * **Register(Name, Passwd, CreditRate, Phone, Email, Address)** - This function performs user registration, adding the user’s profile to the list of users who are able to log in to the system. Once the user has registered, that user can log in to the auction site.

  * **RemoveAuction(AuctionID)** - The purpose of this function is to remove an auction from the system. System administrators can do this at any time, the system itself can do this when an auction has ended, and the auction creator can do this.

  * **RemoveUser(UserID)** - The purpose of this function is to remove a user account from the system. System administrators have this privilege.

  * **SuspendUser(UserID)** - The suspension of a user results in that user no longer being allowed to place bids or create auctions while logged in to the system. The user account still exists, however, and the user can still log in.

  * **ViewBidHist(UserID)** - This function displays a comprehensive list of the entire bid history of a user, including time of bid, item being bid upon and amount of bid.

  * **ViewReverseAuctions()** - Reverse auctions will be displayed to the user who invokes this function. The user will be able to choose an auction to enter from among this list.

  * **ViewStandardAuctions()** - Standard auctions will be displayed to the user who invokes this function. The user will be able to choose an auction to enter from among this list.

- **Auction** - This class serves as a grouping of attributes and operations that define an auction.

  - **Attributes**

    * **AuctionID: int** - This is the auction’s unique ID, which is used for quick identification by other objects to avoid passing the entire auction object around.

    * **BestBid: Bid** - The auction maintains the current best bid that has been accepted by its auctioneer. This is an object of type Bid that is compared with incoming bids to determine their acceptability.
* BestBidder: int - This is the user ID of the bidder who currently holds the best bid.
* EndTime: datetime - A date and time representing the auction end point.
* isActive: bool - Indicates the current state of the auction. This attribute must be
true in order for bids to be placed with the auction’s auctioneer.
* isReserveMet: bool - Indicates whether or not the reserve on an auction has been
met. For a reverse auction, this will indicate whether or not a bid that is lower than
the maximum buying price of the item has been received. For a standard auction,
it indicates whether a bid that is higher than the minimum selling price of the item
has been received.
* isReverse: bool - If true, this auction is a reverse auction. Otherwise, it is a standard
auction.
* NextBestBidder: int - The user ID of the bidder who was most recently outbid.
* StartTime: datetime - A date and time value representing the auction start point.
* WinningBid: Bid - This is an object of type Bid, used to maintain the bid that won
the auction. The Bid object contains information on the winning user, the time of
bid, and the bid amount.

Operations
* Activate() - This tells the auction to switch its IsActive member to true, thereby
allowing users to bid on this auction.
* Deactivate() - This tells the auction to switch its IsActive member to false, thereby
barring users from bidding on this auction.

• User - The User class encapsulates an auction site user. Each user must be registered and
logged in to use the auction site.
  - Attributes
    * Address: string - The address of the user, to be used for shipping goods.
    * ContactMethod: enum - The preferred method of contact for users. This can be any
of e-mail, snail mail, phone, etc.
    * CreditRating: real - Represents the credit rating of the user. This attribute is used
in determining whether or not a user may register with the site, and which items
they may bid on.
    * Email: string - The email address of the user, to be used for notification of auction
wins, and auction status.
    * UserID: int - A unique identification number.
    * Phone: string - The user’s phone number.
    * Password: string - The user’s password for gaining access to the system.
    * UserName: char - The user’s name.
    * Type: enum - The type of user this is. This can be one of (1) Bidder, (2) Auction-
Creator or, (3) Administrator.
  - Operations
    * EnterAuction(AuctionID) - Enters a user into an auction, which is specified by the
auction ID number. Once the user has entered the auction, s/he is allowed to place
bids on the auction.
* LeaveAuction(AuctionID) - This function lets a user leave an auction that s/he has previously entered.

* Type() - Returns the type of user this is to the caller.

- **Administrator** - A user who is an administrator will return the administrator enumerated value when function User::Type() is called. Users of this type are granted special privileges such as user account removal, auction removal, and user suspension.

- **AuctionCreator** - A user who is an auction creator will return the auction creator enumerated value when function User::Type() is called. Users become this type when in the context of an auction that they have created, and they have the ability to request auction removal from the system for their own auctions.

  - **Attributes**

    * LastCreationTime: datetime - The date and time of the last auction creation by this user.

    * MyAuctionID: int - The ID of the auction created by this user.

- **Bidder** - A user who is a bidder will return the bidder enumerated value when function User::Type() is called. Users become bidders when in the context of an auction that they have bid on, and they have the ability to view their bid history.

  - **Attributes**

    * CurrentAuction: int - The ID of the auction that the bidder is currently bidding upon.

    * LastBidTime: datetime - The date and time of the previous bid placed by this user.

- **Auctioneer** - The auctioneer acts as the mediator between its auction and a user. Its chief responsibilities are accepting bids, notifying auction creators of incoming bids, and notifying active bidders of new bids. This class is the abstract base class for the derived classes StandardAuctioneer and ReverseAuctioneer.

  - **Attributes**

    * AuctionType: enum - The type of auction this auctioneer is mediating. This can be either (1) Standard, or (2) Reverse auction.

    * MyAuctionID: int - The unique ID number of the auction that this auctioneer is mediating.

  - **Operations**

    * AcceptBid(Bid, UserID) - The purpose of this function is to ensure that the bid received is acceptable, and ensure that the user placing the bid is authorized to do so. A minimum credit rating for each auction must be met by each potential bidder.

    * NotifyActiveBidders(NewBestBid) - When a new high bid is accepted, this function notifies all active bidders.
* NotifyAuctionRemove(Item, CreatorID) - This function notifies an auction creator that their auction has been removed from the system. Typically, this function is associated with a call to ContactUser() to let the user know why their auction is being removed.

* NotifyCreator(NewBestBid, isReserveMet, AuctionID) - This function notifies the creator of an auction whenever a new best bid is accepted. The creator is also notified of whether or not the reserve price has been met.

* NotifyWinner(WinningBid, CreatorID) - Performs notification of the winning bidder, telling s/he what the winning price was.

* Reconcile() - Tells the auction that the user that created it has reconciled payment with the winning bidder. This allows the auction to be removed from the system.

- **StandardAuctioneer** - This is a child class of Auctioneer, and will be instantiated whenever a standard auction is created. It contains the polymorphic functions DisplayTOS(), PlaceBid(), and WithdrawBidder as described below. The functions also appear in the ReverseAuctioneer class, but have different implementations.

  - **Operations**

    * DisplayTOS() - Each time a bid is placed, the owner of the auction system displays their terms of service to the bidder. This is legal information supplied by the company’s attorneys, not programmers.

    * PlaceBid(UserID, Bid) - This operation receives all user bids and responds that the bid has been either accepted or rejected. The incoming bid must be higher than the current best bid to be accepted.

    * WithdrawBidder(UserID) - The purpose of this function is to allow auction bidders to remove themselves from an auction’s active bidder list. Thus, users who have placed a bid on an auction but are no longer interested in purchasing the item can avoid further notification from the system about the auction.

- **ReverseAuctioneer** - This is a child class of Auctioneer, and will be instantiated whenever a reverse auction is created. It is similar to the StandardAuctioneer class, but its function implementation is slightly different.

  - **Operations**

    * DisplayTOS() - Each time a bid is placed, the owner of the auction system displays their terms of service to the bidder. This is legal information supplied by the company’s attorneys, not programmers.

    * PlaceBid(UserID, Bid) - This operation receives all user bids and responds that the bid has been either accepted or rejected. The incoming bid must be lower than the current best bid to be accepted.

    * WithdrawBidder(UserID) - The purpose of this function is to allow auction bidders to remove themselves from an auction’s active bidder list. Thus, users who have placed a bid on an auction but are no longer interested in purchasing the item can avoid further notification from the system about the auction.
• **Item** - This class encapsulates all of the products that are either being sold or sought after on the auction site. Each auction will have its own item object.

  – **Attributes**

    * **ShortDescription**: string - This is a short description of the item, used for previewing auctions in the `ViewStandardAuctions` or `ViewReverseAuctions` list.
    * **LongDescription**: string - This is a long description of the item, used for fully explaining what the item is to interested buyers.
    * **HasReserve**: bool - Indicates whether or not the item has a minimum selling (standard auction) or maximum buying (reverse auction) price.
    * **OwnerID**: int - The user ID of the user who owns the item.
    * **ReservePrice**: real - The reserve price of the auction, as described above. Note that this is only meaningful if `HasReserve` is true.

• **Bid** - This class encapsulates a bid object. Bid objects have a bid placer ID, a bid amount and a placement time.

  – **Attributes**

    * **BidPlacerID**: int - The user ID of the user who placed this bid.
    * **Amount**: real - The amount that the bid placer has agreed to pay for the item being sold (standard auction) or sought after (reverse auction).
    * **PlacementTime**: timeval - A timestamp indicating when the bid was received by the auctioneer.

4.4 **Dynamic Model**

This section illustrates the dynamic behavior of the system. State diagrams for each active class, as well as sequence diagrams for important scenarios, are used to demonstrate this behavior graphically.

4.5 **State Diagrams**

4.5.1 **Auction Site State Diagram**

Figure 3 is the AuctionSite class state diagram. The authorizer begins in the `Ready` state, from which it can receive a `Login()` event. The `Logging In` state performs various checks on the user attempting to log in, such as whether or not the user is registered with the system. Once the `User Logged In` state has successfully been reached, a number of possible events can lead to transitions. The user may log out, effectively bringing them back to the `Ready` state. Also, a user may send the `CreateAuction()` event, which will turn that user into an `AuctionCreator` type and add the auction to the system. Further, a user may request auction removal or user removal. In either case, the system checks to ensure that the user requesting the removal has administrator privileges. If so, the removal takes place and the user proceeds back to the `User Logged In` state. Finally, `ViewReverseAuctions` and `ViewStandardAuctions` events will place the system into those states in which the auctions are displayed. All of these states proceed to the `Done` fork, which returns to the `User Logged In` state.
Figure 3: AuctionSite State Diagram
4.5.2 User State Diagram

Figure 4 is the User class state diagram. Once the user has logged in, the User object will be in the 
*Ready* state. The user has little functionality of its own, other than to enter itself into auctions, 
and to leave auctions. Upon receiving an *EnterAuction()* event, a user becomes entered into that 
auction and can bid on it. After receiving the *LeaveAuction()* event, the user can no longer bid on 
the item until he/she has entered the auction again.

![User State Diagram](image)

Figure 4: User State Diagram

4.5.3 Auctioneer State Diagram

The diagram in figure 5 displays the possible states and state transitions of Auctioneer class objects. 
Note that this includes both *Standard* and *Reverse* auctioneers. Once the auctioneer has been 
instantiated, it is in the ready state. A *PlaceBid* event will transition the Auctioneer to the 
DisplayTOS state, which displays the system’s terms of service. When the user has accepted 
the TOS, the system transitions to the Accept Bid composite state, from where the auction type 
(standard or reverse) is checked. There are two transition possibilities from here:

- If the auction is a reverse auction, the auctioneer will transition and check the submitted bid. 
  If the bid amount is greater than the auction’s best bid, or if the bid amount is greater than 
or equal to the maximum price the buyer is willing to pay, the bid is rejected and failure is 
returned to the caller. The system transitions to the *Done* state, and back to the *Ready* state.
Figure 5: Auctioneer State Diagram
Otherwise, the former best bid becomes the next best bid and the submitted bid becomes the best bid in the *Bid Validated* state. The Auctioneer performs the Notifying operations, notifying the client and the auction’s active bidders of a new best bid. A success signal is returned to the caller and the auctioneer goes back to the *Ready* state.

- If the auction is a standard auction, the auctioneer will transition and check the submitted bid. If the bid amount is less than or equal to the current best bid, the bid is rejected and a failure signal is returned to the caller. Otherwise, the auctioneer checks whether or not the reserve has been met, notifies the client and active bidders, and transitions back to the *Ready* state.

The auctioneer also performs the WithdrawBidder and Reconcile functions. Thus, if an auction creator requests that the auction be reconciled, the auctioneer transitions to the *Reconciling* state where it removes the auction from the system, as long as the auction is no longer active. If a bidder requests withdrawal from an auction, the auctioneer transitions to the *Withdraw Bidder* state, and removes the bidder from its list of active bidders as long as the bidder is not the current best bidder.

### 4.5.4 Auction State Diagram

![Auction State Diagram](image)

*Figure 6: Auction State Diagram*

The Auction class state diagram of figure 6 displays the possible states of objects of type Auction. Upon instantiation, the object will be in the *Ready* state, from which it can transition to
the *Active* state with an *Activate* event. This will set the `isActive` member to true. Once active, the
auction remains in that state until it receives a *Deactivate* event, causing it to enter the *Inactive*
state, setting the `isActive` member to false. The auction will remain in the *Inactive* state until the
auction’s creator issues a *Reconcile* event, at which point the auction is removed from the system.

4.6 **Sequence Diagrams**

The sequence diagrams demonstrate system interaction in the chronological order in which the
events occur. The squares in the top row of a diagram are objects from the class diagram. The
dashed lines going down from the objects are timelines. The arrows show messages passing between
objects in the sequence they occur. Only a limited representation of the possible sequence diagrams
are shown.

4.6.1 **Create Auction and View Results**

![Sequence Diagram](image)

Figure 7: Create Auction and View Results

Figure 7 shows a user creating an auction and viewing the results.
1. First the User submits a login request to the Auction Site.
2. The Auction Site returns a successful login.
3. The User then sends a signal to create an auction to the Auction Site.
4. The User signals that it wants to view standard auctions to the Auction Site.
5. The Auction Site responds by displaying the standard auctions to the User.
6. The User signals to the Auction that it is entering.
7. The Auctioneer notifies the winner of the auction outcome.
8. The Auctioneer deactivates the auction.
9. The Auctioneer send the Auction Site the Reconcile signal.
10. The Users leave the auction.
11. The Auction Site removes the auction.
12. The User sends the logout signal to the Auction Site.

4.6.2 Winning an Auction

Figure 8 shows a User winning an auction.
1. First the User submits a login request to the Auction Site.
2. The Auction Site returns a successful login.
3. The User signals that it wants to view standard auctions to the Auction Site.
4. The Auction Site responds by displaying the standard auctions to the User.
5. The User signals to the Auction that it is entering.
6. The User sends a place bid signal to the Auctioneer.
7. The Auctioneer Displays the TOS to the User.
8. The User responds that it accepts the TOS.
9. The Auctioneer signals the Auction that it wants to check the auction type.
10. The Auction responds with the auction type.
11. The Auctioneer validates the bid.
12. The Auctioneer notifies the winner of the auction outcome.
13. The Auctioneer deactivates the auction.
14. The Auctioneer send the Auction Site the Reconcile signal.
15. The Users leave the auction.
16. The Auction Site removes the auction.
17. The User sends the logout signal to the Auction Site.
4.6.3 Remove User and Auction

Figure 9 shows an Administrator Removing a User and an Auction.

1. First the Administrator submits a login request to the Auction Site.
2. The Auction Site returns a successful login.
3. The Administrator signals that it wants to view standard auctions to the Auction Site.
4. The Auction Site responds by displaying the standard auctions.
5. The Administrator signals the Auction site to remove a user.
6. The Auction Site Contacts the User.
7. The Auction Site Deletes the User.
8. The Auction Site responds to the Administrator that the User was removed successfully.
9. The Administrator signals the Auction site to remove a Auction.
10. The Auction Site Contacts the Auction.
11. The Auction Site Deletes the Auction.
12. The Auction Site responds to the Administrator that the Auction was removed successfully.
13. The Administrator sends the logout signal to the Auction Site.

5 Promela and Spin Analysis

The customer developed two state diagrams and requested we perform model analysis on them. One diagram is for the auctioneer and another is for the client. The promela code was written for one auctioneer and two clients, with the clients being set up to bid against each other. Two tests were performed on the models, and both failed. One tested an LTL formula while the other tested a simulation of one client submitting a bid and then another client submitting a second bid.
The promela code for the analysis can be found in the Appendices. Code for the auctioneer and clients can be found in Appendix A. Appendix B contains the code for the initialization of the LTL analysis. The initialization code for the simulation is in Appendix C.

5.1 Simulation Results

Figure 10: Simulation Message Sequence Chart
When the simulation was run, the auctioneer ended up being stuck in the High_Bid state. One client was able to finish and return to the Idle state, however the other client was stuck in the User_Bid state. In addition, none of the components were able to transition to their end states once the auction_time_elapse signal was sent, which should have told the auctioneer to send the signal to the clients to end, as well as to end itself. To test this simulation further, the simulation was set up to run with a different random seed. This time, each client ended in the state expected of them. As a result, the conclusion was reached that the problem has to do with the timing of the events that arrive.

The message sequence chart for this simulation is included in figure 10. The chart shows the auction_time_elapse signal being sent, however its being sent does not trigger any actions. The next step was to run a trace through the simulation and follow along with the machine and see exactly what was going on.

The first thing that happened was that each client submitted a bid, one right after the other. This moved them both to their User_Bid state, where they each sent a bidrqst event to the auctioneer, and then waited for a bidok response from the auctioneer. The auctioneer received the bidrqst event from the first client and sent a bidok event back to the client, then transitioned to the Check_OK state. The first client processed its bid and sent that bid to the auctioneer, which received it and sent back the ack event. The first client then transitioned to its Idle state. The auctioneer, having seen that this new bid was higher than previously submitted bids, transitioned to the High_Bid state where it sent a high_bid event to each of the clients. The first client received the event and processed it as it was supposed to. However, the channel between the auctioneer and the second client already had one event on it (the second client’s bidrqst event). As a result, the auctioneer had to wait to send the high_bid event for the bidrqst event to be processed, but that would never happen because the auctioneer was the component that would need to process it. As a result, the system was deadlocked there.

5.2 LTL Results

The LTL formula which was run tested to be sure the first client would always arrive at its end state some time after the auctioneer received the auction_time_elapse event. Upon running this verification, it proved to be invalid. This was the expected result, as the simulation previously run had shown this did not hold true for that case.

The message sequence chart, shown in figure 11, revealed that the auctioneer actually received the auction_time_elapse event and transitioned to its end state. This was somewhat unexpected, as this test was run expecting the client would not arrive at its end state for the same reason it did not arrive there in the simulation. So it was once again time to run a trace and see what was going on in the system.

As the trace was run, the trouble started with a bid being submitted that was the same value as the current high bid. When the auctioneer reached the Save_Bid state, the only transitions it could take were for the case of the new bid being higher than the current high bid or being lower than the high bid. As a result, the auctioneer was stuck in this state. The clients then were stuck waiting in their User_Bid state for a bidok event to be received back from the auctioneer, which would never come. Then the auction_time_elapse event arrived, and the auctioneer was actually able to transition on this event because the event goes from the border of the composite machine which was set up for the auctioneer’s state diagram. As a result, the auctioneer sent the signal to the clients to end and ended itself. The clients, however, were still in their User_Bid state and so were unable to receive the event and never ended.
Figure 11: LTL Analysis Message Sequence Chart

6 Appendices

6.1 Appendix A (State Machine Code Without Init)

/* States */
mtype {sWait, sCheck_ok, sSave_bid, sHigh_bid, sEnd_all}
mtype {sHi_bid, sIdle, sUser_bid, sGet_bid, sSent_bid, sClient_end, 
  sAuctioneer_end}
/* events */
mtype {user_bid_request, bidok, ack, auctiontimeout, high_bid, 
  end_auction, bidrqst, bidmsg, auction_time_elapse, ending, get_bid_amount}
mtype client1state;
mtype client2state;
mtype auctioneerstate;
/* channel site is for auction site/auctioneer communication */
chan site = [1] of {mtype, int};

/* channel system is for system/client communication */
chan system = [1] of {mtype, int};

/* channel screen1 is for interface screen1/client1 communication */
chan screen1 = [1] of {mtype, int};

/* channel screen2 is for interface screen2/client2 communication */
chan screen2 = [1] of {mtype, int};

/* channel event1 is for auctioneer/client1 communication */
chan event1 = [1] of {mtype, int};

/* channel event2 is for auctioneer/client2 communication */
chan event2 = [1] of {mtype, int};

/* the client variable is so we know which client we're communicating with */
int theclient = 0;

int maxbid = 0;
int newbid = 0;
int amt = 0;
int bid = 0;

/* ourbid is the bid that's being generated to send while driving through all scenarios - start with a value of 1 just for fun */
int ourbid = 1;

/* junk variable is to receive 0s for events without a parameter */
int junk;

active proctype client1()
{
    client1state = sIdle;
    do
        :: (client1state == sIdle) -> printf("in state idle\n");
        if
            :: event1?high_bid,amt ->
                printf("receiving high_bid event1\n");
                client1state = sHi_bid;
            :: screen1?user_bid_request,junk ->
                printf("receiving user_bid_request event1\n");
                client1state = sUser_bid;
    od
}

/* channel site is for auction site/auctioneer communication */
chan site = [1] of {mtype, int};

/* channel system is for system/client communication */
chan system = [1] of {mtype, int};

/* channel screen1 is for interface screen1/client1 communication */
chan screen1 = [1] of {mtype, int};

/* channel screen2 is for interface screen2/client2 communication */
chan screen2 = [1] of {mtype, int};

/* channel event1 is for auctioneer/client1 communication */
chan event1 = [1] of {mtype, int};

/* channel event2 is for auctioneer/client2 communication */
chan event2 = [1] of {mtype, int};

/* the client variable is so we know which client we're communicating with */
int theclient = 0;

int maxbid = 0;
int newbid = 0;
int amt = 0;
int bid = 0;

/* ourbid is the bid that's being generated to send while driving through all scenarios - start with a value of 1 just for fun */
int ourbid = 1;

/* junk variable is to receive 0s for events without a parameter */
int junk;

active proctype client1()
{
    client1state = sIdle;
    do
        :: (client1state == sIdle) -> printf("in state idle\n");
        if
            :: event1?high_bid,amt ->
                printf("receiving high_bid event1\n");
                client1state = sHi_bid;
            :: screen1?user_bid_request,junk ->
                printf("receiving user_bid_request event1\n");
                client1state = sUser_bid;
    od
}
:: event1?end_auction, junk ->
    printf("receiving end_auction event1\n");
client1state = sClient_end;
    fi
:: (client1state == sUser_bid) -> printf("in state user_bid\n");
    printf("sending bidrqst event1\n");
    event1!bidrqst, 0;
if
:: event1?bidok, junk ->
    printf("receiving bidok event1\n");
client1state = sGet_bid;
    fi
:: (client1state == sGet_bid) -> printf("in state get_bid\n");
    if
:: screen1?get_bid_amount, bid ->
    printf("receiving get_bid_amount event1\n");
    printf("sending bidmsg event1\n");
    event1!bidmsg, bid;
    client1state = sSent_bid;
    fi
:: (client1state == sSent_bid) -> printf("in state sent_bid\n");
    if
:: system?auction_timeout, junk ->
    printf("receiving timeout event\n");
    client1state = sUser_bid;
:: event1?ack, junk ->
    printf("receiving ack event1\n");
    client1state = sIdle;
    fi
:: (client1state == sHi_bid) -> printf("in state hi_bid\n");
    printf("High bid is: \%d\n", amt);
    client1state = sIdle;
:: (client1state == sClient_end) -> printf("in state client_end\n");
    /* try to receive ending to keep from cycling through the do */
    event1?ending;
    od;
}

active proctype client2()
{
    client2state = sIdle;
    do
:: (client2state == sIdle) -> printf("in state idle\n");
    if
:: event2?high_bid, amt ->
    printf("receiving high_bid event2\n");
    client2state = sHi_bid;
    od;
:: screen2?user_bid_request, junk ->
printf("receiving user_bid_request event2\n");
client2state = sUser_bid;
:: event2?end_auction, junk ->
printf("receiving end_auction event2\n");
client2state = sClient_end;
fi
:: (client2state == sUser_bid) -> printf("in state user_bid\n");
printf("sending bidrqst event2\n");
event2!bidrqst, 0;
if
:: event2?bidck, junk ->
printf("receiving bidok event2\n");
client2state = sGet_bid;
fi
:: (client2state == sGet_bid) -> printf("in state get_bid\n");
if
:: screen2?get_bid_amount, bid ->
printf("receiving get_bid_amount event2\n");
printf("sending bidmsg event2\n");
event2!bidmsg, bid;
client2state = sSent_bid;
fi
:: (client2state == sSent_bid) -> printf("in state sent_bid\n");
if
:: system?auctiontimeout, junk ->
printf("receiving timeout event2\n");
client2state = sUser_bid;
:: event2?ack, junk ->
printf("receiving ack event2\n");
client2state = sIdle;
fi
:: (client2state == sHi_bid) -> printf("in state hi_bid\n");
printf("High bid is: %d\n", amt);
client2state = sIdle;
:: (client2state == sClient_end) -> printf("in state client_end\n");
/* try to receiv ending to keep from cycling through the do */
event2?ending;
\od;
}

active proctype auctioneer(){
auctioneerstate = sWait;
do
:: (auctioneerstate == sWait) -> printf("in state wait\n");
if
:: event1?bidrqst, junk ->
printf("receiving bidrqst event1\n");
printf("sending bidok event1\n");
event1!bidok,0;
auctioneerstate = sCheck_ok;
  :: event2?bidrqst,junk ->
    printf("receiving bidrqst event2\n");
printf("sending bidok event2\n");
event2!bidok,0;
auctioneerstate = sCheck_ok;
  :: site?auction_time_elapse,junk ->
    printf("receiving auction_time_elapse event\n");
auctioneerstate = sEnd_all;
fi
  :: (auctioneerstate == sCheck_ok) -> printf("in state check_ok\n");
if
  :: event1?bidmsg,newbid ->
    printf("receiving bidmsg event1\n");
theclient = 1;
auctioneerstate = sSave_bid;
  :: event2?bidmsg,newbid ->
    printf("receiving bidmsg event2\n");
theclient = 2;
auctioneerstate = sSave_bid;
  :: site?auction_time_elapse,junk ->
    printf("receiving auction_time_elapse event\n");
auctioneerstate = sEnd_all;
fi
  :: (auctioneerstate == sSave_bid) -> printf("in state save_bid\n");
if
  :: (theclient == 1) ->
    event1!ack,0;
  :: (newbid > maxbid) ->
    printf("newbid was higher than maxbid\n");
auctioneerstate = sHigh_bid;
  :: (newbid < maxbid) ->
    printf("newbid was lower than maxbid\n");
auctioneerstate = sWait;
  :: site?auction_time_elapse,junk ->
    printf("receiving auction_time_elapse event\n");
auctioneerstate = sEnd_all;
fi
  :: (theclient == 2) ->
    event2!ack,0;
  :: (newbid > maxbid) ->
    printf("newbid was higher than maxbid\n");
```c
auctioneerstate = sHigh_bid;
    :: (newbid < maxbid) ->
        printf("newbid was lower than maxbid\n");
auctioneerstate = sWait;
    :: site?auction_time_elapse,junk ->
        printf("receiving auction_time_elapse event\n");
auctioneerstate = sEnd_all;
    fi
fi

:: (auctioneerstate == sHigh_bid) -> printf("in state high_bid\n");
    maxbid = newbid;
    printf("sending high_bid event\n");
event1!high_bid,maxbid;
event2!high_bid,maxbid;
auctioneerstate = sWait;
:: (auctioneerstate == sEnd_all) -> printf("in state end_all\n");
    printf("sending end_auction event\n");
event1!end_auction,0;
event2!end_auction,0;
auctioneerstate = sAuctioneer_end;
:: (auctioneerstate == sAuctioneer_end) ->
    printf("in state auctioneer_end\n");
    /* try to receive ending to keep from cycling through the do */
    site?ending,0;
    od
}

6.2 Appendix B (Init Code for LTL Analysis)

init
{
    do
        :: ourbid++;
        if
            :: (ourbid > 10) ->
                site!auction_time_elapse,0;
            break;
            :: (ourbid <= 10) ->
                screen1!user_bid_request,0;
                screen1!get_bid_amount,ourbid;
            fi
        :: ourbid++;
        if
            :: (ourbid > 10) ->
                site!auction_time_elapse,0;
            break;

33
```
:: (ourbid <= 10) ->
  screen2!user_bid_request,0;
  screen2!get_bid_amount,ourbid;
  fi
:: ourbid--;
  if
    :: (ourbid < 0) ->
      site!auction_time_elapse,0;
      break;
    :: (ourbid >= 0) ->
      screen1!user_bid_request,0;
      screen1!get_bid_amount,ourbid;
    fi
:: ourbid--;
  if
    :: (ourbid < 0) ->
      site!auction_time_elapse,0;
      break;
    :: (ourbid >= 0) ->
      screen2!user_bid_request,0;
      screen2!get_bid_amount,ourbid;
    fi
  :: ourbid = ourbid;
  screen1!user_bid_request,0;
  screen1!get_bid_amount,ourbid;
:: ourbid = ourbid;
  screen2!user_bid_request,0;
  screen2!get_bid_amount,ourbid;
:: site!auction_time_elapse,0;
  break;
ode
6.3 Appendix C (Init Code for Simulation)

init
{
/* this section submits a bid of $25 */
printf("sending user_bid_request\n");
screen1!user_bid_request,0;
printf("sending get_bid_amount with bid of 25\n");
screen1!get_bid_amount,25;

/* this section submits a bid of $50 */
printf("sending user_bid_request\n");
screen2!user_bid_request,0;
printf("sending get_bid_amount with bid of 50\n");
screen2!get_bid_amount,50;

/* the rest of the code in the init ends the auction */
timeout ->
printf("timing out\n");

printf("sending auction_time_elapse\n");
site!auction_time_elapse,0;
}