COMP1216 : Software Modelling and Design

Coursework: An Online Auction Service

March 2019

Group Number : 11

Member Names:

Gopika Bejoy (gb1n17), Madihah Moraby (mhm1n18),
Andreea Nechita (an2u18), Palak Jain (ppj1u18)

Part A - Modelling in UML

1. Introduction

We all worked on different sections of part A and part B simultaneously and divided the work according to our strengths and weaknesses in order be more time efficient. We got together to discuss each of our sections, which helped us build and improve on our ideas. Additionally, we met up regularly to discuss the updates we made on our work. When issues occurred we changed our roles, so that we could optimally use our skills and fix the problem as a group. Due to uncertainty of the details in the diagrams in part A, we first drew them by hand and made changes to them on paper before we drew them using the Visual Paradigm software. Similarly, in part B, we first identified and wrote down all the variables that we would need to model the events. Moreover, we tried to identify all the invariants (conditions on the variables which must hold permanently) beforehand. Do this planning beforehand made it easier to model the events.

Clarifications:
Users who bid in auction system = bidders
Users who sell items in auction system = sellers

2. Scope of the system

Need: We need to develop an online Auction System service

Goals: It should allow users to submit items for auction and to bid for items that are being auctioned.

Business Case: To create an effective and competitive system for auctioning

Stakeholders: new and existing users

High-level operational concepts:

- Users get registered into the system, using username, status, login id and password
- Seller adds name and the price of the item
- Seler defines start and end time
- Seller can auction
- Seller can see progress of the item at any time
- Other users in the system see the auctioned item and suppliers' information (feedback and penalty, if present)
- Users can only bid on a price higher than the last bid
- Seller can cancel auction at any point
- After end time is reached, bidders and sellers are informed
- Auction is closed
- Bidders can give the supplier reviews

Assumptions:

Users will use an electronic device - PC, tablet, smartphone etc.
Users will have Internet access.

Constraints:

New system has to be operational in 5 months
Budget: £100,000

3. Three Full Scenarios

3.1 Successful Auction

- Seller submits item for auction
- Item becomes visible to all users (except the user who submitted item)
- Bidding starts
- The end time is reached and the current highest bid is locked into the system
- Bid reaches or exceeds the original reserve price
- The bidder with highest bid is informed
- Supplier is informed
- The auction has been successful

3.2 Failed Auction

- Seller submits item for auction
- Item becomes visible to all users (except the user who submitted item)
- Bidding starts
- The end time is reached and the current highest bid is locked into the system
- Highest bid fails to reach the original reserve price
- Supplier is informed
- The auction has failed

3.3 Cancelled Auction

Scenario 1: User who starts auction cancels auction before bid accepted is less than reserve price

- Seller submits item for auction
- Item becomes visible to all users (except the user who submitted item)
- Bidding starts
- Bid at less than reserve price is accepted
- Seller cancels without penalty
- Auction closes
- The auction has been cancelled
- All bidders are informed

Scenario 2: User who starts auction cancels auction before bid accepted is more than reserve price

- Seller submits item for auction
- Item becomes visible to all users (except the user who submitted item)
- Bidding starts
- Bid at a price higher than reserve price is accepted
- Seller cancels
- Auction closes
- The seller receives a penalty point
- The auction has been cancelled
- All bidders are informed

4. Two use case in the Bruegge & duToit format

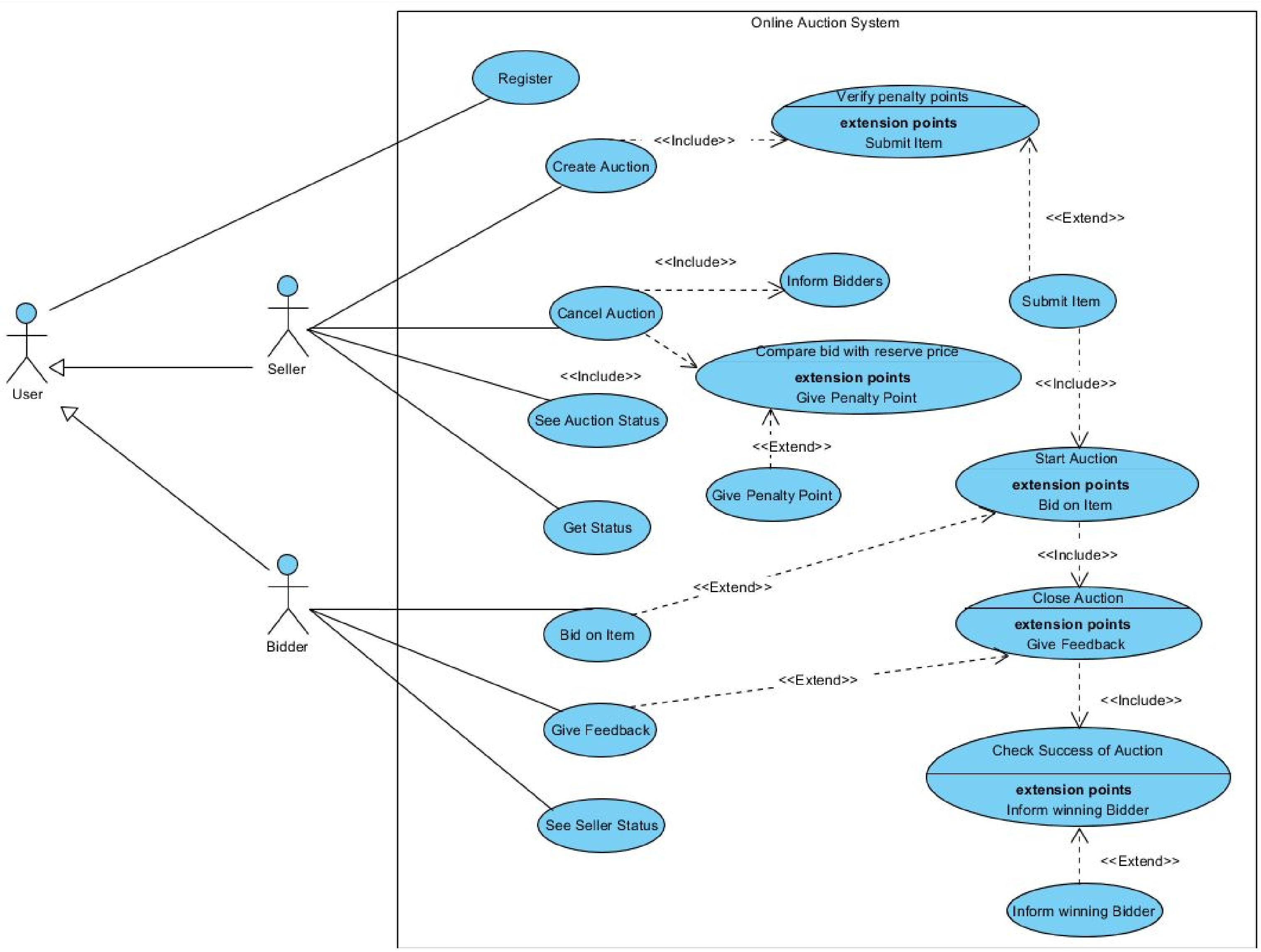
First use case :

Use case name	StartAuction
Participating actors	Initiated by Seller Bidders
Flow of events	1. The Seller submits an item for the auction. 2. The Seller provides a name for the item, the start and end time of auction, and the reserve price. 2. AuctionSystem responds by creating a new auction, and opens the auction to bids from Bidders. 3. AuctionSystem makes the item visible to the Bidders. 4. The Seller sees the status of the auction during the action. 5. AuctionSystem provides information about the Seller's feedback and penalties. 6. The Bidders bid for item within the BidForItem use case. 7. AuctionSystem closes the action whether because the duration has passed or the Seller cancels it. If the Seller cancels the auction after a bid higher than the reserve price a Bidder has made, the Seller receives a penalty point through the AuctionSystem. 8. AuctionSystem informs all the Bidders if the Seller cancels the auction. AuctionSystem informs the winning Bidder if the auction succeeds.
Entry condition	· The Seller is logged into the AuctionSystem. · The Seller has no more than two penalty points.
Exit condition	· The duration of the auction has passed, OR · The Seller has cancelled the auction.
Quality requirements	· The AuctionSystem should update the information about the auction in real time.

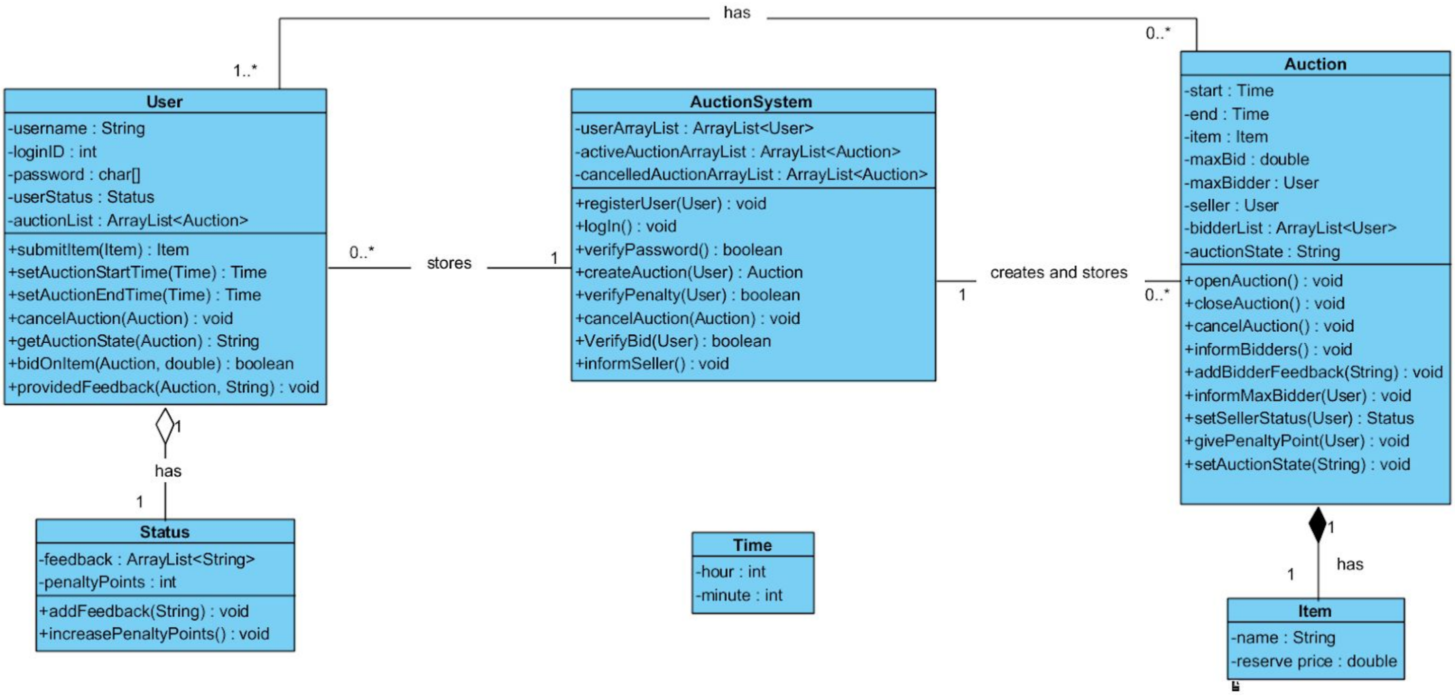
Second use case:

Use case name	`BidForItem`
Participating actors	Initiated by `Bidder` `Seller`
Flow of events	1. `AuctionSystem` makes the item visible to the `Bidder`. 2. The `Bidder` makes bids that are higher than the current highest bid of the auction. 3. `AuctionSystem` informs the `Bider` if the auction is cancelled. When the auction succeeds (the highest bid is at least as high as the reserve price), `AuctionSystem` informs the `Bidder` if he is the winning bidder.
Entry condition	· The `Bidder` is registered with the `AuctionSystem`. · The `Bidder` is not the `Seller` of the auction.
Exit condition	· The duration of the auction has passed, OR · The `Seller` has cancelled the auction.
Quality requirements	· The `Seller` should be able to see the status of the auction updated in real time.

5. Use Case Diagram

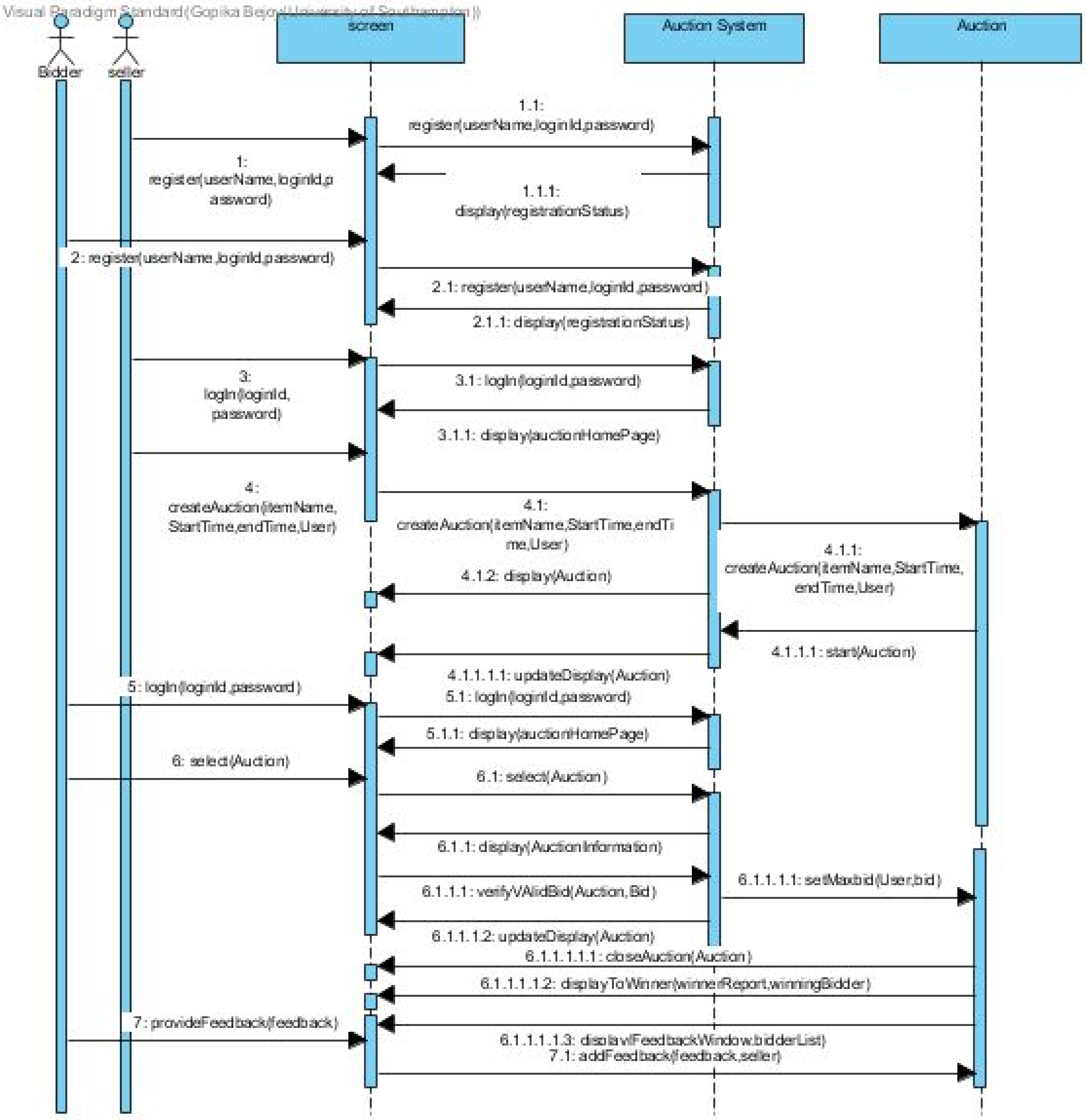


6. Class Diagram

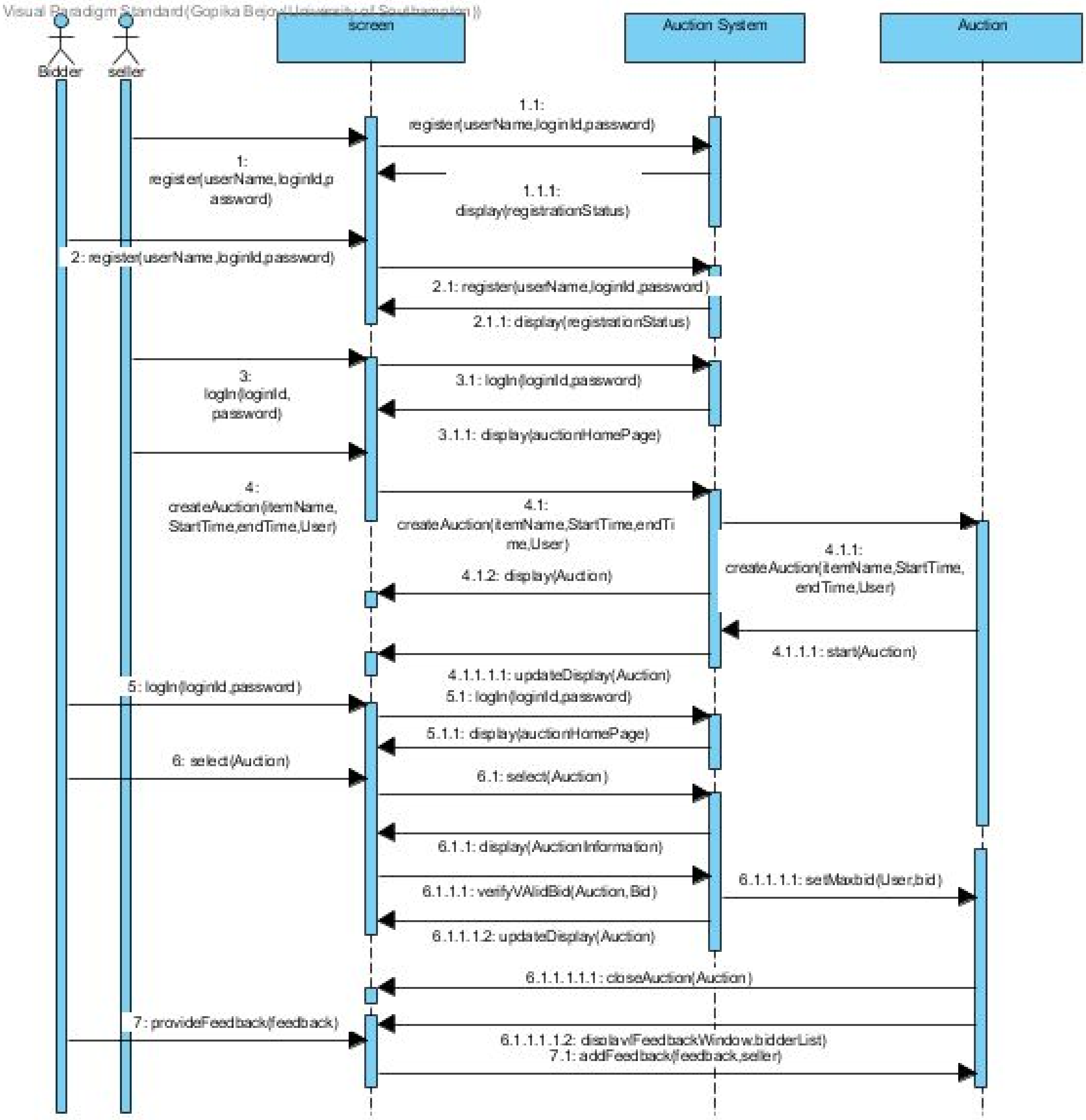

has
1..*
0..*
User
-username : String
-loginID : int
-password : char[]
-userStatus : Status
-auctionList : ArrayList<Auction>
+submitItem(Item) : Item
+setAuctionStartTime(Time) : Time
+setAuctionEndTime(Time) : Time
+cancelAuction(Auction) : void
+getAuctionState(Auction) : String
+bidOnItem(Auction, double) : boolean
+providedFeedback(Auction, String) : void
0..*
stores
1
AuctionSystem
-userArrayList : ArrayList<User>
-activeAuctionArrayList : ArrayList<Auction>
-cancelledAuctionArrayList : ArrayList<Auction>
+registerUser(User) : void
+logIn() : void
+verifyPassword() : boolean
+createAuction(User) : Auction
+verifyPenalty(User) : boolean
+cancelAuction(Auction) : void
+VerifyBid(User) : boolean
+informSeller() : void
creates and stores
1
0..*
Auction
-start : Time
-end : Time
-item : Item
-maxBid : double
-maxBidder : User
-seller : User
-bidderList : ArrayList<User>
-auctionState : String
+openAuction() : void
+closeAuction() : void
+cancelAuction() : void
+informBidders() : void
+addBidderFeedback(String) : void
+informMaxBidder(User) : void
+setSellerStatus(User) : Status
+givePenaltyPoint(User) : void
+setAuctionState(String) : void
1
has
1
Status
-feedback : ArrayList<String>
-penaltyPoints : int
+addFeedback(String) : void
+increasePenaltyPoints() : void
Time
-hour : int
-minute : int
1
has
1
Item
-name : String
-reserve price : double

7. Sequence Diagrams

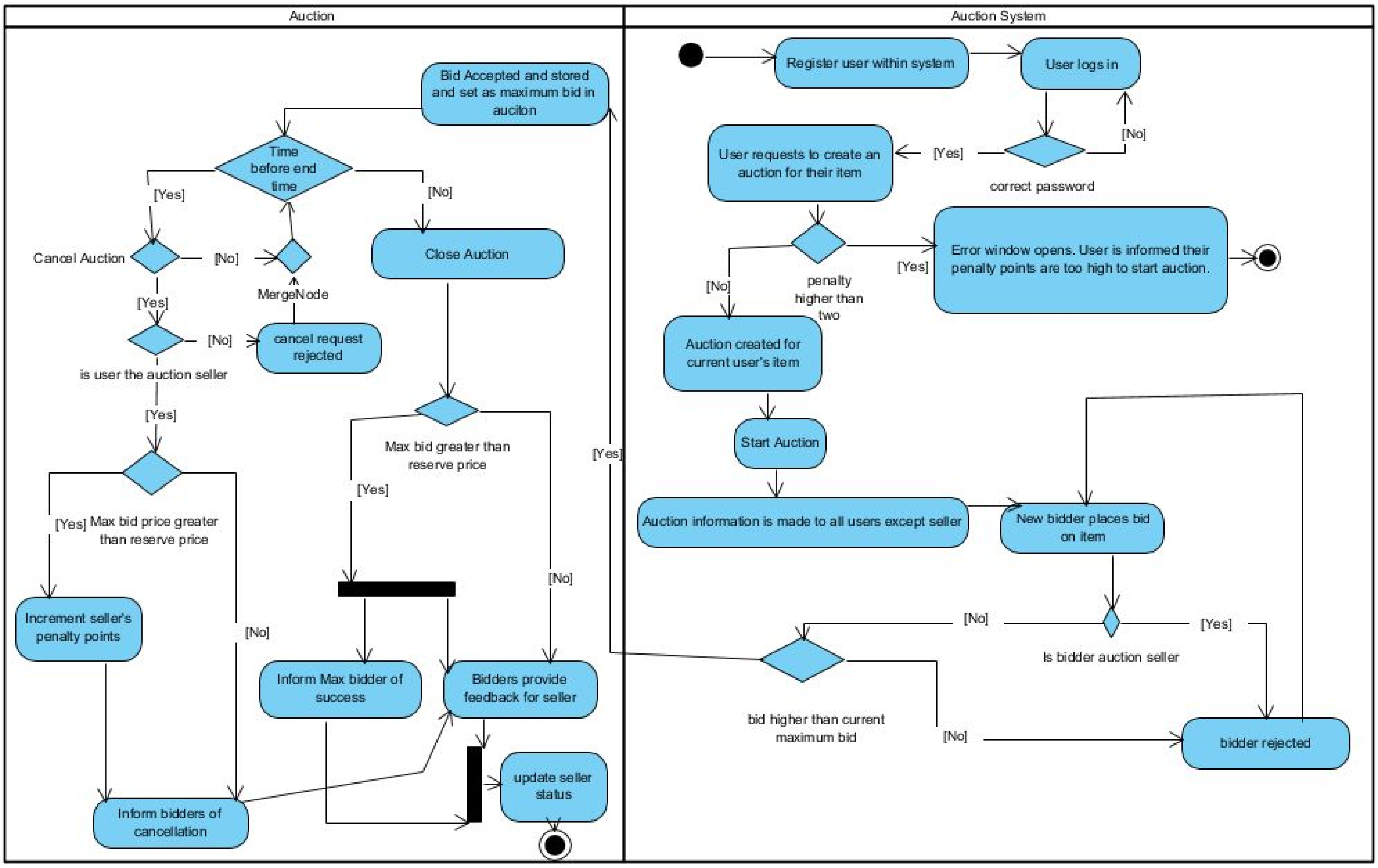
7.1 Successful Auction Scenario


Visual Paradigm Standard (Gopika Bejoy(University of Southampton))
screen
Auction System
Auction
Bidder
seller
1.1: register(userName,loginId,password)
1: register(userName,loginId,password)
1.1.1: display(registrationStatus)
2: register(userName,loginId,password)
2.1: register(userName,loginId,password)
2.1.1: display(registrationStatus)
3: logIn(loginId, password)
3.1: logIn(loginId,password)
3.1.1: display(auctionHomePage)
4: createAuction(itemName, StartTime,endTime,User)
4.1: createAuction(itemName,StartTime,endTime,User)
4.1.1: createAuction(itemName,StartTime, endTime,User)
4.1.2: display(Auction)
4.1.1.1: start(Auction)
4.1.1.1.1: updateDisplay(Auction)
5: logIn(loginId,password)
5.1: logIn(loginId,password)
5.1.1: display(auctionHomePage)
6: select(Auction)
6.1: select(Auction)
6.1.1: display(AuctionInformation)
6.1.1.1: verifyVAlidBid(Auction,Bid)
6.1.1.1.1: setMaxbid(User,bid)
6.1.1.1.2: updateDisplay(Auction)
6.1.1.1.1.1: closeAuction(Auction)
6.1.1.1.1.2: displayToWinner(winnerReport,winningBidder)
7: provideFeedback(feedback)
6.1.1.1.1.3: displayFeedbackWindow,bidderList)
7.1: addFeedback(feedback,seller)

7.2 Failed Auction Scenario


Visual Paradigm Standard(Gopika Bejoy(University of Southampton))
Bidder
seller
screen
Auction System
Auction
1: register(userName,loginId,password)
1.1: register(userName,loginId,password)
1.1.1: display(registrationStatus)
2: register(userName,loginId,password)
2.1: register(userName,loginId,password)
2.1.1: display(registrationStatus)
3: logIn(loginId, password)
3.1: logIn(loginId,password)
3.1.1: display(auctionHomePage)
4: createAuction(itemName, StartTime,endTime,User)
4.1: createAuction(itemName,StartTime,endTime,User)
4.1.1: createAuction(itemName,StartTime, endTime,User)
4.1.2: display(Auction)
4.1.1.1: start(Auction)
4.1.1.1.1: updateDisplay(Auction)
5: logIn(loginId,password)
5.1: logIn(loginId,password)
5.1.1: display(auctionHomePage)
6: select(Auction)
6.1: select(Auction)
6.1.1: display(AuctionInformation)
6.1.1.1: verifyVAlidBid(Auction,Bid)
6.1.1.1.1: setMaxbid(User,bid)
6.1.1.1.2: updateDisplay(Auction)
6.1.1.1.1.1: closeAuction(Auction)
7: provideFeedback(feedback)
6.1.1.1.1.2: display(FeedbackWindow,bidderList)
7.1: addFeedback(feedback,seller)

8. Activity Diagram


Auction
Auction System
Register user within system
User logs in
[No]
correct password
[Yes]
User requests to create an auction for their item
penalty higher than two
[Yes]
Error window opens. User is informed their penalty points are too high to start auction.
[No]
Auction created for current user's item
Start Auction
Auction information is made to all users except seller
New bidder places bid on item
[No]
[Yes]
Is bidder auction seller
bidder rejected
bid higher than current maximum bid
[No]
[Yes]
Bid Accepted and stored and set as maximum bid in auciton
Time before end time
[Yes]
[No]
Cancel Auction
[No]
MergeNode
Close Auction
[Yes]
[No]
cancel request rejected
is user the auction seller
[Yes]
Max bid greater than reserve price
[Yes]
[Yes] Max bid price greater than reserve price
[No]
Increment seller's penalty points
[No]
Inform Max bidder of success
Bidders provide feedback for seller
update seller status
Inform bidders of cancellation

Part B - Modelling in Event B

Context

```
context Context

sets AUCTION USER

constants PENALTY_POINTS AUCTION_STATUS

axioms
  @axm1 PENALTY_POINTS = 0‥3
  @axm2 AUCTION_STATUS = 0‥3
end
```

Machine without refinement

```
machine Machine sees Context

variables users // registered users in the system
          auctions
          active_auctions // auctions that are in progress
          seller bidder
          reserve_price // price set by the seller
          status // status of the auction active, successful, unsuccessful or cancelled
          penalty // penalty points for each user
          bids // bids of all auctions
          max_bid // highest bid of an auction


invariants
  @inv1 users ⊆ USER
  @inv2 auctions ⊆ AUCTION
  @inv3 active_auctions ⊆ auctions
  @inv4 seller ∈ active_auctions → users // total and functional relation between active auctions and users
  @inv5 bidder ∈ auctions ↔ users // a relation between auctions and users
  @inv6 reserve_price ∈ active_auctions → ℕ // total and functional relation between active_auctions and their reserve price
  @inv7 bids ∈ auctions ↔ ℕ // a relation between auctions and its bids
  @inv8 max_bid ∈ auctions → ℕ // total and functional relation between auctions and their max_bid
  @inv9 status ∈ auctions → AUCTION_STATUS // total and functional relation between auction and its status
  @inv10 penalty ∈ users → PENALTY_POINTS // total and functional relation between users and penalty points

events
  event INITIALISATION
    then
      @act1 users ≔ ∅
      @act2 auctions ≔ ∅
      @act3 active_auctions ≔ ∅
      @act4 seller ≔ ∅
      @act5 bidder ≔ ∅
      @act6 reserve_price ≔ ∅
      @act7 bids ≔ ∅
      @act8 status ≔ ∅
      @act9 penalty ≔ ∅
      @act10 max_bid ≔ ∅
  end

  event CreateAuction // Creates a new auction
    any a // auction
       s // seller
       p // price

    where
      @grd1 a ∉ auctions // for auction to be created, it must be verified that it is not an existing auction
      @grd2 s ∈ users // checks that seller is registered
      @grd3 p ∈ ℕ // checks price is a natural number
      @grd4 penalty(s) ≤ 2 // seller's penalty points must be less than or equal to 2
    then
      @act1 active_auctions ≔ active_auctions ∪ {a} // the new auction is added to the set of active auctions
      @act2 seller(a) ≔ s // new seller is added
      @act3 status(a) ≔ 0 // change status to active auction
      @act4 reserve_price(a) ≔ p // reserve price is set
      @act5 auctions ≔ auctions ∪ {a} // new auction is also added to the entire set of auctions
      @act6 max_bid(a) ≔ 0 // maximum bid of the auction is initialised to 0
      @act7 bids ≔ bids ∪ {a ↦ 0} // maximum bid is added to all the bids
  end

  event BidOnAuction // User bids on the auction
    any a // auction
       b // bidder
       bid // bid

    where
      @grd1 a ∈ active_auctions // user can only bid if auction exists
      @grd2 seller(a) ≠ b // the seller of the auction cannot be a bidder too
      @grd3 bid ∈ ℕ // bid has to be greater than 0
      @grd4 max_bid(a) < bid // the bidder can only bid if the bid proposed is greater than the last bid
      @grd5 b ∈ users // the bidder must be a registered user
    then
```

```
      @act1 bidder ≔ bidder ∪ {a ↦ b} // add bidder to list of bidders
      @act2 bids ≔ bids ∪ {a ↦ bid} // add bid to list of bids
      @act3 max_bid(a) ≔ bid // set the current bid as the maximum bid
  end

  event CancelAuctionWithPenalty // User cancels an auction with penalty points
    any a // auction
        s // seller

    where
      @grd1 a ∈ active_auctions // auction must be active
      @grd2 max_bid(a) ≥ reserve_price(a) // for penalty to be given to seller, the bid
accepted must be greater than reserve price
      @grd3 s = seller(a) // the seller must be recognized as the seller of the auction
when it was created
    then
      @act1 active_auctions ≔ active_auctions ∖ {a} // remove the auction from the list
of active auctions
      @act2 seller ≔ {a} □ seller // remove the seller from the auction
      @act3 bidder ≔ {a} □ bidder // remove the bidder from the auction
      @act4 penalty(s) ≔ penalty(s) + 1 // increase penalty of seller by 1
      @act5 status(a) ≔ 1 // set auction status to be 1, which signifies cancelled
      @act6 reserve_price ≔ {a} □ reserve_price // remove the reserve price from the
auction
  end

  event CancelAuctionWithoutPenalty // User cancels an auction without penalty
points
    any a // auction

    where
      @grd1 a ∈ active_auctions // auction must be active
      @grd2 max_bid(a) < reserve_price(a) // for penalty to be given to seller, the bid
accepted must be lower than reserve price
    then
      @act1 active_auctions ≔ active_auctions ∖ {a} // remove the auction from the list
of active auctions
      @act2 seller ≔ {a} □ seller // remove the seller from the auction
      @act3 bidder ≔ {a} □ bidder // remove the bidder from the auction
      @act4 status(a) ≔ 1 // set auction status to be 1, which signifies cancelled
      @act5 reserve_price ≔ {a} □ reserve_price // remove the reserve price from the
auction
```

```
  end

  event GetAuctionStatus // Holds the status of the auction
    any a // auction
       s // status

    where
      @grd1 a ∈ auctions // the auction must already be in the set of all auctions
      @grd2 status(a) = s // the status has to be @s
  end

  event CloseSuccessfulAuction // Closes a successful auction
    any a // auction

    where
      @grd1 a ∈ active_auctions // auction must be active
      @grd2 max_bid(a) ≥ reserve_price(a) // the highest bid proposed must be higher
than or equal to the reserve price
    then
      @act1 active_auctions ≔ active_auctions ∖ {a} // removes this auction from the
list of active auctions
      @act2 seller ≔ {a} □ seller // removes seller from auction
      @act3 bidder ≔ {a} □ bidder // removes bidder from auction
      @act4 status(a) ≔ 2 // sets the status as successful, denoted by integer 2
      @act5 reserve_price ≔ {a} □ reserve_price // remove reserve price from the
auction
  end

  event CloseUnsuccessfulAuction // Closes an unsucessful auction
    any a // auction

    where
      @grd1 a ∈ active_auctions // auction must be active
      @grd2 max_bid(a) < reserve_price(a) // the highest bid proposed must be less
than the reserve price
    then
      @act1 active_auctions ≔ active_auctions ∖ {a} // removes this auction from the
list of active auctions
      @act2 seller ≔ {a} □ seller // removes seller from auction
      @act3 bidder ≔ {a} □ bidder // removes bidder from auction
      @act4 status(a) ≔ 3 // sets the status as unsuccessful, denoted by integer 3
      @act5 reserve_price ≔ {a} □ reserve_price // remove reserve price from the auction
  end

  event GetBidsHistory // Gets all the bids on an auction
    any a // auction
        h // history of bids

    where
      @grd1 a ∈ auctions // the auction exists
      @grd2 bids[{a}] = h // all the bids of the auction have to be @h
  end

  event CreateUser // Creates a new user
    any u // user to be created

    where
      @grd1 u ∈ USER // @u is of type USER
      @grd2 u ∉ users // @u is not a registered user
    then
      @act1 users ≔ users ∪ {u} // @u is added to the list of registered users
      @act2 penalty(u) ≔ 0 // penalty is set to 0 for a new user
  end
end
```

Machine with refinement

```
machine Machine0 refines Machine  sees Context

variables users auctions active_auctions seller bidder reserve_price status penalty
bids max_bid
        time // current time
        start_time // time at which the auction is started
        end_time // time at which the auction is ended



invariants
  @inv1 time ∈ ℕ // time is modelled as a natural number
  @inv2 start_time ∈ active_auctions → ℕ
  @inv3 end_time ∈ active_auctions → ℕ
  @inv4 ∀a·a ∈ active_auctions ⇒ start_time(a) < end_time(a) // each active
auction must start before it ends

events
  event INITIALISATION extends INITIALISATION
    then
      @act11 time ≔ 0 // time starts at 0
      @act12 start_time ≔ ∅ // start time is initially an empty set
      @act13 end_time ≔ ∅ // end time is initially an empty set
  end

  event CreateAuction extends CreateAuction
    any s_time // start time
        e_time // end time

    where
      @grd5 s_time ∈ ℕ // start time belongs to the set of natural numbers
      @grd6 e_time ∈ ℕ // end time belongs to the set of natural numbers
      @grd7 s_time ≥ time // time must be started after or at the current time
      @grd8 e_time > s_time // the end time must be after the start time
    then
      @act8 start_time(a) ≔ s_time // set the start time of the auction
      @act9 end_time(a) ≔ e_time // set the end time of the auction
  end

  event BidOnAuction extends BidOnAuction
    where
```

```
      @grd6 start_time(a) ≤ time // user can only bid after the start time
      @grd7 end_time(a) ≥ time // the end time of bidding must be after the current
time
  end

  event CancelAuctionWithPenalty extends CancelAuctionWithPenalty
    where
      @grd4 start_time(a) < time // start time must have been before the current time
      @grd5 end_time(a) > time // end time must be after the current time
    then
      @act7 start_time ≔ {a} □ start_time // remove the start time from the auction
      @act8 end_time ≔ {a} □ end_time // remove the end time from the auction
  end

  event CancelAuctionWithoutPenalty extends CancelAuctionWithoutPenalty
    where
      @grd3 start_time(a) < time // start time must have been before the current time
      @grd4 end_time(a) > time // end time must be after the current time
    then
      @act6 start_time ≔ {a} □ start_time // remove the start time from the auction
      @act7 end_time ≔ {a} □ end_time // remove the end time from the auction
  end

  event GetAuctionStatus extends GetAuctionStatus
  end

  event CloseSuccessfulAuction extends CloseSuccessfulAuction
    where
      @grd3 start_time(a) < time // start time must have been before the current time
      @grd4 end_time(a) = time // end time must be the current time
    then
      @act6 start_time ≔ {a} □ start_time // remove the start time from the auction
      @act7 end_time ≔ {a} □ end_time // remove the end time from the auction
  end

  event Clock
    then
      @act1 time ≔ time + 1 // increment the time by 1
  end

  event CloseUnsuccessfulAuction extends CloseUnsuccessfulAuction
    where
```

```
      @grd3 start_time(a) < time // start time must have been before the current time
      @grd4 end_time(a) = time // end time must be the current time
    then
      @act6 start_time ≔ {a} □ start_time // remove the start time from the auction
      @act7 end_time ≔ {a} □ end_time // remove the end time from the auction
  end

  event GetBidsHistory extends GetBidsHistory
  end

  event CreateUser extends CreateUser
  end
end
```