Andreas Nearchou, UUN: s1722294
Petros Zantis, UUN: s1703728

# 1 Introduction

The following report is the design of a system concerning an Auction House that wants to run auctions at its premises, while allowing buyers to bid using electronic devices. Further uses of the system are to store information about sellers, buyers and lots, in addition to controlling bank transfers and messages.
More details on the specific instructions and guidance for this design can be found in Coursework 1 and 2.

# 2 Static model

## 2.1 UML class model

See Figure 1, page 4 and Figure 2, page 5.

## 2.2 High-level description

- **Buyer**, **Seller**, **MemberOfPublic** and **Auctioneer** are actor classes which contain methods representing interactions of each actor with the system.
  Each of the actor classes can interact with the system by sending in a message, calling the **AuctionHouse** class. In turn, the **AuctionHouse** class forwards that message to a class suitable for executing that message.

  An alternate design of this would have been to link the actor classes directly with the suitable classes, but we are instructed to follow the Facade pattern design.

- **BuyerInfo**, **SellerInfo** and **AuctioneerInfo** classes contain as their attributes all the required information for each of the actors in the system.
  Note that the **BuyerInfo** class contains an *authorisation* attribute, which ensures that the buyer has given the Auction House authorisation during registration to collect payments.

- A class called **Catalogue** is created, with a single operation which takes objects of type **CatalogueEntry,** and adds it in a list of Catalogue Entries (i.e. the **Catalogue**).

  An alternative design option would be if an attribute called *catalogue*, which is a list of Catalogue Entries, was created inside the **AuctionHouse** class and the **AuctionHouse** class was linked to the **CatalogueEntry** class with a one-to-many relationship.

- In this UML class diagram, Buyers and Sellers choose their unique usernames during their registration. But no information is given on how Auctioneers are entered into the system, so a unique username for each Auctioneer is assumed.

Each operation can be called only by a specific user, via entering their unique username as a parameter, which is used to validate the identity of the caller.

- When a user, or any member of the public, wishes to view the online catalogue, they simply call the **AuctionHouse** class through the *viewCatalogue* method, which returns a list of Catalogue Entries (i.e. the catalogue).

- When a **Buyer** wants to note their interest in a lot, they call the **AuctionHouse** class through the *noteInterest* method, by providing their unique username and the *lotID* as arguments. Then, the **AuctionHouse** class calls the **CatalogueEntry** class through the *addInterestedBuyer* method, which adds the username of the interested buyer in a list called *interestedBuyers*, which is an attribute of the **CatalogueEntry** class.

- This diagram contains an **Auction** class, referring to each unique auction session taking place in the Auction House premises. Each session is given a unique identification number (*auctionID*) and contains the *lotID* of the lot that is being auctioned, as an attribute.
  Each time a **Buyer** makes a new bid, by calling the **AuctionHouse** class through the *makeBid* method, *currentBid* and *currentBidderUsername* attributes of the **Auction** class are updated through *setCurrentBid* and *setCurrentBidder* methods respectively.

- When an **Auctioneer** opens an auction, they call the **AuctionHouse** class through the *openAuction* method, while passing to it arguments such as their unique username, the *lotID*, the *auctionID* and the *initialBiddingPrice* decided using the low estimate given by the seller of the lot. Consequently, the **AuctionHouse** class obtains a list of the Messaging Addresses of the interested buyers via the *getInterestedBuyers* method in the **Catalogue** class and merges it with the messaging addresses of the **Auctioneer** and the **Seller**, into a list called *usersInvolved*. It then invokes the constructor in the **Auction** class by passing it these parameters and setting the value of *isOpen* to true.

  Similarly, to close an auction, an auctioneer calls the **AuctionHouse** class through the *closeAuction* method, by passing to it arguments such as their unique username, the *lotID*, the *auctionID* and the *hammerPrice.* The *reservePrice* is compared with the hammer price (the last *currentBid*), and if it is lower than or equal, the **AuctionHouse** class calls the **Auction** class through the *close* method, which also sets the value of *isOpen* to false. The **BankingService** class is called through the *transfer* method, and the appropriate money transfers are performed via a method in this class.

- In the cases of *transfer*, *noteInterest* and *makeBid* methods, an instance of type **Status** is returned, to indicate if the methods were carried out successfully.

- The class called **MessagingService** in this design, contains three distinct methods that correspond to different use cases, and are used to notify the relevant users (from the list of Messaging Addresses), and give them information required.

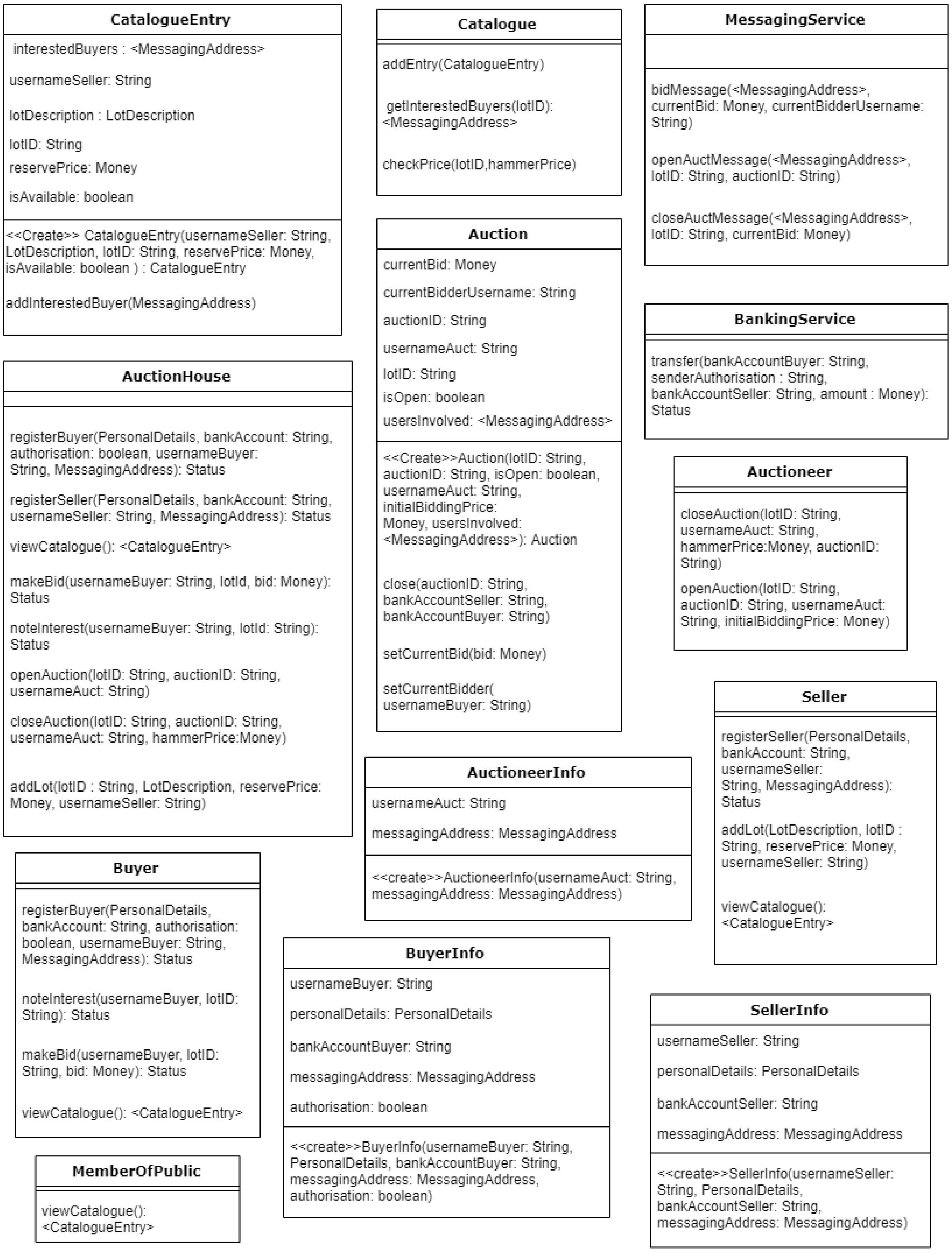
CatalogueEntry
interestedBuyers : <MessagingAddress>
usernameSeller: String
lotDescription : LotDescription
lotID: String
reservePrice: Money
isAvailable: boolean
<<Create>> CatalogueEntry(usernameSeller: String, LotDescription, lotID: String, reservePrice: Money, isAvailable: boolean ) : CatalogueEntry
addInterestedBuyer(MessagingAddress)
Catalogue
addEntry(CatalogueEntry)
getInterestedBuyers(lotID): <MessagingAddress>
checkPrice(lotID,hammerPrice)
MessagingService
bidMessage(<MessagingAddress>, currentBid: Money, currentBidderUsername: String)
openAuctMessage(<MessagingAddress>, lotID: String, auctionID: String)
closeAuctMessage(<MessagingAddress>, lotID: String, currentBid: Money)
Auction
currentBid: Money
currentBidderUsername: String
auctionID: String
usernameAuct: String
lotID: String
isOpen: boolean
usersInvolved: <MessagingAddress>
<<Create>>Auction(lotID: String, auctionID: String, isOpen: boolean, usernameAuct: String, initialBiddingPrice: Money, usersInvolved: <MessagingAddress>): Auction
close(auctionID: String, bankAccountSeller: String, bankAccountBuyer: String)
setCurrentBid(bid: Money)
setCurrentBidder( usernameBuyer: String)
BankingService
transfer(bankAccountBuyer: String, senderAuthorisation : String, bankAccountSeller: String, amount : Money): Status
AuctionHouse
registerBuyer(PersonalDetails, bankAccount: String, authorisation: boolean, usernameBuyer: String, MessagingAddress): Status
registerSeller(PersonalDetails, bankAccount: String, usernameSeller: String, MessagingAddress): Status
viewCatalogue(): <CatalogueEntry>
makeBid(usernameBuyer: String, lotId, bid: Money): Status
noteInterest(usernameBuyer: String, lotId: String): Status
openAuction(lotID: String, auctionID: String, usernameAuct: String)
closeAuction(lotID: String, auctionID: String, usernameAuct: String, hammerPrice:Money)
addLot(lotID : String, LotDescription, reservePrice: Money, usernameSeller: String)
Auctioneer
closeAuction(lotID: String, usernameAuct: String, hammerPrice:Money, auctionID: String)
openAuction(lotID: String, auctionID: String, usernameAuct: String, initialBiddingPrice: Money)
Seller
registerSeller(PersonalDetails, bankAccount: String, usernameSeller: String, MessagingAddress): Status
addLot(LotDescription, lotID : String, reservePrice: Money, usernameSeller: String)
viewCatalogue(): <CatalogueEntry>
AuctioneerInfo
usernameAuct: String
messagingAddress: MessagingAddress
<<create>>AuctioneerInfo(usernameAuct: String, messagingAddress: MessagingAddress)
Buyer
registerBuyer(PersonalDetails, bankAccount: String, authorisation: boolean, usernameBuyer: String, MessagingAddress): Status
noteInterest(usernameBuyer, lotID: String): Status
makeBid(usernameBuyer, lotID: String, bid: Money): Status
viewCatalogue(): <CatalogueEntry>
BuyerInfo
usernameBuyer: String
personalDetails: PersonalDetails
bankAccountBuyer: String
messagingAddress: MessagingAddress
authorisation: boolean
<<create>>BuyerInfo(usernameBuyer: String, PersonalDetails, bankAccountBuyer: String, messagingAddress: MessagingAddress, authorisation: boolean)
SellerInfo
usernameSeller: String
personalDetails: PersonalDetails
bankAccountSeller: String
messagingAddress: MessagingAddress
<<create>>SellerInfo(usernameSeller: String, PersonalDetails, bankAccountSeller: String, messagingAddress: MessagingAddress)
MemberOfPublic
viewCatalogue(): <CatalogueEntry>

Figure 1 – UML Class Diagram (full information)

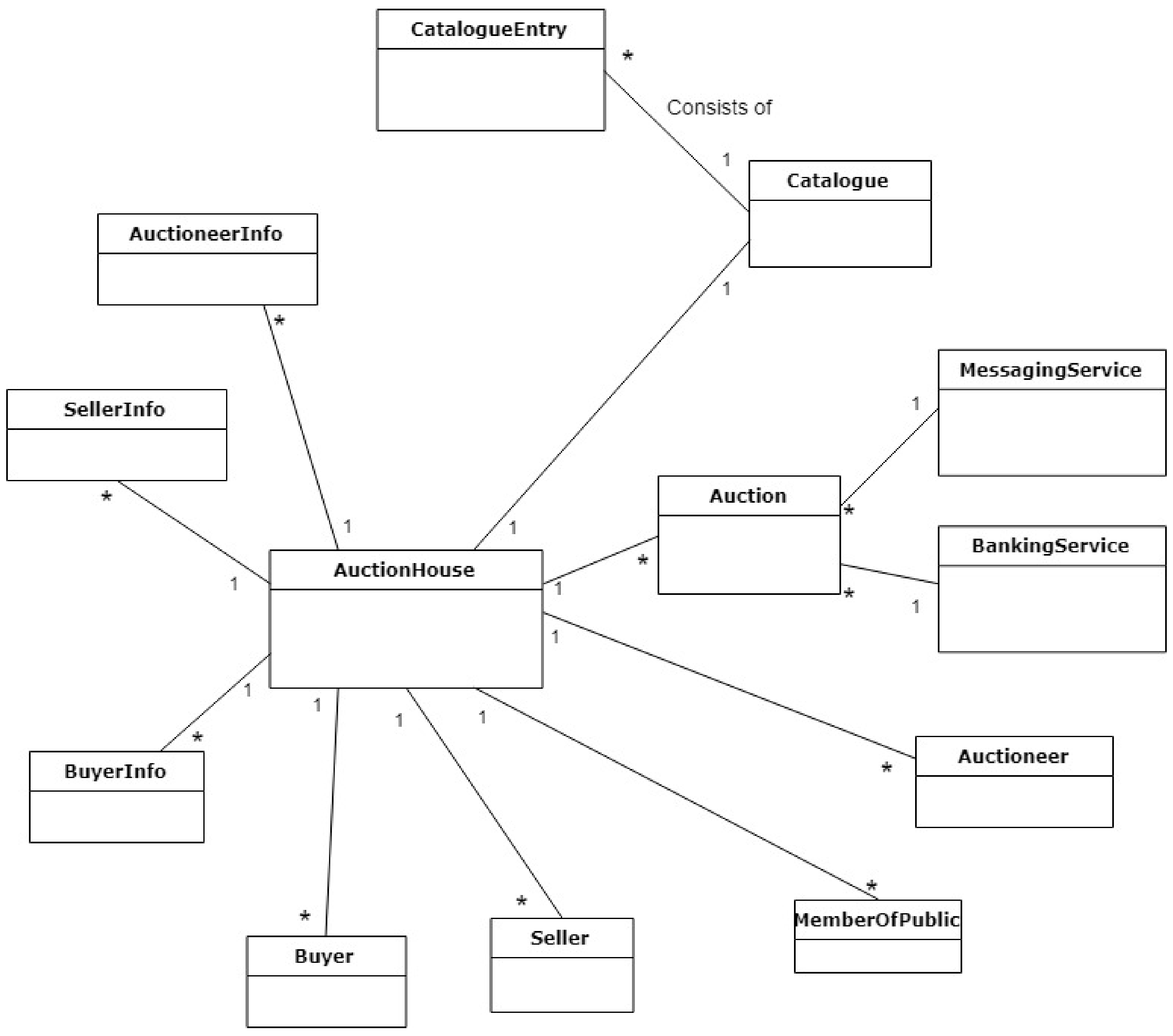

Figure 2 – UML Class Diagram (associations and dependency information)

# 3 Dynamic models

## 3.1 UML sequence diagram

See Figure 3, page 8 for a sequence diagram showing the dynamic behaviour of the Close Auction use case.

- This sequence diagram does not show how the methods in **BankingService** and **MessagingService** classes function, since their implementation is abstract for this Coursework.
- The function of the loop is to match the *lotID* of the auctioned lot with the *lotID* of a **CatalogueEntry** in the **Catalogue**, and retrieve its *reservePrice*, so it can be compared to the hammer price.
- If the hammer price is higher than or equal to the reserve price, the username of the seller is returned to the **AuctionHouse** class, so it can be used to retrieve the seller's Bank Account number.
- This sequence diagram does not show what will happen if the hammer price is lower than the reserve price. This case is one of the ambiguities of Coursework 1.

## 3.2 Behaviour descriptions

### i. addLot:

```
aSeller -- addLot() --> theAuctionHouse
        theAuctionHouse -- addEntry() --> theCatalogue
                theCatalogue -- <<create>>CatalogueEntry --> aCatalogueEntry
                theCatalogue <-- newCatalogueEntry -- aCatalogueEntry
```

## ii. noteInterest:

```
aBuyer -- noteInterest(usernameBuyer, lotID) -->  theAuctionHouse
        theAuctionHouse -- getMessagingAddress() --> aBuyerInfo
        theAuctionHouse <-- MessagingAddress -- aBuyerInfo

        LOOP [For each CatalogueEntry in the Catalogue]
        theAuctionHouse -- getLotID --> aCatalogueEntry
        theAuctionHouse <-- lotID -- aCatalogueEntry
        END LOOP

        //For the specific lot intended by the buyer
        theAuctionHouse -- addInterestedBuyer(MessagingAddress) --> aCatalogueEntry
        theAuctionHouse <-- -- -- aCatalogueEntry

aBuyer <-- Status – theAuctionHouse
```

## iii. makeBid:

```
aBuyer -- makeBid(usernameBuyer , bid, lotID) -- > theAuctionHouse

        theAuctionHouse -- getIsOpen() -- > anAuction
        theAuctionHouse <-- -- --  anAuction
        IF [ isOpen == true ]
        theAuctionHouse -- setCurrentBid(bid) -- > anAuction
        theAuctionHouse < -- -- -- anAuction
        theAuctionHouse -- setCurrentBidder(usernameBuyer) -- > anAuction

                anAuction -- bidMessage(usersInvolved, currentBid, currentBidder) -->
        theMessagingService

aBuyer <-- Status -- theAuctionHouse
```

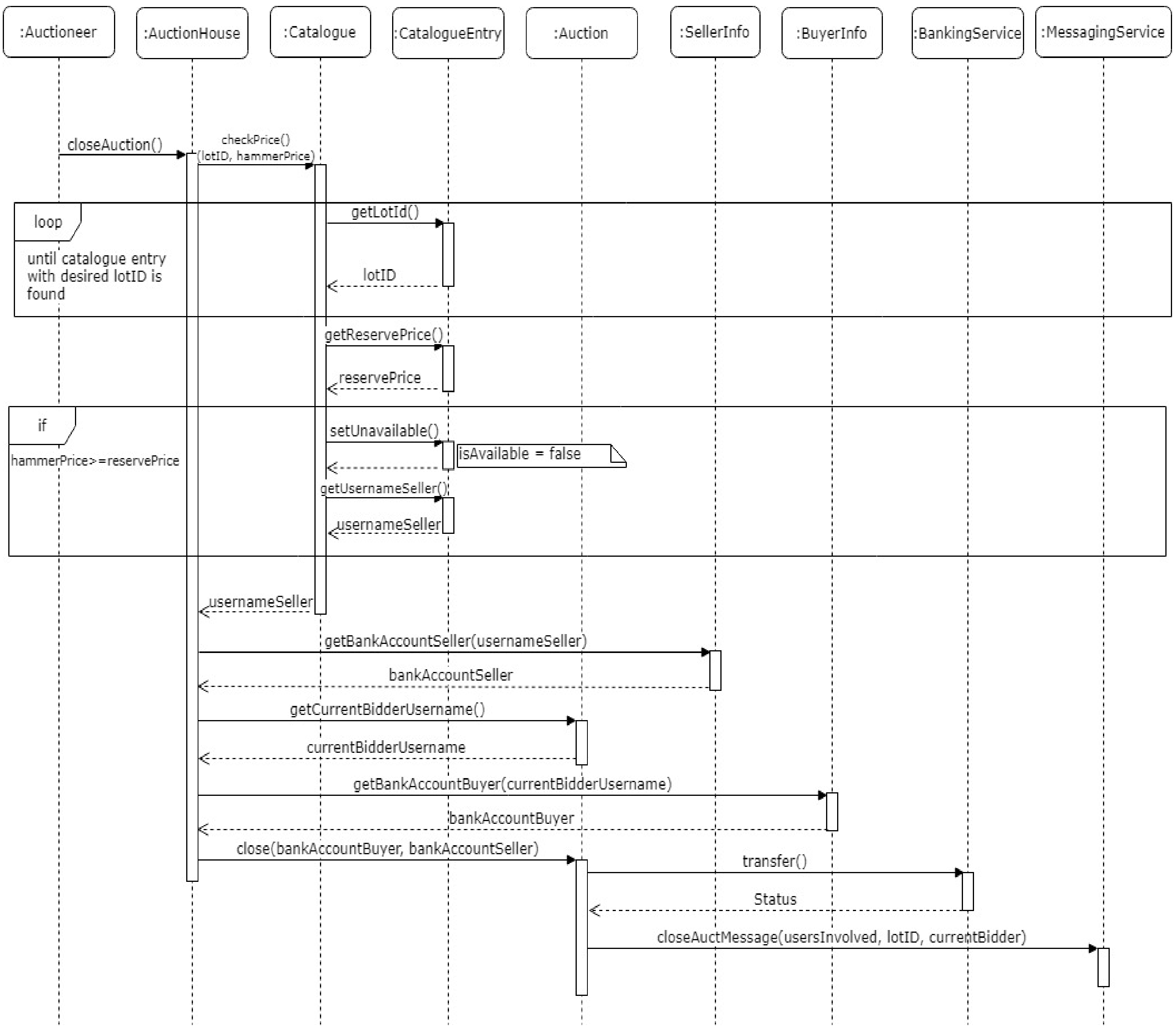

Figure 3 - UML sequence diagram