Bryan Vullo (bv1g22)
James Staton (js5n21)
Mohannad Elsaadawy (me1n21)
Rabia Bilmen (rmb1n21)
Shaho Zagrosi (sz10g21)

COMP1216. Software Modelling and Design (2022-23)

Group 32: Silk Road - Online Auction Service

Submission Date: 12 May 2023

1 Introduction

1.1 Objective

The aim of this coursework is to model an online auction service, self-named 'Silk Road', with Event-B. In this coursework, we aim to develop a formal model of an online auction service, named 'Silk Road', using the Event-B modeling language. The primary objective is to provide a rigorous and comprehensive representation of the system's key components, such as user registration, item submission, bidding, auction management, and user feedback. By employing Event-B, we ensure a systematic approach to specifying, designing, and verifying the correctness of our model, ultimately enhancing the reliability and robustness of the Silk Road online auction service.

2 Event-B model

```
context c0

sets
 USER
 NAME
 PASSWORD

end
```

```
machine m0
sees c0

variables
 users
 name
 password
 loginID
 loggedIn

invariants
 @inv−users: users ⊆ USER
 @inv−name: name ∈ users → NAME
 @inv−password: password ∈ users → PASSWORD
 @inv−loginID: loginID ∈ users ⤔ ℕ
 @inv−logged−in: loggedIn ⊆ users

events

 event INITIALISATION
 begin
  @init−users: users := ∅
  @init−name: name := ∅
  @init−password: password := ∅
  @init−loginID: loginID := ∅
  @init−logged−in: loggedIn := ∅
 end

 // Event to register a user
 event RegisterUser
  any
   u n p l
  where
   @grd1: u ∈ USER
   @grd2: u ∉ users
   @grd3: n ∈ NAME
   @grd4: p ∈ PASSWORD
   @grd5: l ∈ ℕ
   @grd6: l ∉ ran(loginID)
  then
   @act−add−user: users := users ∪ {u}
   @act−set−name: name(u) := n
```

```
    @act−set−password: password(u) := p
    @act−set−loginID: loginID(u) := l
  end

  // Event to log in a user
  event LogIn
   any
    u p ID
   where
    @grd1: u ∈ users
    @grd2: u ∉ loggedIn
    @grd3: password(u) = p
    @grd4: loginID(u) = ID
   then
    @act−add−user: loggedIn := loggedIn ∪ {u}
  end

  // Event to log out a user
  event LogOut
   any
    u
   where
    @grd1: u ∈ loggedIn
   then
    @act−log−out: loggedIn := loggedIn \ {u}
  end

end
```

```
context c1
extends c0

sets
 ITEM
 AUCTION
 BID
 FEEDBACK

end
```

```
machine m1
refines m0
sees c1
variables
 users
 name
 password
 loginID
 loggedIn
 userPenalty
 auctions
 item
```

```
13  seller
14  topBidder
15  bidders
16  reservePrice
17  currentBid
18  bids
19  bidsOfAuction
20  bidValue
21  bidder
22
23 invariants
24  @inv−userPenalty: userPenalty ∈ users → ℕ
25  @inv−auctions: auctions ⊆ AUCTION
26  @inv−auction−item: item ∈ auctions ⤔ ITEM
27  @inv−seller: seller ∈ auctions → users
28  @inv−reserve−price: reservePrice ∈ auctions → ℕ1
29  @inv−top−bidder: topBidder ∈ auctions ⇸ users
30  @inv−bidders: bidders ∈ auctions ↔ users
31  @inv−bids: bids ⊆ BID
32  @inv−current−bid: currentBid ∈ auctions ⇸ bids
33  @inv−bids−auction: bidsOfAuction ∈ auctions ↔ bids
34  @inv−bid−value: bidValue ∈ bids → ℕ
35  @inv−bidder: bidder ∈ bids → users
36 events
37  // Event to initialize the system
38  event INITIALISATION extends INITIALISATION
39  then
40   @init−userPenalty: userPenalty := ∅
41   @init−auctions: auctions := ∅
42   @init−item: item := ∅
43   @init−seller: seller := ∅
44   @init−currentBids: currentBid := ∅
45   @init−reservePrice: reservePrice := ∅
46   @init−topBidder: topBidder := ∅
47   @init−bidders: bidders := ∅
48   @init−bids: bids := ∅
49   @init−bids−auction: bidsOfAuction := ∅
50   @init−bid−value: bidValue := ∅
51   @init−bidder: bidder := ∅
52  end
53
54  // Event to create an auction
55  event CreateAuction
56  any
57   a
58   u
59   r
60   i
61   b
62  where
63   @grd1: a ∉ auctions
64   @grd2: u ∈ loggedIn
65   @grd3: userPenalty(u) ≤ 2
66   @grd4: r ∈ ℕ1
67   @grd5: i ∉ ran(item)
68   @grd6: b ∉ bids
```

```
then
 @act1: auctions := auctions ∪ {a}
 @act2: seller(a) := u
 @act3: reservePrice(a) := r
 @act4: item(a) := i
 @act5: bids := bids ∪ {b}
 @act6: bidsOfAuction := bidsOfAuction ∪ {a ↦ b}
 @act7: bidValue(b) := 0
 @act8: bidder(b) := u
 @act9: currentBid(a) := b
end

// Event to cancel an auction without penalty
event CancelAuctionWithoutPenalty
any
 a
 u
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ auctions
 @grd3: a ∈ dom(seller)
 @grd4: seller(a) = u
 @grd5: a ∈ dom(currentBid)
 @grd6: (currentBid; bidValue)(a) < reservePrice(a)
end

// Event to cancel an auction with penalty
event CancelAuctionWithPenalty
any
 a
 u
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ auctions
 @grd3: a ∈ dom(seller)
 @grd4: seller(a) = u
 @grd5: a ∈ dom(currentBid)
 @grd6: (currentBid; bidValue)(a) ≥ reservePrice(a)
then
 @act1: userPenalty(u) := userPenalty(u) + 1
end

// Event to close an auction successfully
event CloseAuctionSuccess
any
 a
 u
where
 @grd1: a ∈ auctions
 @grd2: a ∈ dom(currentBid)
 @grd3: (currentBid; bidValue)(a) ≥ reservePrice(a)
 @grd4: u ∈ users
 @grd5: a ∈ dom(topBidder)
 @grd6: topBidder(a) = u
end
```

```
// Event to close an auction unsuccessfully
event CloseAuctionFail
any
 a
where
 @grd1: a ∈ auctions
 @grd2: a ∈ dom(currentBid)
 @grd3: (currentBid; bidValue)(a) < reservePrice(a)
end

// Event to place a bid
event Bid
any
 a
 u
 b
 v
where
 @grd1: a ∈ auctions
 @grd2: u ∈ loggedIn
 @grd3: a ∈ dom(seller)
 @grd4: seller(a) ≠ u
 @grd5: b ∈ BID
 @grd6: v ∈ ℕ
 @grd7: a ∈ dom(currentBid)
 @grd8: (currentBid; bidValue)(a) < v
then
 @act1: topBidder(a) := u
 @act2: bidders := bidders ∪ {a ↦ u}
 @act3: bids := bids ∪ {b}
 @act4: currentBid(a) := b
 @act5: bidsOfAuction := bidsOfAuction ∪ {a ↦ b}
 @act6: bidValue(b) := v
 @act7: bidder(b) := u
end

// Event to get the history of an auction
event getAuctionHistory
any
 a
 result
where
 @grd1: a ∈ auctions
 @grd2: result = bidsOfAuction[{a}]
end

// Event to log in
event LogIn extends LogIn
end

// Event to log out
event LogOut extends LogOut
end

// Event to register a new user
// It also sets the user's initial penalty to 0
```

```
181  event RegisterUser extends RegisterUser
182   then
183     @act−set−penalty: userPenalty(u) := 0
184  end
185
186 end
```

```
1  machine m2
2  refines m1
3  sees c1
4
5  variables
6   users
7   name
8   password
9   loginID
10  loggedIn
11  userPenalty
12  auctions
13  createdAuctions
14  liveAuctions
15  closedFailed
16  closedSuccess
17  cancelled
18  item
19  seller
20  topBidder
21  bidders
22  reservePrice
23  currentBid
24  bids
25  bidsOfAuction
26  bidValue
27  bidder
28
29  time
30  startTime
31  endTime
32  feedbackDuration
33  winnerNotif
34  cancelledNotif
35  status
36
37 invariants
38   @inv−time: time ∈ ℕ
39   @inv−feedback−duration: feedbackDuration ∈ ℕ1
40   @inv−start−time: startTime ∈ auctions → ℕ
41   @inv−end−time: endTime ∈ auctions → ℕ
42   @inv−live−auction−duration: ∀a · a ∈ auctions ⇒ startTime(a) < endTime(a)
43   @inv−partition−auctions: partition(auctions, createdAuctions, liveAuctions, closedFailed, closedSuccess,
         cancelled)
44   @inv−winner−notif: winnerNotif ∈ closedSuccess ↔ users
45   @inv−cancelled−notif: cancelledNotif ∈ cancelled ↔ users
46   @inv−status: status ∈ users ↔ FEEDBACK
```

```
47
48 events
49  // Initialisation event to set starting state of variables
50  event INITIALISATION extends INITIALISATION
51  then
52   @begin−time: time := 0
53   @init−feedback−duration: feedbackDuration := 5
54   @init−startTime: startTime := ∅
55   @init−endTime: endTime := ∅
56   @init−winner−notif: winnerNotif := ∅
57   @init−cancelled−notif: cancelledNotif := ∅
58   @init−status: status := ∅
59   @init−: createdAuctions := ∅
60   @init−liveAuctions: liveAuctions := ∅
61   @init−closedFailed: closedFailed := ∅
62   @init−closedSuccess: closedSuccess := ∅
63   @init−cancelled: cancelled := ∅
64  end
65
66  // Event for increasing the system time
67  event Clock
68  then
69   @act1: time := time + 1
70  end
71
72  event CreateAuction extends CreateAuction
73  any
74   startT
75   endT
76  where
77   @grd7: startT ∈ ℕ
78   @grd8: endT ∈ ℕ
79   @grd9: startT < endT
80   @grd10: startT ≥ time
81  then
82   @act10: createdAuctions := createdAuctions ∪ {a}
83   @act11: startTime(a) := startT
84   @act12: endTime(a) := endT
85  end
86
87  // Event for starting an auction
88  event StartAuction
89  any
90   a
91  where
92   @grd1: a ∈ createdAuctions
93   @grd2: startTime(a) = time
94  then
95   @act1: createdAuctions := createdAuctions \ {a}
96   @act2: liveAuctions := liveAuctions ∪ {a}
97  end
98
99  // Event for cancelling an auction without a penalty
100 event CancelAuctionWithoutPenalty extends CancelAuctionWithoutPenalty
101 where
102  @grd7: a ∈ liveAuctions
```

```
103   @grd8: endTime(a) > time
104   @grd9: startTime(a) < time
105  then
106   @act1: liveAuctions := liveAuctions \ {a}
107   @act2: cancelled := cancelled ∪ {a}
108   @act3: endTime(a) := time
109   @act4: cancelledNotif := cancelledNotif ∪ ({a} × bidders[{a}])
110  end
111
112  // Event for cancelling an auction with a penalty
113  event CancelAuctionWithPenalty extends CancelAuctionWithPenalty
114  where
115   @grd7: a ∈ liveAuctions
116   @grd8: endTime(a) > time
117   @grd9: startTime(a) < time
118  then
119   @act2: liveAuctions := liveAuctions \ {a}
120   @act3: cancelled := cancelled ∪ {a}
121   @act4: endTime(a) := time
122   @act5: cancelledNotif := cancelledNotif ∪ ({a} × bidders[{a}])
123  end
124
125  // Event for closing an auction successfully
126  event CloseAuctionSuccess extends CloseAuctionSuccess
127  where
128   @grd7: a ∈ liveAuctions
129   @grd8: a ∈ dom(endTime)
130   @grd9: endTime(a) ≤ time
131  then
132   @act1: liveAuctions := liveAuctions \ {a}
133   @act2: closedSuccess := closedSuccess ∪ {a}
134   @act3: winnerNotif := winnerNotif ∪ {a ↦ u}
135  end
136
137  // Event for closing an auction without a successful bid
138  event CloseAuctionFail extends CloseAuctionFail
139  where
140   @grd4: a ∈ liveAuctions
141   @grd5: a ∈ dom(endTime)
142   @grd6: endTime(a) ≤ time
143  then
144   @act1: liveAuctions := liveAuctions \ {a}
145   @act2: closedFailed := closedFailed ∪ {a}
146  end
147
148  // Event for placing a bid on an auction
149  event Bid extends Bid
150  where
151   @grd9: a ∈ liveAuctions
152  end
153
154  // Event for getting auction history
155  event getAuctionHistory extends getAuctionHistory
156  end
157
158  // Event for viewing status of a created auction
```

```
event ViewStatusCreatedAuction
any
 a
 u
 result
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ createdAuctions
 @grd3: seller(a) = u
 @grd4: result = a
end

// Event for viewing status of a live auction
event ViewStatusLiveAuction
any
 a
 u
 result
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ liveAuctions
 @grd3: seller(a) = u
 @grd4: result = a
end

// Event for viewing status of a cancelled auction
event ViewStatusCancelledAuction
any
 a
 u
 result
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ cancelled
 @grd3: seller(a) = u
 @grd4: result = a
end

// Event for viewing status of a failed auction
event ViewStatusClosedFailedAuction
any
 a
 u
 result
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ closedFailed
 @grd3: seller(a) = u
 @grd4: result = a
end

// Event for viewing status of a successfully closed auction
event ViewStatusClosedSuccessAuction
any
 a
 u
```

```
 result
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ closedSuccess
 @grd3: seller(a) = u
 @grd4: result = a
end

// Event for giving feedback on an auction
event GiveFeedback
any
 u
 a
 f
where
 @grd1: u ∈ loggedIn
 @grd2: a ∈ closedSuccess ∪ closedFailed ∪ cancelled
 @grd3: seller(a) ≠ u
 @grd4: u ∈ bidders[{a}] // checks if the user is a bidder of the closed auction
 @grd5: (endTime(a) + feedbackDuration) ≥ time
 @grd6: endTime(a) < time
 @grd7: f ∉ ran(status)
then
 @act2: status := status ∪ {seller(a) ↦ f}
end

// Event for logging in a user
event LogIn extends LogIn
end

// Event for logging out a user
event LogOut extends LogOut
end

// Event for registering a new user
event RegisterUser extends RegisterUser
end

end
```