University of Southampton

Anuj Rudra (ar28g22@soton.ac.uk)
Aryan Patel (ap14g22@soton.ac.uk)
Ruichong Peng (rp10g22@soton.ac.uk)
Jincheng Guo (jg16n22@soton.ac.uk)
XiangWei Low (xwl1n21@soton.ac.uk)

COMP1216. Software Modelling and Design (2022-23)

Group 50: An Online Auction Service

Submission date: 12 May 2023

1 Introduction

Anuj Rudra - create an auction, start an auction
Aryan Patel - user register with the system, user logs in, user logs out
Ruichong Peng - gives feedback to the seller, view the history of all bids on an auction
Jincheng Guo - view the status of an auction, close an auction
XiangWei Low - bid on an auction, cancel an auction

2 Event-B model

You are recommended to use the lstEventB.sty package for listing your Event-B model. We use the sample solution for Lab 7 (a hotel reception system) as an example here. The magic command is \EventBinputlisting to include any CamilleX source files (*.bucx, *.bumx) into your report.

```
context UserContext
sets
 USERNAME //The set of usernames
 PASSWORD //The set of passwords
end
```

```
context AuctionContext
sets
 AUCTIONS // The set of Auctions
 AUCTIONID // The set of AuctionIDs that are given to the auctions
 AUCTIONRESERVEPRICE // The reserve price of the auction
 ACTIVEAUCTION // The set of Live Auction
end
```

```
context TokenContext
sets
TOKEN //the set of tokens
end
```

```
context UserStatusContext
sets
 FEEDBACK
 PENALTYPOINTS
constants
 maxpenaltypoints
axioms
 theorem @axm1: maxpenaltypoints ∈ ℤ //The user has penalty points which are integers
 @axm2: maxpenaltypoints < 3 // The user can only have a max of 2 penalty points otherwise they cannot
      make an auction
end
```

```
context viewStatusContext
sets
 BIDS
 STATUS
 ITEM
constants
 reservePrice
 ongoing
 successful
```

```
  failed
  cancelled
axioms
  @axm1: reservePrice ∈ ℕ
  @axm2: STATUS = {ongoing,successful,failed,cancelled}
end
```

```
machine m0
sees UserContext
variables
  username //The set of usernames
  password //The set of passwords
  users //The set of users
  login //Set containing users that are logged in
  logout //Set containing users that are logged out
invariants
  @inv1: username ⊆ USERNAME //Set of usernames
  @inv2: password ⊆ PASSWORD //Set of passwords
  @inv3: users ∈ username → password //Set of users where each username is linked to 1 password
  @inv4: partition(username, login, logout) //All users are put into either login set or logout set, a user
       cannot be in both sets
events
  event INITIALISATION
  then
    @init−login: login := ∅ //There are no users logged in
    @init−logout: logout := ∅ //There are no users logged out
    @init−users: users := ∅ //There are no users in the system registered
    @init−username: username := ∅ //There are no usernames registered
    @init−password: password := ∅ //There are no passwords registered
  end

  event login
  any
    name //Username for login
    pass //Password for login
  where
    @grd1: name ∉ login //The user is not already logged in
    @grd2: name ↦ pass ∈ users //The username and password combination match and exist in users set
  then
    @act1: login := login ∪ {name} //Add the user to the login set
    @act2: logout := logout \ {name} //Remove the user from the logout set
  end

  event logout
  any
    u //User who will be logged out
  where
    @grd1: u ∈ login //The user must already be an element of the login set
  then
    @act1: logout := logout ∪ {u} //Add the user to the logout set
    @act2: login := login \ {u} //Remove the user from the login set
  end

  event register
```

```
47  any
48   name //Username for user
49   pass //Password for user
50  where
51   @grd1: name ∉ dom(users) //The username must be new and unique, doens't exist already
52   @grd2: pass ∈ password ∨ pass ∉ password //The password can already exist or be new
53  then
54   @act1: users := users ∪ {name ↦ pass} //Add the user to the users set, mapping the username to the
         password
55  end
56
57 end
```

```
1  machine m1
2  refines m0
3  sees UserContext AuctionContext TokenContext UserStatusContext
4  variables
5   username
6   password
7   users
8   login
9   logout
10  auctionID // Set of AuctionIDs
11  auction // Set of auction
12  auctionReservePrice // Set of auction reserve price
13  userAuctions // the set of auctions linked to the user
14 invariants
15  @inv1: auctionID ⊆ AUCTIONID
16  @inv2: auctionReservePrice ⊆ AUCTIONRESERVEPRICE // The set of reserve prices of an auction
17  @inv4: auction ∈ auctionID ⤖ auctionReservePrice // Auction are assigned an auction id and a reserve
         price
18  @inv5: username ⊆ USERNAME // The set of users
19  @inv10: ∀ a1,a2 · a1 ∈ dom(auction) ∧ a2 ∈ dom(auction) ⇒ a1 ≠ a2 //This allows for the auction id to be
         unique for each auction
20  @inv11: userAuctions ∈ username → auctionID // The user will have auction assigned to them
21 events
22  event INITIALISATION extends INITIALISATION
23  then
24  @init−auction: auction := ∅ // There are no active auction
25  @init−auctionID: auctionID := ∅ // The auctionIDs will be of a natural number
26  @init−auctionReservePrice: auctionReservePrice := ∅ // There has been no reserve price set
27  @init−userAuctions: userAuctions := ∅
28  end
29
30  event login extends login
31  end
32
33  event logout extends logout
34  end
35
36  event register extends register
37  end
38
39  event createAuction
```

```
40  any
41  i // i is an auctionID that needs to be set
42  r // r is a reserve price
43  u // u is a username
44  where
45  @grd2: i ∈ auctionID // i is not already a part of the auctionID set so
46  @grd3: r ∈ auctionReservePrice ∨ r ∉ auctionReservePrice // r is either set as the reserve price or not
47  @grd4: u ∈ username // u is a username of the user
48  then
49  @act1: auction := auction ∪ {i ↦ r} // The unique auction id is linked to the reserve price and is added to
        the auction
50  @act2: userAuctions := userAuctions ∪ {u ↦ i} // The username and the auction id are linked and are
        added to the user auction
51  end
52
53 end
```

```
1  machine m2
2  refines m1
3  sees UserContext AuctionContext TokenContext UserStatusContext
4  variables
5   username
6   password
7   users
8   login
9   logout
10  auctionID
11  auction
12  auctionReservePrice
13  userAuctions
14  time // The current time
15  token // The token to make the auction live
16  token_valid_from // The time from when the token is valid from
17  token_valid_until // The time from when the token is valid until
18 invariants
19  @inv12: time ∈ ℕ // The time is part of the natural numbers set
20  @inv13: token ⊆ TOKEN // The tokens are part of the token set
21  @inv14: token_valid_from ∈ token → ℕ // The tokens that are valid from a certain time are part of the
        token set
22  @inv15: token_valid_until ∈ token → ℕ // The tokens that are valid until a certain point are also part of
        the token set
23  @inv16: ∀ t · t ∈ token ⇒ token_valid_from(t) < token_valid_until(t) // The token valid from are always
        less than the token valid until
24 events
25  event INITIALISATION extends INITIALISATION
26  then
27  @init−token: token := ∅
28  @init−token_valid_from: token_valid_from := ∅ // Initially there are no information about tokens
        validity
29  @init−token_valid_until: token_valid_until := ∅ // Initially there are no information about tokens
        validity
30  @init−time: time := 0 // Initially the time is set to 0
31  end
32
```

```
event login extends login
end

event logout extends logout
end

event register extends register
end

event createAuction extends createAuction
end

event clock
then
@act1: time := time + 1 // This allows the time to increase
end

event startAuction
any
a // a is the start time of the auction
b // b is the end time of the auction
t // t is the token given to the user when the auction is live to indicate it is open
where
@grd1: time ≤ a // The time is either the same as the start time or less than it
@grd2: a ≤ b // The end time is after the start time of the auction
@grd3: t ∈ token // The t is part of the token set
then
@act1: token_valid_from(t) := a // Set the valid from time for t
@act2: token_valid_until(t) := b // Set the valid until time for t
end

end
```

```
machine m3
refines m2
sees UserContext AuctionContext TokenContext UserStatusContext
variables
username
password
users
login
logout
auctionID
auction
auctionReservePrice
userAuctions
time
token
token_valid_from
token_valid_until
bids // The bids
invariants
@inv1: bids ∈ users ⇸ ℕ // The bidders have at most 1 bid.
events
```

```
22  event INITIALISATION extends INITIALISATION
23  then
24   @init−bids: bids := ∅ // Initially, there are no bids.
25  end
26
27  event login extends login
28  end
29
30  event logout extends logout
31  end
32
33  event register extends register
34  end
35
36  event createAuction extends createAuction
37  end
38
39  event clock extends clock
40  end
41
42  event startAuction extends startAuction
43  end
44
45  event BIDDING
46  any
47   bidder // The current bidder.
48  where
49   @grd1: bidder ∈ users // Checks whether bidder is in users.
50  then
51   @act1: bids(bidder) := bids(bidder) + 1 // Add 1 toward the bidder that bid.
52  end
53
54 end
```

```
1  machine m4
2  refines m3
3  sees UserContext AuctionContext TokenContext UserStatusContext
4  variables
5   username
6   password
7   users
8   login
9   logout
10  auctionID
11  auction
12  auctionReservePrice
13  userAuctions
14  time
15  token
16  token_valid_from
17  token_valid_until
18  bids
19  activeAuction // Active auction.
20 invariants
```

```
  @inv1: activeAuction ⊆ AUCTIONS // Active auction is in AUCTION.
 events
  event INITIALISATION extends INITIALISATION
  then
   @init−activeAuction: activeAuction := ∅ // Initially, there are no active auction.
  end

  event login extends login
  end

  event logout extends logout
  end

  event register extends register
  end

  event createAuction extends createAuction
  end

  event clock extends clock
  end

  event startAuction extends startAuction
  end

  event BIDDING extends BIDDING
  end

  event Cancel
  any
   a // The auction to be cancelled.
  where
   @grd1: a ∈ activeAuction // Checks whether the auction is an active auction.
  then
   @act1: activeAuction := activeAuction \ {a} // Removes the auction from the set of active auction.
  end

 end
```

```
machine m5
refines m4
sees UserContext AuctionContext TokenContext UserStatusContext viewStatusContext
variables
 username
 password
 users
 login
 logout
 auctionID
 auction
 auctionReservePrice
 userAuctions
 time
 token
```

```
16  token_valid_from
17  token_valid_until
18  bids
19  activeAuction
20  currentBids // Current bid
21  status // Status of auction.
22  viewStatus // View status.
23  viewItem // View auction item.
24  viewBids // View bids.
25  item // Auction item.
26 invariants
27  @inv1: currentBids ⊆ BIDS // Current bids of the auction.
28  @inv2: status ⊆ STATUS // Status of the auction.
29  @inv3: item ⊆ ITEM // Auction Item.
30  @inv4: viewStatus ∈ auctionID → STATUS // View the current bids and the status of the auction.
31  @inv5: viewBids ∈ auctionID → BIDS // View the current bids of the auction.
32  @inv6: viewItem ∈ auctionID → ITEM // View the current item of the auction.
33 events
34  event INITIALISATION extends INITIALISATION
35  then
36   @init−currentBids: currentBids := ∅
37   @init−status: status := ∅
38   @init−viewStatus: viewStatus := ∅
39   @init−viewItem: viewItem := ∅
40   @init−viewBids: viewBids := ∅
41   @init−item: item := ∅
42  end
43
44  event login extends login
45  end
46
47  event logout extends logout
48  end
49
50  event register extends register
51  end
52
53  event createAuction extends createAuction
54  end
55
56  event clock extends clock
57  end
58
59  event startAuction extends startAuction
60  end
61
62  event BIDDING extends BIDDING
63  end
64
65  event Cancel extends Cancel
66  end
67
68  event viewStatus
69  any
70   b // Current bids.
71   a // Current auction ID.
```

```
72    s // Current status.
73    i // Current item.
74   where
75    @grd1: b ∈ currentBids // If the type of b is BIDS.
76    @grd2: a ∈ auctionID // If the type of a is AUCTIONS.
77    @grd3: s ∈ status // If the type of s is STATUS.
78    @grd4: i ∈ item // If the type of i is ITEM.
79   then
80    @act1: viewStatus(a) := s // View the status.
81    @act2: viewBids(a) := b // View the current bids.
82    @act3: viewItem(a) := i // View the item.
83   end
84
85 end
```

```
1  machine m6
2  refines m5
3  sees UserContext AuctionContext TokenContext UserStatusContext viewStatusContext
4  variables
5   username
6   password
7   users
8   login
9   logout
10  auctionID
11  auction
12  auctionReservePrice
13  userAuctions
14  time
15  token
16  token_valid_from
17  token_valid_until
18  bids
19  activeAuction
20  currentBids
21  status
22  viewStatus
23  viewItem
24  viewBids
25  item
26  currentBidsValue // Current Bids Value
27 invariants
28  @inv1: currentBidsValue ⊆ ℕ // Current bids values of the auction.
29 events
30  event INITIALISATION extends INITIALISATION
31  then
32   @init−currentBidsValue: currentBidsValue := ∅ // Initially, there are no current bids value.
33  end
34
35  event login extends login
36  end
37
38  event logout extends logout
39  end
```

```
40
41  event register extends register
42  end
43
44  event createAuction extends createAuction
45  end
46
47  event clock extends clock
48  end
49
50  event startAuction extends startAuction
51  end
52
53  event BIDDING extends BIDDING
54  end
55
56  event Cancel extends Cancel
57  end
58
59  event viewStatus extends viewStatus
60  end
61
62  event closeSuccessfulAuction
63  any
64   b // Current bids.
65   a // Current auction ID.
66  where
67   @grd1: a ∈ auction // If the type of a is AUCTiON.
68   @grd2: b ∈ currentBidsValue // If the type of b is ℕ.
69   @grd3: b ≥ reservePrice // If b is not smaller than the reserve price.
70   @grd4: status = {ongoing} // If the status of auction is ongoing.
71  then
72   @act1: auction := auction \ {a} // Close the current auction.
73   @act2: status := {successful} // Set the status to successful.
74  end
75
76  event closeFailAuction
77  any
78   b // Current bids.
79   a // Current auction ID.
80  where
81   @grd1: b ∈ currentBidsValue // If the type of b is ℕ.
82   @grd2: b < reservePrice // If the current bids is smaller than reserve price.
83   @grd3: a ∈ auction // If the type of a is AUCTION.
84   @grd4: status = {ongoing} // If the status of auction is ongoing.
85  then
86   @act1: auction := auction \ {a} // Close the current auction.
87   @act2: status := {failed} // Set the status to failed.
88  end
89
90 end
```

```
1 machine m7
2 refines m6
```

```
sees UserContext AuctionContext TokenContext UserStatusContext viewStatusContext
variables
  username
  password
  users
  login
  logout
  auctionID
  auction
  auctionReservePrice
  userAuctions
  time
  token
  token_valid_from
  token_valid_until
  bids
  activeAuction
  currentBids
  status
  viewStatus
  viewItem
  viewBids
  item
  currentBidsValue
  feedbacks // Feedbacks
  message // Messages
invariants
  @inv1: feedbacks ⊆ FEEDBACK // Declare the type of feedbacks.
  @inv2: message ∈ auctionID → feedbacks // Using auctionID from auctionIDs to find feedback.
events
  event INITIALISATION extends INITIALISATION
  then
    @init−feedbacks: feedbacks := ∅ // Initially, there are no feedbacks.
    @init−message: message := ∅ // Initially, there are no message.
  end

  event login extends login
  end

  event logout extends logout
  end

  event register extends register
  end

  event createAuction extends createAuction
  end

  event clock extends clock
  end

  event startAuction extends startAuction
  end

  event BIDDING extends BIDDING
  end
```

```
59
60  event Cancel extends Cancel
61  end
62
63  event viewStatus extends viewStatus
64  end
65
66  event closeSuccessfulAuction extends closeSuccessfulAuction
67  end
68
69  event closeFailAuction extends closeFailAuction
70  end
71
72  event sendFailedFeedback
73  any
74   auctionid // Auction.
75   s // Whether the auction ends successfully or not.
76   feedback // Information of the auction.
77  where
78   @grd1: auctionid ∉ auctionID // If auctionID is not in the set of auctionIDs.
79   @grd2: s ∈ BOOL // If type of status is BOOL.
80   @grd3: s = FALSE // If status equal to false.
81   @grd4: feedback ∉ feedbacks // If feedback does not exist.
82  then
83   @act1: auctionID := auctionID ∪ {auctionid} // Add auctionID to auctionIDs.
84   @act2: feedbacks := feedbacks ∪ {feedback} // Add feedback to feedbacks.
85   @act3: message := message ∪ {auctionid ↦ feedback} // Add the order pair to message.
86  end
87
88  event sendSucceedFeedback
89  any
90   auctionid // Auction.
91   s // Whether the auction is a success or failure.
92   feedback // Information of the auction.
93  where
94   @grd1: auctionid ∉ auctionID // If auctionID do not exist.
95   @grd2: s ∈ BOOL // If type of status is BOOL.
96   @grd3: s = TRUE // If status equals to true.
97   @grd4: feedback ∉ feedbacks // If feedback is not in the set of feedbacks.
98  then
99   @act1: auctionID := auctionID ∪ {auctionid} // Add auctionID to auctionIDs.
100  @act2: feedbacks := feedbacks ∪ {feedback} // Add feedback to feedbacks.
101  @act3: message := message ∪ {auctionid ↦ feedback} // Add the order pair to message.
102  end
103
104 end
```

```
1  machine m8
2  refines m7
3  sees UserContext AuctionContext TokenContext UserStatusContext viewStatusContext
4  variables
5   username
6   password
7   users
```

```
  login
  logout
  auctionID
  auction
  auctionReservePrice
  userAuctions
  time
  token
  token_valid_from
  token_valid_until
  bids
  activeAuction
  currentBids
  status
  viewStatus
  viewItem
  viewBids
  item
  currentBidsValue
  feedbacks
  message
  bidHistory // History of bids.
  viewBid // Bids to be shown.
invariants
  @inv1: bidHistory ∈ auctionID ⇸ ℙ(BIDS) // Using auctionID to get set of bids.
  @inv2: viewBid ⊆ BIDS // Declare type of viewBid.
events
  event INITIALISATION extends INITIALISATION
  then
    @act1: bidHistory := ∅ // Initially, there are no bidHistory.
    @act2: viewBid := ∅ // Initially, there are no viewBid.
  end

  event login extends login
  end

  event logout extends logout
  end

  event register extends register
  end

  event createAuction extends createAuction
  end

  event clock extends clock
  end

  event startAuction extends startAuction
  end

  event BIDDING extends BIDDING
  end

  event Cancel extends Cancel
  end
```

```
64
65  event viewStatus extends viewStatus
66  end
67
68  event closeSuccessfulAuction extends closeSuccessfulAuction
69  end
70
71  event closeFailAuction extends closeFailAuction
72  end
73
74  event sendFailedFeedback extends sendFailedFeedback
75  end
76
77  event sendSucceedFeedback extends sendSucceedFeedback
78  end
79
80  event addBidHistorySet
81  any
82   bid // Set of bids.
83   auctionid // Auction.
84  where
85   @grd1: bid ⊆ BIDS // If type of set is BID.
86   @grd2: auctionid ∉ auctionID // If auctionID is not in set auctinIDs.
87  then
88   @act1: auctionID := auctionID ∪ {auctionid} // Add auctionID to auctionIDs.
89   @act2: bidHistory := bidHistory ∪ {auctionid ↦ bid} // Add set of bids to bidHistory.
90  end
91
92  event addSingleBidHistory
93  any
94   bid // Single bid.
95   auctionid // Auction.
96  where
97   @grd1: bid ∈ BIDS // If type of bid is BID.
98   @grd2: auctionid ∈ auctionID // If auctionID is already in set auctionIDs.
99  then
100  @act1: bidHistory(auctionid) := bidHistory(auctionid) ∪ {bid} // Add bid in set of bids with given
       auctionID.
101 end
102
103 event wiewBidHistory
104 any
105  auctionid // Auction.
106 where
107  @grd1: auctionid ∈ auctionID // If auctionID is in the set of auctionIDs.
108 then
109  @act1: viewBid := bidHistory(auctionid) // Give set of bids with given auctionID to the set of bids that
       will be shown to user.
110 end
111
112 end
```