CONCORDIA UNIVERSITY

Deliverable #4 (State Machines)

Project Topic: Online Flight Reservation System

Team: 6 Ideas

Team Members:

Family Name	First Name	Student ID
Afrasiabi	Seyedeh Nazanin	40059181
Afrasiabi	Seyedeh Negar	40057810
Ahmed	Owais	40040018
Promwongsa	Nattakorn	40051020
Rasouli	Farinaz	40047293
Azadiabad	Siamak	26592856

Project GitHub Link: https://github.com/Sazadiabad/OFRS

March 18, 2018

Model Driven Software Engineering Course

Table of Contents

Table of Figures

1. “ScheduledFlight” Class

1.1 State Diagram for “ScheduledFlight” Class

Following diagram depicts the state diagram for “ScheduledFlight” class. After a flight is scheduled, it is available to book until there is no more seat or booking time is finished. Moreover, it is possible to cancel a flight (for more detailed description, see section 1.2).

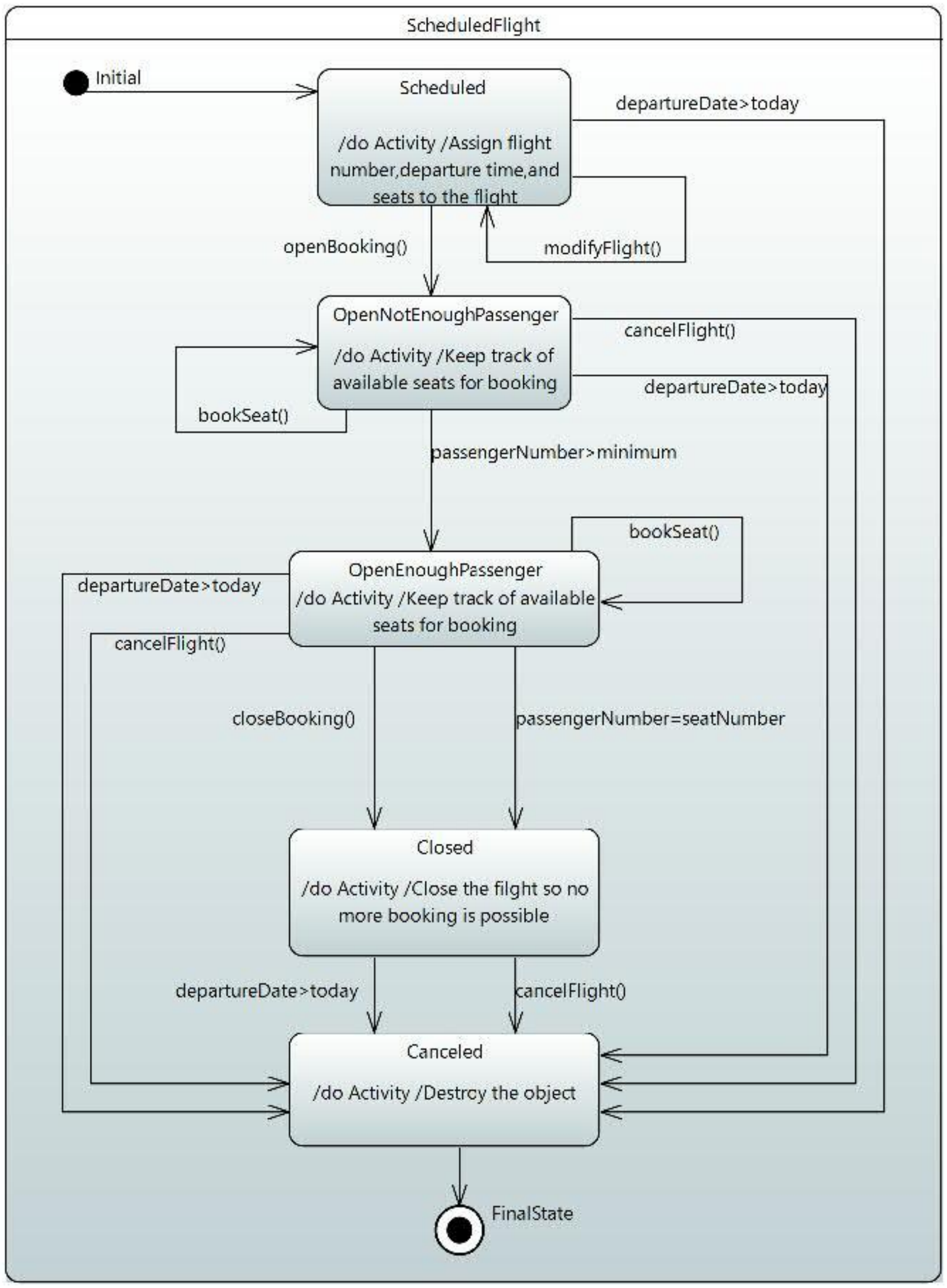
ScheduledFlight
Initial
Scheduled
/do Activity /Assign flight number,departure time,and seats to the flight
departureDate>today
openBooking()
modifyFlight()
OpenNotEnoughPassenger
/do Activity /Keep track of available seats for booking
cancelFlight()
departureDate>today
bookSeat()
passengerNumber>minimum
bookSeat()
OpenEnoughPassenger
/do Activity /Keep track of available seats for booking
departureDate>today
cancelFlight()
closeBooking()
passengerNumber=seatNumber
Closed
/do Activity /Close the filght so no more booking is possible
departureDate>today
cancelFlight()
Canceled
/do Activity /Destroy the object
FinalState

Figure 1: State Diagram for “Scheduled Flight” Class

1.2 State Description

States for this class are:

1. **Scheduled**: The flight has assigned time and flight number for a one flight template. After initiation, each object is in this state. It is possible to modify the flight in this state, since it is not open for booking yet and there is no assigned customer to it.

2. **OpenNotEnoughPassenger**: The flight is open to book, however there aren't enough passengers for the flight yet. So, the flight cannot be closed until the flight reaches the minimum number of passengers. If so, then the state will change to the OpenEnoughPassenger state.

3. **OpenEnoughPassenger**: There are enough passengers in this state, however there are free seats to book yet. Thus, if the booking deadline gets expired, the flight can be closed (instead of canceling the flight). The airline employee can also cancel the flight even it is in this state.

4. **Closed**: In this state the flight is closed and getting ready to fly. Therefore, no more booking is possible. The flight could be cancelled in this state before departure, based on airline decision. Moreover, when the flight is expired after flying, it needs to be deleted.

5. **Canceled**: Flight can be cancelled before departure, whenever airline wants so. Furthermore, when the flight is expired after flying, its state will change to this state in order to be deleted.

2. “Booking” Class

2.1 State Diagram for “Booking” Class

Following diagram depicts the state diagram for “Booking” class (for description, see section 2.2).

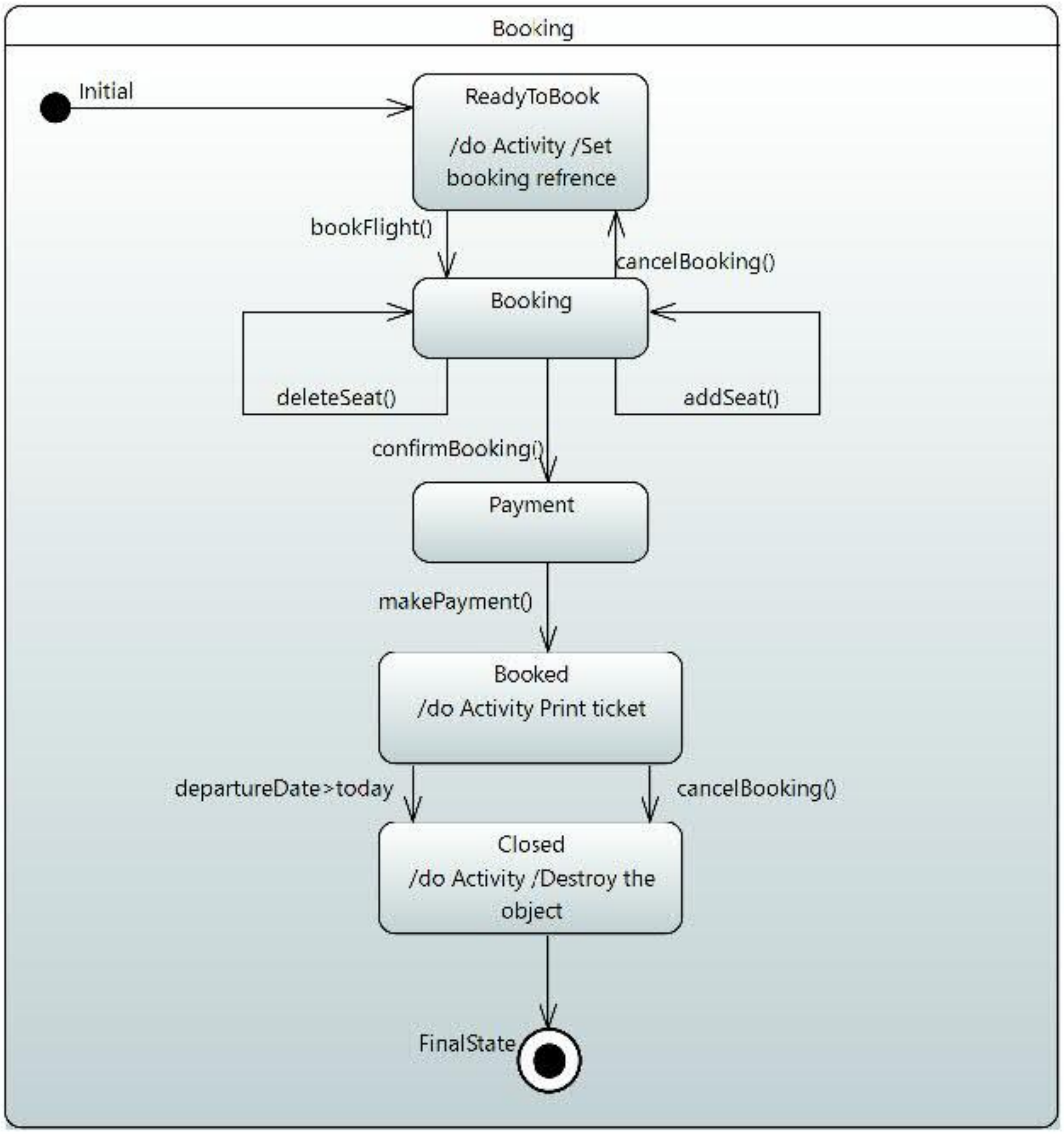

Figure 2: State Diagram for “Booking” Class

2.2 State Description

1. **ReadytoBook**: In this state the object of booking class is initiated.

2. **Booking**: In this state a customer is booking a flight which means that he/she is adding and deleting seats. Then he/she can confirm the booking.

3. **Payment**: After confirming the booking, the customer must pay for the booked flight in this state.

4. **Booked**: By making a payment for the desired booking, in this state the booking is done and ticket is issued. This booking then can be cancelled later, or after the expiration its state will be changed to the “Closed” state.

5. **Closed**: The booking is cancelled or expired and it needs to be destructed.

3. “Customer” Class

3.1 State Diagram for “Customer” Class

Following diagram depicts the state diagram for “Customer” class (for description, see section 3.2).

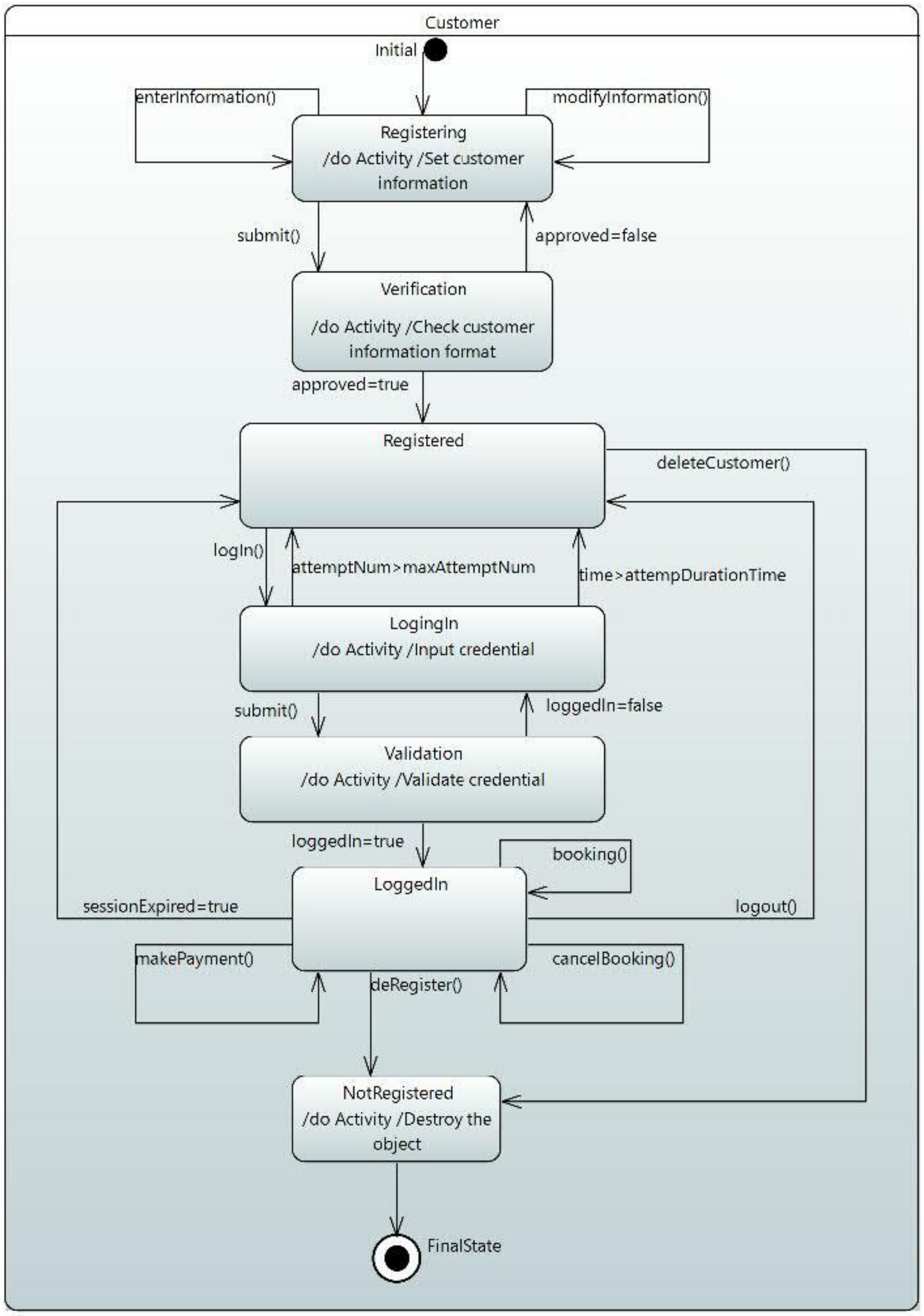

Figure 3: State Diagram for “Customer” Class

3.2 State Description

1. **Registering**: In this state the customers can enter and modify their information for registering.

2. **Verification**: After submitting the registering information, entered data must be verified. During this process, the object is in “Verification” state. If some data is not approved, it’s possible to go to the previous state to modify them to be corrected.

3. **Registered**: If the information of the customer is valid, he/she will reach to this state, and it means that the person is now a registered customer in the system. From now on, customer can log into the system and book or cancel flights. Moreover, the airline employee can delete customers and change its state to “NotRegistered”.

4. **LoggingIn**: In this state the registered customer is performing the login to the system, which means he/she is inserting login credentials to the system. If the number of login attempts and elapsed time for inserting data are less than their thresholds, by submitting the data, the state will be changed to “Validation” state.

5. **Validation**: The customers enter the username and password and if they are correct, customer will be logged in. Unless, he/she will be taken back to the previous state.

6. **LoggedIn**: In this state the customers would be able to book flights, cancel flights and make payments. If the session is expired or the customers want to log out, he/she will go to the “Registered” state, to be ready to log in if needed.

7. **NotRegistered**: The airline might want to delete a customer account or the logged in customer may want to deregister himself/herself. Then the state would be “NotRegistered” to destroy the object.

4. “Seat” Class

4.1 State Diagram for “Seat” Class

Following diagram depicts the state diagram for “Seat” class (for description, see section 4.2).

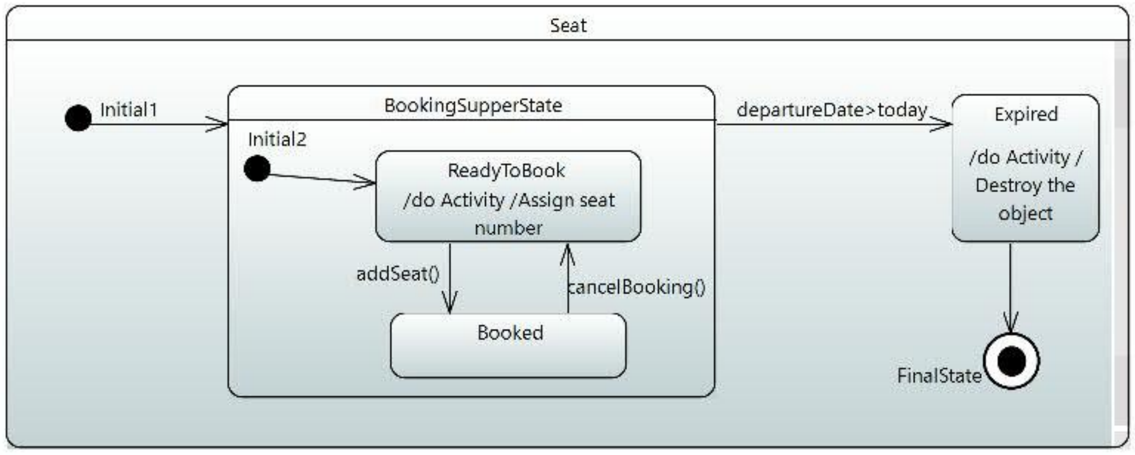

Figure 4: State Diagram for “Seat” Class

4.2 State Description

1. **ReadytoBook**: In this state the object of class seat is instantiated and it is available to be booked.

2. **Booked**: In this state the seat is booked which means is associated to a booking. It will become available again if the booking get cancelled.

3. **Expired**: After the departure the flight is expired and by consequence, the related seats are expired and no more valid to be booked or cancelled. So, in this state, the seat object will be destroyed.

5. Action Specifications

5.1 Action for “bookSeat” Operation

```
//Action specification for “bookSeat” operation in “ScheduledFlight” class
ScheduledFlight::bookSeat(seat:Seat):void
{
   if (state == ‘OpenNotEnoughPassenger’)
   {
            seat.isAvailable = false;
            passengerNumber ++;
   }

   if (passengerNumber > minimum)
            state = “OpenEnoughPassenger”; //change the state to “OpenEnoughPassenger”
}
```

5.2 Action for “bookFlight” Operation

```
//Action specification for “bookFlight” operation in “Booking” class
Booking::bookFlight(flight:ScheduledFlight):void
{
   if (state == "ReadyToBook")
   {
            flight.booking.add(this); //add this booking to the associated flight
            this.flight = flight; //set the flight attribute to the associated flight

            state = "Booking"; //change the state to “Booking”
   }
}
```

5.3 Action for “addSeat” Operation

```
//Action specification for “addSeat” operation in “ScheduledFlight” class
Booking::addSeat(seat:Seat):boolean
{
   if (state == "Booking")
            if (seat.isAvailable)
            {
                     this.seats.add(seat); //add the booking seat to the list of passenger seats
                     flight.bookSeat(seat); //make the seat of the associated flight unavailable

                     return true;
```

```
                }
                else
                        return false;
        }
```

5.4 Action for “enterInformation” Operation

```
        //Action specification for “enterInformation” operation in “Customer” class
        Customer::enterInformation(name:String, address:String, email:String, gender:String,
birthDate: String, paymentInfo: String):void
        {
           if (state == "Registering")
           {
                this.name = name; //Set the name as inputted name
                this.address = address; //Set the address as inputted address
                this.email = email; //Set the email as inputted email
                this.gender = gender; //Set the gender as inputted gender
                this.birthDate = birthDate; //Set the birthDate as inputted birthDate
                this.paymentInfo = paymentInfo; //Set the paymentInfo as inputted paymentInfo
           }
        }
```

5.5 Action for “approve” Operation

```
        //Action specification for “approve” operation in “Customer” class
        Customer::approve():boolean
        {
           boolean approved = true;
           if (state == "Verification")
           {
                //check the inputed values length
                if (name.lenght > 30)
                        approved = false;
                if (address.lenght > 200)
                        approved = false;
                if (email.lenght > 50)
                        approved = false;
                if (birthDate.lenght > 20)
                        approved = false;
                if (paymentInfo.lenght > 100)
                        approved = false;
```

```
            //check the format
            if (name.exist(numbers) || name.exist(special_characters))
                    approved = false;
            if (!email.format("*@*.*"))
                    approved = false;
            if (!birthDate.format("dd/mm/yyyy"))
                    approved = false;
            if (!paymentInfo.exist(numbers))
                    approved = false;

            //set the state based on the approvement
            if (approved)
            {
                    state = "Registered";
                    return true;
            }
            else
            {
                    state = "Registering";
                    return false;
            }
    }
}
```