**CSE 522: Project**

**Tariq Siddiqui: 50208476**

**Junaid Shaikh: 50208476**

We have Implemented Bus Transport System Something Similar to UB Transport System.

As Any Bus System, our Bus System Consists of

1.Buses

2.Vans

3.Drivers

4.Maintainenace Staff

5. Schedule Manager and

6.Commuters

Our implementation Implements all above as Separate Classes with each interacting with others as part of various use cases implementation.

Also, since entities like Commuters, Drivers, Maintenance Staff **Are Independent**, our program **extensively uses Multi-threading**. Hence, each entity above operates automatically and triggers Event Correspondingly when corresponding events are trigger.

**Final Remarks:**

We have tried to design and build the project using the Use-Case driven methodology. We have come to see the obvious benefits of following this methodology in a way that it warns you about the upcoming challenges in earlier stages and you can prepare better for the future phases and adjust the requirements analysis appropriately.

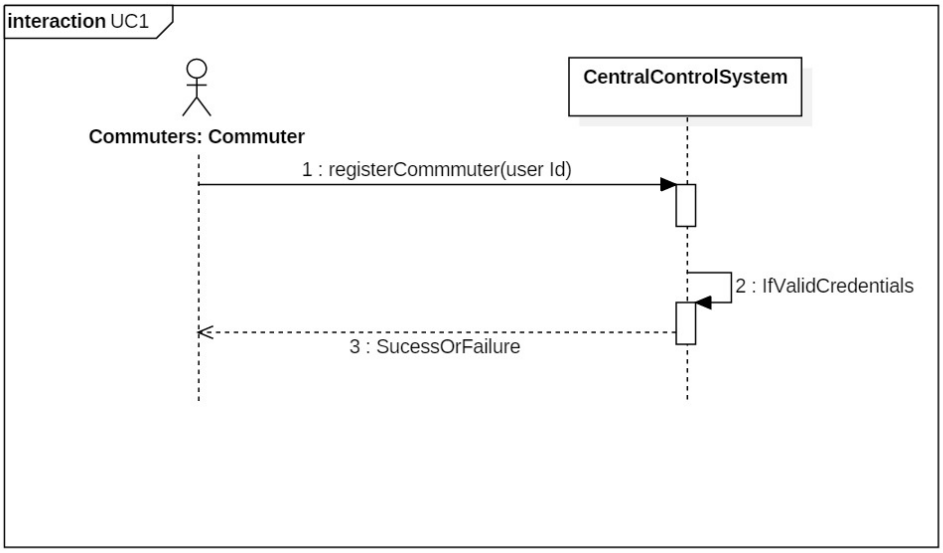
**Use Case 1(UC1):**

**Summary:**

i. Use Case 1 was about registering a new commuter to the database of commuters in the Central Control System. This was required since the next use case we want to implement, UC2, requires a commuter to retrieve the best route, fare, schedule and location updates for their intended trip.

ii. As explained in the Use Case diagram, the commuter object calls the Central Control System to request to register himself to the database. The commuter is registered to the database by the CCS and control is passed back.

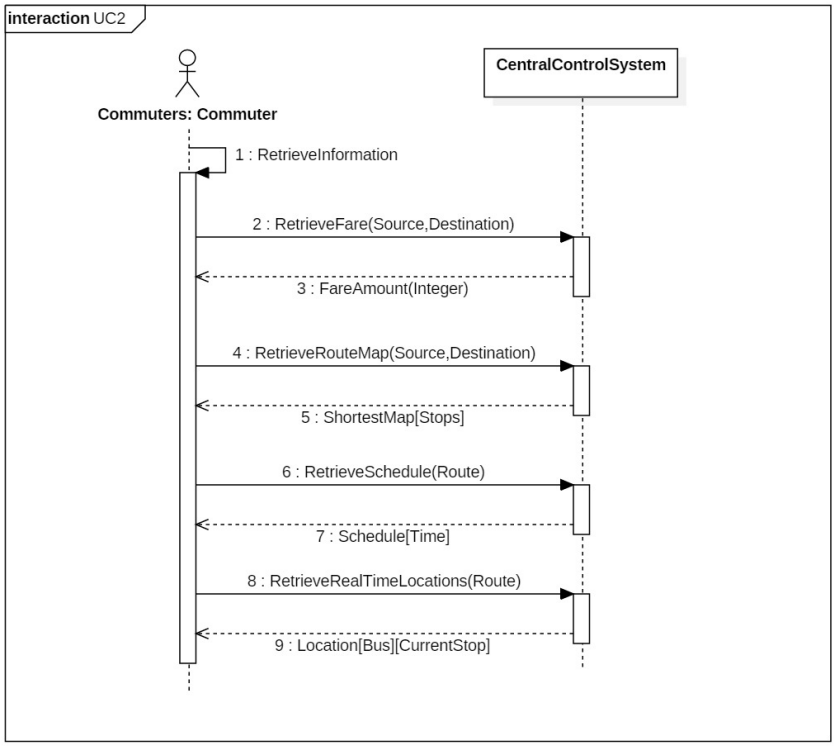
**Interaction Diagrams**



**Use Case 2(UC2):**

**Summary:**

The registered commuter wants to enquire about the best route from his source to intended destination. The object commuter sends a retrieveRouteMap, retrieveFare, retrieveSchedule and retrieveRealTimeLocation request to the CCS along with the source and destinations as shown below:

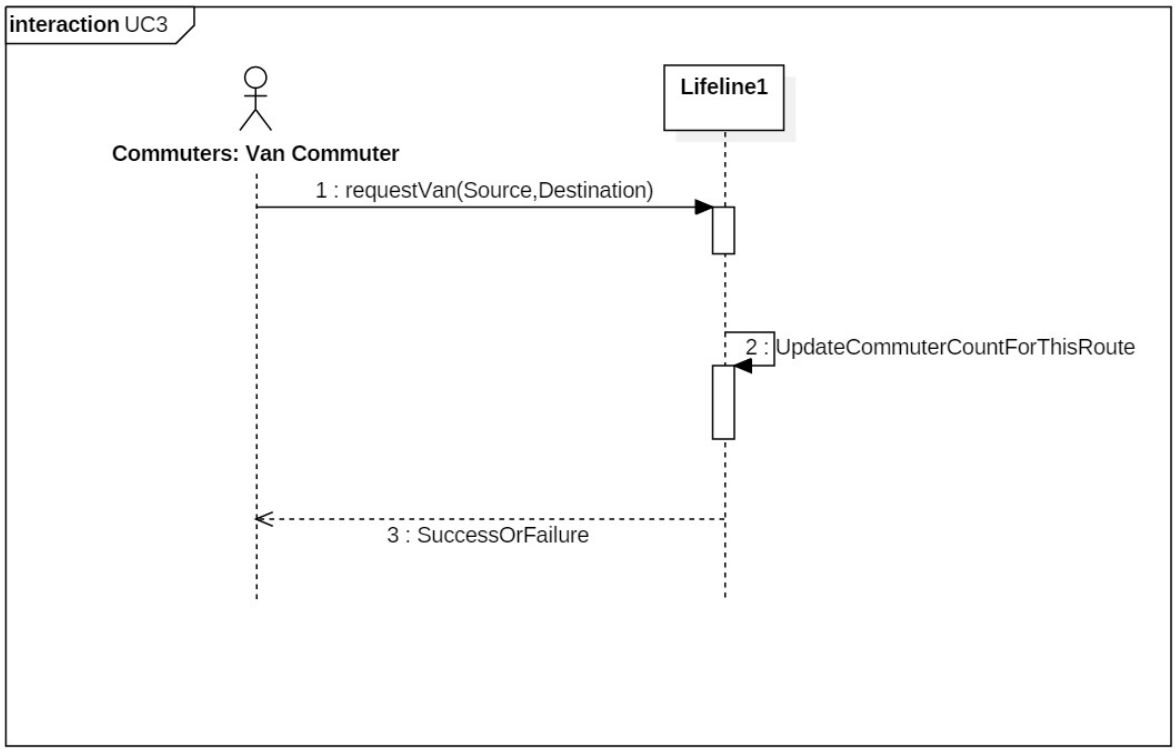


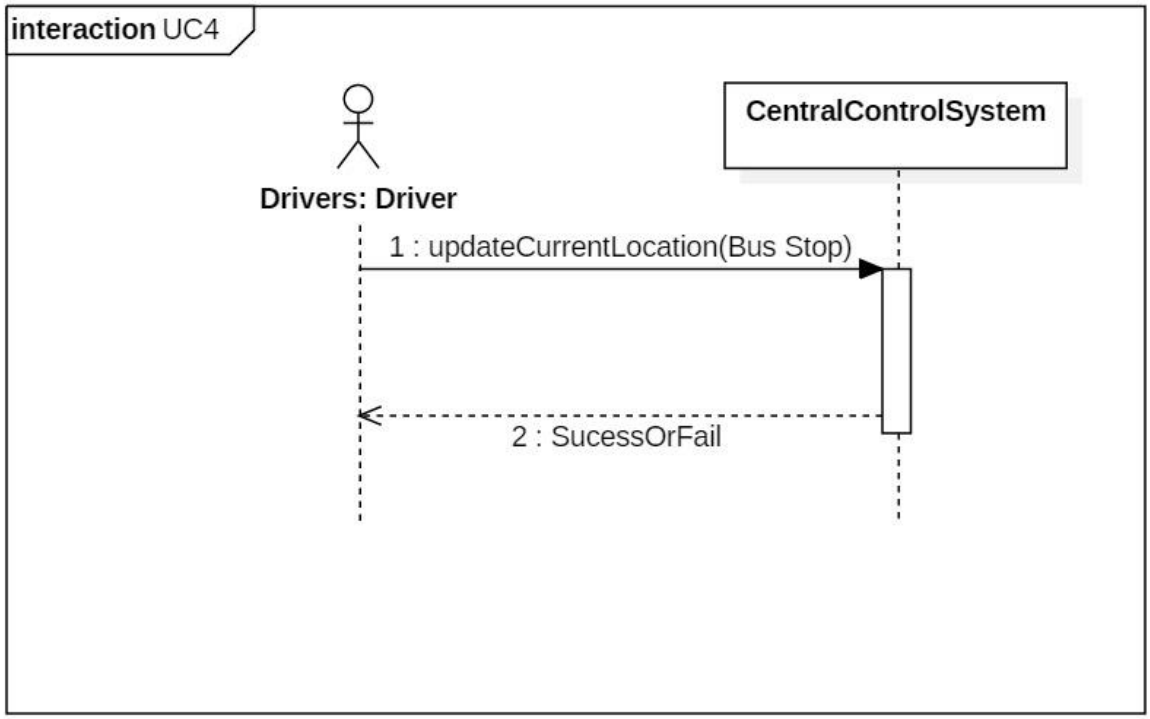

We have implemented the code best route retrieval and the fare by defining the routes as a collection of stops and searching for the source and destination stops among the routes and returned the best route available (i.e. least no of stops). The fare is calculated as $2 per stop. Schedule and realTimeLocation updates will be implemented in Phase4.

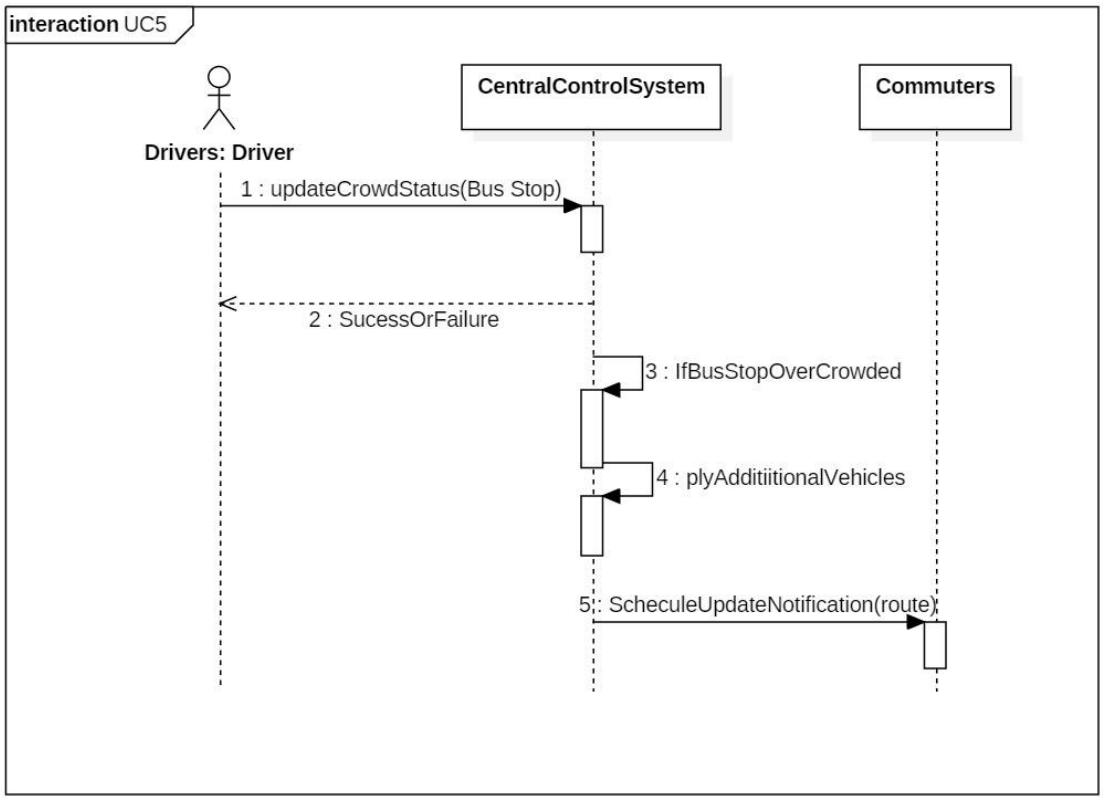
**Use Case3(UC3):**

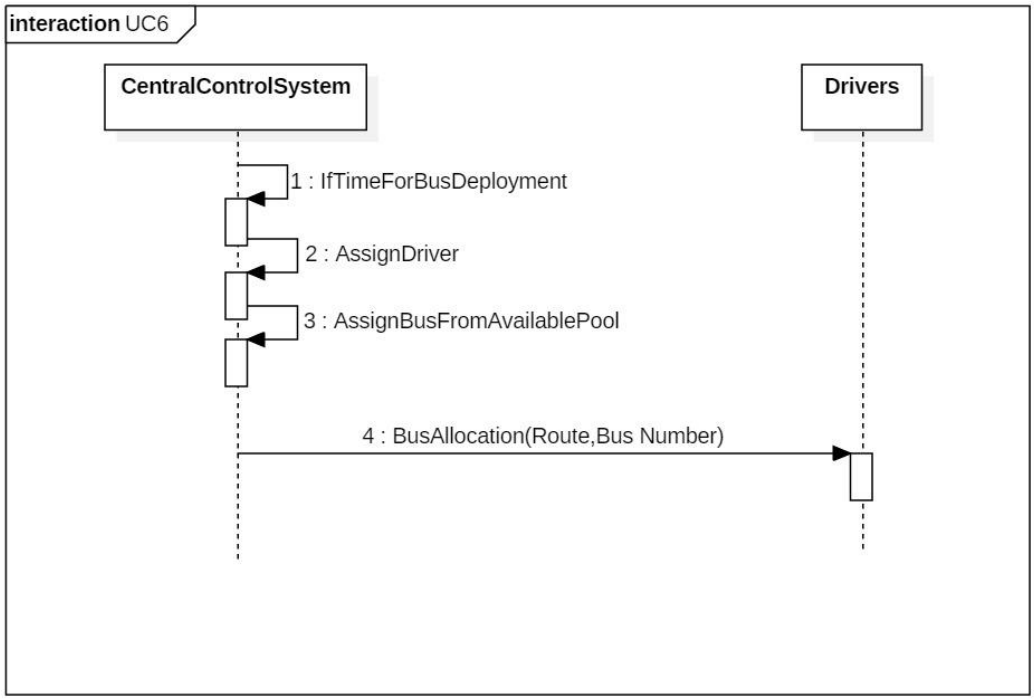
**Summary:**

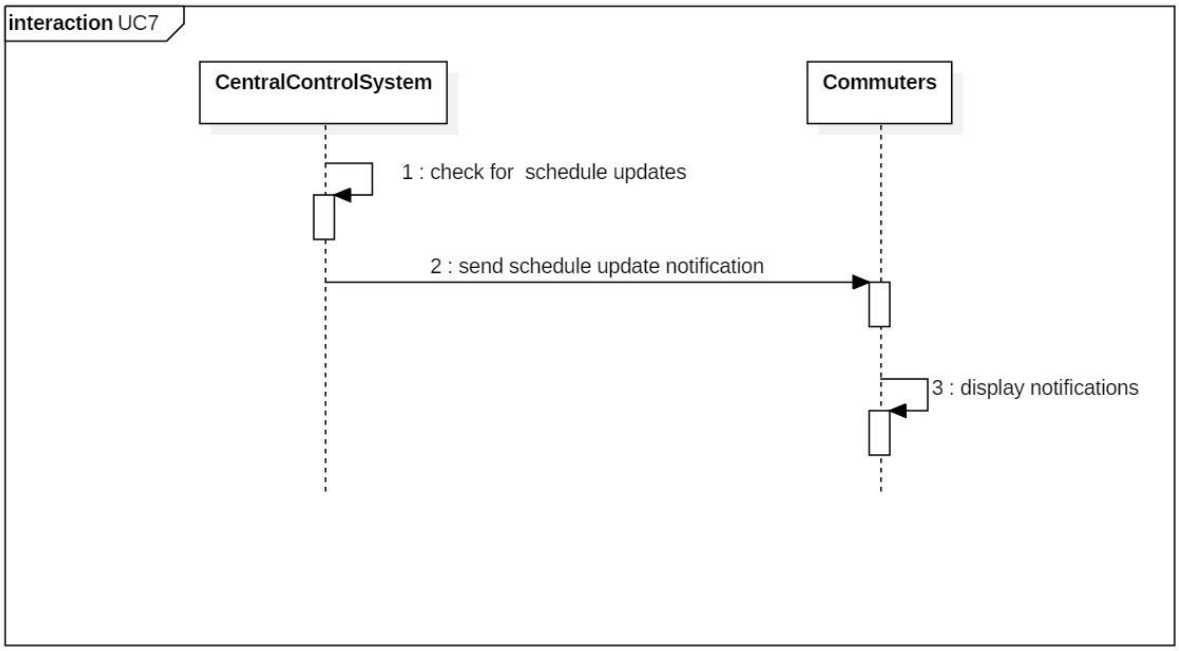
This Case deals with User Requesting Van on demand. When requested, System checks for available Van and assigns for a route. After trip over, Van is removed and added back to free List.

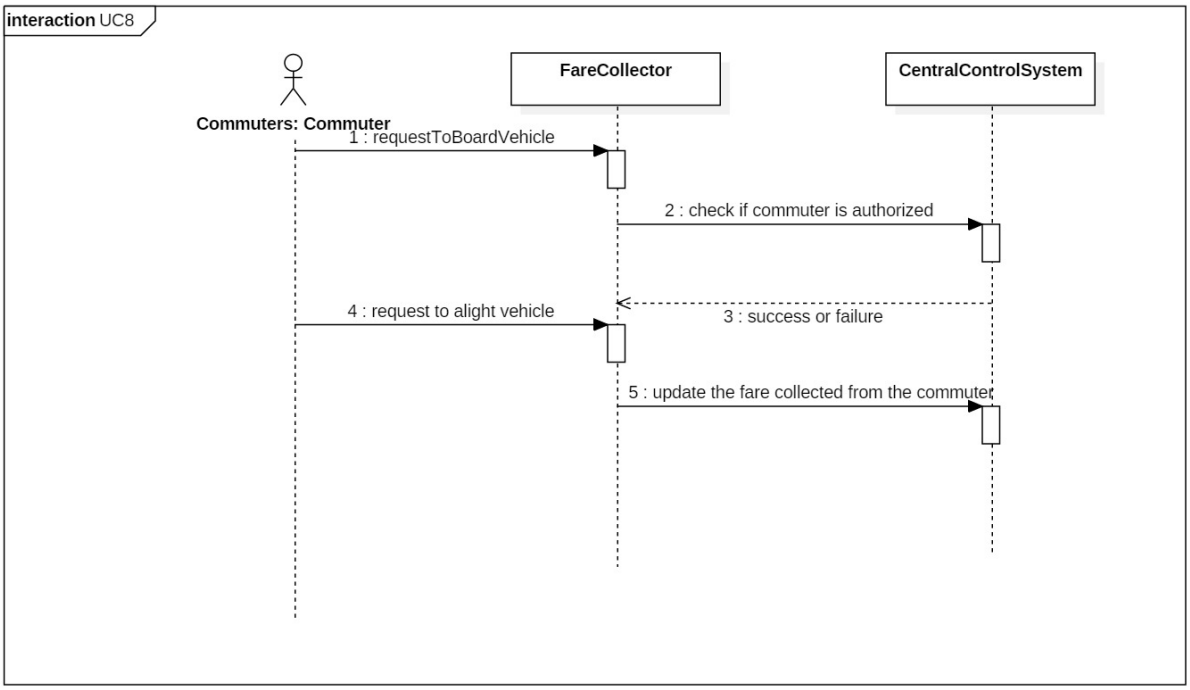


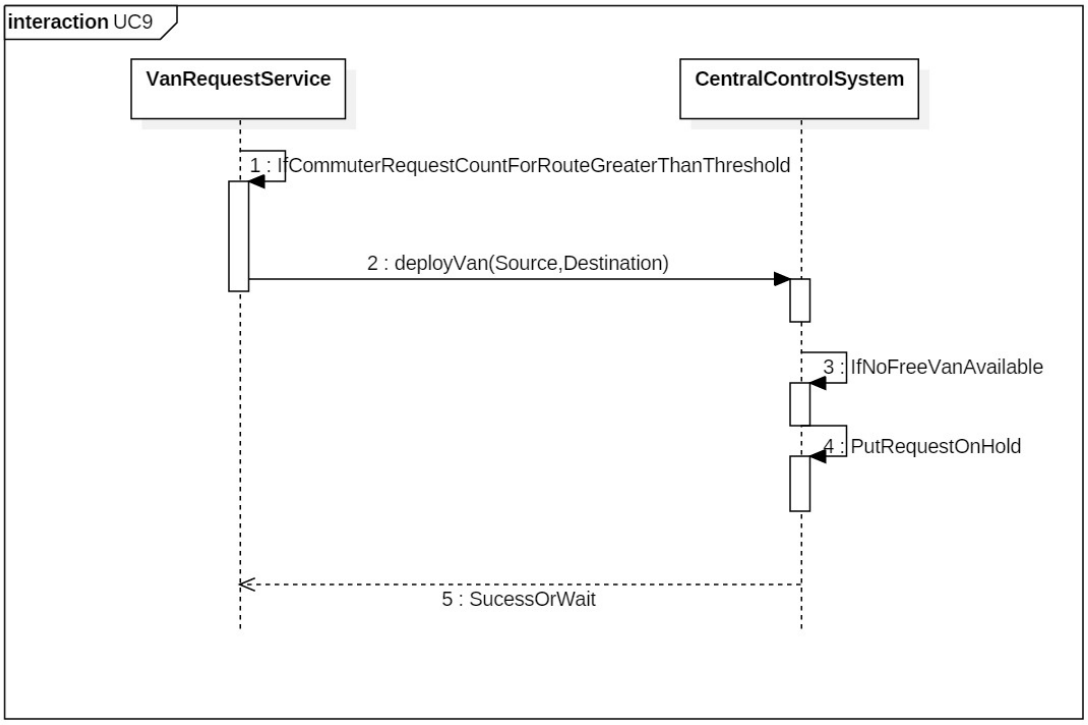

interaction UC4
CentralControlSystem
Drivers: Driver
1 : updateCurrentLocation(Bus Stop)
2 : SucessOrFail


interaction UC5
CentralControlSystem
Commuters
Drivers: Driver
1 : updateCrowdStatus(Bus Stop)
2 : SucessOrFailure
3 : IfBusStopOverCrowded
4 : plyAdditiitionalVehicles
5 : ScheculeUpdateNotification(route)


interaction UC6
CentralControlSystem
Drivers
1 : IfTimeForBusDeployment
2 : AssignDriver
3 : AssignBusFromAvailablePool
4 : BusAllocation(Route,Bus Number)


interaction UC7
CentralControlSystem
Commuters
1 : check for schedule updates
2 : send schedule update notification
3 : display notifications

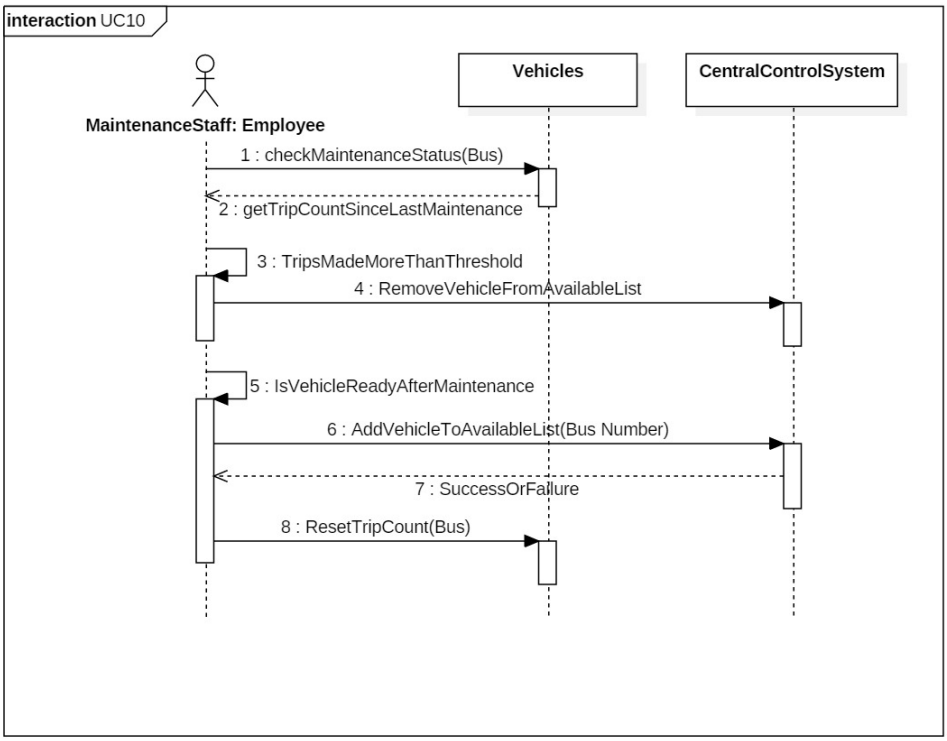

interaction UC8
Commuters: Commuter
FareCollector
CentralControlSystem
1 : requestToBoardVehicle
2 : check if commuter is authorized
3 : success or failure
4 : request to alight vehicle
5 : update the fare collected from the commuter


interaction UC9
VanRequestService
CentralControlSystem
1 : IfCommuterRequestCountForRouteGreaterThanThreshold
2 : deployVan(Source,Destination)
3 : IfNoFreeVanAvailable
4 : PutRequestOnHold
5 : SucessOrWait

**Use Case10(UC 10):**

**Summary:**

First Use Case we implemented(UC10) is related to periodic checking of TripCount of each vehicle since last it was Serviced in Garage. As show below, maintenance Staff (who implements Employee interface), periodically inquires TripCount for each Bus. Whenever, this TripCount increases beyond some threshold, Vehicle is called to Dockyard/Garage and is removed from Available Service List.
Once Servicing for Vehicle is Complete, it is sent to Field for service again. Since bus is serviced again, TripCount for this bus is reset. Also, Bus is added to available Service list.




**Details:**

1. **checkMaintenanceStatus :** periodically check for TripCount of buses since last service periodically(Say every night when all buses are in DockYard)
2. **getTripCount :** Retrieves TripCount since last maintenance for all Vehicles.
3. **RemoveVehicleFromavailableList:** Removes vehicle from available vehicle list when Bus/Van is called to Garage for Servicing.
4. **SendVehicleToGarage:** Vehicle is kept in Garage for duration till it is ready for Service again. NOTE: We have used timer value of 3 seconds (Thread.sleep(3000)) to simulate the duration which is needed to Service the Bus.
5. **AddVehicletoavailablelist:** Once Vehicle is ready for Service after servicing, add this vehicle to available service list.
6. **resetTripCount:** In addition to point 5 above, also reset “trip count since last maintenance” for that vehicle

**iii. Modifications to Class Diagram:**

The following two new classes have been added to the Class Diagram as per suggestion to increase the classes in the project.

a. VanRequestService
b. MaintenanceStaff as an implementation of Employees interface

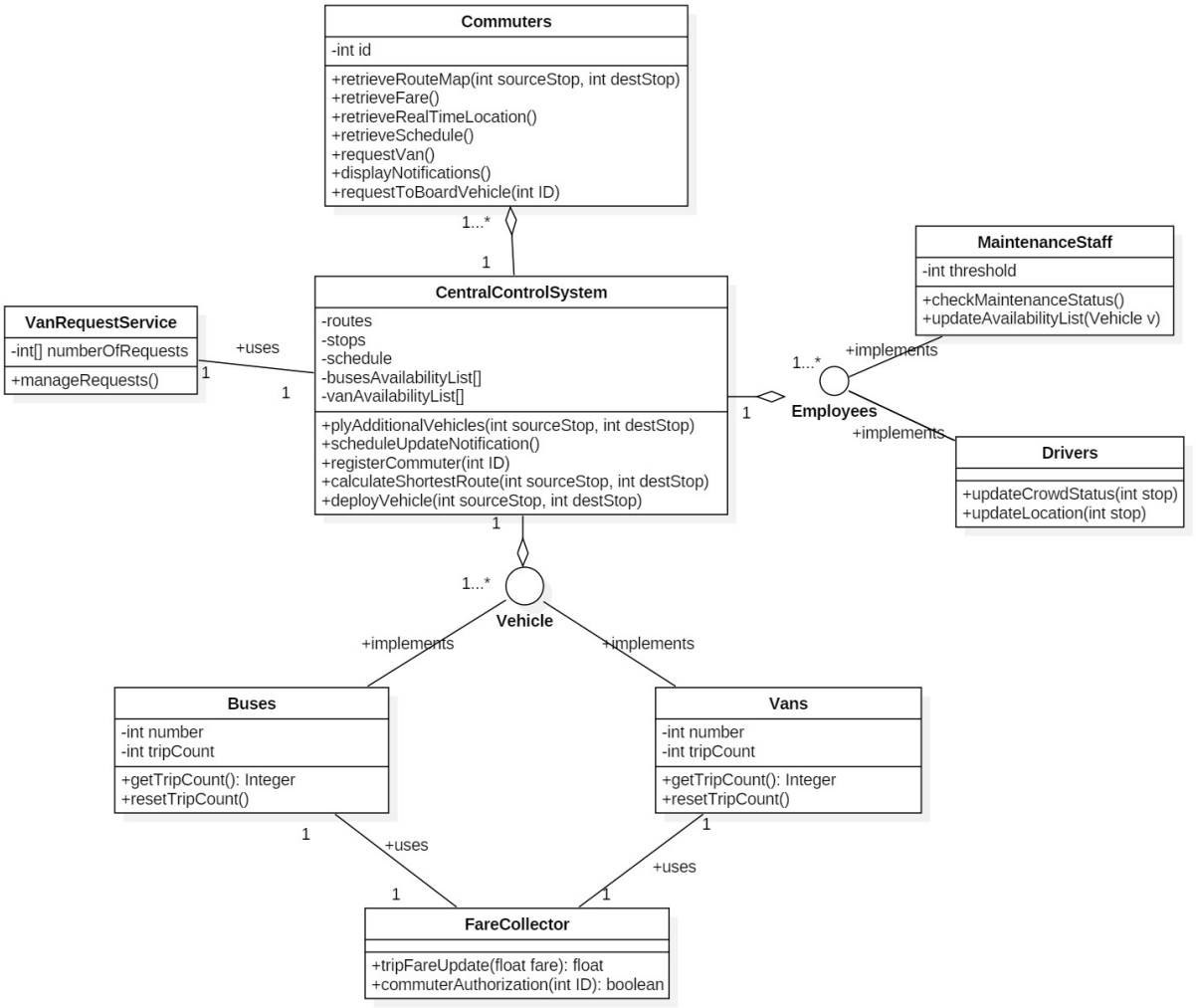
**VanRequestService** will handle requests from Commuters for vans during odd hours when the bus frequency is low or closed. It will wait for an appropriate time or number of requests before deciding to deploy a van to the requested stop. It will interact with the CentralControlSystem and Commuter classes.

**MaintenanceStaff** will decide when a vehicle is due for servicing and will request the CentralControlSystem to deliver the particular vehicle. After the servicing is done it will allocate the vehicle back to the system to use for deployment.

Accordingly, the use case diagram has been updated as follows, some use cases have been merged:

| ID | Use Case | Actor |
|---|---|---|
| UC1 | Register commuter to the transport system | Commuter |
| UC2 | Retrieve route map, fare, schedule and real time location | Commuter |
| UC3 | Request van | Commuter |
| UC4 | Post real time location updates | Driver |
| UC5 | Ply additional vehicles depending on crowd status | Driver, CentralControlSystem |
| UC6 | Deploy vehicles as per schedule, assign a driver to the vehicle | CentralControlSystem |
| UC7 | Send notifications for schedule updates | CentralControlSystem |
| UC8 | Authorize commuters to ride the bus and update fare collected | Fare Collector |
| UC9 | Manage requests for van | VanRequestService |
| UC10 | Manage vehicle servicing system | MaintenanceStaff |

All other notations like multiplicity annotations, attributes and relationships have been added to the Class Diagram.




## Class Diagram

## iv. Table for use-cases and classes

| | Buses | CentralControlSystem | Commuters | Drivers | FareCollector | MaintenanceStaff | VanRequestService | Vans |
|---|---|---|---|---|---|---|---|---|
| UC1 | - | registerCommuter() | - | - | - | - | - | - |
| UC2 | - | calculateShortestRoute() | retrieveRouteMap(), retrieveFare(), retrieveRealTimeLocation(), retrieveSchedule() | - | - | - | - | - |
| UC3 | - | - | requestVan() | - | - | - | manageRequests() | - |
| UC4 | - | - | - | updateLocation() | - | - | - | - |
| UC5 | - | plyAdditionalVehicles(), scheduleUpdateNotification() | - | updateCrowdStatus() | - | - | - | - |
| UC6 | - | deployVehicle() | - | assignDriver(int busNumber, Route) | - | - | - | - |
| UC7 | - | scheduleUpdateNotification() | displayNotifications() | - | - | - | - | - |
| UC8 | - | - | requestToBoardVehicle() | - | commuterAuthorization(), tripFareUpdate() | - | - | - |
| UC9 | - | plyAdditionalVehicles() | - | - | - | - | manageRequests() | - |
| UC10 | getTripCount(), resetTripCount() | - | - | - | - | checkMaintenanceStatus(), updateAvailibilityList() | - | getTripCount(), resetTripCount() |