CSE 522: Project

Tariq Siddiqui: 50208470

Junaid Shaikh: 50208476

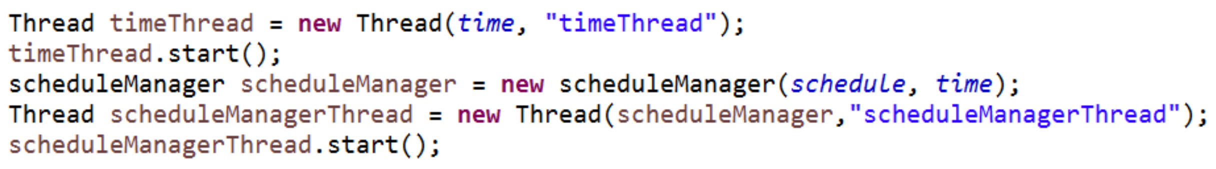
We have Implemented Bus Transport System in a similar fashion to UB Transport System.

<u>As Any Bus System, our Bus System Consists of below nouns:</u>

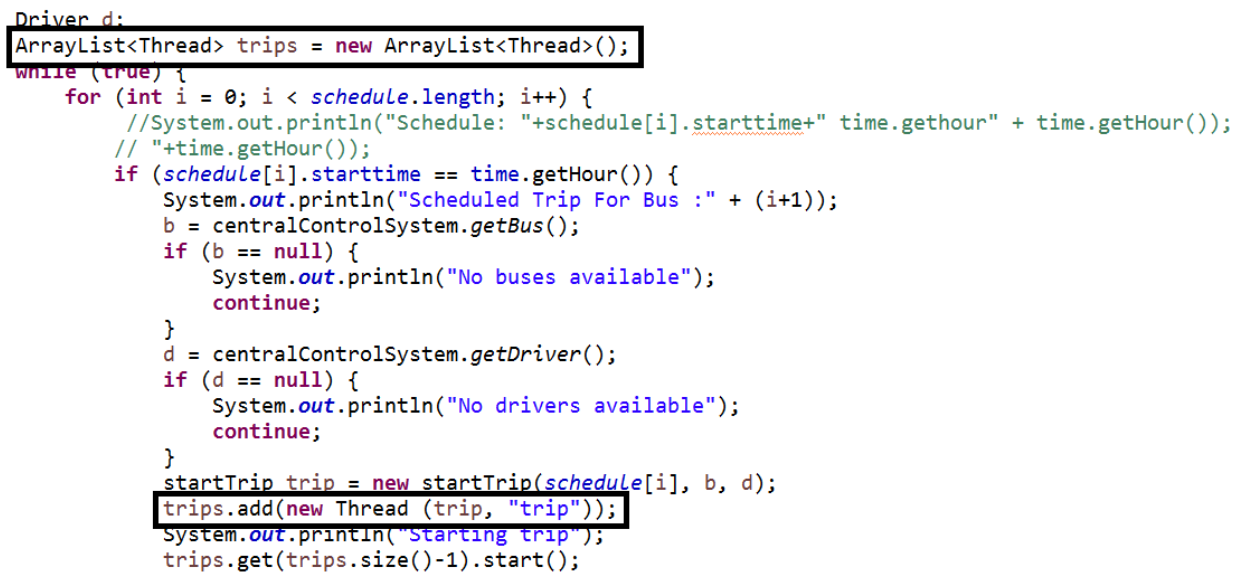
1. Buses/Vans
2. Drivers
3. Maintenance Staff
4. Schedule Manager
5. Commuters
6. Schedule
7. Stop
8. Route
9. Central Control System

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

Since depicting all the bus trip according to schedules times is a set of events that **Are Independent**, our program **extensively uses Multi-threading**. Hence, all bus trips start automatically according to schedule and trigger events correspondingly.



```
Thread timeThread = new Thread(time, "timeThread");
timeThread.start();
scheduleManager scheduleManager = new scheduleManager(schedule, time);
Thread scheduleManagerThread = new Thread(scheduleManager,"scheduleManagerThread");
scheduleManagerThread.start();
```



```
Driver d;
ArrayList<Thread> trips = new ArrayList<Thread>();
while (true) {
    for (int i = 0; i < schedule.length; i++) {
         //System.out.println("Schedule: "+schedule[i].starttime+" time.gethour" + time.getHour());
        // "+time.getHour());
        if (schedule[i].starttime == time.getHour()) {
            System.out.println("Scheduled Trip For Bus :" + (i+1));
            b = centralControlSystem.getBus();
            if (b == null) {
                System.out.println("No buses available");
                continue;
            }
            d = centralControlSystem.getDriver();
            if (d == null) {
                System.out.println("No drivers available");
                continue;
            }
            startTrip trip = new startTrip(schedule[i], b, d);
            trips.add(new Thread (trip, "trip"));
            System.out.println("Starting trip");
            trips.get(trips.size()-1).start();
```

<u>Above Nouns makeup a dynamic system through Implementation of below uses Cases:</u>

1. Commuter Registration
2. Add Bus/Van to Current Schedule on Trip Start.
3. Real Location Bus/Van Update
4. User Triggered Shortest_RouteMap retrieval
5. User Triggered RealTimeLocation retrieval
6. User triggered Fare Retrieval
7. User triggered Schedule retrieval
8. Fare Collection System depending on Commuter's Source and Destination
9. On Schedule Arrival, pick available driver and available Driver to start Trip.
10. Request for Additional Buses/Vans deployment when Requests for particular Stops More than Threshold
11. System Schedule Update on above requests for additional Buses/Vans
12. Notification update to all commuters on Any Schedule Update
13. Recall to garage for maintenance when Bus Reaches Some Trip Threshold.
14. Deployment back in field after Maintenance Complete
15. Maintenance of Bus and Van Availability List
16. Maintenance of Drivers Availability List
17. Add Vehicle to availability List on Trip Completion
18. Remove Vehicle from availability list on Trip Start

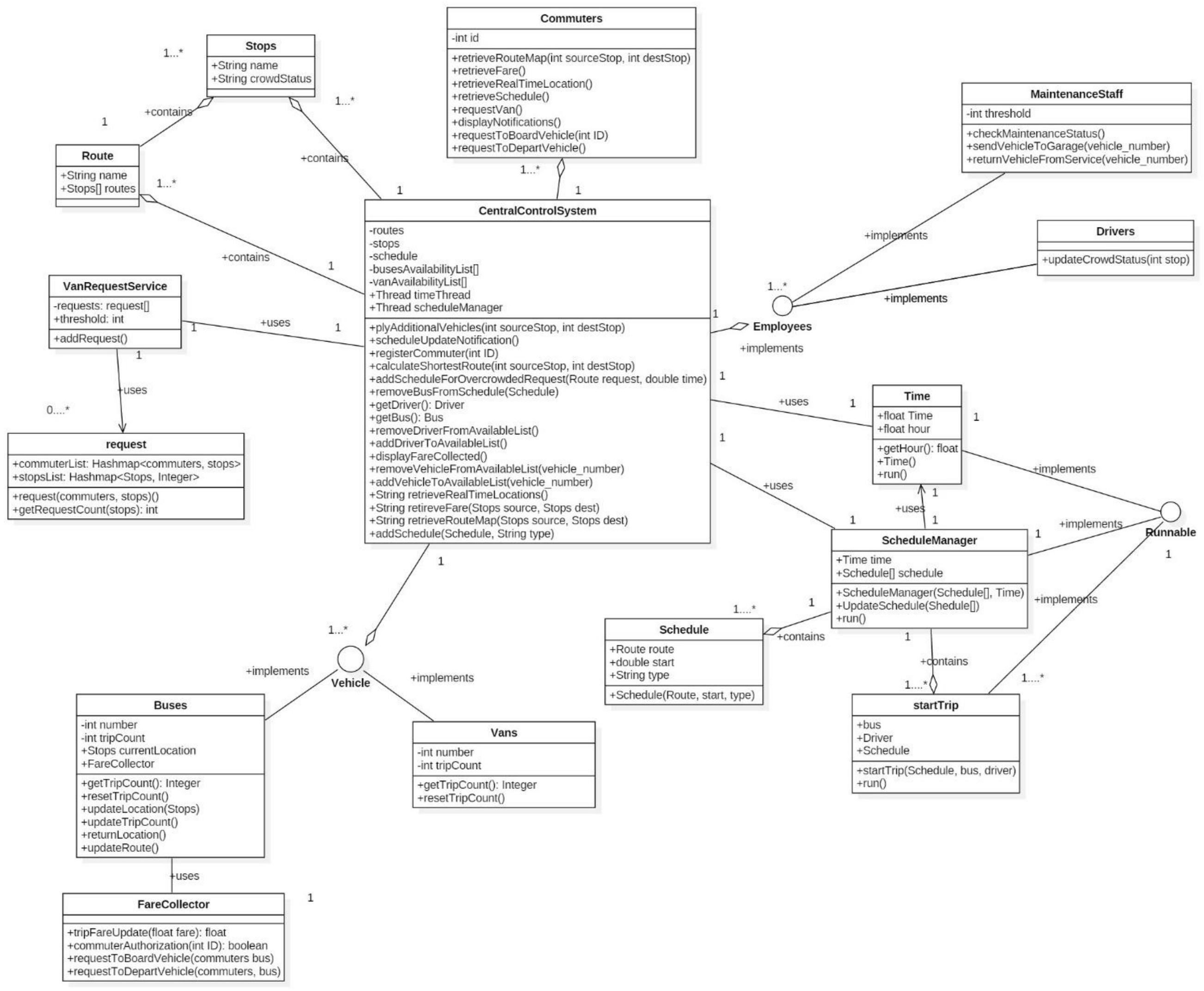

Stops
+String name
+String crowdStatus
Commuters
-int id
+retrieveRouteMap(int sourceStop, int destStop)
+retrieveFare()
+retrieveRealTimeLocation()
+retrieveSchedule()
+requestVan()
+displayNotifications()
+requestToBoardVehicle(int ID)
+requestToDepartVehicle()
MaintenanceStaff
-int threshold
+checkMaintenanceStatus()
+sendVehicleToGarage(vehicle_number)
+returnVehicleFromService(vehicle_number)
Route
+String name
+Stops[] routes
CentralControlSystem
-routes
-stops
-schedule
-busesAvailabilityList[]
-vanAvailabilityList[]
+Thread timeThread
+Thread scheduleManager
+plyAdditionalVehicles(int sourceStop, int destStop)
+scheduleUpdateNotification()
+registerCommuter(int ID)
+calculateShortestRoute(int sourceStop, int destStop)
+addScheduleForOvercrowdedRequest(Route request, double time)
+removeBusFromSchedule(Schedule)
+getDriver(): Driver
+getBus(): Bus
+removeDriverFromAvailableList()
+addDriverToAvailableList()
+displayFareCollected()
+removeVehicleFromAvailableList(vehicle_number)
+addVehicleToAvailableList(vehicle_number)
+String retrieveRealTimeLocations()
+String retireveFare(Stops source, Stops dest)
+String retireveRouteMap(Stops source, Stops dest)
+addSchedule(Schedule, String type)
Drivers
+updateCrowdStatus(int stop)
VanRequestService
-requests: request[]
+threshold: int
+addRequest()
request
+commuterList: Hashmap<commuters, stops>
+stopsList: Hashmap<Stops, Integer>
+request(commuters, stops)()
+getRequestCount(stops): int
Time
+float Time
+float hour
+getHour(): float
+Time()
+run()
ScheduleManager
+Time time
+Schedule[] schedule
+ScheduleManager(Schedule[], Time)
+UpdateSchedule(Shedule[])
+run()
Schedule
+Route route
+double start
+String type
+Schedule(Route, start, type)
startTrip
+bus
+Driver
+Schedule
+startTrip(Schedule, bus, driver)
+run()
Buses
-int number
-int tripCount
+Stops currentLocation
+FareCollector
+getTripCount(): Integer
+resetTripCount()
+updateLocation(Stops)
+updateTripCount()
+returnLocation()
+updateRoute()
Vans
-int number
-int tripCount
+getTripCount(): Integer
+resetTripCount()
FareCollector
+tripFareUpdate(float fare): float
+commuterAuthorization(int ID): boolean
+requestToBoardVehicle(commuters bus)
+requestToDepartVehicle(commuters, bus)
Employees
Vehicle
Runnable
+contains
+contains
+contains
+contains
+contains
+uses
+uses
+uses
+uses
+uses
+uses
+implements
+implements
+implements
+implements
+implements
+implements
+implements
+implements
1...*
1...*
1...*
1...*
1...*
1...*
0....*
1....*
1....*
1....*
1
1
1
1
1

Figure 1: Final Class Diagram

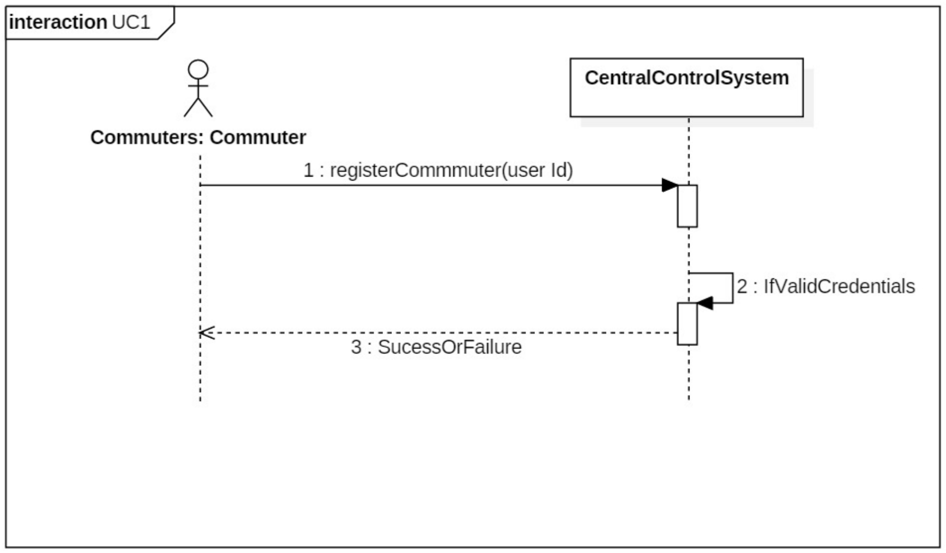
Use Case 1(UC1):

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.

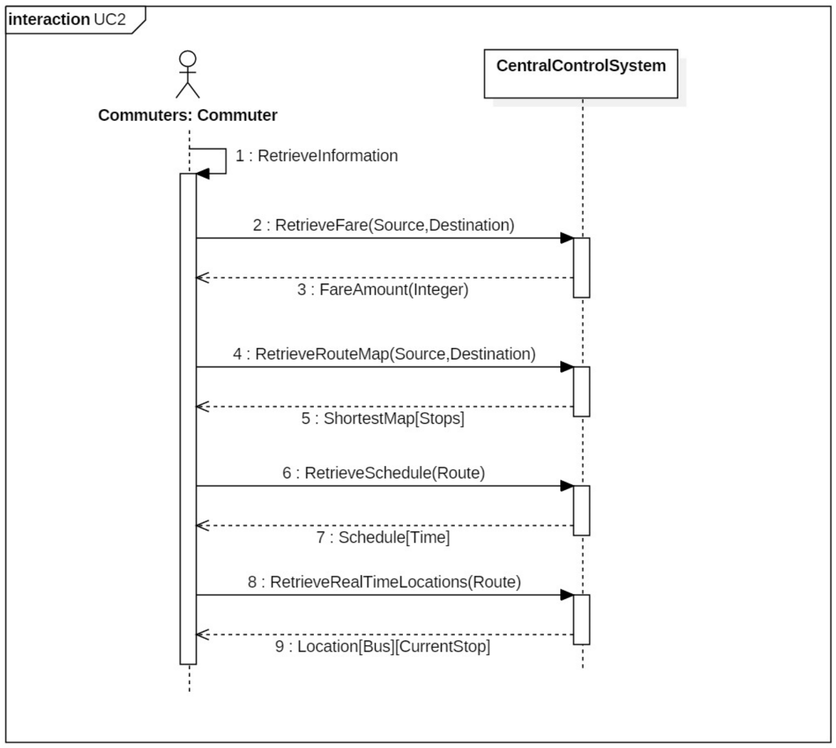
ii. As explained in the Use Case diagram, the commuter object calls the Central Control System to request to register himself to the database. The commuter is registered to the database by the Central Control System and control is passed back.

Interaction Diagrams



Use Case 2(UC2):

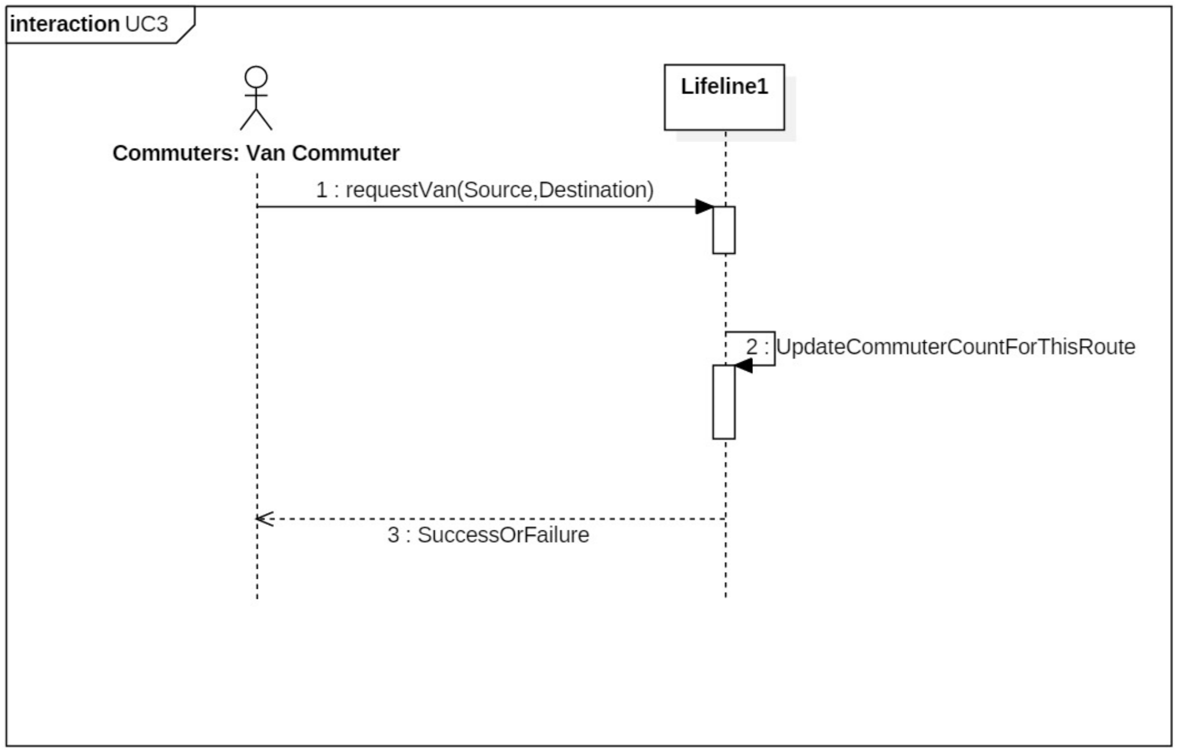
Summary:
The registered commuter wants to enquire about the best route from his source to intended destination. The object commuter sends a retrieveRouteMap, retrieveFare, retrieveSchedule and retrieveRealTimeLocation request to the Central Control System along with the source and destinations as shown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


interaction UC2
Commuters: Commuter
CentralControlSystem
1 : RetrieveInformation
2 : RetrieveFare(Source,Destination)
3 : FareAmount(Integer)
4 : RetrieveRouteMap(Source,Destination)
5 : ShortestMap[Stops]
6 : RetrieveSchedule(Route)
7 : Schedule[Time]
8 : RetrieveRealTimeLocations(Route)
9 : Location[Bus][CurrentStop]

We have implemented code for best route retrieval, Schedule retrieval, Route retrieval and cheapest fare retrieval 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.

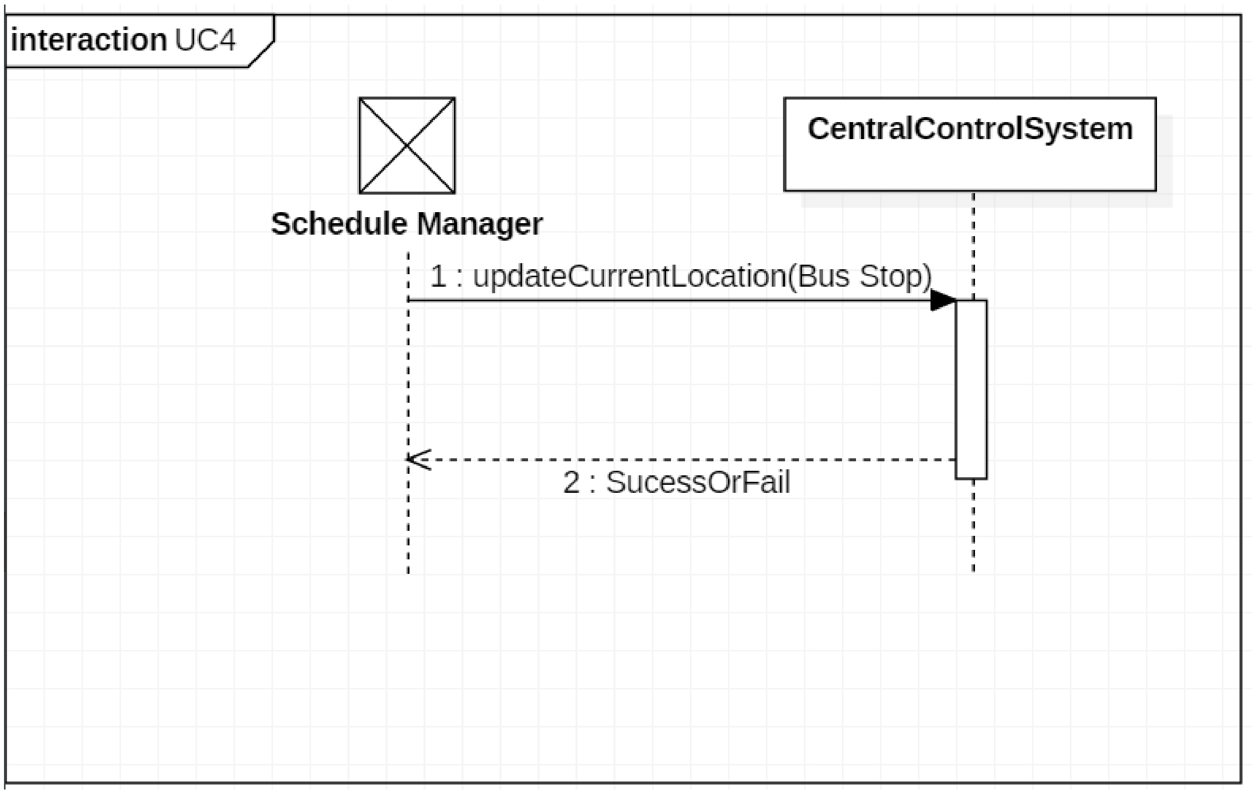
Use Case3(UC3):

Summary:
This Case deals with User Requesting Van on demand. When requests for a van service arrive, they are added to the existing requests categorized by stops.


interaction UC3
Commuters: Van Commuter
Lifeline1
1 : requestVan(Source,Destination)
2 : UpdateCommuterCountForThisRoute
3 : SuccessOrFailure

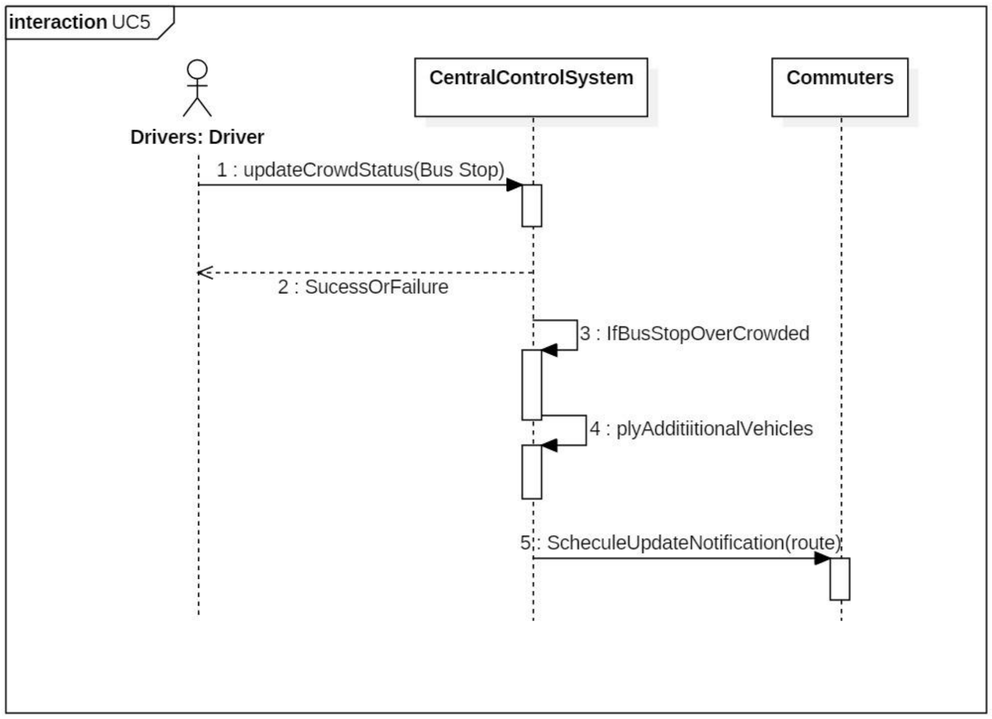
Use Case4(UC4):

Summary: This Case deals with Automatic Real Time location update of Bus to Central System something similar to Real Time GPS location update. Once this location is updated to Control System, any further calls to this return updated location of Vehicle.



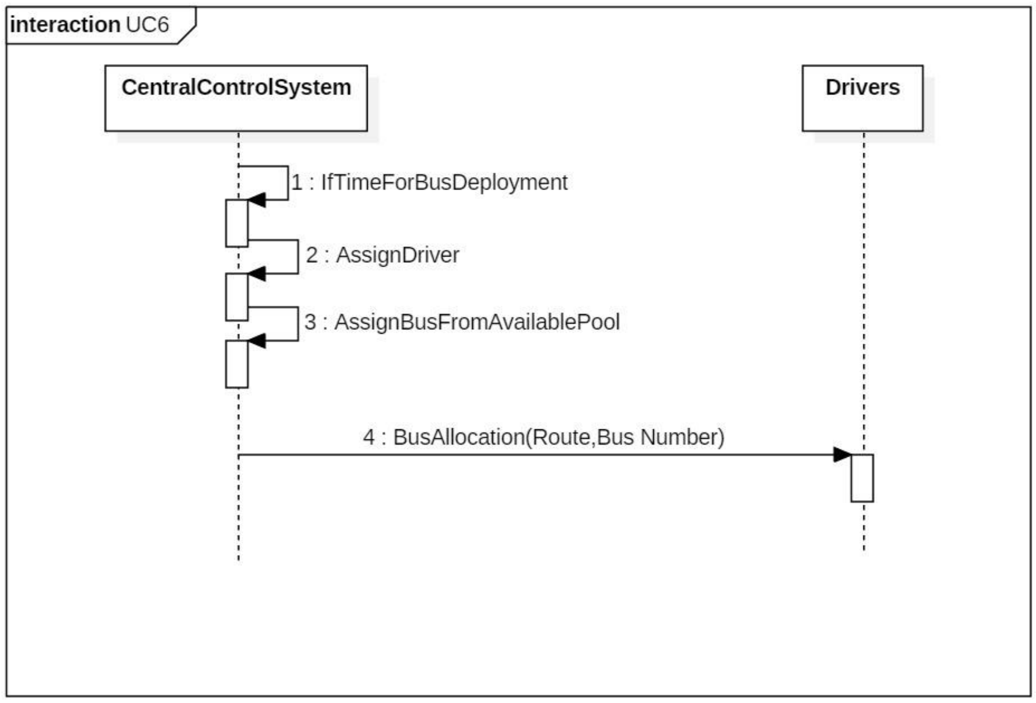
Use Case5(UC5):

Summary: This Case deals with Crowd Status update to Central Control System when Driver observes that there are more number of people wanting to use transport system for Particular Route. On receiving this Request, Central System checks for available Driver and Vehicle and if any available is retrieved deploys same on field.



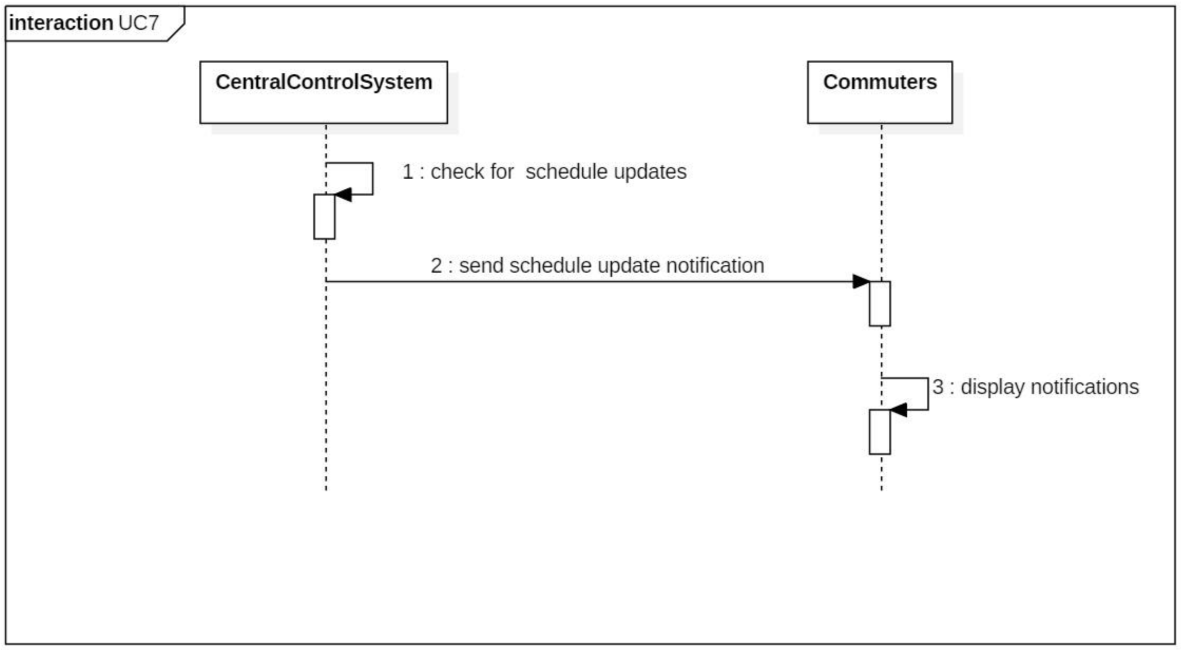
<u>Use Case6(UC6):</u>

<u>Summary:</u> This case deals with the deployment of a bus as per schedule. Central Control System creates a Time thread and a scheduleManager thread at the beginning. The scheduleManager thread keeps checking for upcoming trips and when it is time for a scheduled trip, it fetches an available bus and a driver and creates a startTrip thread to depict the journey of the bus along the scheduled route. The real time location of the bus is updated periodically. The bus and the driver are returned to the availability lists at the conclusion of the trip and the trip fare is updated to the central control system.


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

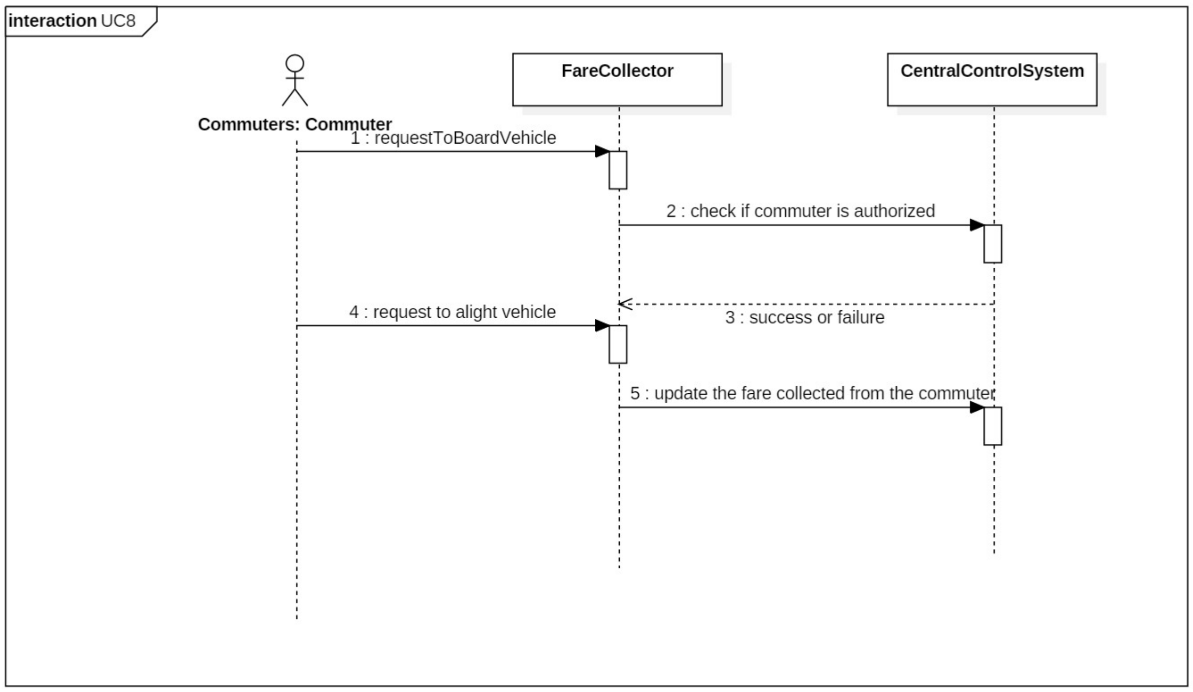
Use Case7(UC7):

Summary: This Case is about sending Schedule update Notification to all registered commuters when there is change in Schedule such as when any planned trip is started or when schedule is updated after Vehicle is deployed on request.


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

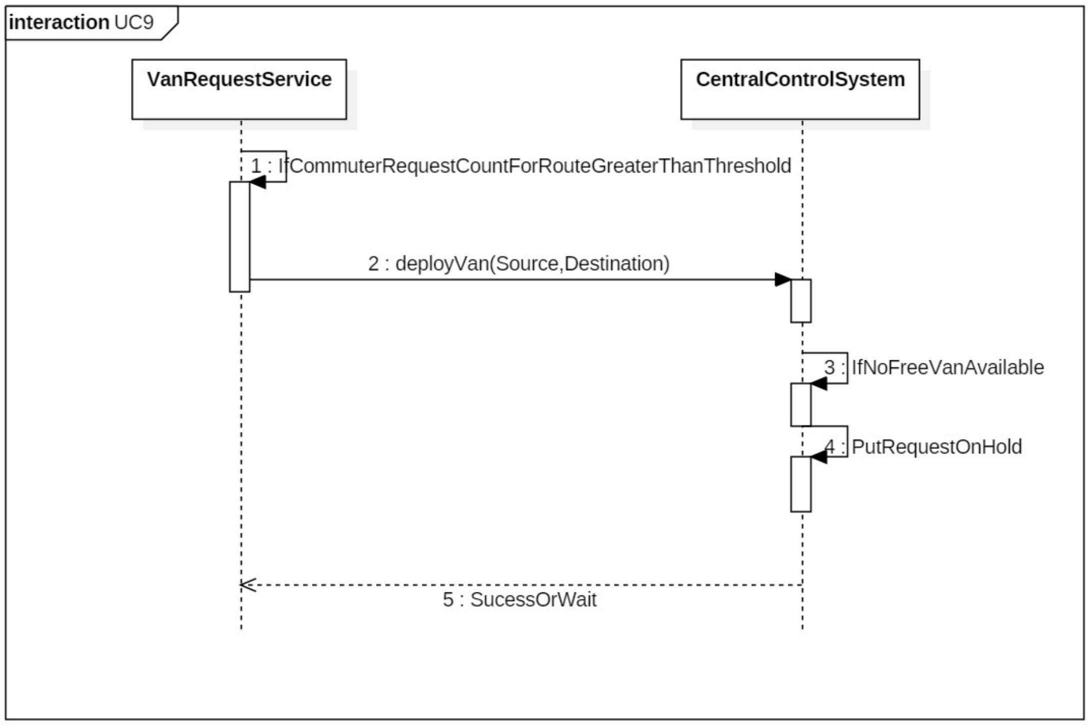
<u>Use Case8(UC8):</u>

<u>Summary:</u>
This Case is about authorizing Commuter to use transport System on Card/Ticket Swipe. Central Control system performs authorization of Commuter boarding request by cross-checking request against registered Commuters database.


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

Use Case9(UC9):

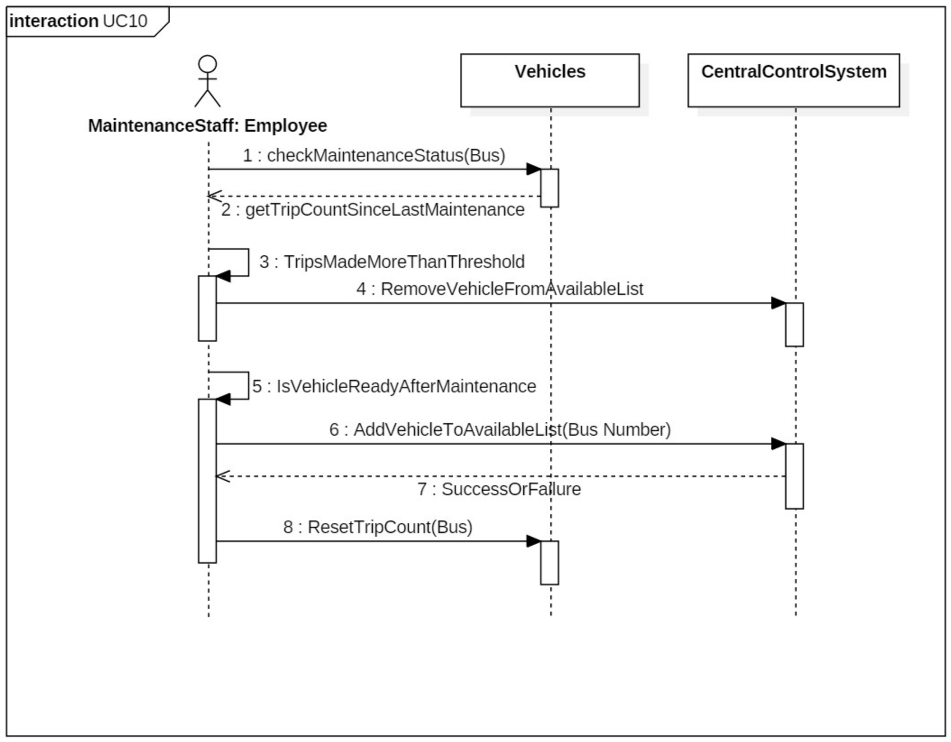
Summary:
This Case is about User Requesting Vehicle on need. When requests for a particular stop reach a threshold, a van is deployed to the stop and a notification is sent to commuters who have sent van requests from the particular stop.



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


interaction UC10
MaintenanceStaff: Employee
Vehicles
CentralControlSystem
1 : checkMaintenanceStatus(Bus)
2 : getTripCountSinceLastMaintenance
3 : TripsMadeMoreThanThreshold
4 : RemoveVehicleFromAvailableList
5 : IsVehicleReadyAfterMaintenance
6 : AddVehicleToAvailableList(Bus Number)
7 : SuccessOrFailure
8 : ResetTripCount(Bus)

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	ScheduleManager
UC5	Ply additional vehicles depending on crowd status	Driver, CentralControlSystem
UC6	Deploy vehicles as per schedule, assign a driver to the vehicle	CentralControlSystem, ScheduleManager
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.

iv. Table for use-cases and classes

	Buses	CentralControlSystem	Commuters	Drivers	FareCollector	MaintenanceStaff	VanRequestService	Request	startTrip	scheduleManager	Time
UC1	-	registerCommuter()	registerCommuter	-	-	-	-	-			
UC2	-	retrieveRouteMap(), retrieveFare(), retrieveRealTimeLocation(), retrieveSchedule()	retrieveRouteMap(), retrieveFare(), retrieveRealTimeLocation(), retrieveSchedule()	-	-	-	-	-			
UC3	-	-	requestVan()	-	-	-	addRequest()	-			
UC4	updateLocation()	-	-	-	-	-	-	-	Run()		
UC5	-	plyAdditionalVehicles(), addScheduleForOverCrowdRequest() scheduleUpdateNotification()	-	plyAdditionalVehicles()	-	-	-	-			
UC6	-	ScheduleManger.start() Time.start(), getBus(), removeVehicleFromAvailabilityList(),getDriver(), removeDriverFromAvailabilityList() addVehicleToAvailabilityList(), addDriverToAvailabilityList(),	-		-	-	-	-	Run()	Run() startTrip.start()	Run()
UC7	-	scheduleUpdateNotification()	displayNotifications()	-	-	-	-	-			
UC8	-	-	requestToBoardVehicle(), requestToDepartVehicle()	-	commuterAuthorization(), tripFareUpdate()	-	-	-			
UC9	-	-	displayNotitifications()	-	-	-	getRequestCount()	-			
UC10	getTripCoun	-	-	-	-	checkMaintenanceStatus(),	-	getTripCount(),			

	t(), resetTripCount()					updateAvailibilityList()		resetTripCount()			

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. Implementing the entire project part by part in decreasing order of use-case complexity helped us tackle the humongous task in an efficient manner without getting overwhelmed.

One of the ways in which we understood the importance of the methodology is when we overlooked implementing the classes for the live demonstration, as these were not officially a part of the use cases decided at design time. We did not implement them during Phase 3 and they proved to be the biggest challenge going into the final phase.

It has been a great learning experience as we attempted multithreading for the first time and came to know of the various issues it can cause to the project implementation. We have written efficient code to implement the same using a limited number of classes implementing the runnable interface and creating multiple threads for each of the trips according to schedule to accurately depict a real life bus transport system with its various complications like the non-availability of buses or drivers and changes in schedules and such.

We have also learnt about writing code contracts and class invariants which have helped to always keep an eye on the entry, exit and boundary conditions of the methods we wrote.

The implementation expanded into something much bigger than what we envisioned at the start, but we tried to keep up with the increasing requirements and tried to implement as much as of it as possible to not compromise on the completeness and quality of the implementation. We have attached the final class diagram above which depicts the difference between the class diagram at design time and after final implementation. The change is not drastic, however the core classes remained the same, but for the live demonstration of the transport system, the classes implementing the runnable interface came into picture. Implementing those classes took time, effort and debugging, but it also covered a few use cases directly like updating the real time locations of all the buses, fare collection from commuters, etc.

In conclusion, we would like to say thank you for giving us a free hand and letting us implement a project which we conceptualized from the start to finish. We are also grateful for your inputs which have helped us implement the project in a better way.