Concordia Institute for Information Systems Engineering (**CIISE**)

System Design Specification Document

Noble Team

Yu Jin (23753034)

Gurpreet Kaur (40106011)

Danie(A)Zakerifar (40054463)

INSE 6260

Submitted To: Dr. Chun Wang

Table of Content

1 Overview

.

1.1 Goals and Objectives

The purpose of system design specification documents is to completely describe the system at architecture level, including subsystems and their mapping, database mapping, interface design.

The design of the simulator should be done in order to satisfy following criteria:

1. Usability: The software simulator should be easy to use along with meeting all the customer requirements.
2. Efficiency: The software should be design considering the fact that computation part (#of chargers required, # of buses required, schedule of the buses) takes less time once the input is provided.

1.2 Statement of Scope

This software can fit to the any schedule. The constraints are that it can works for individuals' schedules and cannot consider the coverage of several schedule at the same time. But if we define some changes, we can improve this software to consider the whole schedule in the city.

The input is the schedule of East and West terminals. The output is the numbers of buses and chargers that we need in each terminals. It also produces the schedule for the each bus and charger.

The software simulates the process of traveling and charging to produce the outputs

1.3 Terminology & Definitions

1.3.1 Abbreviations

HS:	High Speed Charger
ON:	Overnight Charger

2 Functional Descriptions

The Simulator software system is composed of the following basic components:

1. User Interface
2. Computational Part
3. Database

User Interface

It will be a console based graphical user application that will take inputs from users like manufacture of charger, battery size of bus, cost of bus, chargers. Once input is provided by the user it will display the result accordingly.

Computational Part

In this part, algorithm of our simulator will calculate number of buses required, number of chargers of each type at two terminus and suspected budget to complete the electrification of 211 bus route.

Database

The database used as part of the software. Bus transit schedule, User inputs are stored in database.

3 Non-Functional Descriptions

Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.

3.1 Solution quality

The software should **show** us the number of buses that satisfy all the time slots in the schedule. The solution is for one day schedule trip.

3.2 Flexibility

Software should be able to fit with all the schedule from the other routs. For here we are just using rout 211 but we can adjust to any routs.

It is also compatible with every schedule. The only thing that we have to do is putting the schedule of bus in the file for reading through the software.

3.3 Real time responsiveness

For Windows 10 and Java Application, 8 gigs ram it should take 10 seconds. No application should be open, when they are using this application.

3.4 Reliability

If the user input the data according to the options that it has, the software should work properly. It the user input wrong data it should run it again.

4 System Architecture

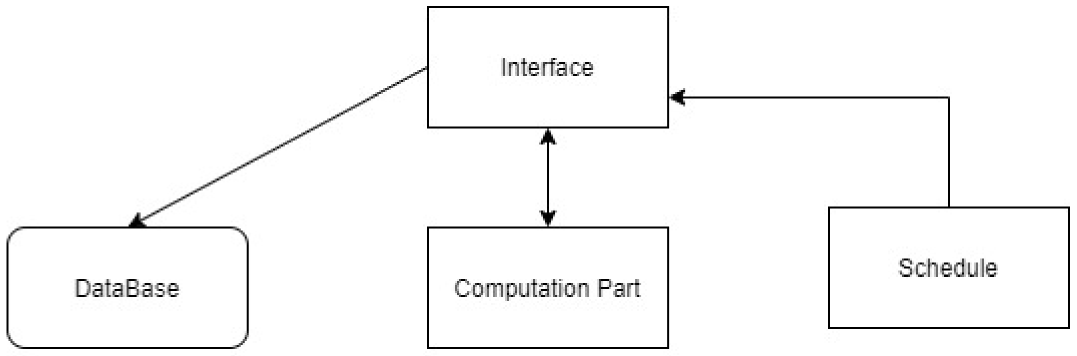


Figure 1 System Architecture

5 Detailed Architecture

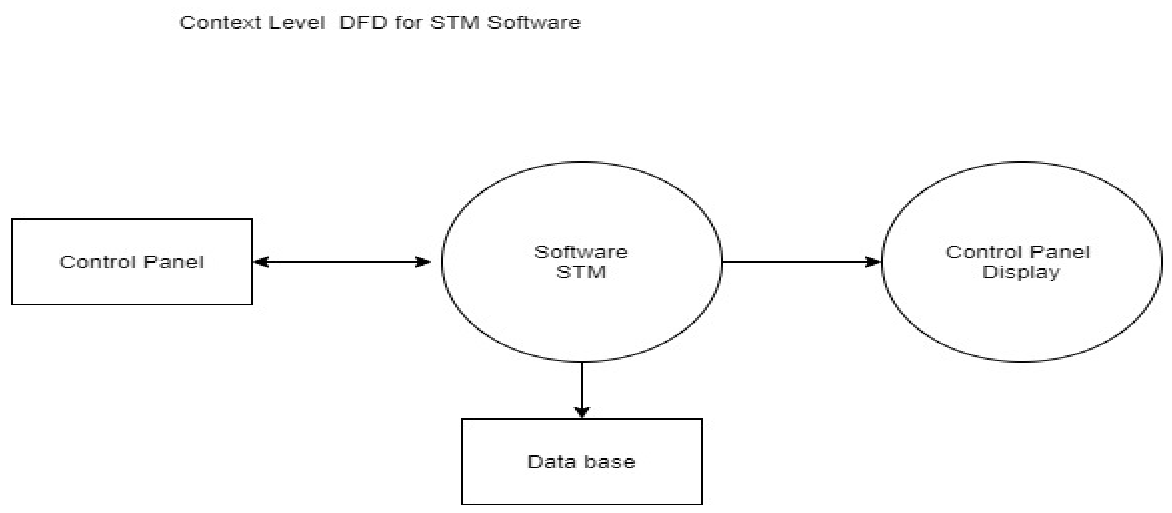


Figure 2Context Level DFD STM Software

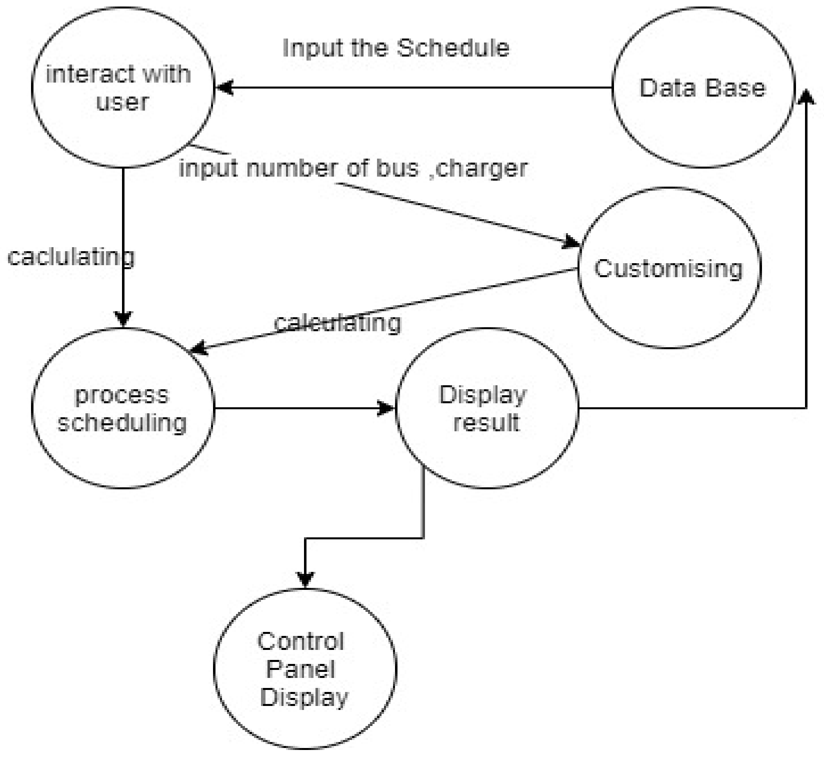
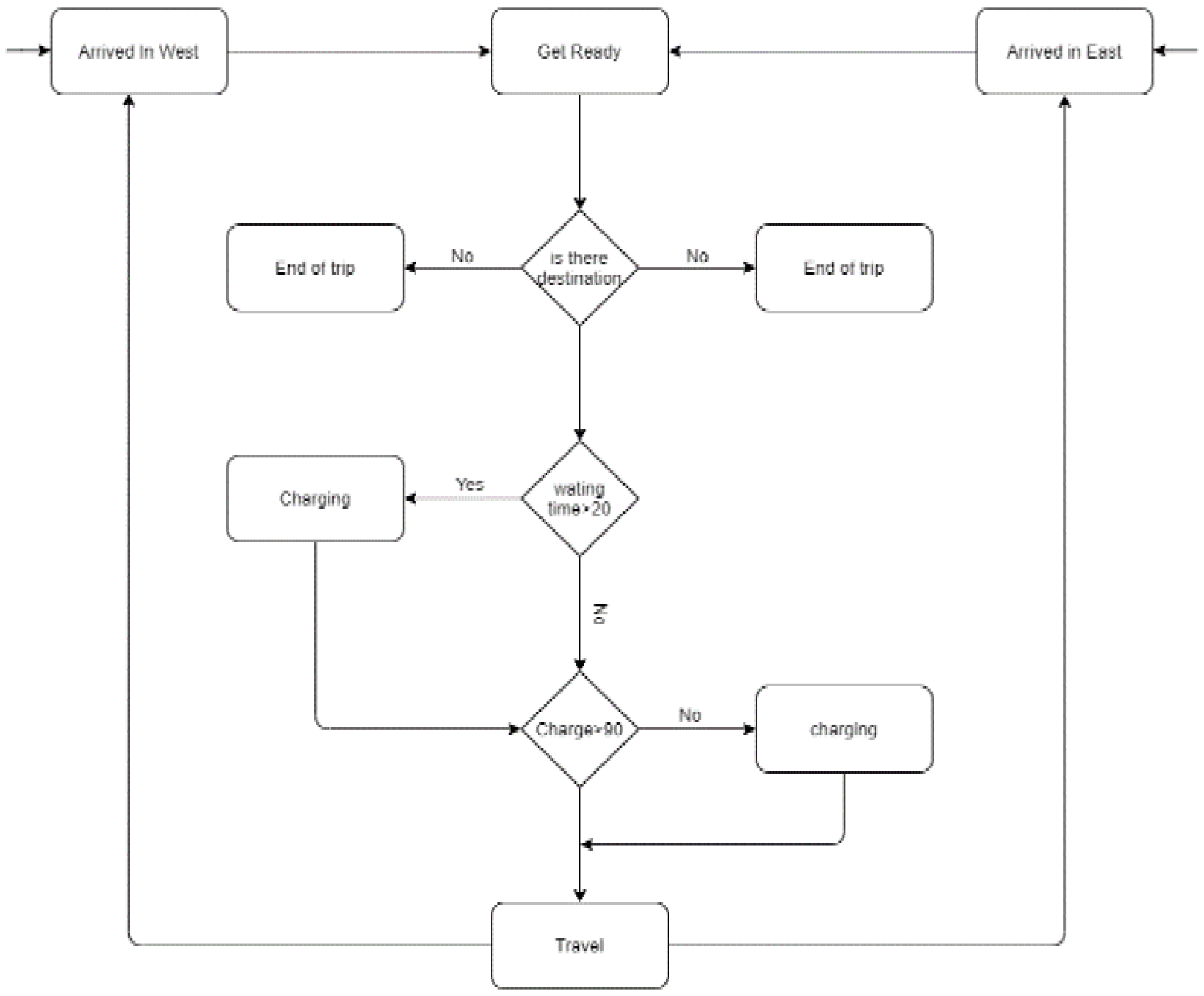


Figure 3: Level 1 DFD for STM Software

5.1 Process Scheduling

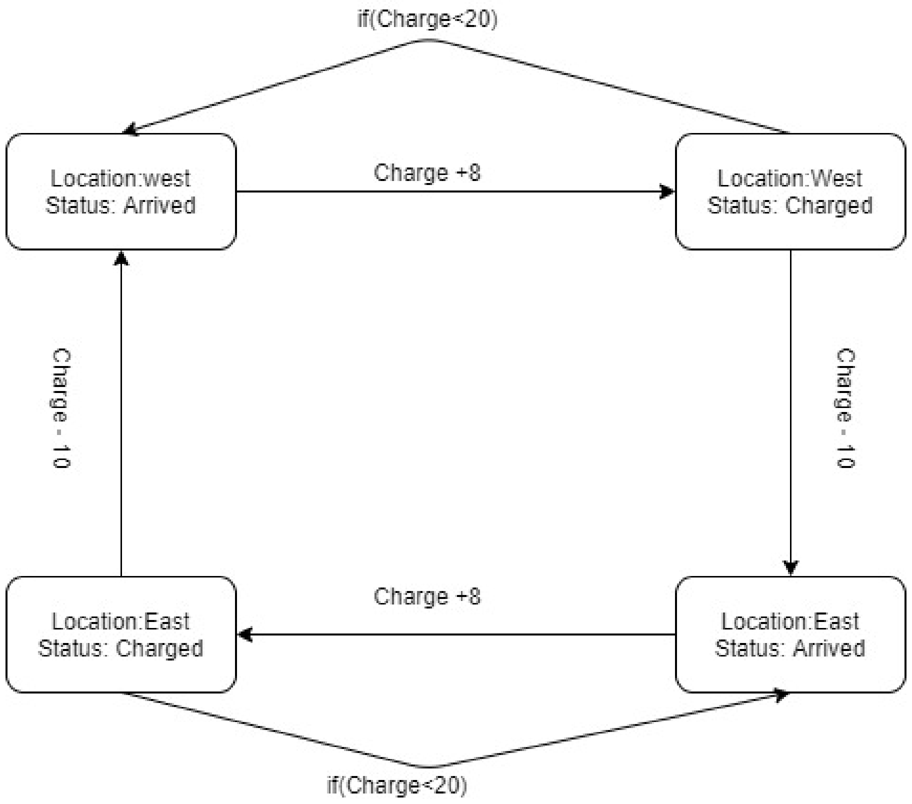
At first, we must find the algorithm. The algorithm that we are using is the following:



The formula for charging the battery of bus is, P(kwh) = E(kw)T(h)

Where P(kwh) = Increased battery level after charging,
E(kw) = Power of charger,
T(h) = Time spent for charging

The model of algorithm is given below.



Policies for our software:

1. When to charge the bus?
   - if, (waiting time < 20 minutes & battery remaining > 90) then, no charging required
   - If, (waiting time >20 minutes & battery remaining >90) then, charge using high speed charger
   - If, (battery remaining <90) then, charge using low speed charger
2. Charging Time Required by slow speed charger
   At least 30 minutes
3. Charging Time Required by high speed charger
   At most 1 minutes
4. What is the charging rate of chargers?
   - ABB = 1 minute of charging will charge the battery to 7units
   - HELIOX = 1 minute of charging will charge the battery to 5 units
5. What is the utilization of bus battery?
   1km of distance will utilizes 1kwh of energy

The following are our requirements (numbers are in percent):

1. The charge of the battery should be always more than zero. This is because whenever the charge of battery become zero, it cannot move anymore. The formal language for that is (G(StateOfCharging>0))

2. Moreover, to make sure that the bus has enough energy to reach charger station, we have to define the limit for that. For doing that we consider that limit as the amount of energy needed for reaching from one station to another station. We consider that limit equals 10. So, the charge of battery should always be more than 10. The formal language for that is $\neg(F(x < 10)$

3. The energy consuming for one trip is 10 so for the condition that energy more than 10, we can have a trip with bus when it has at least 20. When is 20, after a one trip the energy become 10. So, it must go charging when it is 20. So the charge of battery often goes less than 20 but it always more than 10. The formal language for that is (GF(x<20)

4. The charge of the battery cannot go more than the capacity of battery which is 100. The formal language for that is F(StateOfCharger>100) is false.

5. We cannot move anywhere until we have sufficient charging. The formal language for that is check_ltlspec -p"!(status!=charged U status=arrived)“ Is true

6. The bus cannot be in two stations at the same time. The formal language to represent this is, !(location==west and location==east)

7. Whenever the charge of the battery become less than 20, it continue charging until it increase energy to 20. The formal language to represent this is, G(stateOfCharging<20andstatus=charging XXstatus=charging)

8. When charging become equals to 100, the charging stops. The formal language to represent this is (G(StateOfCharging ==100→b.charging.stops))

9. The bus cannot arrive anywhere if does not move.

10. When bus arrived, there is no possibility that it stops for ever.

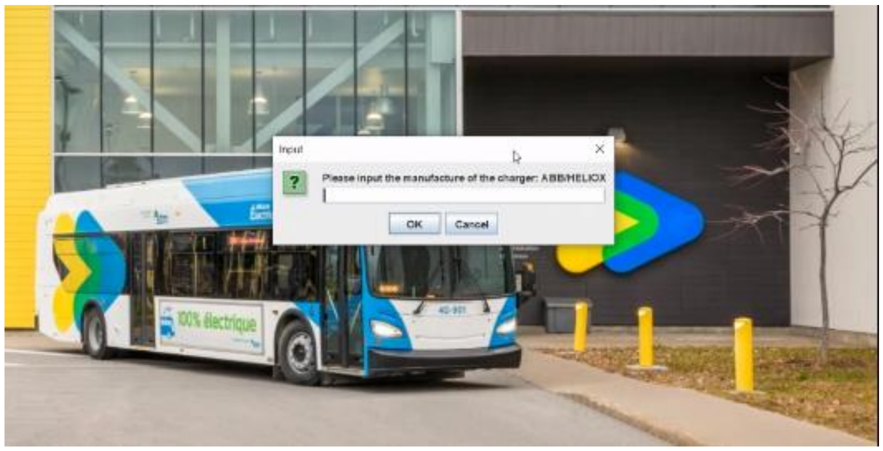
Then we used a model checker like NUSMV to check the our requirements are satisfied. After checking some time and improving model eventually we made an algorithm that satisfied our problem.

6 Detailed System Design

.

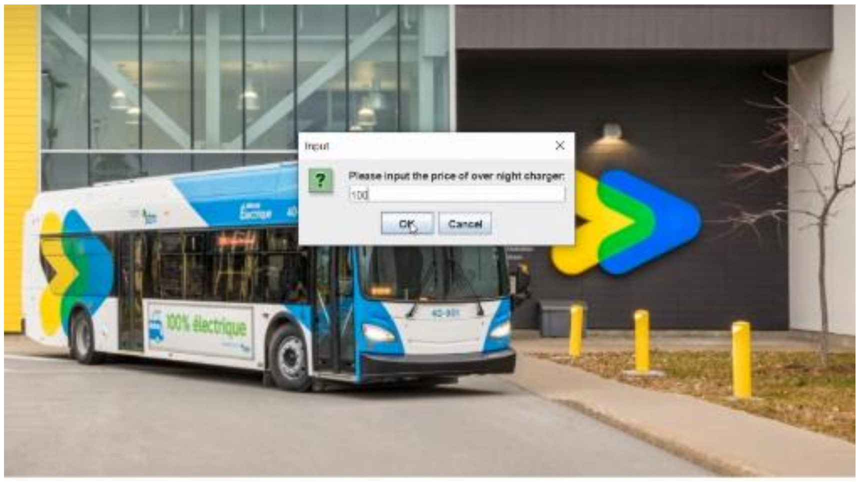
6.1 Graphic User Interfaces and a Basic Demo Scenario

At first it asked user to input manufacture of Chargers between ABB and HELIOX



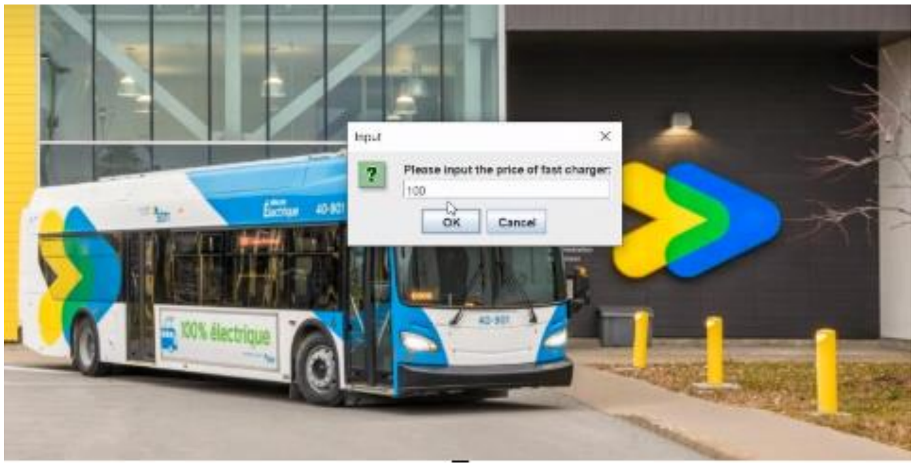
Please input the manufacture of the charger: ABB/HELIOX

Then it asked user to input the price of overnight charger



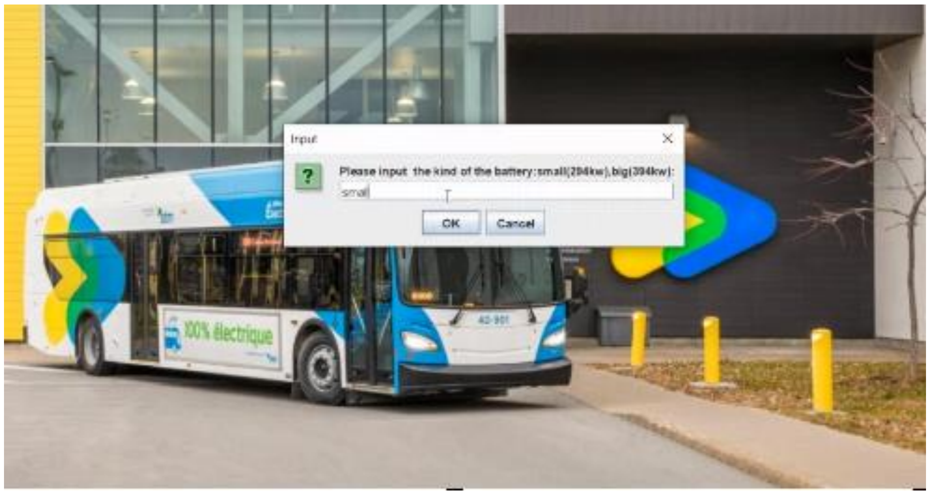
Please input the price of over night charger:
100

In the next step it asked user to input the price of fast chargers



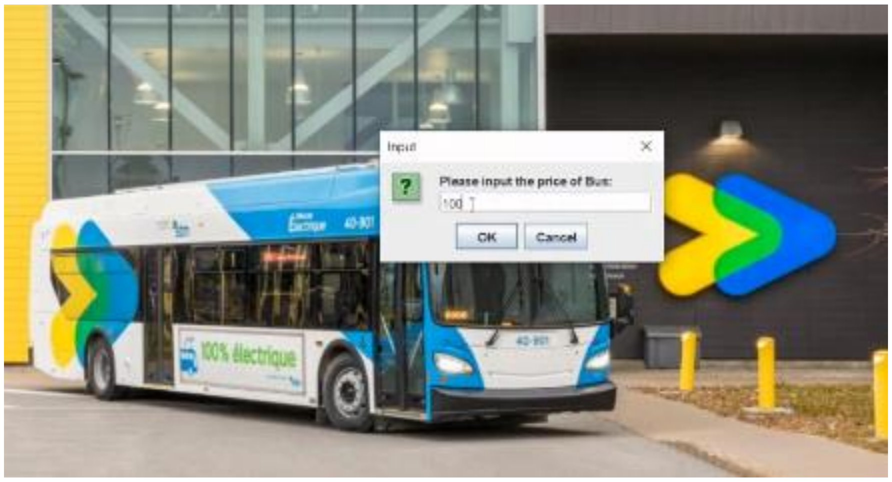
Please input the price of fast charger:
100

Then it ask user for the kind of bus.



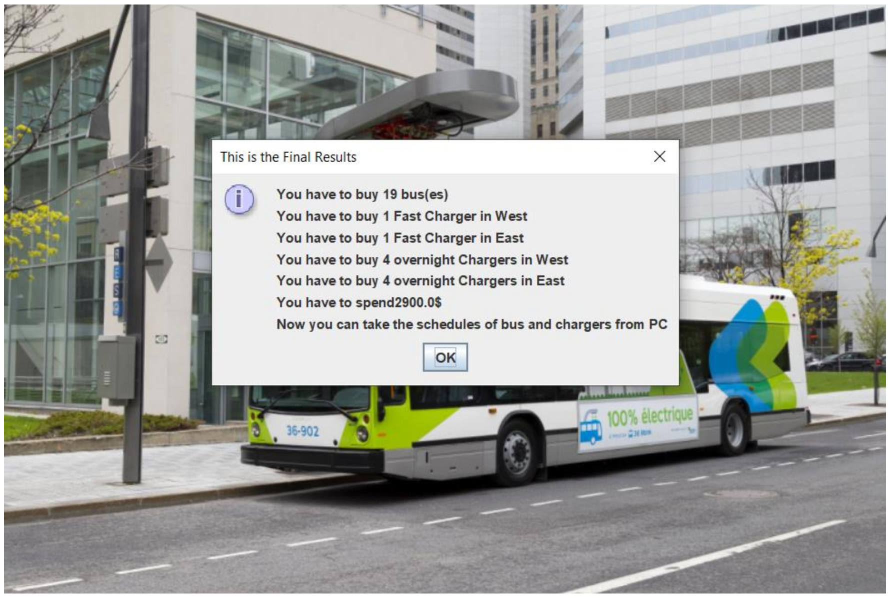
```
Input
Please input  the kind of the battery:small(294kw),big(394kw):
small
```

Then it asked user to input the price of bus



```
Input
Please input the price of Bus:
100
```

And eventually it is going to show how many buses, and charger you need. It also say about the amount of money that you have to spend for that.



```
This is the Final Results
You have to buy 19 bus(es)
You have to buy 1 Fast Charger in West
You have to buy 1 Fast Charger in East
You have to buy 4 overnight Chargers in West
You have to buy 4 overnight Chargers in East
You have to spend2900.0$
Now you can take the schedules of bus and chargers from PC
```

6.2 Subsystem

Devices

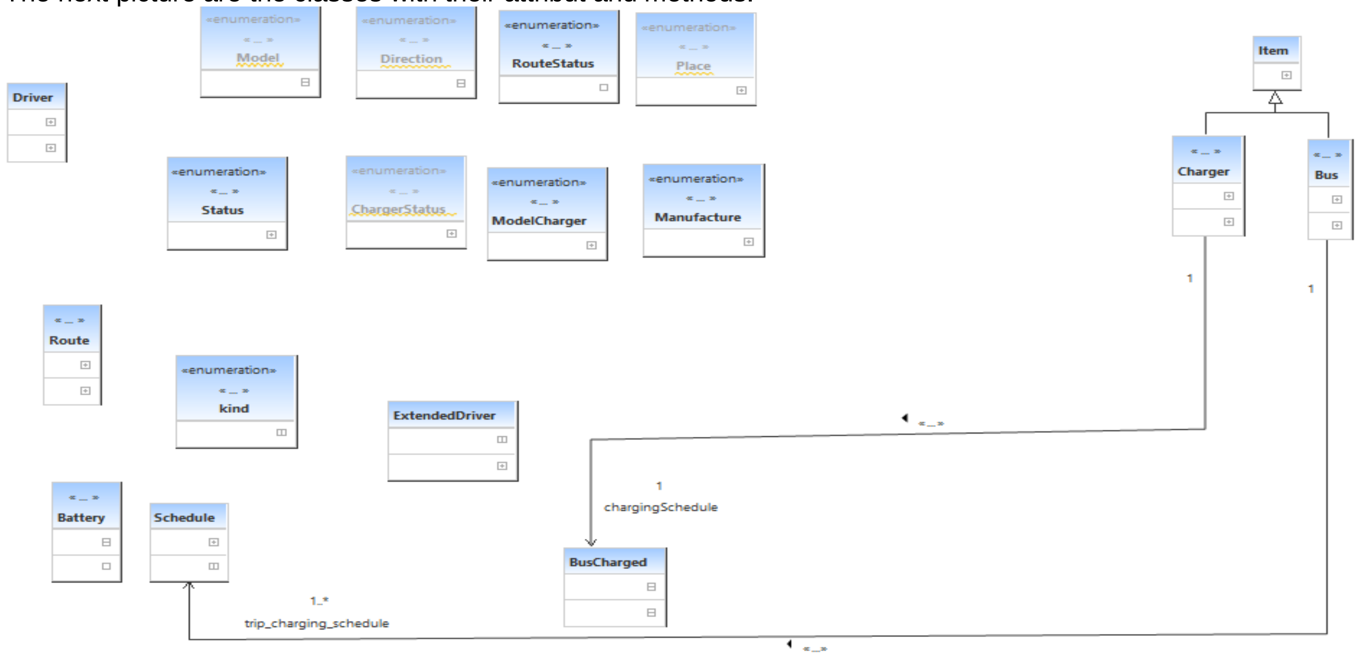
Property

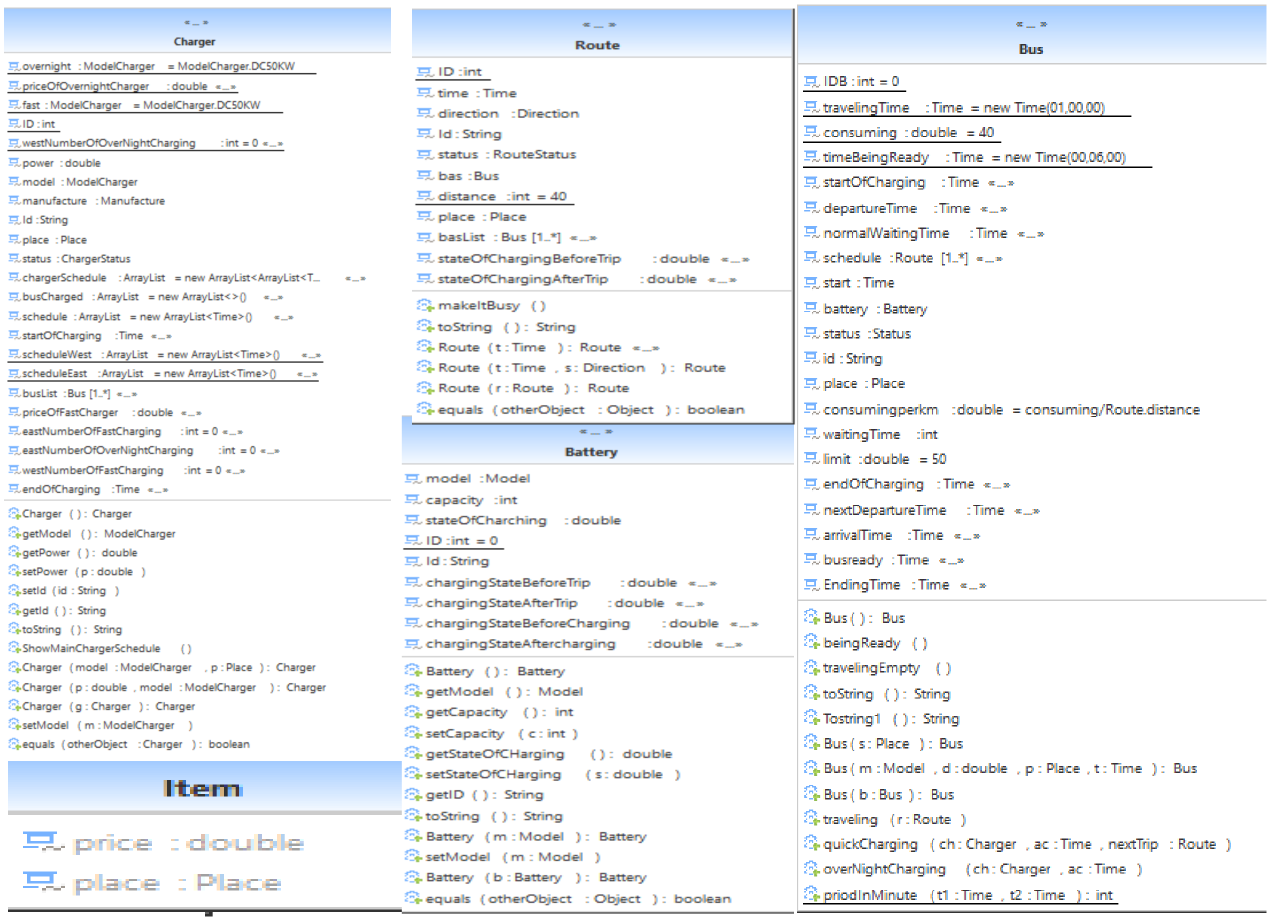
Interconnection

6.3 Class Diagram

The following picture is the diagram of all class.

The next picture are the classes with their attribut and methods.





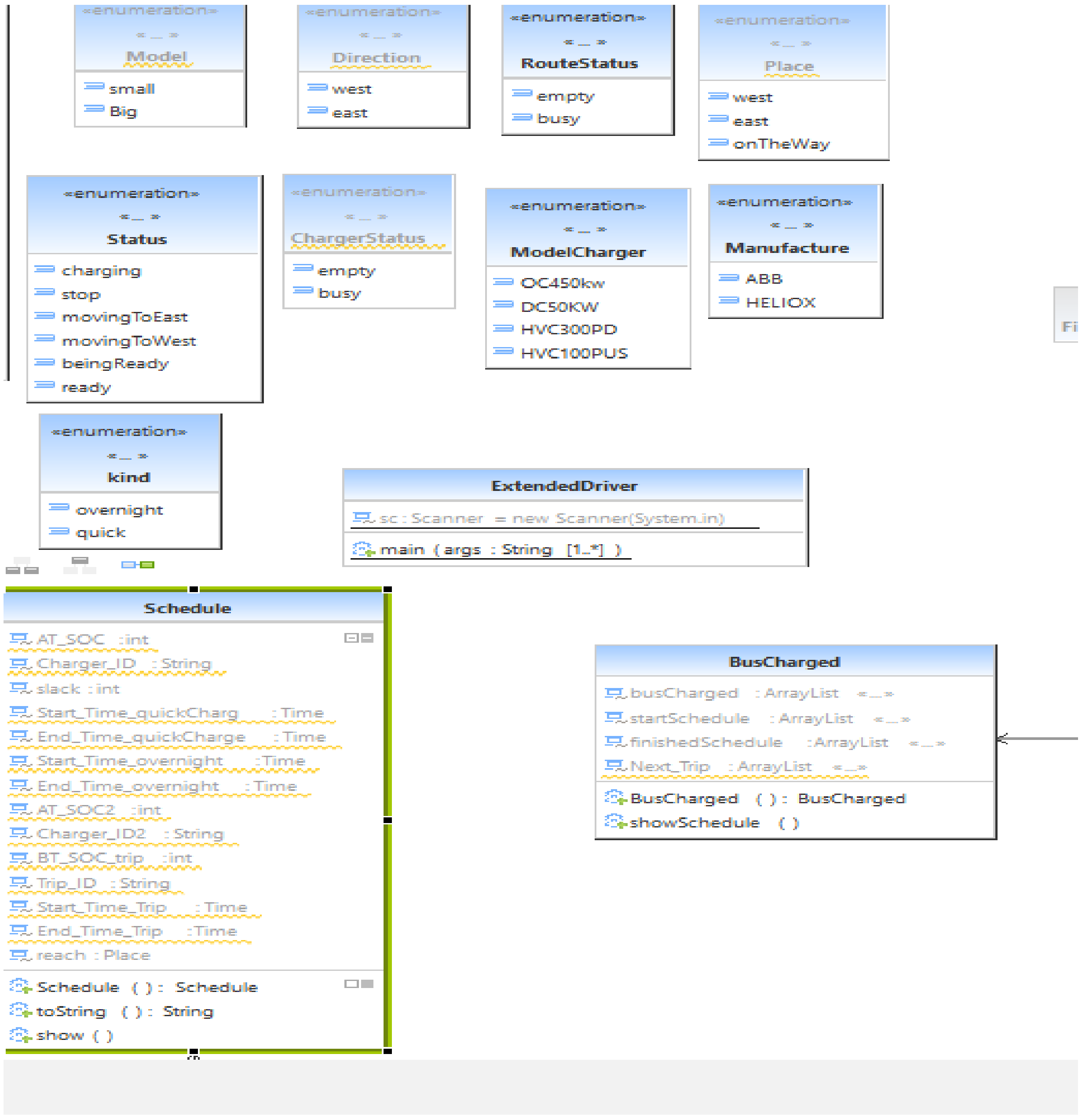
```
«_»
Charger

overnight  : ModelCharger  = ModelCharger.DC50KW
priceOfOvernightCharger  : double  «_»
fast  : ModelCharger  = ModelCharger.DC50KW
ID : int
westNumberOfOverNightCharging  : int = 0 «_»
power  : double
model  : ModelCharger
manufacture  : Manufacture
Id : String
place  : Place
status  : ChargerStatus
chargerSchedule  : ArrayList  = new ArrayList<ArrayList<T...  «_»
busCharged  : ArrayList  = new ArrayList<>()  «_»
schedule  : ArrayList  = new ArrayList<Time>()  «_»
startOfCharging  :Time  «_»
scheduleWest  : ArrayList  = new ArrayList<Time>()  «_»
scheduleEast  :ArrayList  = new ArrayList<Time>()  «_»
busList  :Bus [1..*]  «_»
priceOfFastCharger  : double  «_»
eastNumberOfFastCharging  : int = 0 «_»
eastNumberOfOverNightCharging  :int = 0 «_»
westNumberOfFastCharging  :int = 0 «_»
endOfCharging  :Time  «_»

Charger  ( ) :  Charger
getModel  ( ) :  ModelCharger
getPower  ( ) :  double
setPower  ( p : double  )
setId  ( id : String  )
getId  ( ) :  String
toString  ( ) :  String
ShowMainChargerSchedule  ( )
Charger  ( model  : ModelCharger  , p : Place  ) :  Charger
Charger  ( p : double  , model  : ModelCharger  ) :  Charger
Charger  ( g : Charger  ) :  Charger
setModel  ( m : ModelCharger  )
equals  ( otherObject  : Charger  ) :  boolean
```

```
Item

price  : double
place  : Place
```

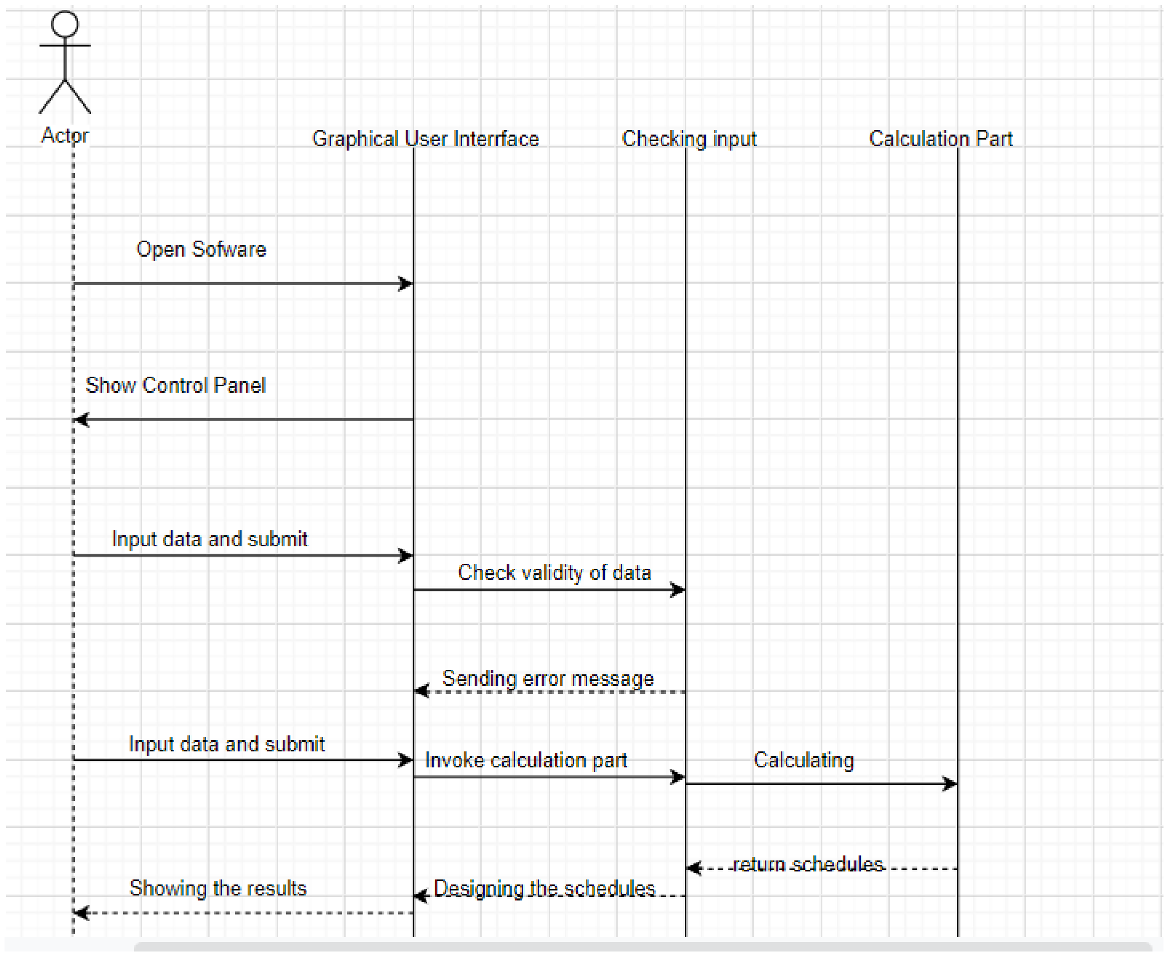
```
«_»
Route

ID : int
time  : Time
direction  : Direction
Id : String
status  : RouteStatus
bas  : Bus
distance  : int  = 40
place  : Place
basList  : Bus [1..*]  «_»
stateOfChargingBeforeTrip  : double  «_»
stateOfChargingAfterTrip  : double  «_»

makeItBusy  ( )
toString  ( ) :  String
Route  ( t : Time  ) :  Route
Route  ( t : Time  , s : Direction  ) :  Route
Route  ( r : Route  ) :  Route
equals  ( otherObject  : Object  ) :  boolean
```

```
«_»
Battery

model  :Model
capacity  :int
stateOfCharching  : double
ID : int  = 0
Id : String
chargingStateBeforeTrip  : double  «_»
chargingStateAfterTrip  : double  «_»
chargingStateBeforeCharging  : double  «_»
chargingStateAftercharging  :double  «_»

Battery  ( ) :  Battery
getModel  ( ) :  Model
getCapacity  ( ) :  int
setCapacity  ( c : int  )
getStateOfCHarging  ( ) :  double
setStateOfCHarging  ( s : double  )
getID  ( ) :  String
toString  ( ) :  String
Battery  ( m : Model  ) :  Battery
setModel  ( m : Model  )
Battery  ( b : Battery  ) :  Battery
equals  ( otherObject  : Object  ) :  boolean
```

```
«_»
Bus

IDB : int = 0
travelingTime  : Time  = new Time(01,00,00)
consuming  : double  = 40
timeBeingReady  : Time  = new Time(00,06,00)
startOfCharging  : Time  «_»
departureTime  :Time  «_»
normalWaitingTime  : Time  «_»
schedule  :Route  [1..*]  «_»
start  : Time
battery  : Battery
status  :Status
id : String
place : Place
consumingperkm  :double  = consuming/Route.distance
waitingTime  :int
limit  :double  = 50
endOfCharging  : Time  «_»
nextDepartureTime  :Time  «_»
arrivalTime  :Time  «_»
busready  : Time  «_»
EndingTime  : Time  «_»

Bus ( ) :  Bus
beingReady  ( )
travelingEmpty  ( )
toString  ( ) :  String
Tostring1  ( ) :  String
Bus ( s : Place  ) :  Bus
Bus ( m : Model  , d : double  , p : Place  , t : Time  ) :  Bus
Bus ( b : Bus ) :  Bus
traveling  ( r : Route  )
quickCharging  ( ch : Charger  , ac : Time  , nextTrip  : Route  )
overNightCharging  ( ch : Charger  , ac : Time  )
priodInMinute  ( t1 : Time  , t2 : Time  ) :  int
```


«enumeration» « _ » Model
small
Big
«enumeration» « _ » Direction
west
east
«enumeration» « _ » RouteStatus
empty
busy
«enumeration» « _ » Place
west
east
onTheWay
«enumeration» « _ » Status
charging
stop
movingToEast
movingToWest
beingReady
ready
«enumeration» « _ » ChargerStatus
empty
busy
«enumeration» « _ » ModelCharger
OC450kw
DC50KW
HVC300PD
HVC100PUS
«enumeration» « _ » Manufacture
ABB
HELIOX
«enumeration» « _ » kind
overnight
quick
ExtendedDriver
sc : Scanner = new Scanner(System.in)
main (args : String [1..*])
Schedule
AT_SOC :int
Charger_ID : String
slack : int
Start_Time_quickCharg : Time
End_Time_quickCharge : Time
Start_Time_overnight : Time
End_Time_overnight : Time
AT_SOC2 :int
Charger_ID2 : String
BT_SOC_trip :int
Trip_ID : String
Start_Time_Trip : Time
End_Time_Trip : Time
reach : Place
Schedule ( ) : Schedule
toString ( ) : String
show ( )
BusCharged
busCharged : ArrayList « _ »
startSchedule : ArrayList « _ »
finishedSchedule : ArrayList « _ »
Next_Trip : ArrayList « _ »
BusCharged ( ) : BusCharged
showSchedule ( )
Fi

6.4 Sequence Diagram



6.5 Suggestions for Web Service Integration

We can use JavaScript to connect this software with web application.