Requirement Analysis and Specification Document for PowerEnJoy

Enrico Migliorini, Alessandro Paglialonga, Simone Perriello

Politecnico di Milano

16 November 2016

What is PowerEnJoy?

PowerEnJoy is an upcoming service for renting electric cars. After reserving and getting on a car from special Parking Area, a User will be able to drive it for as long as necessary, returning it to any other Parking Area. The User will therefore benefit from a high degree of freedom, being able to use PowerEnJoy cars for both short- and long-distance movement.

Glossary

We will now show you some of the terms that we will use most prominently throughout the presentation.

Car-Sharing A Car-sharing service allows Users to rent Cars for a limited amountof time, being charged a Fee according to time and possibly applying a Discount or a Surcharge.

System The software structure this presentation is about.

User A person registered on the System, who has access to the Car-Sharing Service functionalities.

Visitor A person who needs to log in the System to access the Car-Sharing Service functionalities.

Glossary

Car An electric car owned by the Car-sharing service, rented to the User and tracked by the System.The Car communicates to the System its position, the status of its battery, its damages, the connection to an electrical socket and the number of seats occupied. A Car has a status and a Plugged flag, the status can be:

- *In Use*, if the engine is turned on. In this state, it cannot be Reserved by an User.
- *Available*, if it can be Reserved by an User.
- *Reserved*, if an User has reserved it but has still not unlocked it.
- *Unavailable* if it can't be Reserved by any User (for example due to damage, battery exhaustion, maintainance, ...)

Additionally, the *Plugged* flag indicates if the Car is plugged or not to the socket of the Charging Area.

Glossary

Fee The amount of money that the User will be charged for their usage of the Car-sharing service, or for making a Reservation that is not fulfilled.

Discount A reduction in the Fee because of good behaviour on the part of the User, e.g. leaving the Cars plugged or bringing it back with a mostlyfull battery. The actions that constitute good behaviour are determined ad detailed further in the presentation.

Surcharge An increase in the Fee caused by an improper behaviour on the part of the User, e.g. bringing the Cars back with a mostly-empty battery.

Glossary

Parking Area A place where the User can leave their Car and exit it to end the Ride. Parking Areas are predefined by the System.

Charging Area A special Parking Area where Cars plugs can be connected to the socket in order to recharge their Battery.

System Requirements

These are the requirements that the final system should fulfill.

1. PowerEnjoy shall provide Users with the ability to access all the System functionalities reserved to them.
2. PowerEnjoy shall support Users in locating Available Cars within a range of 5 Km from a specific position.
3. PowerEnjoy shall support Users in locating Parking Areas and their free parking slots.
4. PowerEnjoy shall support Users in locating Charging Areas and their free parking slots and free charging sockets.
5. PowerEnjoy shall support Users in reserve Available Cars.
6. PowerEnjoy shall apply a fixed Surcharge of 1 EUR if they have reserved a Car and not unlocked it within a time range of 60 minutes.
7. PowerEnjoy shall support a User in unlocking a Car they have previously reserved when they are within 15 meters of the same Car.

System Requirements

8. PowerEnjoy shall charge the User of a fixed fee per minutes, communicating them the Fee they will be charged at the end of the ride, only considering the driving time and the fee per minute.
9. PowerEnjoy shall be able to know if a User has took in their Car at least two other passengers for at least 3 minutes. If so, PowerEnjoy should apply a 10% Discount on the final Fee of the last ride.
10. PowerEnjoy shall allow the User to end the ride in a Parking or Charging Area.
11. PowerEnjoy shall allow any User who has ended a ride to plug the Car he/she has driven to a Socket in a time rage of 2 minutes since they have ended the ride, in order to get a 30% Discount on the final Fee of the last ride.
12. PowerEnjoy shall apply a 20% Discount on the final Fee of the last ride if the User will end the ride leaving the Car with more than 50% battery charge status.

System Requirements

13. PowerEnjoy shall apply a 30% Surcharge on the final Fee of the last ride if a User leaves the Car at more than 3 km from the nearest Charging Area or with a battery charge status less than 20%.
14. PowerEnjoy shall provide a User the ability to use the "Money Saving Option", telling him/her the position of a Charging Area where they have to park the Car he/she is driving in order to get a Discount on the total Fee. The Charging Area is determined by the System to ensure a uniform distribution of Cars in the city of that address and depends both on the destination of the User and on the availability of Sockets at the selected Charging Area.
15. PowerEnjoy shall interface with an external Mailing System to send emails to Users.
16. PowerEnjoy shall interface with an external Banking System to charge Fee to Users.

System Requirements

17. PowerEnjoy shall interface with an external GPS System to know the positions of Users and Areas.
18. PowerEnjoy shall interface with an external Mapping System to know the positions of Users and Areas.
19. PowerEnjoy shall interface with the existing Car to get their GPS position, damages, connection to an electrical socket, the number of seats occupied.

Domain Assumptions

These are the assumptions that we considered to be valid for the project.

1. The User can only have one Account at time.
2. The Company can decide at any time to block an User from accessing to the System (f.e. for improper behavior, unpaid bill, ...). It will be done by employees or Administrators.
3. The User always provides real correct data in his/her registration form.
4. The Database in which the Cars, Parking Areas, Charging Areas, Users,etc, are stored, is owned and managed from the Company (and not by this System), which is responsible for its security, reliability and availability. Our System is provided by the Company with read/write access to this Database.
5. The Company is responsible for the employees and their actions.

Domain Assumptions

6. The Car has a set of sensors that can detect, in every moment, its position, the status of its battery, the status of the engine, its damages, the connection of its plug to an electrical socket and the number of seats occupied. We assume that these sensors won't ever break down and that their measures are always correct.
7. The Car GPS always detects its position with absolute precision.
8. The User always enters the Car when he/she unlocks its doors.
9. After a Car is Plugged, it will not be maliciously unplugged by the User himself/herself or by other people.
10. After the doors of the cars are unlocked by the User, he/she always enters the Car, ignites the engine and leave the Parking Area.
11. An User can park/stop the Car everywhere and leave the Car at anytime. However, the system will end the ride (i.e. stop charging the User) only if he/she turns the engine off inside a Parking Area.

Domain Assumptions

12. When the User gets at least two Passengers, the corresponding discount on the User's fee will be applied only if the passengers stay together in the Car for more than 3 minutes.
13. When a User will park the Car inside a Parking Area, it will always correctly use one and only one free slot.
14. As soon as the Car battery status gets below 20% of the full capacity AND the Car isn't in a Charging Area AND the Car Status isn't *In Use* OR *Plugged*, there's always an Employee that immediately reaches the Car and recharges it on site; in the meanwhile the Car status is *Unavailable*.
15. When the Car is *In Use* and the battery charge level reaches the 0% of the full capacity the Car stops working and is immediately set as *Unavailable*. If the Car status is *Unavailable*, the Car will be reached by an Employee to consider if the Car needs to be taken in the Company's workshop for repairs or just needs to be recharged.
16. The Car has the ability to detect if it has been damaged.

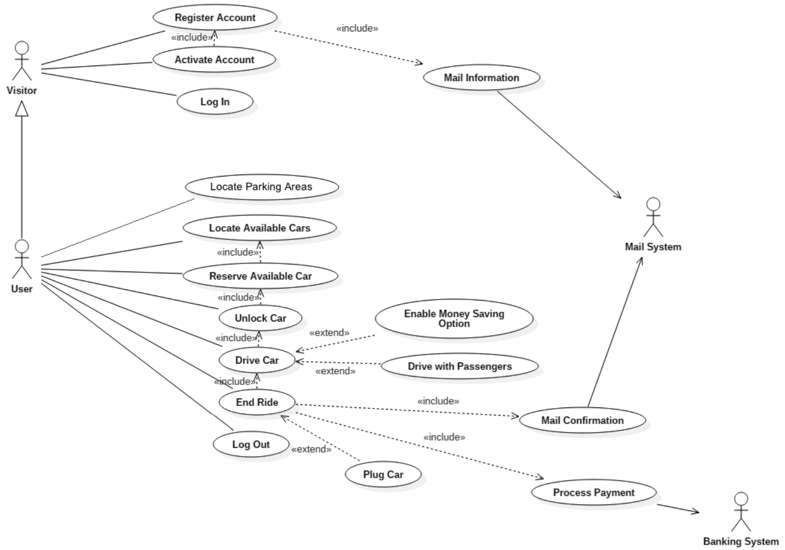
Domain Assumptions

17. If the Car status is *In Use* when a *Minor damage* is detected, the Car status will be set to *Unavailable* at the end of the ride; if a *Major damage* is detected the Car status is immediately set to *Unavailable*. In both cases an employee will reach the car and cope with the damages, deciding if the Car can be immediately used again (sets it to *Available*) or should be moved to the Company's workshop and/or if the User should pay for the damages.
18. A car which is *Available* or *Plugged* can be set as *Unavailable* in every moment by an Employee. This is done through another Company's System as it is not provided in this System.
19. A car which is *Unavailable* can be set to *Available* in every moment by an Employee. This is done through another Company's System and it is not provided in this System.

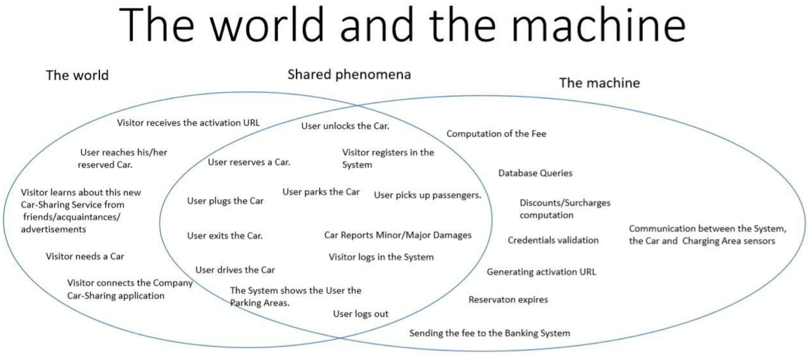
Domain Assumptions

20. If the Car has been left out of a Parking Area there will always be an employee which immediately reaches it, recharges it and move it to a Charging Area.
21. Every fine received by the Company for improper use of the Car will be managed by the Company.

Use Cases


Register Account
«include»
Activate Account
Log In
Visitor
«include»
Mail Information
Locate Parking Areas
Locate Available Cars
«include»
Reserve Available Car
«include»
Unlock Car
«include»
Drive Car
«include»
End Ride
Log Out
User
Enable Money Saving Option
«extend»
Drive with Passengers
«extend»
«include»
Mail Confirmation
«include»
«extend»
Plug Car
Process Payment
Mail System
Banking System

The world and the Machine



The world and the machine

The world

Shared phenomena

The machine

Visitor receives the activation URL

User reaches his/her reserved Car.

Visitor learns about this new Car-Sharing Service from friends/acquaintances/ advertisements

Visitor needs a Car

Visitor connects the Company Car-Sharing application

User unlocks the Car.

User reserves a Car.

Visitor registers in the System

User plugs the Car

User parks the Car

User picks up passengers.

User exits the Car.

Car Reports Minor/Major Damages

Visitor logs in the System

User drives the Car

The System shows the User the Parking Areas.

User logs out

Computation of the Fee

Database Queries

Discounts/Surcharges computation

Credentials validation

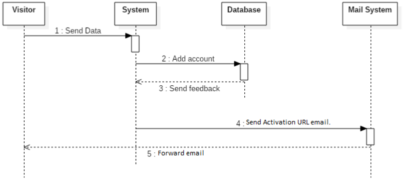
Communication between the System, the Car and Charging Area sensors

Generating activation URL

Reservaton expires

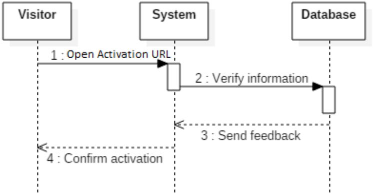
Sending the fee to the Banking System

Register Account (UC 1)



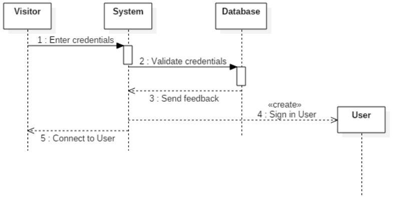
A Visitor wants to create an Account on the System, necessary to gain User privileges.

Activate Account (UC 2)



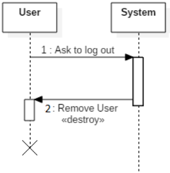
A Visitor receives an e-mail containing an activation link that will allow them to become a User.

Log In (UC 3)



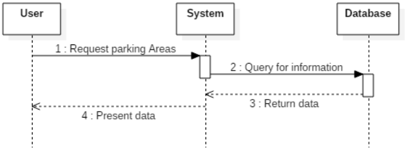
A Visitor accesses the System, gaining User privileges.

Log Out (UC 4)



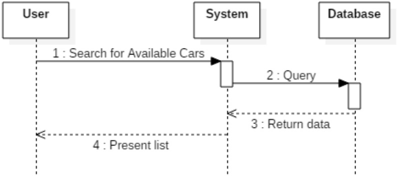
A User exits the System, becoming a Visitor and preventing themselves from using all the User-only functionalities.

Show Parking Areas (UC 5)



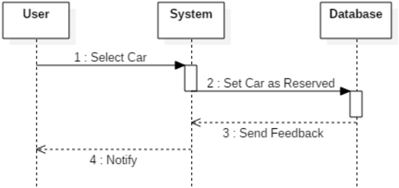
A User asks the System to show them a list of nearby Parking Areas.

Locate Available Cars (UC 6)



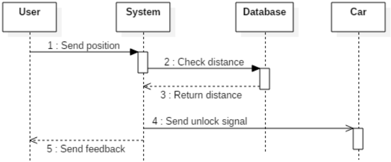
A User asks the System to show them a list of Available Cars near a position.

Reserve Car (UC 7)



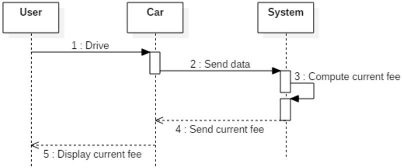
A User chooses a Car from those shown by UC 6, reserving it for themselves for a period of 1 hour.

Unlock Car (UC 8)



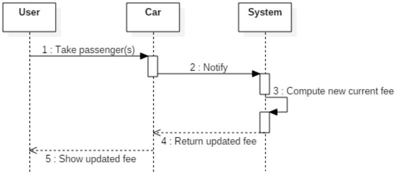
A User unlocks the Car they reserved through UC 7 when sufficiently close to it.

Drive Car (UC 9)



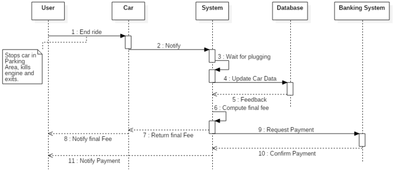
A User drives the Car they unlocked through UC 8, meanwhile the System keeps a timer on how long the Car has been rented.

Drive with Passengers (UC 10)



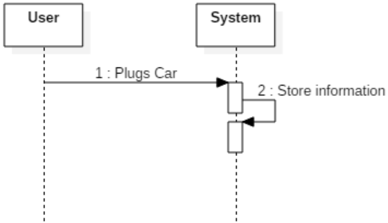
While driving (UC 9), the User shares the ride with a number of passengers.

End Ride (UC 11)



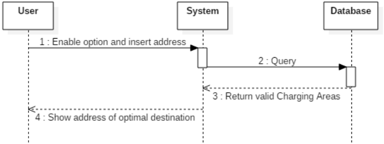
The User parks and exits the Car. The System calculates the final Fee and charges the User through the Banking System.

Plug Car (UC 12)



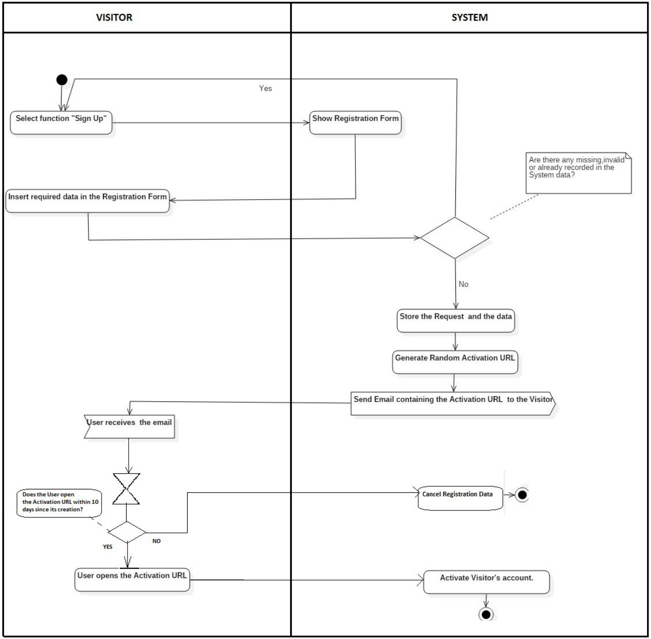
Immediately after the end of the ride (UC 11), the User plugs it into the socket of a Charging Area.

Enable Money Saving Option (UC 13)


User
System
Database
1 : Enable option and insert address
2 : Query
3 : Return valid Charging Areas
4 : Show address of optimal destination

The User inserts an address, receiving the address of a nearby Charging Station where to leave and Plug the Car (UC 11 and 12).

Activity Flow (Registration)



VISITOR

SYSTEM

Yes

Select function "Sign Up"

Show Registration Form

Are there any missing,invalid or already recorded in the System data?

Insert required data in the Registration Form

No

Store the Request and the data

Generate Random Activation URL

Send Email containing the Activation URL to the Visitor

User receives the email

Does the User open the Activation URL within 10 days since its creation?

Cancel Registration Data

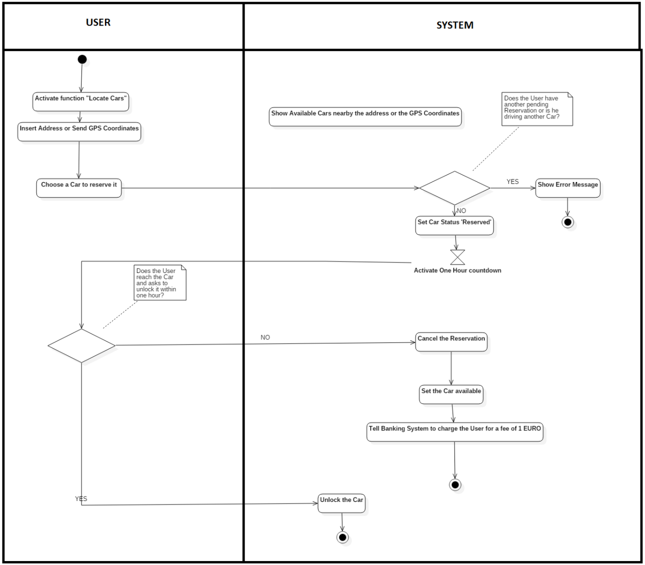
NO

YES

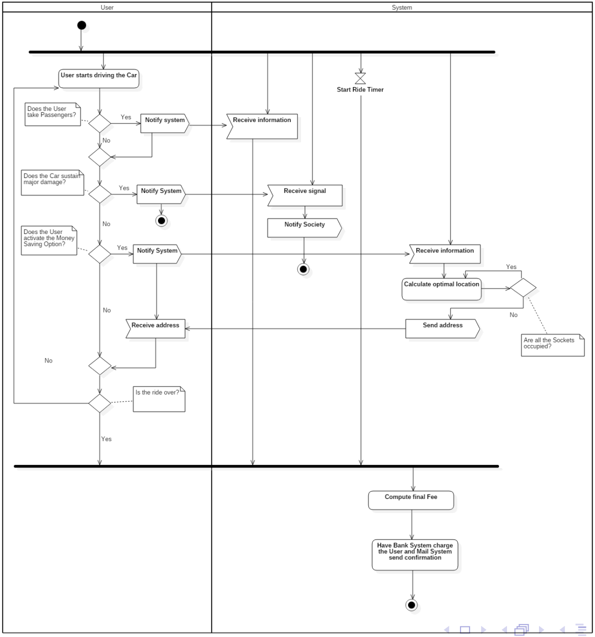
User opens the Activation URL

Activate Visitor's account.

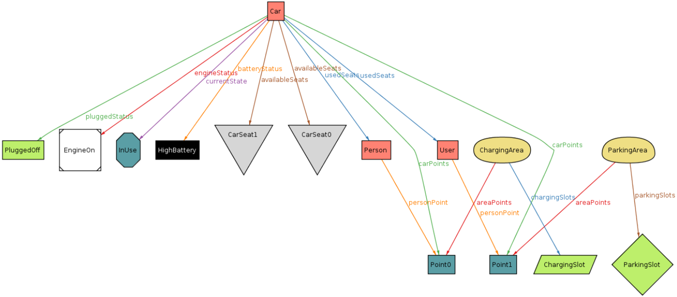
Activity Flow (Reservation)


USER
SYSTEM
Activate function "Locate Cars"
Insert Address or Send GPS Coordinates
Choose a Car to reserve it
Show Available Cars nearby the address or the GPS Coordinates
Does the User have another pending Reservation or is he driving another Car?
YES
Show Error Message
NO
Set Car Status 'Reserved'
Activate One Hour countdown
Does the User reach the Car and asks to unlock it within one hour?
NO
Cancel the Reservation
Set the Car available
Tell Banking System to charge the User for a fee of 1 EURO
YES
Unlock the Car

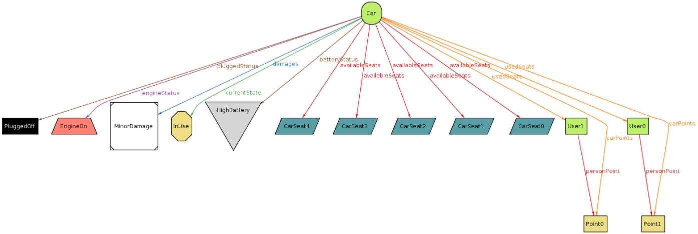
Activity Flow (Ride)


User
System
User starts driving the Car
Start Ride Timer
Does the User take Passengers?
Yes
Notify system
Receive information
No
Does the Car sustain major damage?
Yes
Notify System
Receive signal
Notify Society
No
Does the User activate the Money Saving Option?
Yes
Notify System
Receive information
Yes
Calculate optimal location
No
Are all the Sockets occupied?
Send address
Receive address
No
No
Is the ride over?
Yes
Compute final Fee
Have Bank System charge the User and Mail System send confirmation

Alloy representation of Areas



Alloy Representation of Cars



```
module GeoUtilities


sig Point {}
```

```
module Persons
open GeoUtilities

/**
  SIGNATURES
*/
sig Person {
  // We assume that each Person is identified by only one point
  personPoint: one Point
}
sig User extends Person {}

/**
  FACTS
*/
fact personPositionsDoNotOverlap {
  all disj p1, p2: Person | p1.personPoint ≠ p2.personPoint
}
```

```
/**
  ASSERTS
*/
assert allUsersHaveDifferentPositions {
  no disj p1, p2: Person | p1.personPoint = p2.personPoint
}
check allUsersHaveDifferentPositions for 10

/**
  PREDICATES/FUNCTIONS
*/
pred show() {
  #Person > 0
  #Point = #Person
}

run show for 25
```

```
module Cars
open util/boolean
open GeoUtilities
open Persons

/*
  SIGNATURES
*/
sig Car {
  batteryStatus: one BatteryStatus,
  availableSeats: some CarSeat,
  insidePersons: set Person,
  damages: set Damage,
  currentState: one CarState,
  pluggedStatus: one PluggedStatus,
  engineStatus: one EngineStatus,
  //This means that every car in every moment occupies a range of points
  carPoints: some Point
}
```

```
{
  #insidePersons ≤ #availableSeats
  insidePersons ≠ none implies currentState =
   InUse
  currentState ≠ none
  currentState ≠ InUse implies insidePersons =
   none
  currentState = InUse implies pluggedStatus =
   PluggedOff
  (currentState in Reserved + Available) implies
    batteryStatus = HighBattery
  currentState = InUse implies batteryStatus ≠
   ZeroBattery
  (batteryStatus = LowBattery and
    currentState ≠ InUse and
    pluggedStatus = PluggedOff) implies
    currentState = Unavailable
  engineStatus = EngineOn implies currentState =
    InUse
  currentState ≠ InUse implies engineStatus =
   EngineOff
}
```

```
abstract sig BatteryStatus {}
sig LowBattery, HighBattery extends BatteryStatus {}
sig ZeroBattery extends LowBattery{}

abstract sig EngineStatus {}
sig EngineOn, EngineOff extends EngineStatus {}

abstract sig PluggedStatus {}
sig PluggedOn, PluggedOff extends PluggedStatus {}

abstract sig CarState {}
sig Available, Unavailable, Reserved, InUse extends CarState {}

sig CarSeat {}

abstract sig Damage {}
sig MajorDamage, MinorDamage extends Damage {}
```

```
/*
  FACTS
*/
// Trivial relations
fact allEngineStatusAreAssociatedToSomeCar {
  all es: EngineStatus | es in Car.engineStatus
}

fact allPluggedStatusAreAssociatedToSomeCar {
  all ps: PluggedStatus |  ps in Car.pluggedStatus

}

fact allBatteryStatusMustBeAssociatedToSomeCar {
  all b: BatteryStatus | b in Car.batteryStatus
}

fact allCarStatesMustBeAssociatedToSomeCars {
  all cs: CarState | cs in Car.currentState
}
```

```
fact allCarSeatsMustBeAssociatedToOneCar {
  all cs: CarSeat | one c: Car | cs in c.availableSeats
}

fact damagesMustBeAssociatedToACar {
  all d: Damage | d in Car.damages
}


// Others
fact carsPositionsDoNotOverlap {
  all disj c1, c2: Car | c1.carPoints & c2.carPoints = none
}

fact personsAreNotUbiquituous {
  all disj c1, c2: Car | no p: Person | p in c1.insidePersons and p in c2.insidePersons
}
```

```
fact personsInUsedSeatsHaveSamePositionOfCar {
  all c: Car, p: Person | p in c.insidePersons iff p.personPoint in c.carPoints
}

fact majorDamagesImpliesUnavailableCars {
  all c: Car, m: MajorDamage | m in c.damages implies
     c.currentState = Unavailable
}


/**
  ASSERTS
*/
assert allCarsHaveDifferentPositions {
  all disj c1, c2: Car | no c1.carPoints & c2.carPoints
}
check allCarsHaveDifferentPositions for 10
```

```
assert allPersonsCantBeInDifferentCars {
  all disj c1, c2: Car | no p: Person |
    p in c1.insidePersons and p in c2.
   insidePersons
}
check allPersonsCantBeInDifferentCars for 10

assert allPersonsInACarMustHaveThatCarPosition {
  all p: Person, c: Car | p in c.insidePersons
   implies
    p.personPoint in c.carPoints
}

assert allMajorDamagedCarsAreUnavailable {
  all m: MajorDamage, c: Car | m in c.damages
   implies
    c.currentState = Unavailable
}
check allMajorDamagedCarsAreUnavailable for 10
```

```
assert
    allReservedOrAvailableCarsHaveHighBatteries {
  all c: Car | c.currentState in (Reserved +
    Available) implies
     c.batteryStatus = HighBattery
}
check
    allReservedOrAvailableCarsHaveHighBatteries
    for 3

assert noCarInUseHaveZeroBattery {
  no c: Car | c.currentState = InUse and c.
    batteryStatus = ZeroBattery
}
check noCarInUseHaveZeroBattery for 10

assert allCarWithUsedSeatsShouldBeInUse {
  all c: Car | c.insidePersons ≠ none implies c.
    currentState = InUse
}
check allCarWithUsedSeatsShouldBeInUse for 10
```

```
assert allCarsNotInUseAndNotPluggedAndWithLowBatteryShouldBeUnavailable {
  all c: Car | (c.batteryStatus = LowBattery and
    c.currentState ≠ InUse and
    c.pluggedStatus = PluggedOff) implies
    c.currentState = Unavailable
}
check allCarsNotInUseAndNotPluggedAndWithLowBatteryShouldBeUnavailable for 10

assert noPluggedCarIsInUse {
  all c: Car | c.currentState = InUse implies c.pluggedStatus = PluggedOff
}
check noPluggedCarIsInUse for 10
```

```
assert allEnginesOnAreAssociatedToInUseCars {
  all c: Car | c.engineStatus = EngineOn implies
    c.currentState = InUse
}
check allEnginesOnAreAssociatedToInUseCars for 3

assert allPersonsInsidesHaveSamePositionOfCars {
  all c: Car | c.insidePersons ≠ none implies
    c.insidePersons.personPoint in c.carPoints
}
check allUsedSeatsHaveSamePositionOfCars for 3


/*
  PREDICATES/FUNCTIONS
*/
pred show() {
  #Car > 0
}
run show for 3
```

```
// A car may be perfectly functioning but still unavailable (the external
// employee has manually set the status to Unavailable)
pred showCouldExistSomeUnavailableCarWithNoMajorDamageAndHighBattery {
  #Car > 0
  #Unavailable = #Car
  #MajorDamage = 0
  #LowBattery = 0
  #Person = 0
}
run showCouldExistSomeUnavailableCarWithNoMajorDamageAndHighBattery for 3
```

```
// A car may have minor damages but still
   available (the external
// employee has manually set the

status to Available)
pred
   showCouldExistSomeAvailableCarWithMinorDamages
    {
#MinorDamage = #Car
  #Available = #Car
}
run

showCouldExistSomeAvailableCarWithMinorDamages
   for 3
```

```
// It does mean that a User has turned the engine off outside a parking area
pred showCouldExistSomeInUseCarsWithEngineOff {
  #Car > 0
  #InUse = #Car
  #EngineOff = #Car
}
run showCouldExistSomeInUseCarsWithEngineOff for 3
```

```
// Same as before, all the people have left the car, even it is still in use
pred showCouldExistSomeInUseCarsWithEngineOnAndPersonsOutside {
  #Car > 0
  #InUse = #Car
  #EngineOn = #Car
  #Point = #Person
  #Person > #Car
}
run showCouldExistSomeInUseCarsWithEngineOnAndPersonsOutside for 3
```

```
// Not only users have access to the car. We
   ensure that a User reserve a Car,
// but we don't know if he/she will use it.
pred
   showCouldExistSomeInUseCarsWithAllSeatsOccupiedByN
    {
  #Car > 0
  #Person > 0
  #User = 0
}
run
   showCouldExistSomeInUseCarsWithAllSeatsOccupiedByN
    for 3

// Show that different people can be in the same
    car
pred showMorePersonsInOneCar {
  #Car.insidePersons > 1
  #Car = 1
}
run showMorePersonsInOneCar for 5
```

```
module Areas
open Cars
open GeoUtilities

/**
  SIGNATURES
*/
abstract sig CompanyCarSlot {}
sig ParkingSlot, ChargingSlot extends CompanyCarSlot {}

abstract sig CompanyArea {
  // We assume that a CompanyArea is composed by a non empty set of Points
  // This is enough for our modelation of the world
  areaPoints: some Point
}
```

```
sig ParkingArea extends CompanyArea {
  parkingSlots: set ParkingSlot,
  parkedCars: set Car
}
{
  #parkedCars ≤ #parkingSlots
}

sig ChargingArea extends ParkingArea {
  chargingSlots: some ChargingSlot,
  chargingCars: set Car
}
{
  #chargingCars ≤ #chargingSlots
  #parkedCars ≤ #parkingSlots
  // A car can't be charging and parked at the same time
  // because the two sets are disjoint
  chargingCars & parkedCars = none
}
```

```
/**
  FACTS
*/
// Trivial
fact parkingSlotsAreaAssociatedToExactlyOneArea
   {
  all ps: ParkingSlot | one pa: ParkingArea | ps
    in pa.parkingSlots
}

fact chargingSlotsAreaAssociatedToExactlyOneArea
    {
  all cs: ChargingSlot | one ca: ChargingArea |
   cs in ca.chargingSlots
}
```

```
// Areas do not overlap
fact
   areaPositionsAreAssociatedToExaxtlyOneCompany
    Area {
  all disj a1, a2: CompanyArea | a1.areaPoints &
    a2.areaPoints = none
}

// Parked Cars are in Parking Areas position
fact allParkedCarsAreInsideThoseAreaPositions {
  all pa: ParkingArea, c: Car | c in pa.
   parkedCars implies
    c.carPoints in pa.areaPoints
}

//Charging Cars are in Charging Areas position
fact allChargingCarsAreInsideThoseAreaPositions
   {
  all ca: ChargingArea, c: Car | c in ca.
   chargingCars implies
    c.carPoints in ca.areaPoints
}
```

```
// If a Car is inside an Area but not occupying a slot, it should be in use
fact allCarsInsideAreasButNotParkedOrChargingAreInUse {
  all c: Car |
    (c.carPoints in ParkingArea.areaPoints and
     c not in (ParkingArea.parkedCars + ChargingArea.chargingCars)) implies
     c.currentState = InUse
}
```

```
// I.e. a ParkingArea has always a
    parkingCapacity > 0
fact parkingCapacityZeroCanOnlyBeAssociatedTo
    ChargingArea {
  all p: ParkingArea | p.parkingSlots = none
    implies
     p in ChargingArea
}
```

```
// N.B. Implies and not Iff bcz a car in a ParkingArea can also be
// Unavailable (or Plugged) in a ChargingArea
fact carStateAvailableOrReservedImpliesCarAtOneParkingArea {
  all c: Car, pa: ParkingArea, ca: ChargingArea |
   (c.currentState = Available or c.currentState = Reserved) implies
   ( (c in pa.parkedCars and c.carPoints in pa.areaPoints) or
     (c in ca.parkedCars and c.carPoints in ca.areaPoints) or
     (c in ca.chargingCars and c.carPoints in ca.areaPoints))
}
```

```
// If a car is plugged ≤> it must be in one
    charging area
fact carStatePluggedIffCarInOneChargingCars {
  all c: Car | one ca: ChargingArea |
    c.pluggedStatus = PluggedOn iff c in ca.
   chargingCars
}

fact carParkedInOneParkingArea {
  all pa1, pa2: ParkingArea |
    (pa1 ≠ pa2 implies
      pa1.parkedCars & pa2.parkedCars = none)
}

fact carChargingInOneChargingArea {
  all ca1, ca2: ChargingArea |
    (ca1 ≠ ca2 implies
      ca1.chargingCars & ca2.chargingCars = none
   )
}
```

```
/**
  ASSERTS
*/
assert areaPositionsAreNotOverlapping {
  all disj ca1, ca2: CompanyArea | ca1.
    areaPoints & ca2.areaPoints = none
}
check areaPositionsAreNotOverlapping for 10

assert
    sameCarShouldNotBePluggedAtDifferentCharging
    Area {
  all c: Car | one ca: ChargingArea |
     c.pluggedStatus = PluggedOn iff
     c in ca.chargingCars
}
check
    sameCarShouldNotBePluggedAtDifferentCharging
    Area for 10
```

```
assert sameCarShouldNotBeParkedAtDifferentParkingArea {
  all disj p1, p2: ParkingArea | p1.parkedCars & p2.parkedCars = none
}
check sameCarShouldNotBeParkedAtDifferentParkingArea for 10

// Bcz we assume disjoint sets
assert sameCarShouldNotBeParkedAndChargingAtSameTime {
  no ParkingArea.parkedCars & ChargingArea.chargingCars
}
check sameCarShouldNotBeParkedAndChargingAtSameTime for 10
```

```
assert carsParkedOrChargingAreInsideThoseAreas
    Positions {
  all pa: ParkingArea, ca: ChargingArea |
     (pa.parkedCars.carPoints in pa.areaPoints)
    and
     (ca.chargingCars.carPoints in ca.areaPoints)
}
check carsParkedOrChargingAreInsideThoseAreas
    Positions for 10
```

```
/**
  PREDICATES/FUNCTIONS
*/
pred show() {
  all p: Point | p in Person.personPoint or p in CompanyArea.areaPoints or
    p in Car.carPoints
}

run show for 3
```

```
module CarUsageFunctions
open Cars
open Persons
open Time

/**
  SIGNATURES
*/
abstract sig CarUsageTimes {
  timeDatas: (User lone -> lone Car ) -> Time
}

one sig ReservationDataStartTime extends
   CarUsageTimes {}
{ User.(timeDatas.Time).currentState = Reserved
    }

one sig UsingDataStartTime extends CarUsageTimes
    {}
{ User.(timeDatas.Time).currentState = InUse  }
```

```
fact ifAUserIsInUsingSetItCantBeInReservedSetAndVicever {
  no
    (ReservationDataStartTime.timeDatas.Time).Car &
    (UsingDataStartTime.timeDatas.Time).Car
}

fact usersCantBeInReservingAndUsingCarDatas {
  no User.(UsingDataStartTime.timeDatas.Time) &
     User.(ReservationDataStartTime.timeDatas.Time)
}

fact oneUserOneCarForEachSet {
  all cut: CarUsageTimes, u: User, c: Car |
    lone (cut.timeDatas.Time).c and
    lone u.(cut.timeDatas.Time)
}
```

```
fact anUserCanBeInOnlyOneCarUsageTimes {
  all u: User | all disj t1, t2: Time  |
    no u.(CarUsageTimes.timeDatas.t1) &
          u.(CarUsageTimes.timeDatas.t2)
}


fact aCarCanBeInOnlyOneCarUsageTimes {
  all c: Car | all disj t1, t2: Time |
    no (CarUsageTimes.timeDatas.t1).c &
          (CarUsageTimes.timeDatas.t2).c
}

fact
    ifAUserIsInUsingSetItCantBeInReservedSetAndVicever
     {
  no
     (ReservationDataStartTime.timeDatas.Time).
    Car &
     (UsingDataStartTime.timeDatas.Time).Car
}
```

```
fact carsInUseInUsingDataStartTime {
  all c: Car |
    c.currentState = InUse iff
    c in User.(UsingDataStartTime.timeDatas.Time
   )
}


fact carsReservedInReservationDataStartTime {
  all c: Car |
    c in User.(ReservationDataStartTime.
   timeDatas.Time) iff
    c.currentState = Reserved
}
```

```
assert aCarInOnlyOneSet {
  all c: Car | all disj cut1, cut2:
     CarUsageTimes |
      no (cut1.timeDatas.Time).c & (cut2.timeDatas
     .Time).c
}
check aCarInOnlyOneSet for 5

assert aUserInOnlyOneSet {
  all u: User | all disj cut1, cut2:
     CarUsageTimes |
      no u.(cut1.timeDatas.Time) & u.(cut2.
     timeDatas.Time)
}
check aUserInOnlyOneSet for 5
```