POLITECNICO
MILANO 1863

A better Alloy world for PowerEnJoy

Daniele Riva[*] Marco Sartini[†]

February 8, 2017

version 1.0

[*]matr. 875154
[†]matr. 877979

Contents

1 Introduction

1.1 Purpose

This is an additional document to the project that we felt we were compelled to realize.

The *Alloy* world presented in the RASD is meaningful, but it did not take into account some relevant aspects. The old version was mainly focused on the world in an instant. This new version, instead, basically introduces the time dependence: only ended rentals and ended reservations now have bills (and possibly fees). We also added some states to increase understanding of the events.

1.2 Scope

This document is a part of the Software Engineering II project, which main purpose was to design a platform based on mobile and web application thought to offer a car sharing service with electrical powered cars called *Power EnJoy*.

1.3 Definitions, acronyms, abbreviations

RASD Requirements Analysis and Specifications Document;

Alloy is a language for describing structures and a tool for exploring them.

1.4 Reference documents

- RASD v1.1 available at `https://github.com/marcosartini/PowerEnJoy/blob/master/releases/rasdPowerEnJoy.pdf`

Revision history

Name	Date	Reason For Changes	Version
Marco e Daniele	08/02/2017	Final	1.0

2 Alloy model

2.1 The model

In this section we include the model definition in the *Alloy* language.

```
open util/boolean

/*one sig Company{
    cars: set Car,
    safeAreas: set SafeArea,
    users:set User // ?? Van messi ??
}

fact allUserBelongToCompany{
    no u: User | not (u in Company.users)
}


fact allCarsBelongToCompany{
    no c: Car | not (c in Company.cars)
}


fact allSafeAreaBelongToCompany{
    no s:SafeArea | not (s in Company.safeAreas)
}
*/

//POSITION

sig Position{
    latitude: Int, //should be float
    longitude: Int //should be float
}

//CURRENT TIME

one sig CurrentTime{
    time:Int
}{
    time>0
}

```

```
//Targa and BatteryLevel

sig Targa{}

abstract sig BatteryLevel{}
one sig BATTERYLOW extends BatteryLevel{} // means battery <=20 %
one sig BATTERYMEDIUM extends BatteryLevel{} // means battery >=20 %
    and battery <=50%
one sig BATTERYHIGH extends BatteryLevel{} // means battery >=50 %

// CAR

sig Car {
    targa: Targa,
    position:  Position,
    available: Bool,
    isBatteryCharging:Bool,
    battery: BatteryLevel,
    numberSeat: Int
}{
    numberSeat>=0 and numberSeat <5
}


fact noEqualTarga{
    no c1,c2:Car|( c1!=c2 and c1.targa=c2.targa )
}

// SAFE AREA

sig SafeArea{
    position: Position,
    isBusy: Bool
}


fact noDifferentAreaHasSamePosition
{
    no s1,s2:SafeArea  | (s1.position=s2.position and s1!=s2 )
}

fact noCarInSamePosition{
    no c1,c2:Car | (c1!=c2 and c1.position=c2.position)
}

fact areaIsFree{
    all s:SafeArea |s.isBusy=False iff(no c:Car |
   c.position=s.position)
}
```

```

fact areaIsBusy{
    all c:Car,s:SafeArea | c.position=s.position implies s.isBusy=True
}

fact noAreaBusyByCarOnRental{
    all rent:Rental|rent.state=ONGOING implies (no
   s:SafeArea|rent.car.position=s.position)
}

//PowerGridStation

sig PowerGridStation extends SafeArea{
    isChargingCar: Bool
}


fact carIsCharging{
    all c:Car,pgs:PowerGridStation |
    ( c.isBatteryCharging=True and c.position=pgs.position  )
   implies pgs.isChargingCar=True
}

fact PowerGridStationFreeNoCarInCharging{
    all p:PowerGridStation| p.isBusy=False implies
   p.isChargingCar=False
}

fact carIsChargingTwo{
    all c:Car,pgs:PowerGridStation |
    (pgs.isChargingCar=True  and c.position=pgs.position  ) implies
   c.isBatteryCharging=True
}

fact noCarIsChargingOutOfPGS{
    all c:Car | c.isBatteryCharging=False iff (no p:PowerGridStation
   | c.position=p.position )
}


// USER


sig DriverLicense{}
sig Password{}

sig User{
    driverLicense: DriverLicense,
    //password: Password,                no important to show in the
    complex actual alloy world
    signedIn: Bool// 0=no , 1 =yes
}

fact noSameUser{
    no u1,u2:User| u1.driverLicense=u2.driverLicense and u1!=u2
}

// No user has same password: commented to make more understable
    alloy world
/*fact noSamePassword{
    no u1,u2:User| u1.password=u2.password and u1!=u2
}
*/

// If rent is ongoing, user must be logged
fact noUserCanRentIfIsNotLogged{
    all r:Rental |r.state=ONGOING implies r.user.signedIn=True
}


// If reservation is ongoing, user must be logged
fact noUserCanReserveIfIsNotLogged{
    all r:Reservation |r.state=ONGOING implies r.user.signedIn=True
}

// State of Service where Service can be a Rental or a Reservation
abstract sig StateService{}
one sig ONGOING extends StateService{}
one sig ENDED extends StateService{}


// SERVICE

abstract sig Service{
    user:  User,
    car: Car,
    state:StateService,
    startTime:Int,
    endTime:Int
}{
    startTime>=0 and startTime<endTime
    endTime>0 and endTime<=CurrentTime.time
}

fact ServiceEnded{
    all s:Service|s.endTime<CurrentTime.time implies s.state=ENDED
}
```

```

fact ServiceOnGoing{
    all s:Service|s.endTime=CurrentTime.time implies s.state=ONGOING
}

fact carAreBeenUsing{
    all s:Service|s.state=ONGOING implies s.car.available=False
}

fact carAreAvailable{
    all c:Car|(no s:Service|s.state=ONGOING and s.car=c) implies
   c.available=True
}

fact noServiceOnGoinghasSameCarAndUser{
    all disjoint s1,s2:Service|(s1.state=ONGOING and s2.state=ONGOING)
                 implies (s1.car!=s2.car and s1.user!=s2.user)
}

fact noServiceHasSameCarOrSameUser{
    all disjoint
   s1,s2:Service|((s2.startTime<=s1.endTime)and(s1.startTime<=s2.endTime))
                 implies(s1.car!=s2.car and s1.user!=s2.user)
}


//RESERVATION

sig Reservation extends Service{
    expired: one Bool
}

fact reservationOnGoingNoExpired{
    all r:Reservation|r.state=ONGOING implies r.expired=False
}

// RENTAL

sig Rental extends Service{
    numberPassenger: one Int,
    remainingBattery:BatteryLevel,
    leftCarInCharging:Bool
}{
    numberPassenger>=0 and numberPassenger<car.numberSeat
}

fact noRentalOnGoingHasCarParking{
    all r:Rental|r.state=ONGOING implies(no
   safe:SafeArea|r.car.position=safe.position)
}

fact batteryEqualInOnGoingRental{
    all r:Rental|r.state=ONGOING implies
   r.car.battery=r.remainingBattery
}

fact noCarInChargingInvolvedInOnGoingRental{
    all rent:Rental|rent.state=ONGOING implies
   rent.leftCarInCharging=False
}

fact fromReservationToRental{
    all res:Reservation|(res.expired=False)implies(one
   rent:Rental|res.endTime=rent.startTime and res.car=rent.car and
   res.user=rent.user)
}


// PAYMENT, FEE, BILLS, DISCOUNT and OVERCHARGE DA
   AGGIUNGERE/MODIFICARE

abstract sig  Payment{
    amount: one Int
}

// FEE RESERVATION

sig Fee extends Payment{
    reservation:Reservation
}{
    amount=1
}

fact noFeeAtOnGoingReservation{
    all res:Reservation|res.state=ONGOING implies (no
   fee:Fee|fee.reservation=res)
}

fact noFeeAtNormalEndedReservation{
    all res:Reservation|(res.state=ENDED and res.expired=False)
   implies (no fee:Fee|fee.reservation=res)
}

fact feeAtExpiredReservation{
    all res:Reservation|(res.state=ENDED and res.expired=True)
   implies (one fee:Fee|fee.reservation=res)
}

```

```
fact oneFeeOneReservation{
    all disjoint f1,f2:Fee|f1.reservation!=f2.reservation
}

// BILL

sig Bill extends Payment{
    discount:Discount,
    rental:Rental,
    overcharge:OverCharge
}{
    amount>0
}

fact billAtEndedRental{
    all rent:Rental|rent.state=ENDED implies (one
   b:Bill|b.rental=rent)
}

fact  noBillOnGoingRental{
    all rent:Rental|rent.state=ONGOING implies (no
   b:Bill|b.rental=rent)
}

fact oneBillOneRental{
    all disjoint b1,b2:Bill|b1.rental!=b2.rental
}

// DISCOUNT AND OVERCHARGE

abstract sig Discount{}

abstract sig OverCharge{}

one sig ThirtyIncrement extends OverCharge{}

one sig NoIncrement extends OverCharge{}

one sig NoDiscount extends Discount{}

one sig TenPerHundredDiscount extends Discount{}

sig TwentyPerHundredDiscount extends Discount{}

one sig ThirtyPerHundredDiscount extends Discount{}

// DISCOUNT AND OVERCHARGE CRITERIA

fact noDiscountForUser{
    all bill:Bill |
     (bill.rental.numberPassenger<2 and
    bill.rental.remainingBattery!=BATTERYHIGH and
    bill.rental.leftCarInCharging=False)
     implies bill.discount=NoDiscount
}

fact discountPassenger{
     all bill:Bill |
     (bill.rental.numberPassenger>=2 and
    bill.rental.remainingBattery!=BATTERYHIGH and
    bill.rental.leftCarInCharging=False)
     implies bill.discount=TenPerHundredDiscount
}

fact discountBattery{
     all bill:Bill |
     (bill.rental.remainingBattery=BATTERYHIGH and
    bill.rental.leftCarInCharging=False)  implies
    bill.discount=TwentyPerHundredDiscount
}

fact discountCharging{
     all bill:Bill | bill.rental.leftCarInCharging=True implies
    bill.discount=ThirtyPerHundredDiscount
}

fact payOvercharge{
     all bill:Bill |( bill.rental.remainingBattery=BATTERYLOW //or
    noPwgNear[rental.car]
     ) implies bill.overcharge=ThirtyIncrement
}


fact payNoOvercharge{
     all bill:Bill |( bill.rental.remainingBattery!=BATTERYLOW //or
    noPwgNear[rental.car]
     ) implies bill.overcharge=NoIncrement
}



// PRED

pred userNotRentOnGoing[u:User,c:Car]
{
     all rental:Rental | rental.state=ONGOING implies rental.user!=u
    and rental.car!=c
}
```

```
pred userNotReservationOnGoing[u:User,c:Car]
{
    all res:Reservation | res.state=ONGOING implies res.user!=u and
   res.car!=c
}

pred userCanChooseAservice[u:User,c:Car]
{
    userNotRentOnGoing[u,c] and userNotReservationOnGoing[u,c]
}

// ASSERTS

assert allCarOnRentalAreUnavailable{
    all rental:Rental |rental.state=ONGOING implies
   rental.car.available=False
}

assert allCarReservedAreUnavailable{
    all reserve:Reservation |reserve.state=ONGOING implies
   reserve.car.available=False
}


// RUN

run {} for 5 but exactly 3 Reservation,exactly 3 Rental

run userNotRentOnGoing
run userNotReservationOnGoing
run userCanChooseAservice

check allCarOnRentalAreUnavailable
check allCarReservedAreUnavailable
```

2.2 Results

In this section we include the results of the executions generated by the *Alloy Analyzer* tool.

2.2.1 Run and check

Executing ”Run run$1 for 5 but exactly 3 Reservation, exactly 3 Rental” # Instance. found. Predicate is consistent

Executing ”Run userNotRentOnGoing” # Instance. found. Predicate is consistent

Executing ”Run userNotReservationOnGoing” # Instance. found. Predicate is consistent

Executing ”Run userCanChooseAservice” # Instance. found. Predicate is consistent

Executing ”Check allCarOnRentalAreUnavailable” No counterexample found. Assertion may be valid.

Executing ”Check allCarReservedAreUnavailable” No counterexample found. Assertion may be valid.

2.2.2 Visual worlds

Two different worlds, generated by the *Alloy* tool.

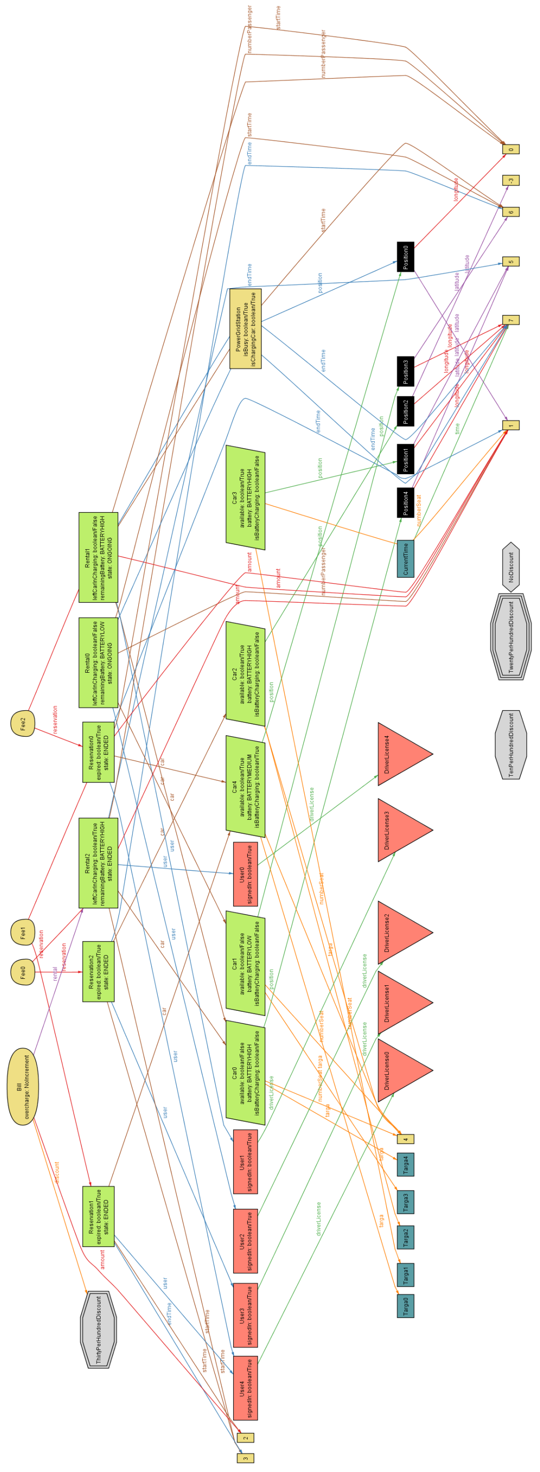

Figure 2.1: First Alloy world, in particular with all the reservations expired

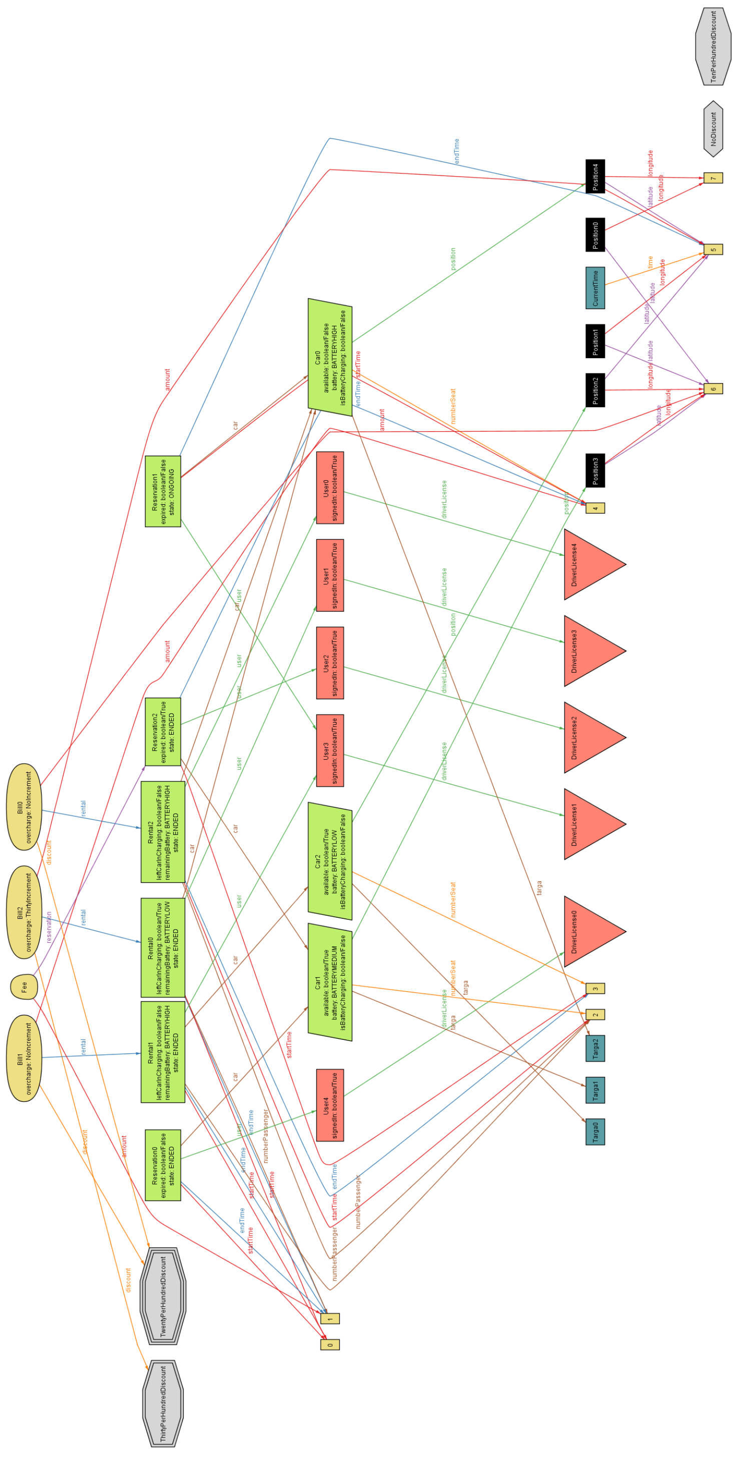

Figure 2.2: Second Alloy world