eMall: e-Mobility for All

Nicolas Benatti (10668784)
Lorenzo Biasiolo (10629367)
Melissande Simonin (10918410)

Requirements Analysis and Specification Document (RASD)
Version 1.0, 23 December 2022

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

This document is intended to guide the development process for *eMall*: an e-mobility platform which allows users to rent electric vehicles and plan charges in order not to interfere with their daily schedule. Smart charging stations allow to automatically decide from which provider to fetch energy, based on the selling price. Furthermore, the user can be notified when it's time for a charge and be driven to the best station thanks to apposite algorithms.

This specification will address problem scope, goals and constraints, but also give an abstract (i.e non-architectural) view of the system, as long as indicating the intended usecases.

1.1.1 Goals

identifier	description
G1	Allow the user to obtain the list of closer stations, their cost and special offers
G2	Allow the user to book a charge in a specific station for a certain amount of time.
G3	Allow the user to start a process at a charging station.
G4	Allow the application to notify the user when the charge process is over.
G5	Allow the user to pay for the charge.
G6	The application must allow to suggest to the user when a charge is needed (based on the status of the car and the user schedule).

1.2 Scope

1.2.1 World Phenomena

Identifier	**Description**
WP1	Road traffic
WP2	EV moves
WP3	EV battery discharges

1.2.2 Shared Phenomena

Identifier	Description
SP1	Vehicle comes to a charging station
SP2	Update on battery charge status
SP3	User books a charge
SP4	User pays for a charge
SP5	User searches for stations
SP6	User registers an account
SP7	User inserts vehicle data in-app
SP8	Energy price update from DSOs
SP9	User gets notified about the need for a charge
SP10	Charging station communicates to the user that the battery is full

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Identifier	**Description**
Charging Station	Set of electrical sockets deployed on the same site and managed by the same CPO
Energy Source	Batteries installed at the charging station or energy acquired from DSO
Reservation	the act of booking a charge in a specific charging Station for a certain timeframe
Energy Source	Can be either DSO-supplied electricity or local batteries inside the charging station

1.3.2 Acronyms

Acronym	**Description**
API	Application Programming Interface
CPMS	Charge Point Managemenet System
CPO	Charging Point Operators
DSO	Distribution System Operator
eMSP	e-Mobility Service Provider

1.3.3 Abbreviations

Abbreviation	**Description**
RASD	Requirements Analysis and Specification Document
WP	World Phenomena
SP	Shared Phenomena
Gn	Goal no. n
Dn	Domain Assumption no. n
eMall	e-Mobility for All
EV	Electric Vehicle
CS	Charging Station

1.4 Revision history

- **23/12/2022**: 1st version

1.5 Reference Documents

The specification document "Assignment RDD AY 2022-2023_v2.pdf".

1.6 Document structure

This document is composed of six sections, detailed below. In the first section the problem is introduced together with the associated goals of the project. Additionally the scope of the project is specified along with the various phenomena occurring. Lastly, the necessary information to read the report is presented, such as definitions and abbreviations. Section two contains an overall description of the system, including a detailing of its users and main functions. Moreover there is the class diagram, descriptions of several scenarios, some statecharts and finally the domain assumptions made in this report. In section three the requirements on the system is specified. This includes func- tional requirements, non-functional requirements and requirements on external interfaces. Furthermore use cases are described, with accompanying use case and sequence diagrams. Section three also contains mappings of functional re- quirements to the goals of the system, and to the use cases. Section four contains a formal analysis with the help of Alloy. Together with the Alloy code, the analysis objective is described. In section five there is a presentation of the project members total effort spent. Section six contains the references used.

2 Overall Description

2.1 Product perspective

2.1.1 Scenarios

1. **Driver wants to start using the application**
   The Driver Paolo, after having bought an electric car, decides that he wants to register to e-Mall to keep monitored his car and to always know where are the charging stations. He download the application on the phone, he launches it, he click on the button 'sign up', he insert all the requested informations and he can use E-Mall. The Driver Luca had already used the application but, for some reasons, he hasn't E-Mall on the phone anymore. He decides to download it another time, he launches it, he click on the button 'sign in', he inserts the password and the username and he is able to use the application again.

2. **User's car is running out of battery**
   After a long trip the driver Marco is coming back home. He hasn't noticed that his car hasn't enough battery to arriving back home. At this point the application (e-Mall), knowing Marco's schedule and after having calculated that his battery level must be low, send to Marco's phone a notification. On it is written 'low charge'. Thanks to internet connection Marco can receive the notification and, after having opened it he can read on the notification some of special offers made available by CPOs. He finds a charging station and he can arrive at home.

3. **A Driver wants to find a charging station**
   Mrs. Smith is driving in a city that offers e-Mobility Service Providers, and she wants to charge her car before going on a trip to meet her daughter. She therefore connects to the application, and obtains the list of available charging stations closest to her current position. For each charging station, she can see the cost of charging, and the special offers they may offer.

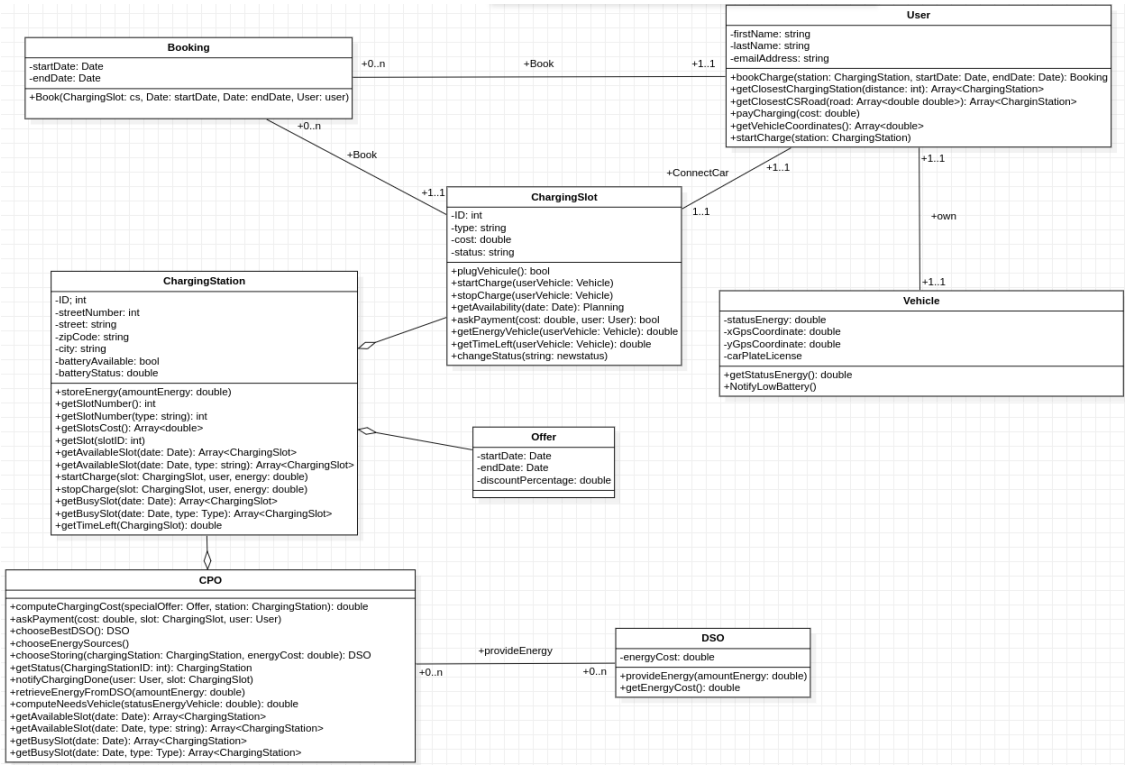
4. **A Driver wants to book a charge**
   Lorenzo knows the city, and he wants to reserve the nearest charging station on his way home, as he is used to do. After finishing his work, he connects to the application, selects the desired charging station, and he books it to recharge his electric car just before arriving home. By doing this, he can choose the exact time of his charge so as not to waste time on his schedule.

5. **A Driver starts the charging process**
   Upon arriving at the charging station, Joe plugs in his car, and the service retrieves the information from it. This way, Joe knows how long he has to wait, and can let the service charge his car independently. Thus, he can leave the war and have a coffee nearby, in order to continue to work on his file.

6. **A Driver has finished the charging process**
   When the charging of Mr. Camozzi's vehicle is complete, e-Mall sends a notification to his phone to warn him that his car battery is now full. The e-Mall service compute the amount of money the user will have to pay, and it adds the special offers if any are present. Mr Camozzi, after returning to the charging station, has to pay, disconnect his vehicle, and then can go on his trip.

7. **A Driver wants to pay**
   Beatrice is to a charging station and has just finished to charge her car. She has recived the notification 'full battery'. She clicks on the button: 'Pay' and she reads the amount. She selects the bank account details to use to pay and the application proceeds to withdraw the money. If the amount of money on the bank account aren't enough Beatrice needs to insert another bank account where to make the application withdraw the money.

2.1.2 Class diagram



The main getter and setter are written because we wanted to highlight them, but we assume that they are all present.

2.1.3 Statecharts

In this section we present statecharts for the most complex aspects of the system. The most trivial ones have been left out for the sake of brevity.

Figure 1: State diagram for the booking process

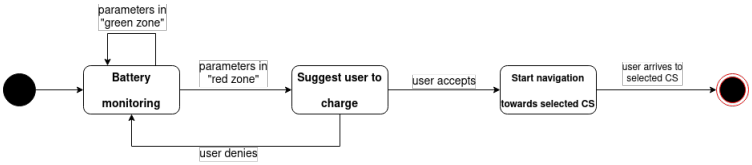


Figure 2: State diagram for the charge-recommendation process

2.2 Product functions

Here we present the main functions of e-Mall in more detail.

2.2.1 Get information about nearby stations

Drivers can know about what are the stations within a specified distance from their position. For each of them, the system will show general information (number of available sockets, their type and, if all are occupied, estimated time before one of each type becomes free) as well as cost of electricity and any special offers applied.

2.2.2 Charge booking system

The system allows the driver to book charges by specifying both the station and the timeframe at which the charge will take place.

2.2.3 Perform a charge at a certain station

The main functionality of the system is to allow Drivers to correctly perform a charge and pay for it, notifying the user when it ends. Users are not forced to book in order to accomplish this

2.2.4 Real-time car status monitoring

eMall monitors the car's battery status in real-time both in normal usage conditions and during charge. In the latter case, the system can also show the remaining time before the charge ends.

2.2.5 Optimal *Time-to-Charge* suggestion

An advanced feature of the eMSP software is the ability to proactively suggest the user when it's time to charge the vehicle, based on the daily schedule of activities, proximity to charging stations, availability of free sockets, battery status and special offers. To take advantage of this feature the user will have to give access permission to both their calendar and navigation system (in order to guide him towards the better CS).

2.3 User characteristics

There is a single user considered in the e-Mall system.

1. **Driver**
   A registered user who drives an electric car and wants to use the system in order to use his electric car at best, to pay as less as possible to mantain it and to reduce the wasted time in their daily schedule. With the application, in fact, he is able to keep checked the level of the battery and to always know where the charging stations are located. He is also able to know where the best offers are and to pay with application.

2.4 Assumptions, dependencies and constraints

2.4.1 Domain assumptions

Identifier	Description
D1	All the charging stations send their information in real time (GPS coordinates, amount of energy available, cost, special offer, ...)
D2	The users provide the right information about their vehicles
D3	The users have enough time to plan a charge in their daily schedule
D4	The charging stations have the required hardware capabilities to provide enough energy to the connected vehicle
D5	The batteries of charging stations are functional
D6	The charging stations and has the required sensors to get battery status during charge
D7	The charging stations know information about energy cost from the energy providers through external APIs
D8	The charging stations have both a POS and a cash manager
D9	The charging stations can connect to the most common payment gateways through apposite APIs
D10	There exists protocols and APIs through which the CPMS can communicate with eMSP, DSOs and CSs
D11	The eMPS software communicates with the charging station (and vice-versa) via CPMS, which offers APIs to do so
D12	The Vehicle is able to communicate with the CS via apposite protocols
D13	CPO operators can access the CPMS through a GUI, either on phone, tablet, or dedicated computers

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The eMSP client software will be available both as a webapp and as a mobile application. It should support all the most used browser and mobile operating systems and be easy-to-use.

3.1.2 Hardware Interfaces

The only hardware interface required to use the application is a PC / smartphone. Instead, the interface with the whole network of charging stations is also the CPMS control panel and the power outlets.

3.1.3 Software Interfaces

All the components communicate using uniform APIs to avoid integration issues. external APIs are needed to communicate with various CPOs and DSOs, as well as to contact payment gateways. the CPMS offers the following services through its API:

- **Get "external" status of a charging station**: provides the number of charging sockets available, their type (slow/fast/rapid), their cost, and, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type becomes free.
- **Start charging a vehicle**: tells the CS to start the charging process at the selected socket. During the charging process, communication between EV and CS allows to know (infer) when the battery is full.
- **Get "internal" status of a charging station**: provides the amount of energy of internal batteries, number of in-charge EVs and estimated time before end of charge for each of them.
- **Get the current price of energy from DSOs**
- **Decide from which DSO to fetch energy**: provides an interface for dynamically deciding the best DSO from which to acquire energy, based on price and what's the current mix of energy sources being used.
- **Decide the optimal energy mix**: computes the optimal combination of DSO-supplied energy and local batteries supply.

3.1.4 Communication Interfaces

The communication between the various components of the infrastructure is carried out through the Internet. There's a variety of protocols in use: `OCPI` (between eMSP and CPMS), `OCPP` (between CPMS and the charging station).

Communication between EV and the charging station happens via the `DIN SPEC 70121` protocol.

3.2 Functional Requirements

Identifier	Description
R1	The system must allow an unregistered user to register by providing his personal information (e-mail, first name, last name)
R2	The system must allow a registered user to log in using the credentials input at registration time
R3	The system must allow a registered user to connect his car
R4	The system must ask the user for permission to access his location data
R5	The system must allow the user to retrieve external information on charging stations within a certain distance from their currint location, such as: location, cost, special offer, number of charging sockets available, their type, and if they are all occupied, the amount of time until the first socket of each type is freed
R6	The system must ask the user for permission to access his schedule on phone's calendar
R7	The system must allow a registered user to choose a charging station and book a charging slot according to his schedule
R8	The system must allow a registered user to connect his car to the charging slot booked and start the charge
R9	The system must allow a registered user to connect his car to a free charging slot even if they didn't book it, and start the charge
R10	The system must be able to retrieve internal information about charging stations (amount of energy available in their batteries (if any), number of vehicles currently charged, and the amount of power absorbed and the time left to the end of the charge of each plugged vehicle)
R11	The system must be able to deliver energy to the connected vehicle as soon as it is plugged-in
R12	The system must be able to stop the charge when the connected vehicle battery is full

R13	The system must notify the user when the charging process is finished
R14	The system must allow a user to pay for the charge
R15	The system must be able to signal the user in the event of an exception
R16	The system must be able to retrieve in real time the battery level of the user's vehicle
R17	The system must be able to proactively suggest the user to charge their vehicle depending on his schedule, battery conditions and available stations
R18	The system must be able to get information from DSOs (current price of energy) if available
R19	The system must be able to accumulate energy in charging station's batteries (if any) when the price is adequately low
R20	The system must be able to choose the cheapest way of acquiring energy during charge(from charging station battery or DSO), according to availability and cost
R21	The system must be able to access phone's navigation system

3.2.1 Mapping on Goals

Goal	**Domain assumptions**	**Requirements**
G1	D1, D3, D7, D10	R1, R2, R3, R4, R5, R10, R15 [, R18]
G2	D1, D2, D3, D13	R1, R2, R3, R6, R7, R14, R15
G3	D1, D4, D5, D6, D12, D13	R1, R2, R3, R8, R9, R10, R11, R15, R16 [, R19], R20
G4	D10, D11, D12, D13	R1, R2, R3, R12, R13, R15, R16
G5	D7, D8, D9, D13	R1, R2, R14, R15, R18
G6	D1, D10, D13	R1, R2, R3, R4, R5, R6, R15, R16, R17, R21

3.2.2 Use cases

1. **User registration**

Actors	User
Entry conditions	The driver is on the initial view of the system and hasn't got any account
Event flow	1. User presses on "Sign up" button 2. User inserts personal data and vehicle data 3. User presses on "Confirm" button 4. eMall processes the request and adds the user to its database
Exit condition	Registration is completed and successfully notified to the user
Exceptions	• Non-existent vehicle license plate • Missing mandatory data upon form submitting eMall will signal an error to the user in any of these cases

2. **User login to eMall**

Actors	User
Entry conditions	The user is registered and on the initial view of the system
Event flow	1. User presses on "Sign in" button 2. User inserts email and password 3. User presses on "confirm" button 4. eMall processes the request, displays a success message and brings the user to the main page
Exit condition	A successful login has been performed
Exceptions	Missing or wrong data

3. **User books a charge**

Actors	User
Entry conditions	The user is registered, logged in and on the CS view page
Event flow	1. User presses on "book charge" button 2. User selects a time range 3. User presses on "confirm" button 4. eMall processes the request and confirms the booking
Exit condition	A successful booking operation has been performed and notified to the user
Exceptions	User selects an already-taken timespan for that station

4. **Get information about nearby stations**

Actors	User
Entry conditions	The user is registered and logged in
Event flow	1. User presses "see nearby stations" button 2. User selects a distance from their current position 3. User selects a CS from the available results
Exit condition	User correctly lands on the CS info page
Exceptions	No GPS granted by User

5. **Access to calendar and navigation system**

Actors	User
Entry conditions	The user is registered and logged in
Event flow	1. eMall asks the permission to access User's calendar and navigation system User accept or reject the proposition
Exit condition	Both permissions are granted
Exceptions	• User doesn't grant any permission, or just 1 • Any of the 2 services is not present on User's phone (e.g. there are no calendar apps installed) eMall will signal an error to the user in any of these cases

6. **User pays for a charge**

Actors	User
Entry conditions	The user is registered, logged in and has completed a charge
Event flow	1. User presses "Pay for the charge" button 2. User connects to its payment gateway and proceeds with payment 3. eMall processes the request and confirms the payment
Exit condition	A successful payment operation has been performed and notified to the user
Exceptions	External errors during payment, invalid credit card data

7. **Set the cost of a charge**

Actors	CPO
Entry conditions	CPO recives a request to set a new price for CS
Event flow	1. the CPO sends to CS its new price 2. the CPO receives a confirmation message
Exit condition	The new price is correctly setted and it is visible from the application
Exceptions	the CPO doesn't receive the confirmation message at point b

8. **Notify user when a charge ends**

Actors	User, CPO
Entry conditions	User is registered, logged in and has a charge in process
Event flow	1. CPMS infers that the battery is full (through communication with the vehicle) 2. CPMS notifies eMall of the new event 3. eMall sends a notification to the user 4. User presses on "confirm" button
Exit condition	The charging process is finished and the notification has been correctly sent
Exceptions	communication errors

9. **Store energy**

Actors	DSO, CS
Entry conditions	CPO can get information from to at least one DSO, and can get information from its Charging Station
Event flow	1. CPMS get the internal battery level of the CS 2. CPMS get energy price from the DSO(s) 3. CPMS compute the best option (store, only use energy for charge or ignore) 4. CPMS store energy or not according to the last condition
Exit condition	Energy is correctly stored in batteries if and only if the system decided to do so, otherwise nothing happens
Exceptions	Communication errors

10. **Acquire energy**

Actors	DSO, CS, CPO
Entry conditions	CPMS has choosen the most interesting DSO (in term of price)
Event flow	1. DSO provide energy to the CPO 2. CPO notify the availability of energy to the CPMS 3. CS acquire the needed energy to charge the vehicles / store energy
Exit condition	CS doesn't need energy anymore, nor for charging, nor for storing.
Exceptions	communication errors

11. **Provide energy price**

Actors	DSO
Entry conditions	CPO is linked to at least one DSO
Event flow	1. CPMS gets price of energy from the DSO 2. DSO returns current price to the CPMS
Exit condition	The price of energy has been correctly sent to the CPMS
Exceptions	Communication errors

12. **Get internal status of a CS**

Actors	User
Entry conditions	The user is registered, logged in and on the CS view page
Event flow	1. The user click on the button 'search the state of a CS' 2. The user selects the CS that he wants to see the status of
Exit condition	The user can see the internal status of a CS (battery level, connected vehicles, time left to charge, ...)
Exceptions	CS not found (invalid ID)

13. **Charge the vehicle**

Actors	User, CS, Vehicle
Entry conditions	User is registered and logged in
Event flow	1. User Requests a charge 2. eMall informs (or reminds, in the case a booking had previously been performed) the User of the slot he/she has been assigned to 3. User plugs the vehicle in
Exit condition	The `StateOfCharge` signal tells the CS that the battery is full, thus terminating the charging process, which is correctly notified to the User
Exceptions	The time slot does not match the one specified in the booking (if the user booked the charge). eMall will signal an error to the user in any of these cases

14. **Provide vehicle status while charging**

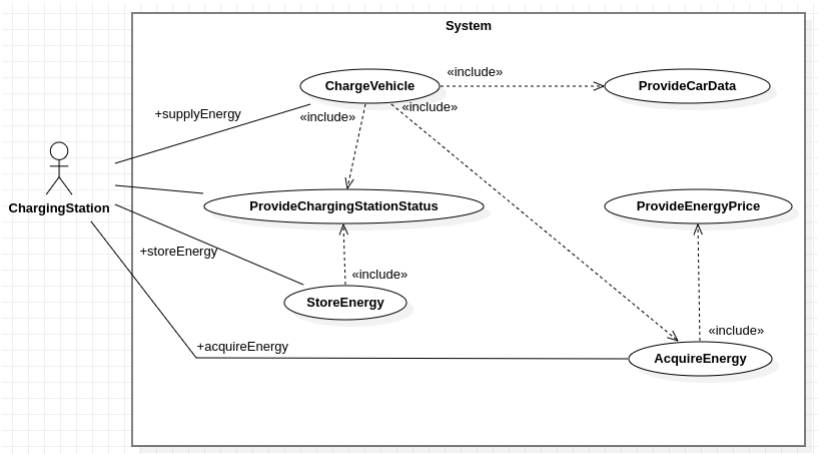
Actors	User, CPO
Entry conditions	The user is registered, logged in and on the Charge Status page
Event flow	1. The user sends a request to the application to know the charging status 2. The user selects the CS where his vehicle is connected between a list of connected vehicle 3. The CPO asks to the CS the status of the charging and sends it to user
Exit condition	The User receives the percentage of the charging level of his vehicle
Exceptions	The selected socket is connected with another vehicle

15. **Set special offer**

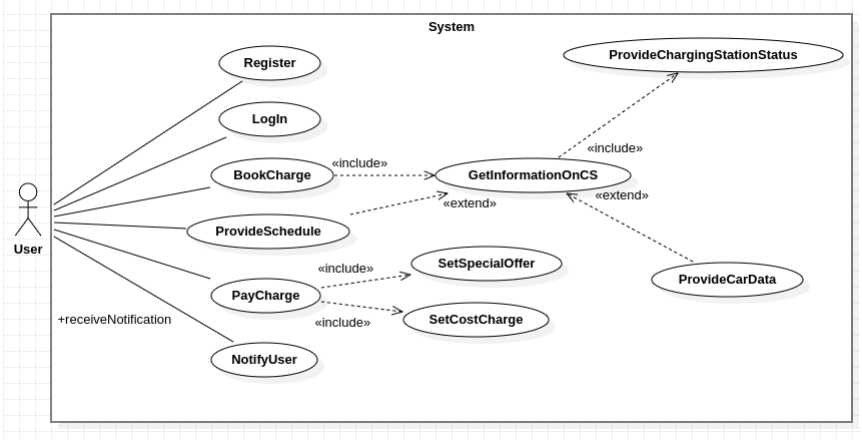
Actors	CPO
Entry conditions	CPO receives a request to set a special offer for CS
Event flow	1. CPO sends to CS the new special offer 2. CPO recives a confirmation message 3. CPO propose this special offer to any user
Exit condition	The special offer is correctly setted and the price is visible from the application
Exceptions	the CPO not receives the confirmation message at point 2

3.2.3 Use case diagrams

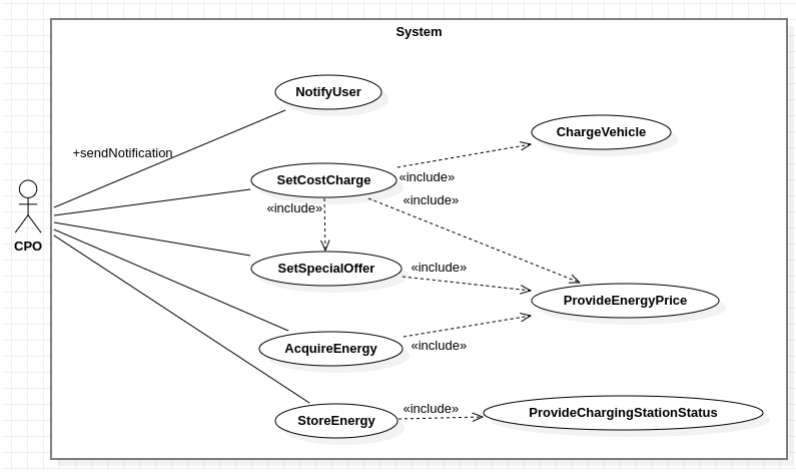
1. **Charging station**



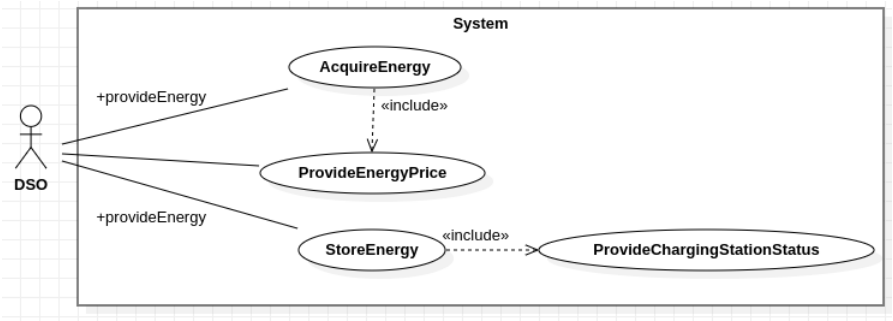
2. **User**



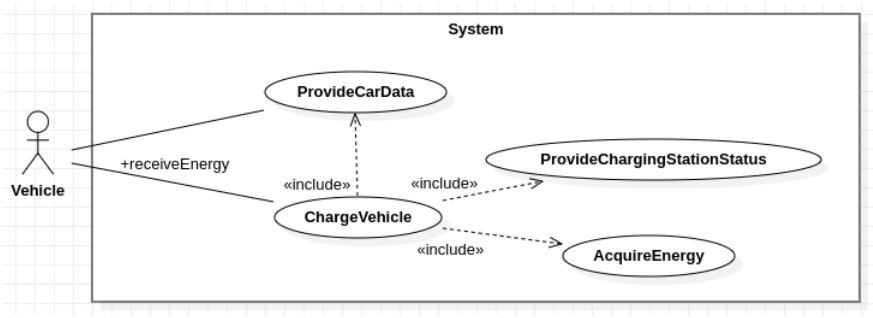
3. **CPO**



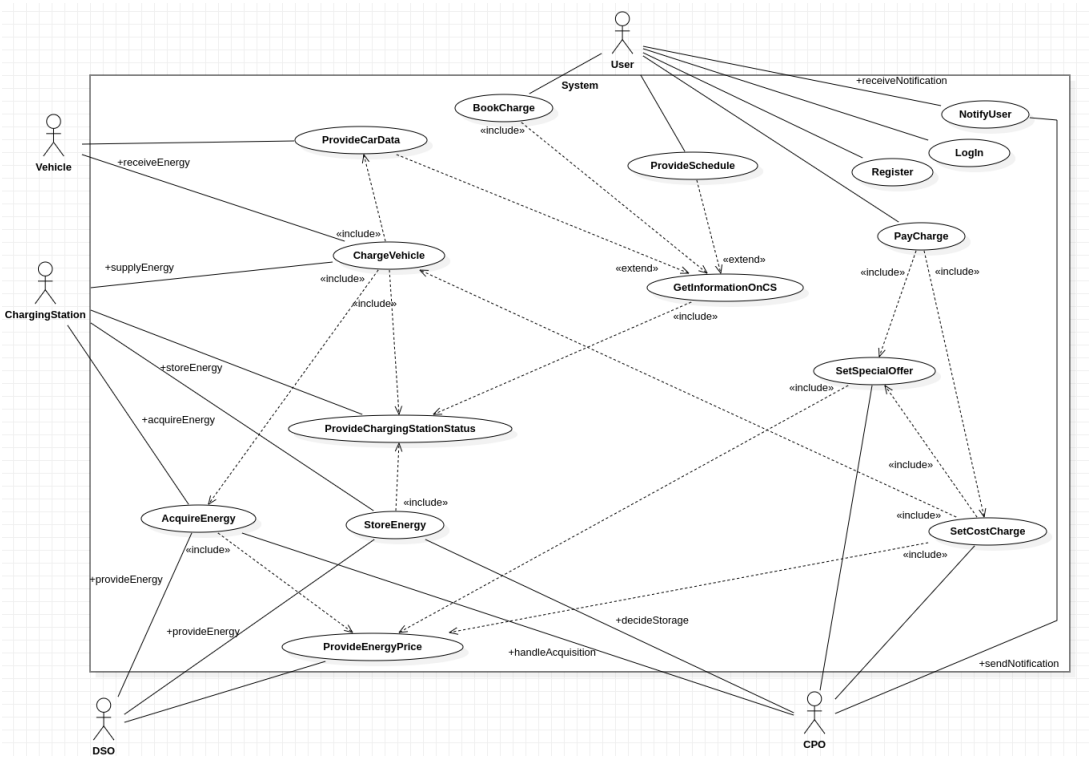
4. **DSO**



5. **Vehicle**

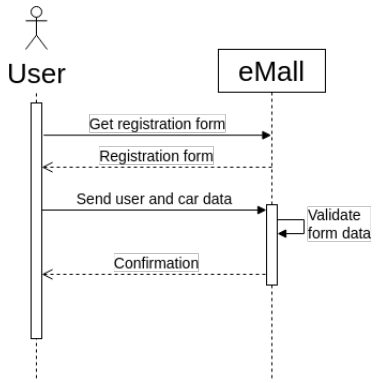


6. General Usecase diagram

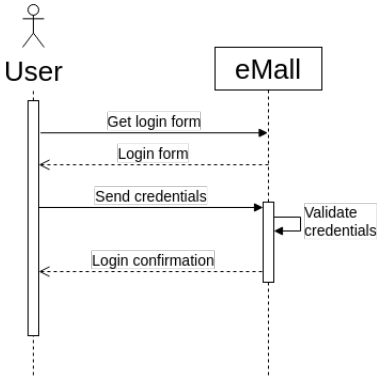


3.2.4 Sequence diagrams

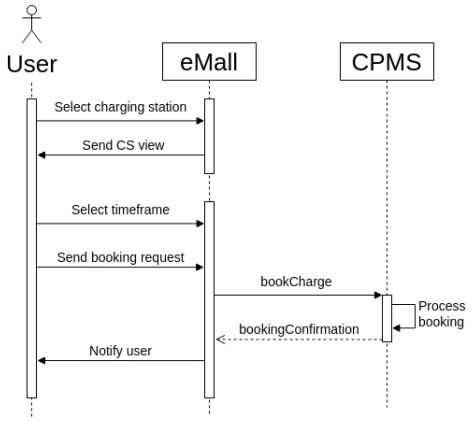
1. **User registration**



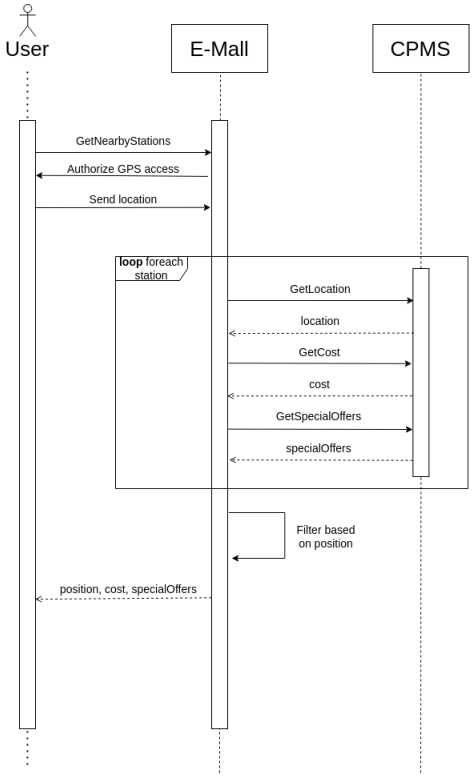
2. **User login to eMall**



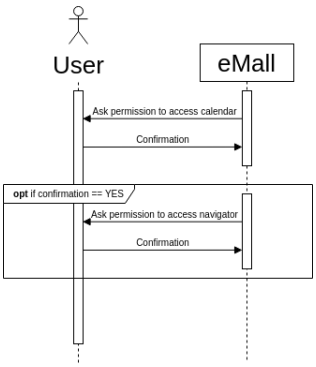
3. **User books a charge**



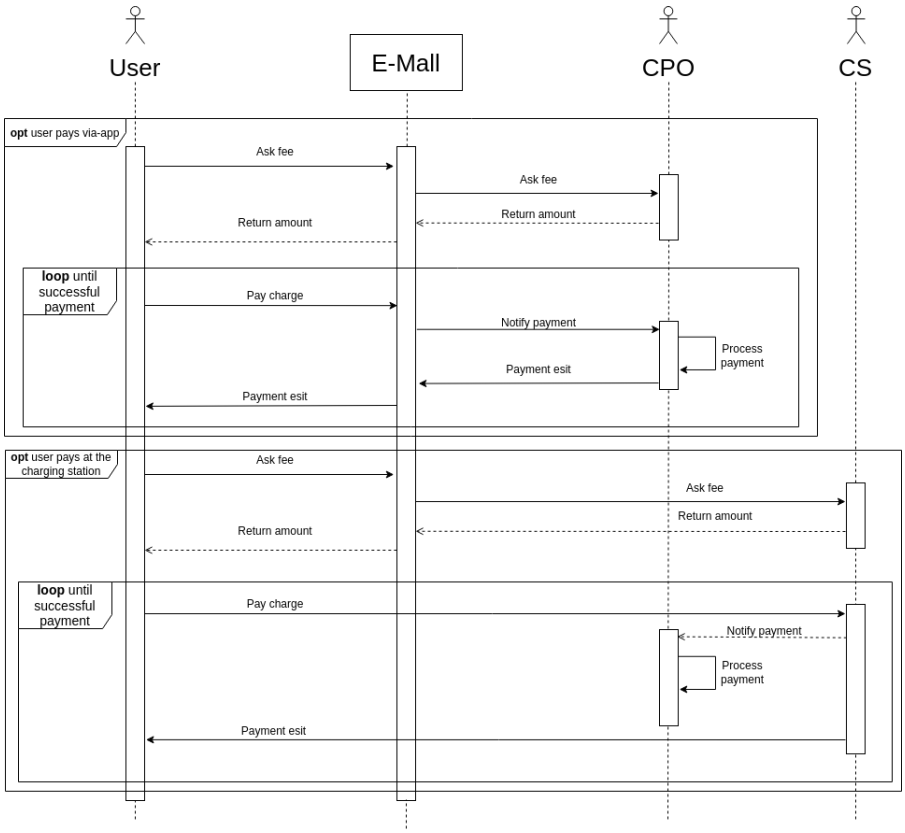
4. **Get the information about nearby stations**



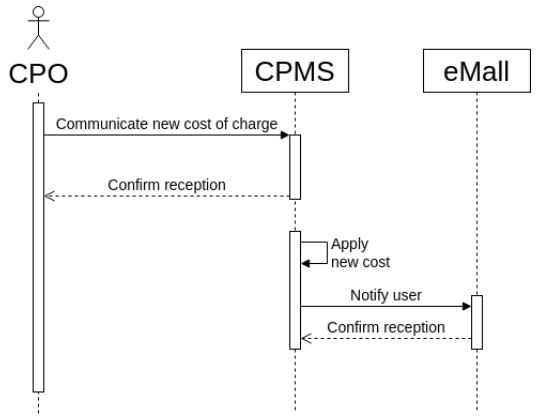
5. **Access to calendar and navigation system**



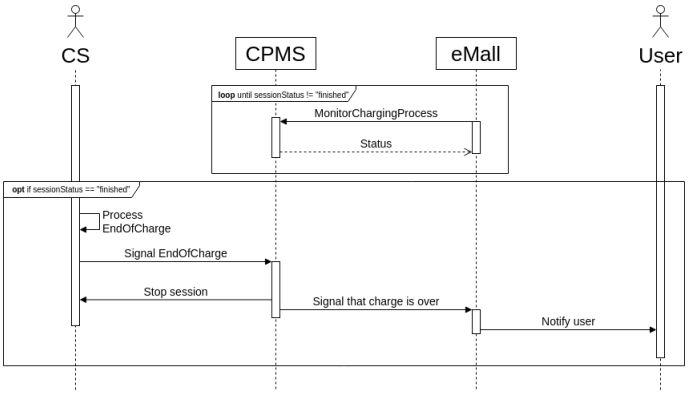
6. **User pays for a charge**



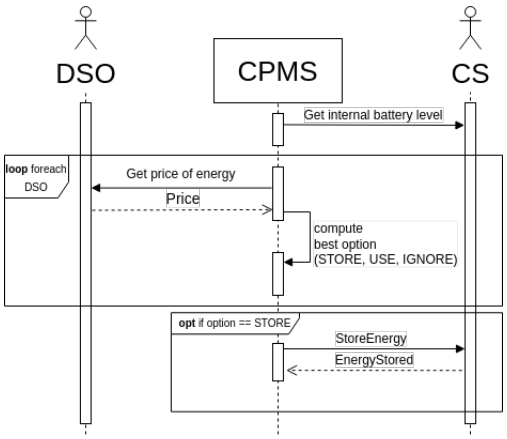
7. **Set the cost of a charge**



8. **Notify user when a charge ends** [1]

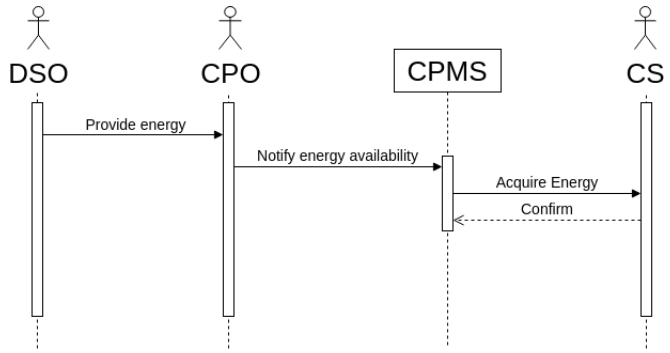


9. **Store energy** [1]

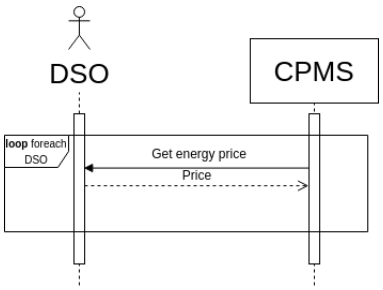


10. **Acquire energy**

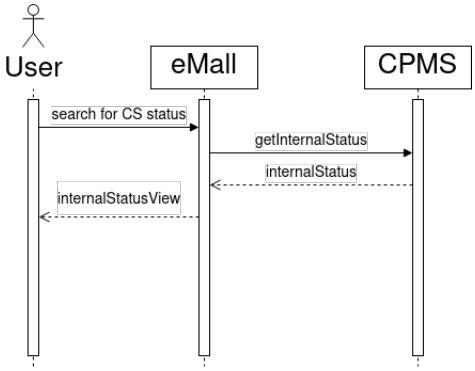
    Note: the diagram assumes that the CPMS has previously selected the DSO for energy aquisition, following the process illustrated in usecase no. 9.



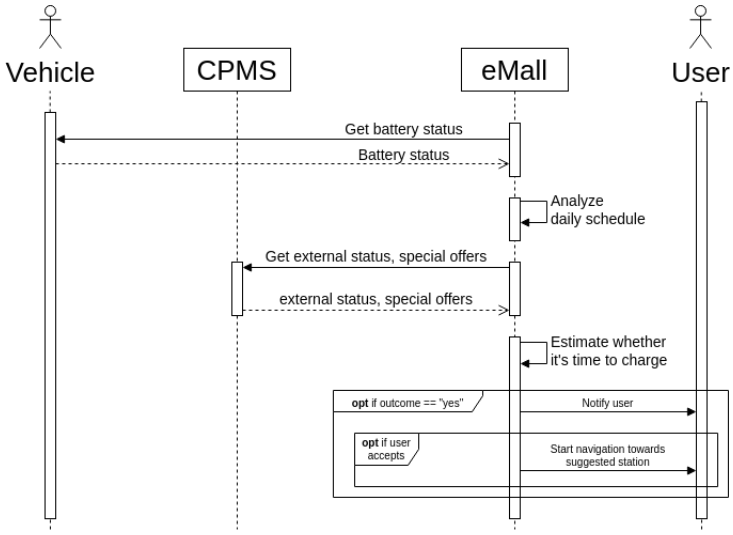
11. **Provide energy price** [1]



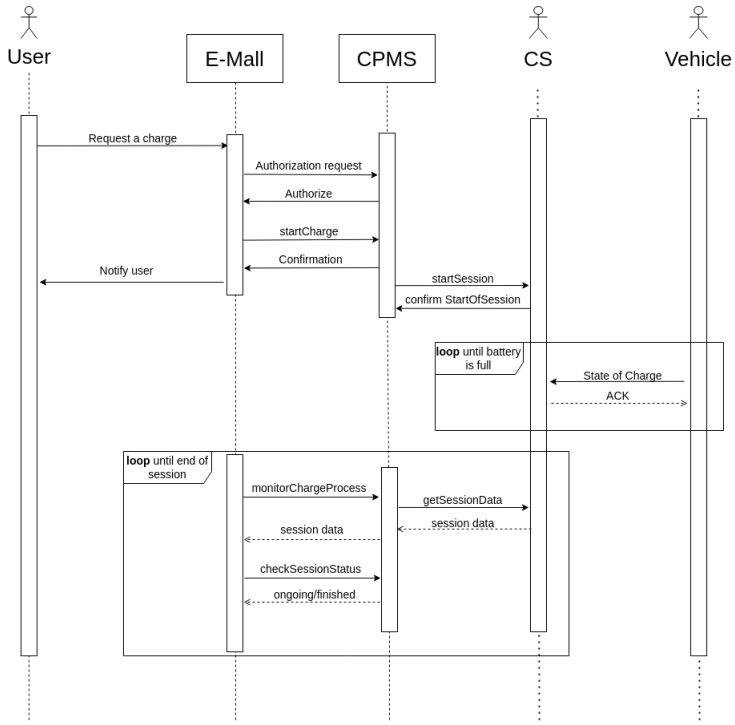
12. **Provide internal status of a CS**



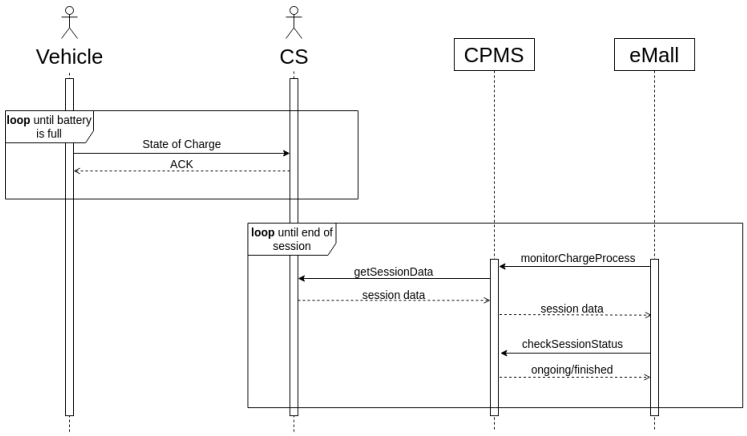
13. **Suggest the user when to charge**


Vehicle
CPMS
eMall
User
Get battery status
Battery status
Analyze
daily schedule
Get external status, special offers
external status, special offers
Estimate whether
it's time to charge
opt if outcome == "yes"
Notify user
opt if user
accepts
Start navigation towards
suggested station

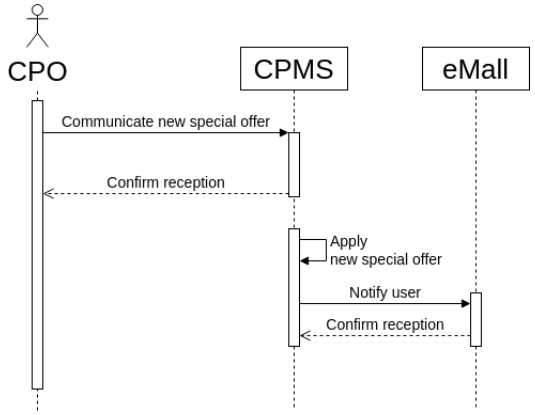
14. **Charge the vehicle**



15. **Provide vehicle status while charging**



16. **Set special offer**



[1]The CPO, although involved as an actor in the usecase, can carry out the task in a fully automated fashion through the CPMS.

3.2.5 Mapping on requirements

Use Case	Requirements
User registration	R1, R15
User login to eMall	R2, R15
User Books a charge	R5, R7, R15
Get information about nearby stations	R4, R5, R15
Access to calendar and navigation system	R6, R15, R21
User pays for a charge	R14, R15
Set the cost of a charge	R5, R18
Notify user when a charge ends	R12, R13, R15
Store energy in CS's batteries	R15, R18, R19
Acquire energy	R15, R20
Provide energy price	R18
Get internal status of a CS	R10, R15
Suggest the user when to charge	R15, R17
Charge the vehicle	R10, R11, R12, R15, R16
Provide vehicle status while charging	R15, R16
Set special offer	R18

3.3 Performance Requirements

The system should not utilize a lot of bandwidth and the client should be reasonably lightweight. By the way, these are not very strict requirements, given that the application will mainly be used in cities and running on standard smartphones. The load on the system can be huge: at least 100.000 simultaneous connections must be supported.

3.4 Design Constraints

3.4.1 Standards compliance

User's data storing policies must be compliant with GDPR. Moreover, all the protocols and hardware devices should be compliant with the according ISO standards.

3.4.2 Hardware limitations

The only hardware requirement to access eMall is owning a smartphone or a laptop and an Internet connection.

3.5 Software System Attributes

3.5.1 Reliability & Availability

The system should have a high reliability and be up and running most of the time. At least 2-nines availability (99%) is required, this means that the total downtime would be of 3.65 $\frac{days}{year}$. In order to achieve this, fault tolerance techniques must be put in place, especially redundancy of servers, as well as predictive maintenance. Hardware Engineers must be notified when a Hardware failure occurs, and be able to recover in minutes. Higher avaiability is not required, being this a service which does not deal with critical events. The system also relies on external APIs which are out of control to us, but we don't tolerate a complete failure because of an external component.

3.5.2 Security

In order to guarantee security properties, information stored on the databases must be encrypted, as well as user's passwords. IT infrastructures of charging stations must be adequatelly protected against intrusions by deploying IPS/IDS systems. Furthermore, DMZ and Firewalls should be used to protect the infrastructure from exposing internal data. Moreover, all the Internet traffic between components must happen via HTTPS.

3.5.3 Maintainability

The system will be written entirely in Java, adhering to the most modern Software Engineering princples. Software components must be reusable, flexible and extendible. Integration of a larger number of functionalities in the future must be taken into account.

3.5.4 Portability

The cilent software must run on a variety of platforms and Operating Systems: desktop (Windows, MacOS, Linux) and mobile (Android, iOS).

4 Formal Analysis

4.0.1 Alloy code

```
// == Types and Constants ==

abstract sig Bool {}
one sig TRUE extends Bool {}
one sig FALSE extends Bool {}

abstract sig VehicleStatus {}
one sig ONGOING extends VehicleStatus {}
one sig CHARGING extends VehicleStatus {}

abstract sig SlotStatus {}
one sig FREE extends SlotStatus {}
one sig OCCUPIED extends SlotStatus {}

abstract sig SlotType {}
one sig SLOW extends SlotType {}
one sig FAST extends SlotType {}
one sig RAPID extends SlotType {}

// === Signatures and Relationships ===

sig User {
        email: one String,
        owns: one Vehicle,
        isChargingTo: one ChargingSlot
}

sig Vehicle {
        licensePlate: one String,
        vehicleStatus: one VehicleStatus
}

sig Dso {
        energyPrice: one Int,
        provideEnergyTo: set Cpo
} {energyPrice > 0}

sig Cpo {
        storeThreshold: one Int,
        manages: some ChargingStation,
        getsEnergyFrom: some Dso,
        storesEnergyFrom: lone Dso
} {storeThreshold > 0}

sig ChargingStation {
        batteryAvailable: one Bool,
        contains: some ChargingSlot,
        appliedOffers: set Offer,
}

sig ChargingSlot {
        type: one SlotType,
        slotStatus: one SlotStatus,
        bookings: set Booking
}

sig Booking {
        performedBy: one User,
        startTime: one Int,
        endTime: one Int,
} {startTime > 0
   endTime > 0
   startTime < endTime}
```

```
sig Offer {
        startDate: one Int,
        endDate: one Int,
        discountPercentage: one Int
} {startDate > 0
   endDate > 0
   endDate > startDate
   discountPercentage > 0}

// === Facts ===

/**
 * If a station doesn't have batteries, it must
 * be connected to a DSO
 */
fact {
        all cpo: Cpo |
                all c: ChargingStation |
                        c in cpo.manages
                        implies
                        (c.batteryAvailable = FALSE
                        implies
                        one d : Dso |
                                d in cpo.getsEnergyFrom)
}

/**
 * If a CPO gets energy from a DSO, then that very same DSO
 * must deliver energy to it
 */
fact {
        all cpo: Cpo |
                all d: Dso |
                        d in cpo.getsEnergyFrom
                        iff
                        cpo in d.provideEnergyTo
}

/**
 * A CPO can store energy only if at least one of the station it manages
 * has got batteries available
 */
fact {
        all cpo: Cpo |
                cpo.storesEnergyFrom != none
                implies
                (some cs: ChargingStation |
                cs in cpo.manages
                implies
                cs.batteryAvailable = TRUE)
}

/**
 * If a CPO stores energy from a certain DSO,
 * its price must be lower than the computed threshold
 */
fact {
        all cpo: Cpo |
                all dso: Dso |
                        dso = cpo.storesEnergyFrom
                        implies
                        dso.energyPrice < cpo.storeThreshold
}

/**
 * Every charging station is managed by a unique CPO
 */
```

```
fact {
        all cs: ChargingStation |
                one cpo: Cpo |
                        cs in cpo.manages
}

/**
 * If the user is charging to a slot, then the slot must be taken
 */
fact {
        all u: User |
                all s: ChargingSlot |
                        s = u.isChargingTo
                        implies
                        s.slotStatus = OCCUPIED
}

/**
 * Every charging slot is connected to a station
 */
fact {
        all s: ChargingSlot |
                one cs: ChargingStation |
                        s in cs.contains
}

/**
 * Each special offer must be associated with a station
 */
fact {
        all o: Offer |
                one cs: ChargingStation |
                        o in cs.appliedOffers
}

/**
 * Validity periods of special offers don't overlap
 */
fact {
        all cs: ChargingStation |
                all disj o1, o2: Offer |
                        (o1 in cs.appliedOffers and o2 in cs.appliedOffers)
                        implies
                        not overlaps[o1.startDate, o1.endDate, o2.startDate,
                            ↪ o2.endDate]
}

/**
 * Every booking must be associated to a slot
 */
fact {
        all b: Booking |
                one s: ChargingSlot |
                        b in s.bookings
}

/**
 * Bookings in the same slots don't overlap
 */
fact {
        all s: ChargingSlot |
                all disj b1, b2: Booking |
                        (b1 in s.bookings and b2 in s.bookings)
                        implies
                        not overlaps[b1.startTime, b1.endTime, b2.startTime,
                            ↪ b2.endTime]
}
```

```
/**
 * EndTime of a booking should respect performance of charging slots
 * (e.g. the endtime for fast charging must be earlier than one for a
 * slow charge)
 */
fact {
        all cs: ChargingStation |
                all disj s1, s2: ChargingSlot |
                        (s1 in cs.contains and s2 in cs.contains)
                        implies
                        all disj b1, b2: Booking |
                                (b1 in (s1.bookings + s2.bookings) and
                                 b2 in (s1.bookings + s2.bookings))
                                implies
                                complyToSlotSpeed[cs, s1, s2, b1, b2]
}

/**
 * Every user has a different email
 */
fact noDuplicateEmail {
        no disj u1, u2: User |
                u1.email = u2.email
}

/**
 * Each vehicle must be owned by a user
 */
fact {
        all v: Vehicle |
                one u: User |
                        v in u.owns
}

/**
 * user email and vehicle's license plate must be
 * different strings
 */
fact {
        all u: User |
                all v: Vehicle |
                        u.email != v.licensePlate
}

/**
 * Each vehicle must have a different license plate
 */
fact {
        all disj v1, v2: Vehicle |
                v1.licensePlate != v2.licensePlate
}

// The following facts make sure that every constant is connected

fact allSlotStatusConnected {
        all ss: SlotStatus |
                some s: ChargingSlot |
                        s.slotStatus = ss
}

fact allSlotTypesConnected {
        all st: SlotType |
                some s: ChargingSlot |
                        s.type = st
}

fact allVehicleStatusConnected {
        all vs: VehicleStatus |
```

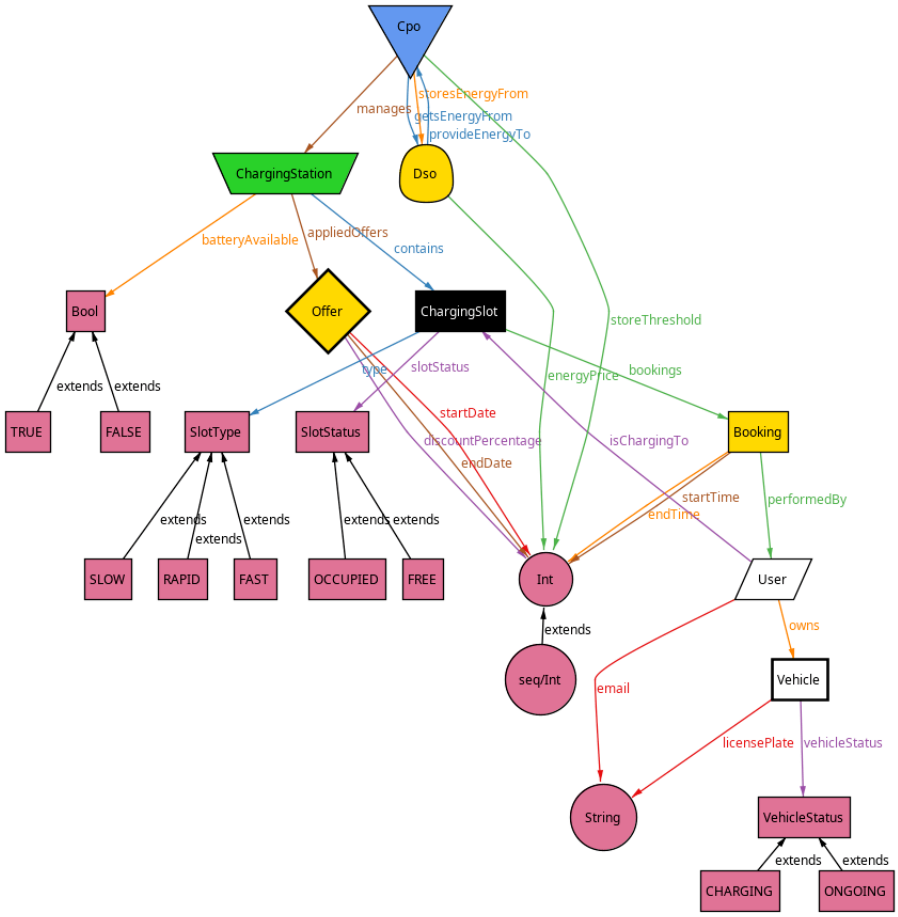
```
                some v: Vehicle |
                        v.vehicleStatus = vs
}

/**
 * Specify which are the strings that should be used in the world
 */
fact makeStringsResolve {
        none != "a" + "b" + "c" + "d"
}

// === Assertions ===



// === Predicates ===

pred overlaps[start1, end1, start2, end2: Int] {
        (start1 = start2) or
        (start1 < start2 and start2 ≤ end1) or
        (start1 > start2 and start1 ≤ end2)
}

/**
 * Sets the rules according to which
 * a booking for a faster slot must finish earlier than for a slower one
 */
pred complyToSlotSpeed[cs: ChargingStation, s1, s2: ChargingSlot, b1, b2:
    ↪ Booking] {
        ((s1.type = RAPID and s2.type = FAST)
        implies
        b1.endTime < b2.endTime)
        or
        ((s1.type = RAPID and s2.type = SLOW)
        implies
        b1.endTime < b2.endTime)
        or
        ((s1.type = FAST and s2.type = SLOW)
        implies
        b1.endTime < b2.endTime)
}

// === Dynamic modeling ===

pred addBooking[b: Booking, u: User, s, s': ChargingSlot, start, end: Int] {
        b.performedBy = u
        b.startTime = start
        b.endTime = end
        s'.bookings = s.bookings + b
}

run addBooking

pred addOffer[cs, cs': ChargingStation, o: Offer, s, e, p: Int] {
        o.startDate = s
        o.endDate = e
        o.discountPercentage = p
        cs'.appliedOffers = cs.appliedOffers + o
}

run addOffer

// === Worlds ===

pred completeWorld {
        #Cpo > 1
        #ChargingStation > 1
        #ChargingSlot > #ChargingStation
```

```
        #Offer > #ChargingStation
        #Vehicle > 1
        #Booking > #ChargingStation

        some cs: ChargingStation | #cs.contains > 1
        some c: Cpo | #c.manages > 1
        some cs: ChargingStation | cs.batteryAvailable = FALSE
        some s: ChargingSlot | #s.bookings > 1
        some disj dso1, dso2: Dso | #dso1.provideEnergyTo > 1 and #dso2.
            ↪ provideEnergyTo > 1
}

run completeWorld for 5
```

4.0.2 Worlds

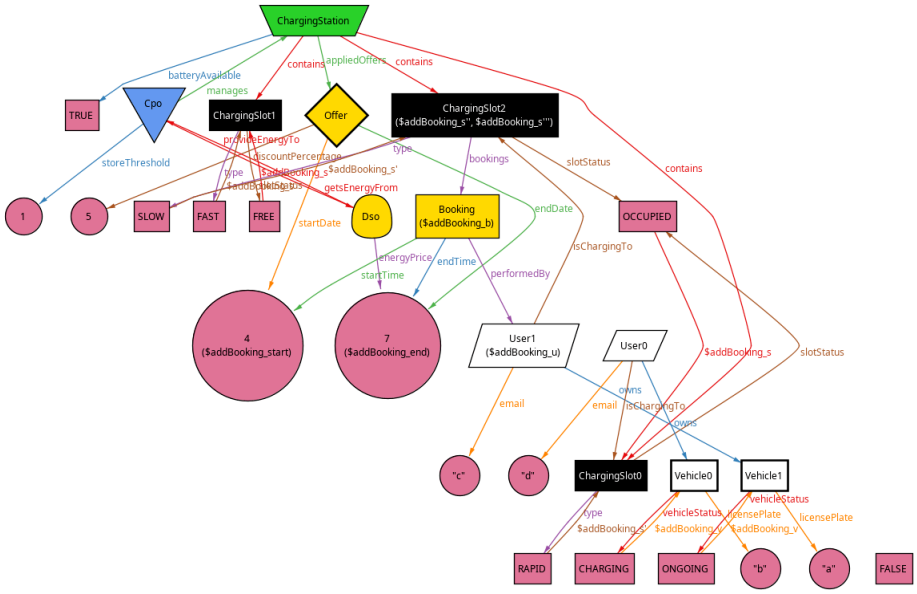
- **World 1 (addBooking)**: This world aims to show the result of booking a charge, verifying that it's correctly linked with the other entities.
- **World 2 (addOffer)**: This world aims to show the result of the introduction of a new special offer for a charging station.
- **World 4 (completeWorld)**: This world aims to show the connections between the infrastructure components such that CPOs, DSOs, stations and slots. In particular
  - A station must be connected to one CPO
  - A CPO can fetch energy from multiple DSOs, or none
  - A slot must be part of one (and only one) station

4.0.3 Metamodel

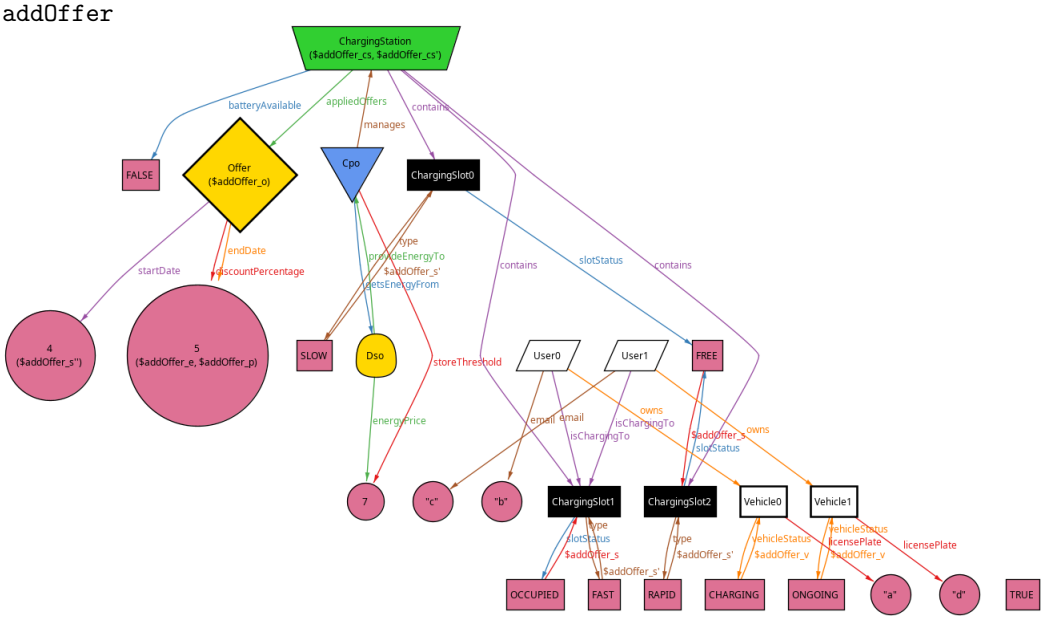


4.0.4 Instances

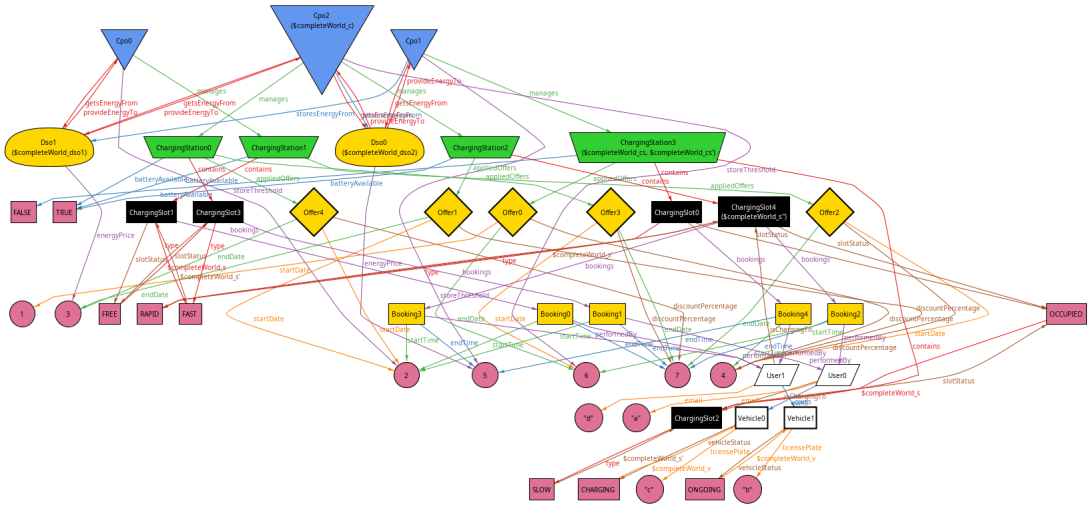
- `addBooking`



- `addOffer`



- **Complete world**



4.0.5 Result of predicates

```
Executing "Run addBooking"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8741 vars. 566 primary vars. 21158 clauses. 132ms.
   Instance found. Predicate is consistent. 26ms.

Executing "Run addOffer"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8846 vars. 579 primary vars. 21423 clauses. 114ms.
   Instance found. Predicate is consistent. 35ms.

Executing "Run completeWorld for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   21384 vars. 1000 primary vars. 45105 clauses. 653ms.
   Instance found. Predicate is consistent. 171ms.
```

5 Effort spent

5.1 Lorenzo

Task	Time spent
Introduction	
Overall description	
Specific requirements	
Formal analysis	
Reasoning	
Document composition	

5.2 Melissande

Task	Time spent
Introduction	
Overall description	
Specific requirements	
Formal analysis	
Reasoning	
Document composition	

5.3 Nicolas

Task	Time spent
Introduction	
Overall description	
Specific requirements	
Formal analysis	
Reasoning	
Document composition	

6 References

- More information about EV charging grids was found on `https://www.current.eco/ev-glossary`
- Presentation about EV charging infrastructure given at PoliMi by the startup Atlante