Travlendar+
Requirement Analysis and Specification Document

Fumagalli Paolo, Grotti Pietro, Gullo Marco

October 28, 2017

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

Many endeavors require scheduling meetings at various locations all across a city, whether in support of a mobile job or a busy parent. The goal of this project is to create a calendar interface that automatically computes and accounts for travel time between appointments to make sure that you're never late for an appointment. The application will also suggest travel means by appointment (e.g., perhaps you drive to the office in the morning but the bus is a better choice between a pair of afternoon meetings) and by day (e.g. working days or weekends, traffic, public transport strikes, weather). The application will support an user interface where complete and automatically well-fitted schedules can be made. System will alert for any appointment overlap and non-doable consecutive appointments (e.g. two meetings really close in time but in locations too far from each other). Furthermore, different preferences, such as travel options and break time, can be expressed by the user.

Goals

The application has the following goals:

G1: Allow a guest user to register to Travlendar+ by filling the registration form with the data needed.

G2: Allow the user to select preferences and modify them whenever he wants.

G3: Allow the user to easily create an organized and customizable agenda based on his preferences.

G4: To help the user plan his movements in a clever and efficient way.

G5: To guarantee the user no to be late for his appointments.

- **G5.1:** The system will take into account the possibility of accidents and plan for the user to arrive at least 10 minutes before the beginning of the event.
- **G5.2:** The user will be notified if an appointment is not reachable in time.

G6: To let the users buy bus or train tickets (both single or seasonal).

G7: To let the user find vehicles from vehicle sharing systems.

G8: Allow the user to buy in advance tickets which can be used later on.

1.2 Scope

Travlendar+ is an application which will be able to manage daily appointments of the users and assist them by creating a specific course around the city which will identify the best mobility option to move from one appointment to the other. It will consider all public transportation options (train, metro, bus, etc.), car and bike sharing systems, the eventuality of a private vehicle and the weather conditions as well, to avoid having the user bike in harsh weather conditions or when it is too hot. It will also be possible to buy tickets and locate cars and bikes directly through the application. Users will have to create meetings specifying the location, the date and the time and the application will automatically calculate the suggested course and travel time, if it is impossible to reach a certain meeting in time the application will send a warning to the user. It will also feature the possibility of selecting a time span in which it will have to save some time for a break in which the user can have lunch (for example if the user wants to have lunch between 11:30 to 2:30 and it should be at least half an hour long, the application will reserve at least 30 minutes for lunch every day). Travlendar+ will also give the possibility to the users to buy tickets, this will make it easier to move with public transportation around the city. To allow these transactions the system will have to work with a bank which will support the possibility of either using a credit/debit card to buy tickets or creating a balance that will be connected to the account. The balance can be charged by credit/debit card or through selected shops which will allow the acquisition of cards containing unique codes, which will load money on the balance.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

1.3.2 Abbreviations

G* : Specific goal
R* : Specific functional requirements
D* : Specific domain assumption
App : Application

1.3.3 Acronyms

ETA : Estimated Time of Arrival
API : Application Programming Interface
PTS : Public Transportation System
CSS : Car-Sharing System
BSS : Bike-Sharing System

1.4 Revision history

Version 1.0.0

1.5 Reference Documents

- IEEE Recommended Practice for Software Requirements Specifications, Software Engineering Standards Committe, 1998
- Specification Document: Mandatory Project Assignments 2017-2018
- API overall description
- RASD structure according IEEE standard

1.6 Document Structure

We decided to structure the document starting from the IEEE standard, but modifying it to make it more readable and linear.
Chapter 2 will give an overall presentation of the application, starting with the purpose of the project and later defining which will be the actors interacting with Travlendar+ and which are the assumptions that we will consider as true for the development of the project.
Later in chapter 3 we will define all the requirements, both functional and non-functional. In this chapter we will include some initial mock-ups of the application.
We've decided to divide the requirements from the diagrams, so chapter 4 will contain all of our UML, Use Cases, Statecharts and Sequence Diagrams, which will be related to the goals and requirements we defined before this chapter.
Chapter 5 will show a formal analysis of the application which was created in Alloy, and it will also show some instances of the application.

2 Overall Description

2.1 Product perspective

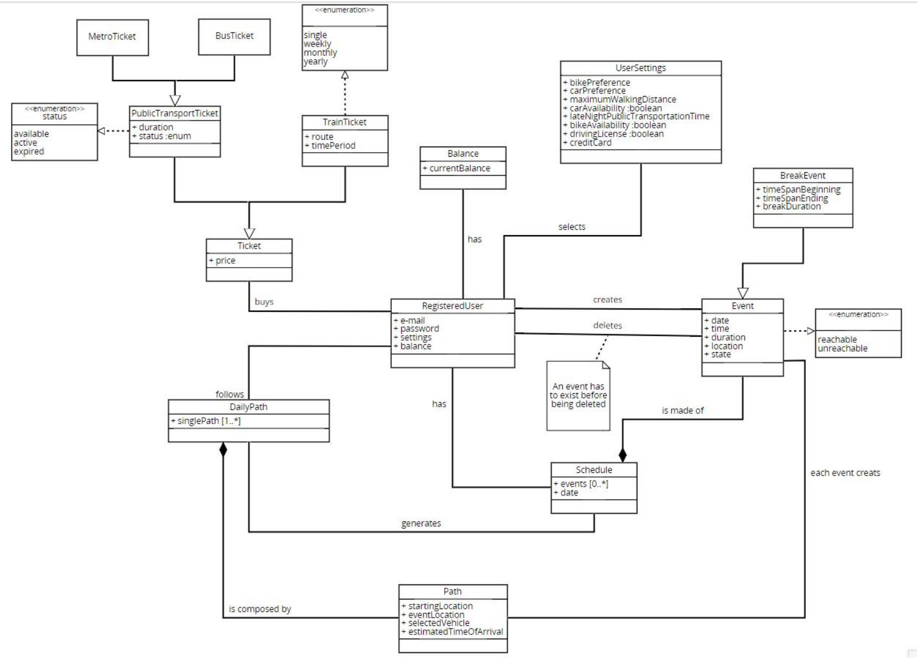


Figure 1: Class Diagram

The products main purpose is to fit in users everyday reality, dealing with a large number of external factors and environments, such as weather conditions, traffic, news, etc. In order to face up all these agents, the system will always cooperate with other existing applications and ask for external help. Travlendar+ will work as an agent-in-the-middle between user and different interfaces, putting together all data required for users planning and traveling needs. All the various interactions will be described in high-level details in UML Section (see Section 4.2: Use Cases).

2.2 Product functions

All main functions will be described in following sections (see Section 3: Specific Requirements) and are the ones that will cover all project goals provided above(see Section 1.1: Purpose).

2.3 User characteristics

Actors:

- **Guest User:** unregistered customer, he has just downloaded the app or visited the website. He wants to try Travlendar+ and needs to find a user-friendly interface.

- **Registered User:** more familiar with the environment, can access all Travlendar+ functionalities and customize his preferences. Expects an efficient service.

- **Public Transportation System:** local public transportation business, is willing to cooperate with the app in order to increase tickets sales and reduce irregular PTS use. Online purchases may also decrease the costs of printed tickets.

- **Car-sharing System:** local private enterprise, is interested in Travlendar+ development to enhance their business reach and visibility.

- **Bike-sharing System:** same as car-sharing system.

- **Google Maps:** is a web mapping service developed by Google. It offers satellite imagery, street maps, 360 panoramic views of streets, real-time traffic conditions, and route planning for traveling by foot, car or public transportation. It is completely free-to-use (its API can be integrated easily in any system) for any system. Travlendar+ will benefit of this.

- **Bank:** it is the entity necessary for Travlendar+ to deal with money transactions (ticket and pass purchases). The bank surely has its own secure channels and count on integrity of data coming from the apps system.

2.4 Assumptions, dependencies and constraints

The following assumptions are to be considered true in the world that we will analyze:

D1: The city is covered by several public and private transport services.

D2: The services are almost always available, and when they arent theres an alternative which can be suggested by Travlendar+ to the user.

D3: A database containing all the correct information needed for transport services (location of the vehicles or the stationts, time tables, etc.) is connected to Travlendar+.

D4: An application able to calculate travel times and distances which can also access current traffic information is connected to Travlendar+.

D5: Some services allow external applications to interact with them to take advantage of certain functions (e.g.: to buy bus tickets from the PTS or to find a car of a BSS) in a transparent way from the user's point of view.

D6. An internet connection is available everywhere and at any given time in the area covered by Travlendar+.

D7: Users of the application have a working GPS that can accurately calculate their position.

D8: The public transportation companies agree to let the application developers sell tickets through Travlendar+.

D9: The transactions are handled through a bank that agrees to work with Travlendar+.

D10: Bikes and cars of sharing-systems are equipped with an accurate GPS that can be traced in the map.

D11: Vehicles sharing systems agree to work with Travlendar+.

D12: "Activated" tickets are accepted by public transportation companies.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

In the following section we will show some mock-ups of how the application will look like from the user's point of view.

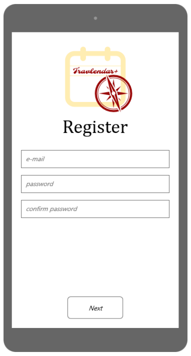
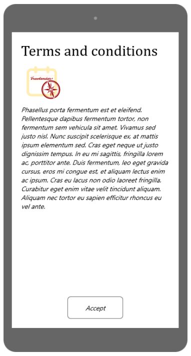


Figure 2: Account Creation



Phasellus porta fermentum est et eleifend. Pellentesque dapibus fermentum tortor, non fermentum sem vehicula sit amet. Vivamus sed justo nisl. Nunc suscipit scelerisque ex, at mattis ipsum elementum sed. Cras eget neque ut justo dignissim tempus. In eu mi sagittis, fringilla lorem ac, porttitor ante. Duis fermentum, leo eget gravida cursus, eros mi congue est, et aliquam lectus enim ac ipsum. Cras eu lacus non odio laoreet fringilla. Curabitur eget enim vitae velit tincidunt aliquam. Aliquam nec tortor eu sapien efficitur rhoncus eu vel ante.

Figure 3: Terms And Conditions

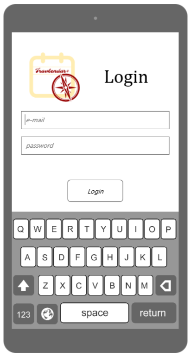


Figure 4: Login

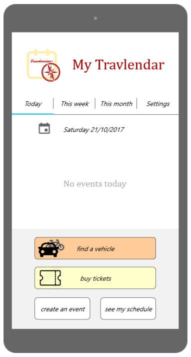
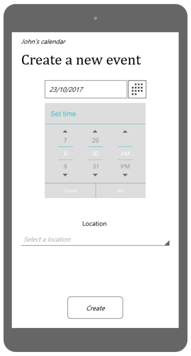


Figure 5: Home Page



John's calendar

Create a new event

23/10/2017

Set time

7 29

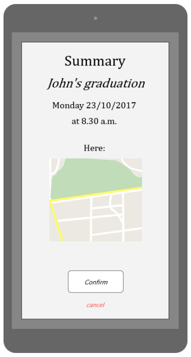
8 : 30 AM

9 31 PM

Location

Select a location

Figure 6: Create Event



Summary

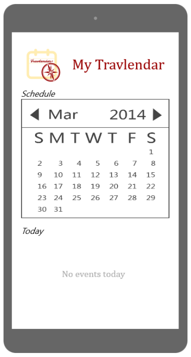
John's graduation

Monday 23/10/2017

at 8.30 a.m.

Here:

Figure 7: Summary



My Travlendar

Schedule

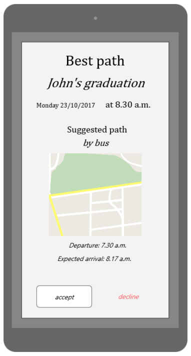
Mar 2014

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Today

No events today

Figure 8: Schedule



Best path

John's graduation

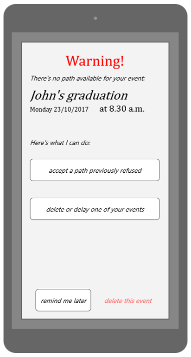
Monday 23/10/2017 at 8.30 a.m.

Suggested path
by bus

Departure: 7.30 a.m.

Expected arrival: 8.17 a.m.

Figure 9: Path



Warning!

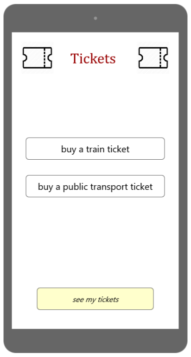
There's no path available for your event:

John's graduation
Monday 23/10/2017 at 8.30 a.m.

Here's what I can do:

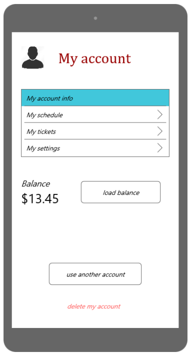
accept a path previously refused
delete or delay one of your events
remind me later
delete this event

Figure 10: Unreachable Path



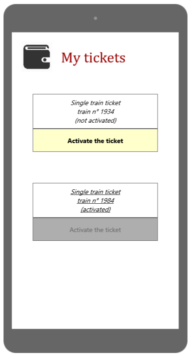
Tickets

Figure 11: Ticket Store



Balance
$13.45

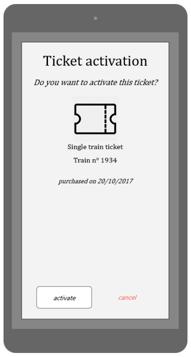
Figure 12: Personal Page



Single train ticket
train n° 1934
(not activated)

Single train ticket
train n° 1984
(activated)

Figure 13: My Tickets



Do you want to activate this ticket?

Single train ticket
Train n° 1934

purchased on 20/10/2017

Figure 14: Ticket Activation

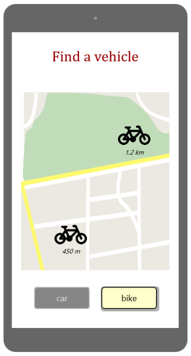


Figure 15: Find a Vehicle

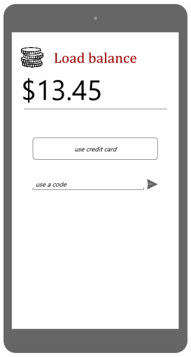


Figure 16: Balance

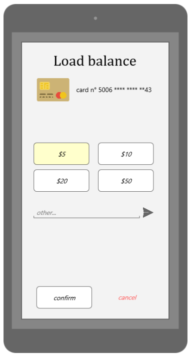


Figure 17: Balance Loading

3.2 Functional Requirements

In this section we will define the functional requirements combined with the domain assumptions that are needed to reach the goals set in the Section 1.1.

G1: Allow a guest user to register to Travlendar+ by filling the registration form with the data needed.

- **R1:** A guest user can create a new account through the registration process by providing his credentials to the system. An e-mail will be necessary and it must be unique.
- **R2:** At the end of the registration process the system will send an e-mail to the user to verify the account.
- **R3:** The account must be verified with the link provided by the system before it becomes active.
- **R4:** A guest user can log in the application with his credentials.

G2: Allow the user to select preferences and modify them whenever he wants.

- **R5:** A registered user can modify his preferences by choosing the preferred movement system at any time.
- **R6:** A registered user can select a specific time during the day (or night) after which he doesn't want to take public transportation.
- **R7:** A registered user can choose a maximum distance that he is willing to walk to move from one place to the other.

G3: Allow the user to easily create an organized and customizable agenda based on his preferences.

- **R8:** A registered user can create an event by filling the time, date, duration and location of the event.
- **R9:** A registered user can select a time slot during the day in which he wants to take a break, he can also choose the length of the break (which doesn't have to necessarily be equal to the time slot).

G4: To help the user plan his movements in a clever and efficient way.

- **R10:** The user has to declare if he has a driving license.
- **R11:** The user can declare if he owns a personal vehicle and indicate which kind of vehicle (car, motorbike, bike).
- **R12:** The system has to calculate a route to move araound the city based on the user's preferences.
- **R13:** The system will have to adjust accordingly to unexpected events (accidents, strikes, weather, natural disasters) changing the route of the day and notifying the user.
- **D1:** The city is covered by several public and private transport services..

- **D2:** The services are almost always available, and when they arent theres an alternative which can be suggested by Travlendar+ to the user.
- **D3:** A database containing all the correct information needed for transport services (location of the vehicles or the stationts, time tables, etc.) is connected to Travlendar+.
- **D4:** An application able to calculate travel times and distances which can also access current traffic information is connected to Travlendar+.

G5: To guarantee the user no to be late for his appointments.

- **R14:** The system will always try to have the ETA to an event at least 10 minutes earlier than the beginning of the event.
- **R15:** The system will notify the user if one appointment is not reachable, this feature has to update the user in real time, which means that if an event becomes unreachable it will notify the user immediately.
- **D4:** An application able to calculate travel times and distances which can also access current traffic information is connected to Travlendar+.
- **D7:** Users of the application have a working GPS that can accurately calculate their position.

G6: To let the users buy bus or train tickets (both single or seasonal).

- **R16:** The system will support the acquisition of valid tickets for public transportation.
- **R17:** The user can link h is account to a credit card to pay for tickets.
- **R18:** The user can open a balance connected to his account that can be used to pay for tickets.
- **R19:** The balance can be charged with credit cards or with cash, by going to shops that are certified by the application.
- **R20:** After being bought a ticket will be flagged as ”Available”.
- **D8:** The public transportation companies agree to let the application developers sell tickets through Travlendar+.
- **D9:** The transactions are handled through a bank that agrees to work with Travlendar+.

G7: To let the user find vehicles from vehicle sharing systems.

- **R21:** The system must be able to locate vehicles from vehicle sharing systems.
- **D10:** Bikes and cars of sharing-systems are equipped with an accurate GPS that can be traced in the map.
- **D11:** Vehicles sharing systems agree to work with Travlendar+.

G8: Allow the user to buy in advance tickets which can be used later on.

R21: An ”Available” ticket can be activated.

R22: After a ticket is activated it will be flagged as ”Active”.

R23: Once the time expires on an ”Active” ticket it will be flagged as ”Expired”.

R24: An ”Expired” ticket can’t be activated.

D12: ”Activated” tickets are accepted by public transportation companies.

3.3 Design Constraints

3.3.1 Regulatory Policy

Any sensible data, such as personal and payment info, will be stored with the purpose of ensure the best service and will not be handed only to third parties. This agreement will be described in better details in the Terms and Conditions the user will accept in the registration process.

3.3.2 Hardware Limitations

Main limitations will come from the mobile version of Travlendar+, which means:

- No Windows Phone OS-supporting version
- 3G/4G/LTE connection
- GPS accuracy

3.3.3 Interfaces With Other Applications

As written above in Product Perspective (section 2.1), the system will be developed to cooperate and interact dynamically with different external applications, so it must be flexible and open to new collaborations with new services in future. Developers will make sure to choose the appropriate middleware for such aim.

3.4 Non-Functional Requirements

3.4.1 Extensibility

Since Travlendar+ will be developed from scratch, it will be first launched with only the essential characteristics, its core goals, but is meant to be open to future additional features and improvements. Therefore, the system shall provide an API, a way for programmers to create own application which can interact with the systems data and functionalities. This choice implies a more complex security system, in order to avoid developers to access sensitive users data. More specific information will be given in the design document.

3.4.2 Performance

The aim of Travlendar+ is to become a large-scale used application. The system must be able to handle a great number of simultaneous requests and respond correctly in the best time possible.

3.4.3 Reliablity

System must work ideally 24/7, but a short period of time should be dedicated to daily maintenance.

3.4.4 Security

Users credentials, personal information, payment data will be stored and must be protected.

3.4.5 Accuracy

System must be accurate in computing routes and in estimating time necessary of any movement from a place to another. For this purposes very precise maps must be used, the system also must take account of GPS accuracy (GPS technology is theoretically able to provide locations with an error less or equal to 7.8 meters).

3.4.6 Interoperability

System must interact efficiently with other applications, such as vehicle sharing apps, to provide a complete service. Travlendar+ will be also mediator in purchases, so needs to be connected to one or more bank/payment system.

4 UML Diagrams, Use Cases, Statecharts And Sequence Diagrams

4.1 Use Case Diagram

We defined the Use Case Diagrams for the actors as Guest Users and Registered User.

4.1.1 Guest User

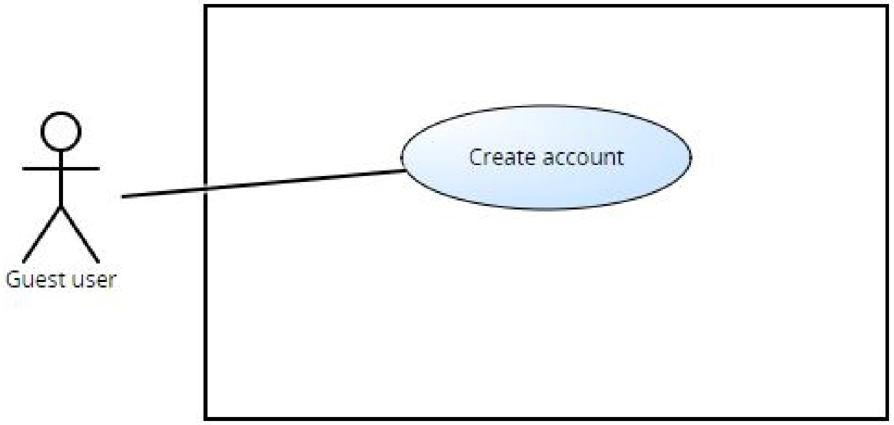


Figure 18: Guest user use case diagram

4.1.2 Registered User

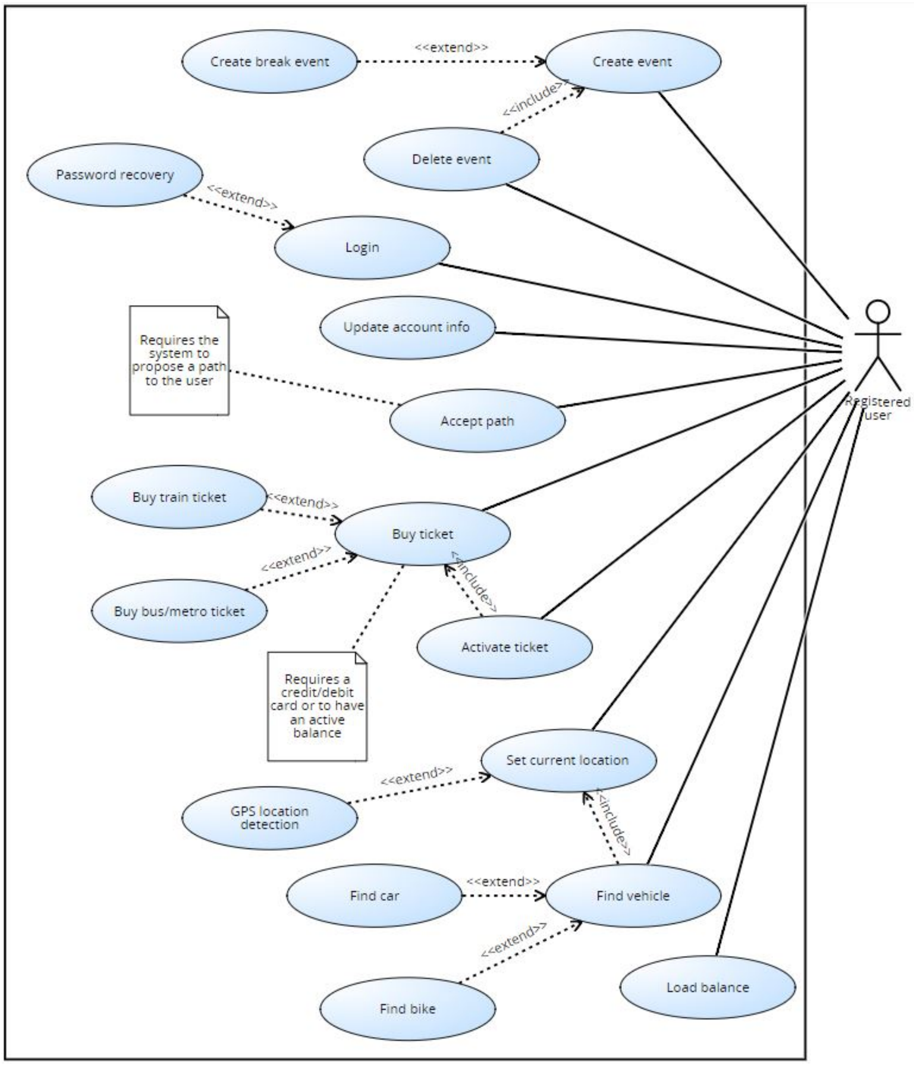


Figure 19: Registered user use case diagram

4.2 Use Cases

In the following section we include the use cases with their relative Sequence Diagrams and Statecharts.

4.2.1 Create Account

Name	Create Account
Actors	Guest User
Assumptions	The user has started a registration form. The user has a valid e-mail address.
Flow Of Events	The user types his e-mail in the form. The user reads the app's terms and conditions. The user accepts the app's terms and conditions. The user submits the information. The system sends an e-mail to the user. The user confirms the account through the link in the e-mail.
Exit Condition	The account is created
Exceptions	The e-mail is already linked to an account. The account is not created.

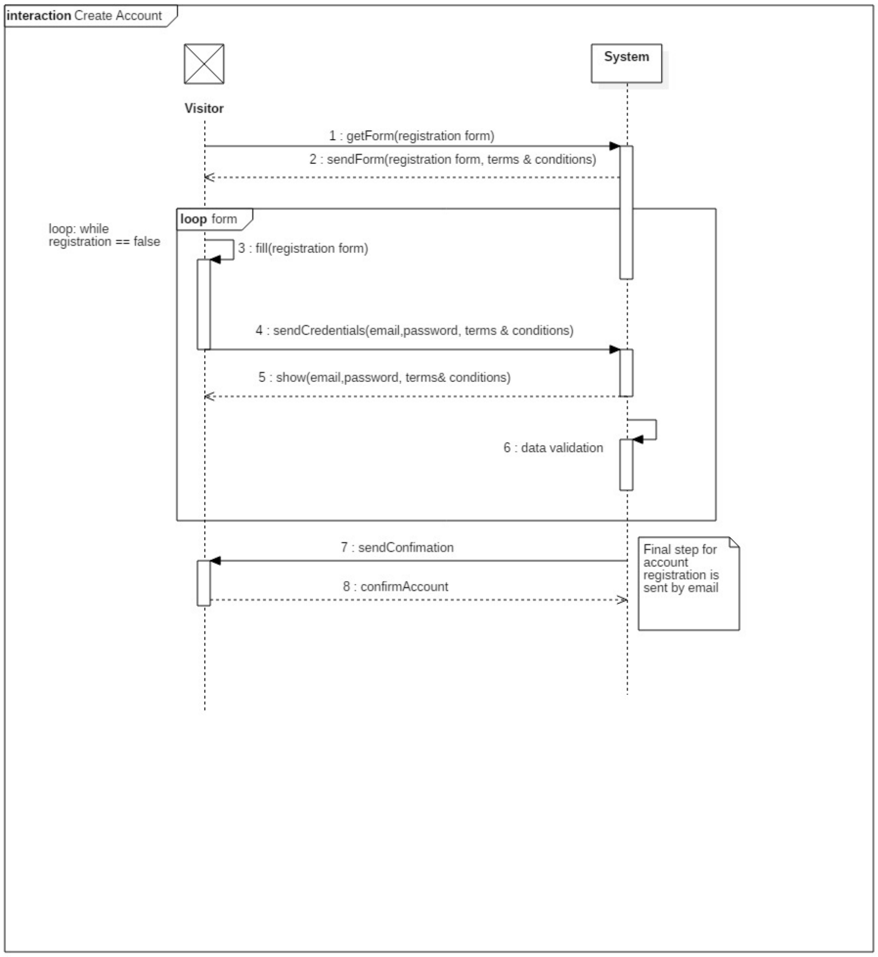

interaction Create Account
Visitor
System
1 : getForm(registration form)
2 : sendForm(registration form, terms & conditions)
loop form
loop: while
registration == false
3 : fill(registration form)
4 : sendCredentials(email,password, terms & conditions)
5 : show(email,password, terms& conditions)
6 : data validation
7 : sendConfimation
8 : confirmAccount
Final step for
account
registration is
sent by email

Figure 20: Create Account Sequence Diagram

4.2.2 Login

Name	Login
Actors	Registered User
Assumptions	The User has alreadty registered an account. The user hasn't logged in yet.
Flow Of Events	The user has to open the app homepage. The user has to input his information (e-mail and password). The system checks if the information sent by the user are valid.
Exit Condition	The user is logged in the system
Exceptions	The information provided by the user are wrong.The user is not logged in. The page will refresh with an error message.

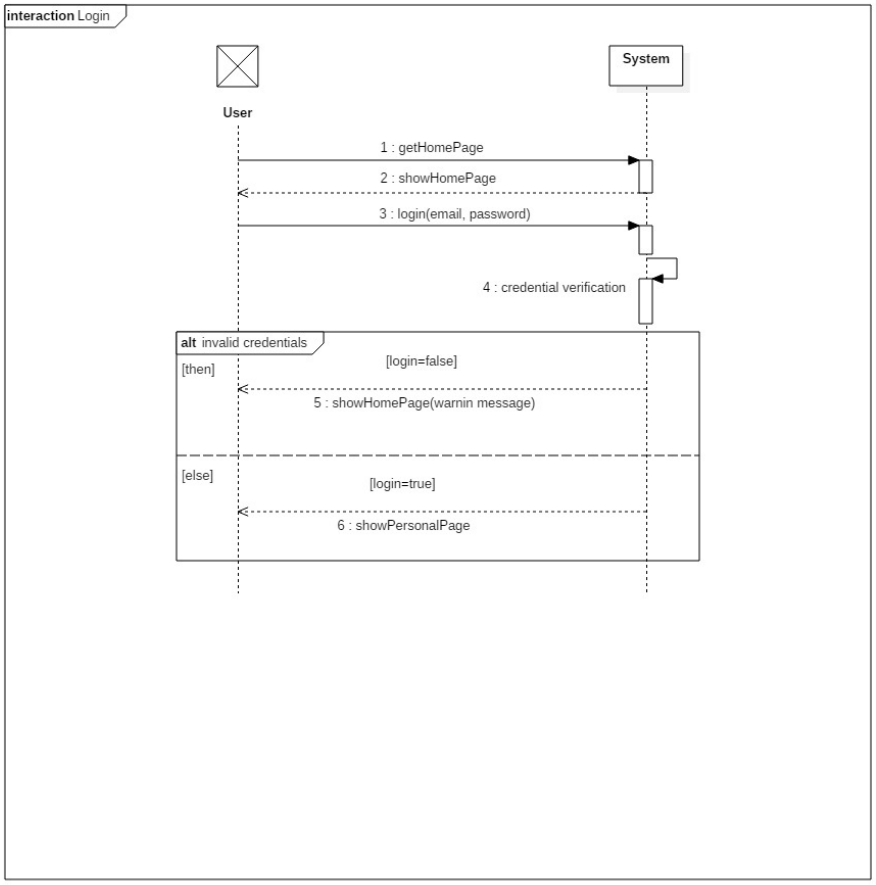


Figure 21: Login Sequence Diagram

4.2.3 Create Event

Name	Create Event
Actors	Registered User
Assumptions	The user has already logged in the account.
Flow Of Events	The user has to go to the homepage of the app The user has to open the "Create an event" page. The user has to fill the information needed for the event (time, date, duration and location). The user submits the information. The system sends a confirmation request under the form of a review of the event. The user confirms the information. The system loads the information on the user's schedule.
Exit Condition	The event is successfully created and added on the schedule.
Exceptions	The user does not confirm the information. The system will go back to the information submission page and ask the user to input the correct information. The connection is lost during the submission of the information. The event will not be created and the system will send an error message to the user who will be notified once the connection is restored. The location does not exist on the map. The system will send an error message and the user will have to update the location. The event is conflicting in time with another event, in this case the event will be created but the system will send the user a notification.

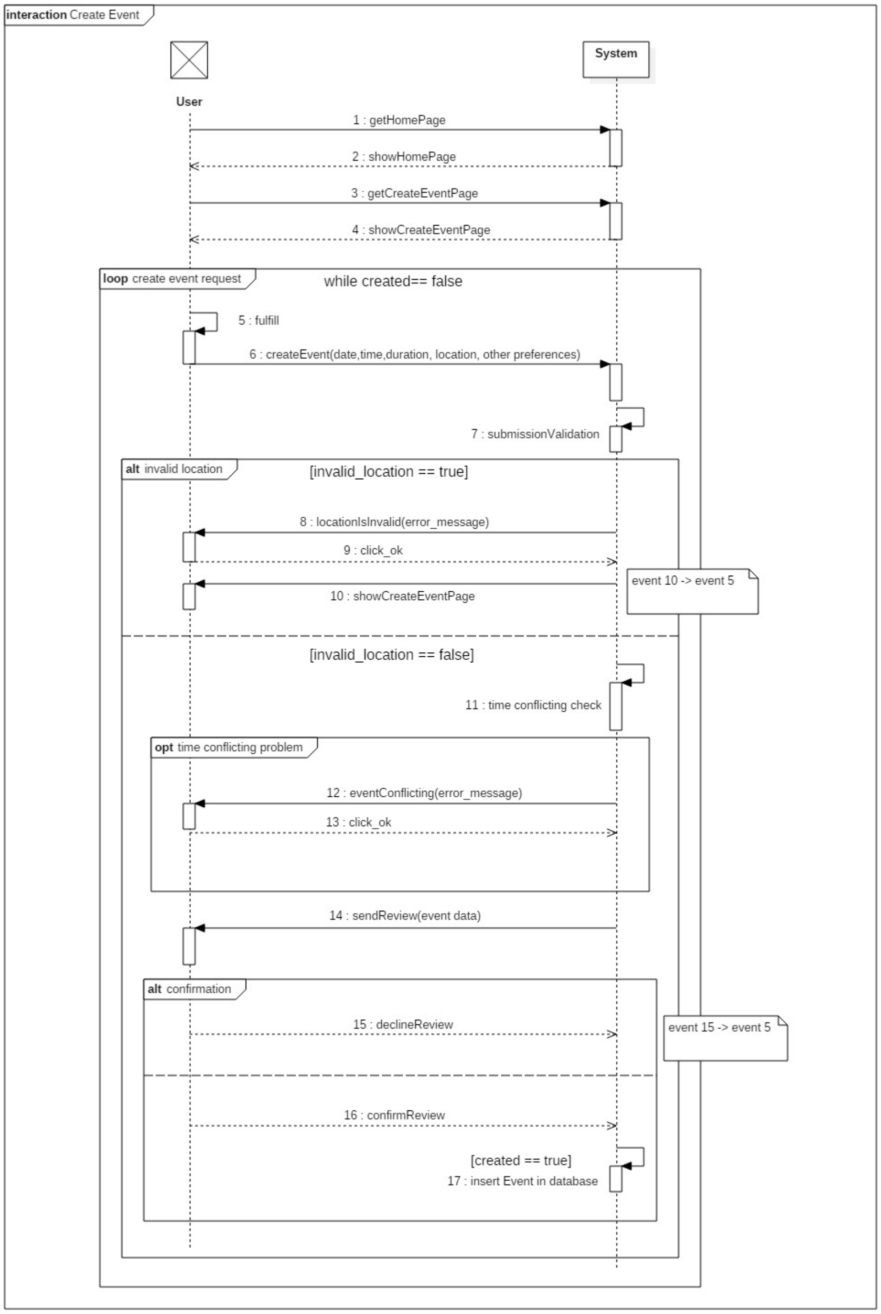


Figure 22: Create Event Sequence Diagram

4.2.4 Accept Path

Name	Accept Path
Actors	Registered User Google Maps
Assumptions	The user is logged in the system. The user has successfully created an event.
Flow Of Events	The system receives the new event The system checks the preferences of the user. The system will send the information of the event to Google Maps. Google Maps will calculate a path. Google Maps will send the path to the system. The system will send the path to the user. The user accepts the path. The system verifies its compatibility in the users' schedule.
Exit Condition	The path is created.
Exceptions	The user doesnt accept the path. The system will be notified and will have to start again by sending the information to Google Maps. The system, after learning from Google Maps, realises there are incompatible events and send a warning to the user. The user will have to select which event is more important. The system will send the new information to Google Maps and start again.

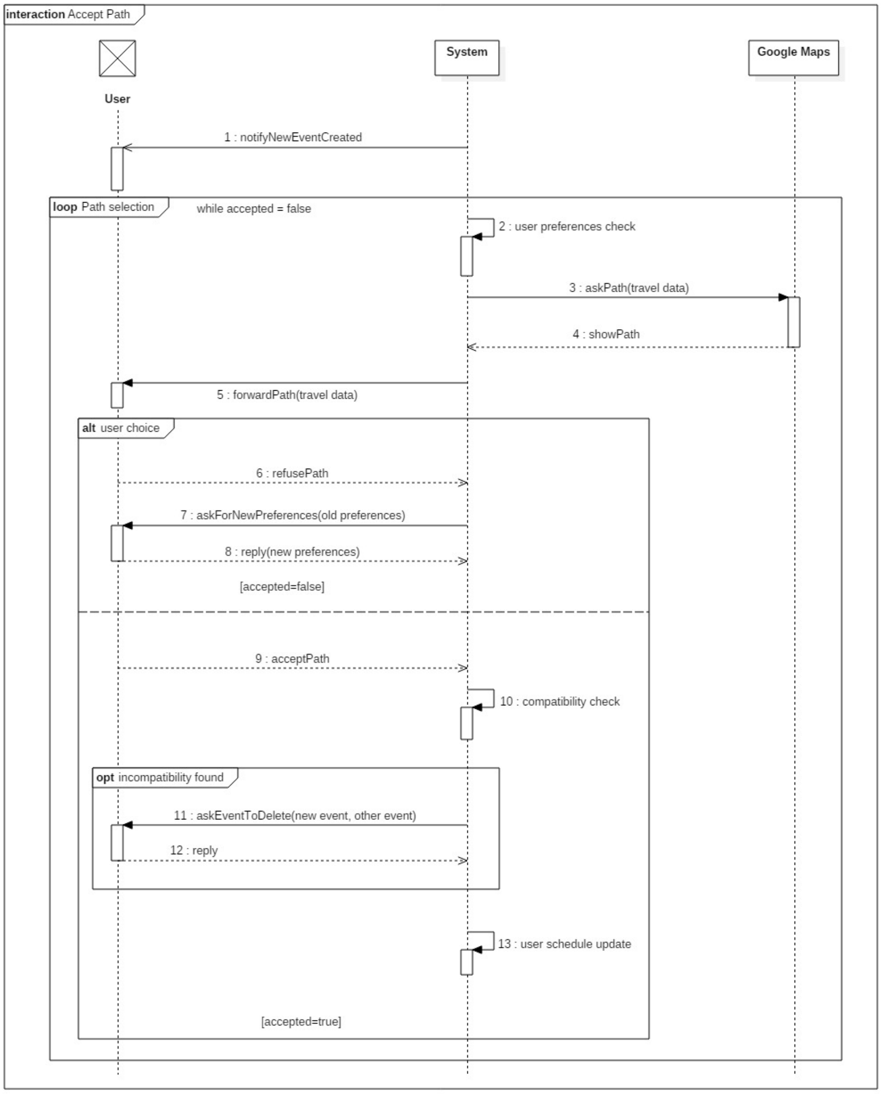

interaction Accept Path
User
System
Google Maps
1 : notifyNewEventCreated
loop Path selection
while accepted = false
2 : user preferences check
3 : askPath(travel data)
4 : showPath
5 : forwardPath(travel data)
alt user choice
6 : refusePath
7 : askForNewPreferences(old preferences)
8 : reply(new preferences)
[accepted=false]
9 : acceptPath
10 : compatibility check
opt incompatibility found
11 : askEventToDelete(new event, other event)
12 : reply
13 : user schedule update
[accepted=true]

Figure 23: Accept Path Sequence Diagram

4.2.5 Buy Ticket

Name	Buy Ticket
Actors	Registered User Public Transportation System
Assumptions	The user is logged in the system. The user has opened a balance in the app or has a credit/debit card connected to it.
Flow Of Events	The user clicks on buy a ticket from the homepage of the app. The user selects the ticket he wants to buy. The system receives the request and forwards it to the correct public transportation system. The public transportation system sends the information to Travlendar+s system to complete the transaction. The system sends the information to the user. The user accepts the payment request. The user selects the payment option. The system sends the payment confirmation to the PTS. The PTS confirms the ticket. The PTS sends the ticket information to the system. The system saves the ticket in the users information.
Exit Condition	The ticket is available to be used by the user.
Exceptions	The balance does not have enough money to buy the ticket. The system will ask the user to select a different payment method. The credit card does not allow transaction. The system will ask the user to select a different payment method. The connection is lost between the system and the PTS. The system will have to ask to the user to start again from the selection of the ticket.

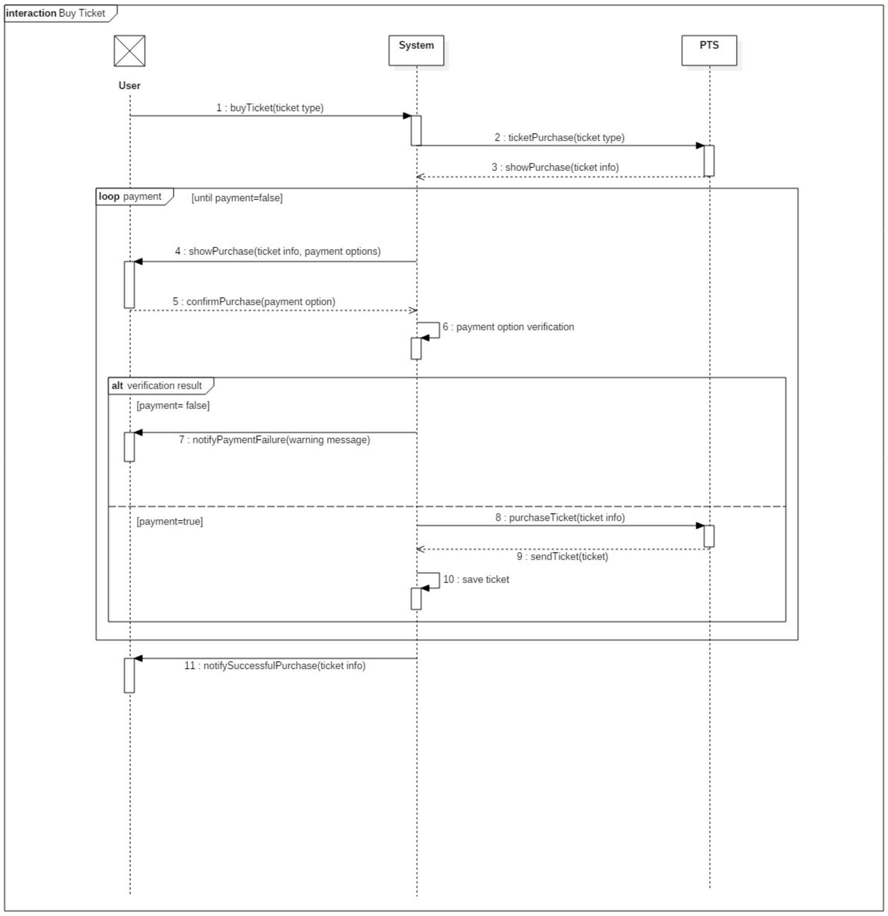

interaction Buy Ticket
User
System
PTS
1 : buyTicket(ticket type)
2 : ticketPurchase(ticket type)
3 : showPurchase(ticket info)
loop payment
[until payment=false]
4 : showPurchase(ticket info, payment options)
5 : confirmPurchase(payment option)
6 : payment option verification
alt verification result
[payment= false]
7 : notifyPaymentFailure(warning message)
[payment=true]
8 : purchaseTicket(ticket info)
9 : sendTicket(ticket)
10 : save ticket
11 : notifySuccessfulPurchase(ticket info)

Figure 24: Buy Ticket Sequence Diagram

4.2.6 Activate Ticket

Name	Activate Ticket
Actors	Registered user PTS
Assumptions	The user has logged in the system. The user has bought at least one ticket. The ticket has not been activated yet.
Flow Of Events	The user has to click on My tickets in his personal page. The system will send to the user the information about his ticket. The user will select the ticket that he wants to activate. The user will press on Activate ticket. The system will send a confirmation message to the user with a summary about the ticket. The user confirms the ticket. The system sends the details about the activation to the PTS. The PTS will send a confirmation of the activation of the ticket. The system sends the confirmation to the user. The system will tag the ticket as activated.
Exit Condition	The ticket is activated and it becomes valid for the duration of the ticket.
Exceptions	The user doesnt confirm the ticket. The system will bring the user back to the selection of the ticket. There is a connection problem between the system and the PTS. The user will be notified of this and he will have to start the process again from the selection of the ticket.

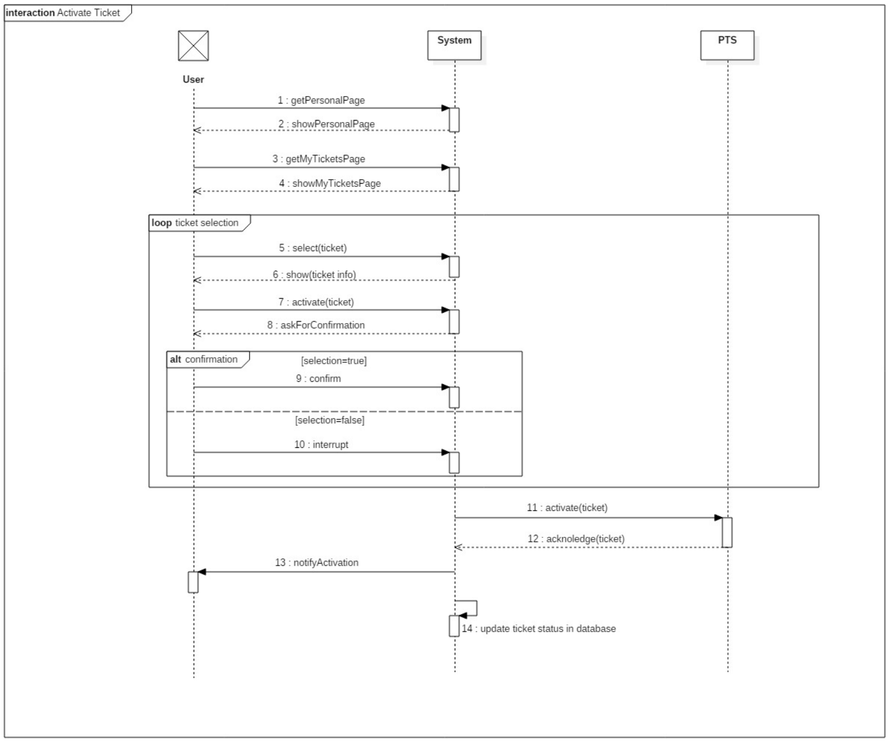


Figure 25: Activate Ticket Sequence Diagram

4.2.7 Find Vehicle And Access Service

Name	Find Car/Bike And Access Service
Actors	Registered user Car-Sharing system (CSS) or Bike-Sharing system (BSS)
Assumptions	The user is logged in the system. The user has a driving license. The information provided by the CSS are true. The users GPS is working and has the correct position.
Flow Of Events	The user has to click on Find a vehicle in the homepage of the app. The user has to select Car or Bike. The system sends a request for information to the CSS/ BSS. The CSS/ BSS sends the information needed to the system. The system will redirect the user to a map of his surroundings, with system. Once a vehicle is chosen, the system will redirect to that vehicles service app.
Exit Condition	The user will be able to see available cars from different car sharing systems near his location and pick one.
Exceptions	There is a connection problem between the CSS(BSS) and Travlendar+s system. The system will notify the user. The user will have to start again.

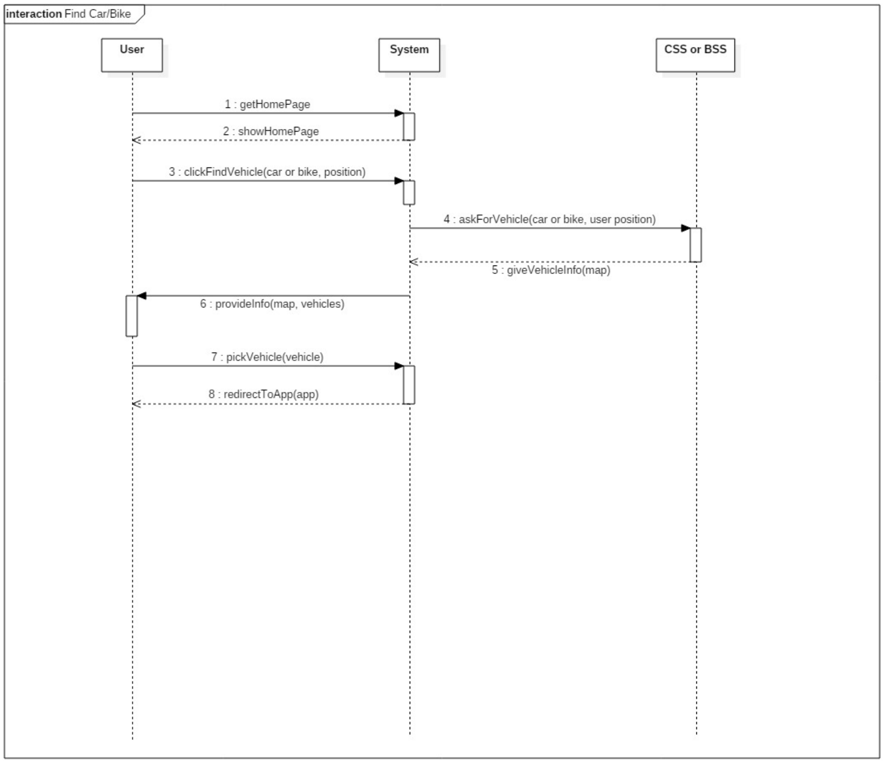

interaction Find Car/Bike
User
System
CSS or BSS
1 : getHomePage
2 : showHomePage
3 : clickFindVehicle(car or bike, position)
4 : askForVehicle(car or bike, user position)
5 : giveVehicleInfo(map)
6 : provideInfo(map, vehicles)
7 : pickVehicle(vehicle)
8 : redirectToApp(app)

Figure 26: Find Vehicle And Acceess Service Sequence Diagram

4.2.8 Load Balance

Name	Load Balance
Actors	Registered User Bank
Assumptions	The user has a credit/debit card connected to the account or The user has bought a code to load the balance from a certified shop.
Flow Of Events	The user has to click on Load balance. The system will redirect him to the loading balance page. The user will have to select Credit/debit Card or Use a code. 1. Credit/Debit Card case: The user has to select the amount of money he wishes to load into the balance. The system sends the request to the bank. The bank confirms the transaction. 2. Use a code: The system asks the user to insert the code written on the card he bought. The user inserts the information requested and confirms. The system checks that the information is correct. The system sends a confirmation message to the user.
Exit Condition	The balance is updated.
Exceptions	The bank does not allow the transaction. The system requests the user to try again. The information sent by the user are incorrect. The system asks the user to check if the code is correct and to try again.

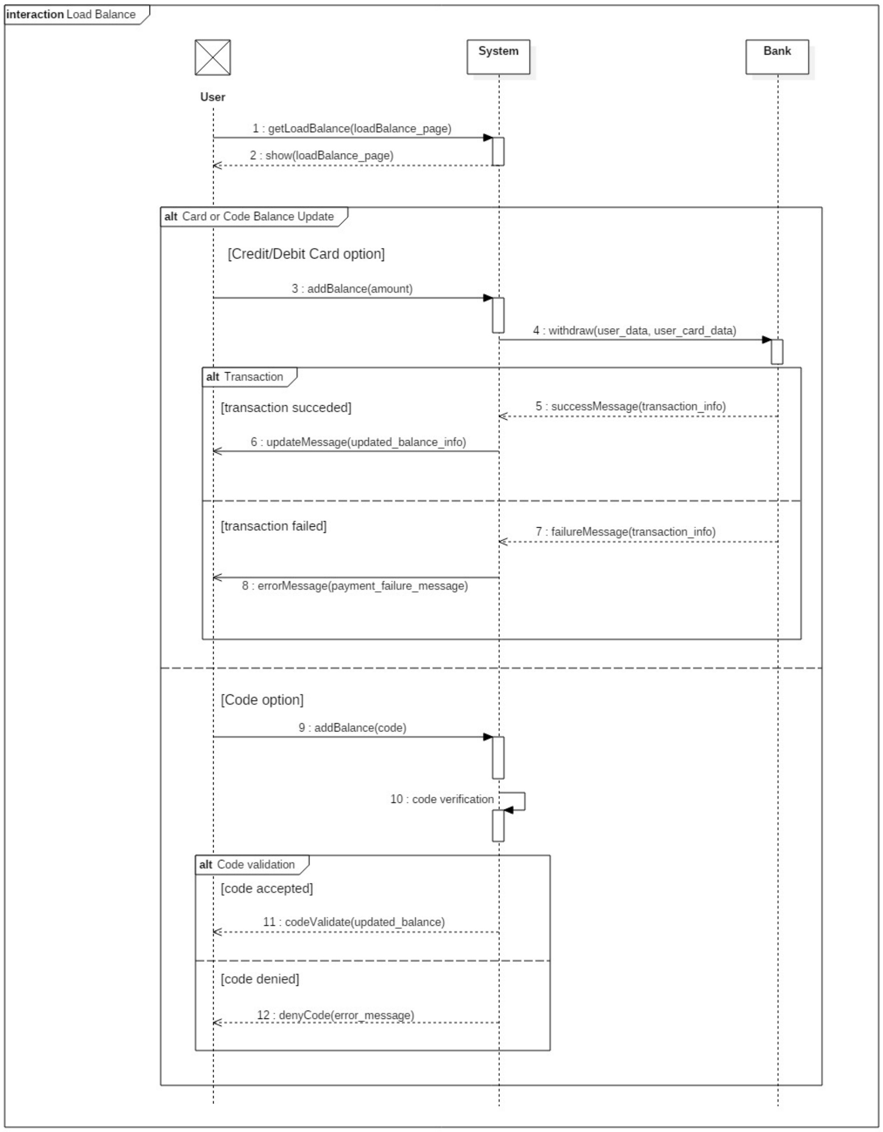

interaction Load Balance
User
System
Bank
1 : getLoadBalance(loadBalance_page)
2 : show(loadBalance_page)
alt Card or Code Balance Update
[Credit/Debit Card option]
3 : addBalance(amount)
4 : withdraw(user_data, user_card_data)
alt Transaction
[transaction succeded]
5 : successMessage(transaction_info)
6 : updateMessage(updated_balance_info)
[transaction failed]
7 : failureMessage(transaction_info)
8 : errorMessage(payment_failure_message)
[Code option]
9 : addBalance(code)
10 : code verification
alt Code validation
[code accepted]
11 : codeValidate(updated_balance)
[code denied]
12 : denyCode(error_message)

Figure 27: Load Balance Sequence Diagram

5 Formal Analysis Using Alloy

5.1 Model Representation

This Alloy document generates a model of the system in a precise moment. It represents a photography of the system and deals more with its global coherence than with the integrity of the transitions. In short, the most important properties of the model are:

- Users and external actors belong to the same system (Travlendar+).
- The users are distinguished in guest and registered users. Since only the registered ones can operate on the system, we are referring to them saying "users".
- Every user can define some settings and the system must not violate them when it suggests some paths.
- Every user owns a schedule containing every event they need to attend.
- The system suggests some paths for the users' events, and those paths are grouped in the users' daily paths.
- The system can suggest an alternative path for an event only if the main path has been refused or it is not "in time".
- The external actors determine the presence of their transports as options in t he system and (some of the actors) the possiblity to buy tickets for those transports.

Alloy Code
The following is the source code of our alloy model.

```
/*
                        Travlendar+
                        ---Alloy project---

                        Paolo Fumagalli
                        Pietro Grotti
                        Marco Gullo
*/

module TravlendarPlus
open util/boolean

                        // SYSTEM

// Encloses all the users and the external actors
one sig System {
        users: set User,
        externals: set Actor
}

                        // EXTERNAL ACTORS

// Represents an external actor
abstract sig Actor {}

// All the kinds of external actors:
// >
sig PublicTransports extends Actor {
        service: one Public, // This relation links the actor with the service provided
        issued: some PublicTransportTicket // This relation defines the origin of the
emitted tickets
}

sig CarSharingService extends Actor {
        service: one CarSharing
}

sig BikeSharingService extends Actor {
        service: one BikeSharing
```

```
}

sig TrainService extends Actor {
        service: one Train,
        issued: some TrainTicket
}

sig TaxiService extends Actor {
        service: one Taxi
}
// <

        // ...facts

// All the actors considered belong to the system
fact actorsInSystem {
        all a: Actor | a in System.externals
}

                        // TRANSPORT

// Represents the generic mean of transport
abstract sig Transport {}

// The transports which don't need an external actor in the system (they always
exist and they are unique)
one sig Car, Walking extends Transport {}

// The transports which DO need an external actor in the system (they exist with
an arity of 1 only if their provider exists)
lone sig Public, CarSharing, BikeSharing, Train, Taxi extends Transport {}

        // ...facts

// The actors and their services exist only in pairs
fact noTransportWithoutService {
        all t: Public | one s: PublicTransports | t in s.service
        all t: CarSharing | one s: CarSharingService | t in s.service
        all t: BikeSharing | one s: BikeSharingService | t in s.service
        all t: Train | one s: TrainService | t in s.service
        all t: Taxi | one s: TaxiService | t in s.service
}

                        // USER

// Represents a user in the system
abstract sig User {}

// A user registered in the system
sig Registered extends User {
        schedule: one Schedule, // Personal schedule containing the events
        dailyPath: one DailyPath, // Personal list of paths
        settings: one Settings, // Personal settings
        tickets: some Ticket, // Bought tickets
        balance: one Balance // Personal balance
}

// This signature represents the balance of the user without specifying it
sig Balance {}

// A user who doesn't own an account
sig Guest extends User {}

        // ...facts

// The considered users are inside the system
fact usersInSystem {
        all u: User | u in System.users
}

// A "balance" belongs to a user
fact balanceOfUser {
        all b: Balance | one u: Registered | b in u.balance
}

                        // DAY TIME
                        // A discrete representation of the day
// Represent a discrete time of the day
abstract sig DayTime {}

// Times of the day considered
one sig Morning, Lunch, Afternoon, Evening, Night extends DayTime {}

                    // DISTANCE

// Represents a generic distance
abstract sig Distance {}

// Kinds of distance
one sig Short, Medium, Long extends Distance {}

                    // USER SETTINGS

// The personal settings
sig Settings {
	refuses: some Transport, // defines the means of transport a user doesn't
want to use
	constraints: some TimeConstraint, // defines some day times a user
doesn't want to use a kind of transport
	lunchConstraint: one Bool, // defines if the user wants to establisha   lunch
break
	walkConstraint: lone Bool, // defines if the user wants to walk only for
short paths
	bikeConstraint: lone Bool // defines if the user wants to use the bike only
for short paths
}

	// ...facts

// Any setting belongs to one user
fact uniqueSettings {
	all s: Settings | one u: User | s in u.settings
}

// A user can't define a constraint on biking or walking if he refuses them
fact walkAndBikeConstraints {
        all s: Settings | Walking in s.refuses => #s.walkConstraint = 0 and
BikeSharing in s.refuses => #s.bikeConstraint = 0
}

                        // TIME CONSTRAINTS
                        // Defines the day times the user wants to avoid a kind of
transport

// Constraint used in personal settings (see Settings)
sig TimeConstraint {
        transport: one Transport,
        time: some DayTime // denied day times
}

        // ...facts

// Every constraint belongs to some settings
fact constraintsForSettings {
        all c: TimeConstraint | one s: Settings | c in s.constraints
}

// A user can't define a constraint on mean of transports he refused
fact noConstraintsOnRefuses {
        all s: Settings | all t: Transport | t in s.refuses => t not in
s.constraints.transport
}

// A user can't define multiple contraints on the same mean of transport
fact noDifferentConstraintsOnTheSameTransport {
        all c1, c2: TimeConstraint | all t: Transport | all s: Settings | (c1 != c2
and c1 in s.constraints and c2 in s.constraints and t in c1.transport) => t not in
c2.transport
}

                        // SCHEDULE AND DAILY PATH

// Gathers all the events belonging to a user
sig Schedule {
        events: set Event
```

```
}

// Gathers all the paths suggested for the events in a schedule
sig DailyPath {
        paths: set Path
}

        // ...facts

// Daily paths belong to one user
fact DailyPathForUsers {
        all d: DailyPath | one u: Registered | d in u.dailyPath
}

// Schedules belong to one user
fact ScheduleForUsers {
        all s: Schedule | one u: Registered | s in u.schedule
}

                        // EVENT

// An event defined by a user in the system
sig Event {
        path: lone Path, // The main path suggested
        alternative: lone Path, // The alternative path suggested
        time: one DayTime, // The time of the day the event occurs
        distance: one Distance // The distance form the previous event or the
expected user's location
}

        // ...facts

// Every event belongs to a schedule
fact eventsAssociation {
        all e: Event | one s: Schedule | e in s.events
}

// If the user set a lunch break, deny any event at that time and deny events at
long distances in the afternoon
```

```
fact eventsAtLunch {
        all u: Registered | all e: Event | (u.settings.lunchConstraint = True and e
in u.schedule.events) => (Lunch not in e.time and (Long in e.distance =>
Afternoon not in e.time))
}

                        // PATH

// A path suggested to go and take part to an event
sig Path {
        transport: one Transport, // The mean of transport
        accepted: one Bool, // Accepted by the user
        inTime: one Bool // Defines if the user will be able to be in time following
this path
}

        // ...facts

// A path belongs to one event, and it can't be the main path and the alternative
at the same time
fact pathUnicity {
        all p: Path | one e: Event | p in e.path or p in e.alternative
        all p: Path | all e: Event | all u: Registered | ((p in e.path or p in
e.alternative) and e in u.schedule.events) => p in u.dailyPath.paths
        all p: Path | all e: Event | (p in e.path => p not in e.alternative) and (p
in e.alternative => p not in e.path)
}

// An alternative is suggested only if another path was suggested before
fact alternativesAfterPaths {
        all e: Event | #e.path = 0 => #e.alternative = 0
}

// An alternative path must consider a different mean of transport from the main
path's one
fact alternativesSuggestDifferentTransports {
        all e: Event | all t: Transport | t in e.path.transport => t not in
e.alternative.transport
}
```

```
// Paths can't contain a refused mean of transport
fact dontSuggestRefusedTransports {
        all u: Registered | all t: Transport | all e: Event | all p: Path | (e in
u.schedule.events and (p in e.path or p in e.alternative) and t in
u.settings.refuses) => t not in p.transport
}

// A path must be by train or car if the distance is "Long"
fact trainOrCarForLongDistances {
        all e: Event | all p: Path | ((p in e.path or p in e.alternative) and Long in
e.distance) => (Train in p.transport or Car in p.transport)
}

// If the user set a walk constrain, consider "Walking" only for short distances
fact walkConstraint {
        all p: Path | all e: Event | all u: Registered | (e in u.schedule.events and
(p in e.path or p in e.alternative) and u.settings.walkConstraint = True and
Walking in p.transport) => Short in e.distance
}

// If the user set a bike constrain, consider "Bike" only for short distances
fact bikeConstraint {
        all p: Path | all e: Event | all u: Registered | (e in u.schedule.events and
(p in e.path or p in e.alternative) and u.settings.bikeConstraint = True and
BikeSharing in p.transport) => Short in e.distance
}

// Don't violate user's time contraints in the suggested paths
fact timeConstraint {
        all u: Registered | all e: Event | all p: Path | all c: TimeConstraint | all d:
DayTime | all t: Transport | (t in c.transport and d in c.time and c in
u.settings.constraints
                and e in u.schedule.events and (p in e.path or p in e.alternative)
and d in e.time) =>
                        t not in p.transport
}

// An alternative is suggested only if the main path is refused or if the user is late
```

```
fact suggestAlternativesIfLateOrRefused {
        all p: Path | all e: Event | (p in e.path and p.accepted = True and
p.inTime = True) => #e.alternative = 0
}

                        // TICKETS

// Represent a ticket bought by a user
abstract sig Ticket {}

// Public transport ticket (bus, undergrount...)
sig PublicTransportTicket extends Ticket {}

// Represent a generic train ticket
abstract sig TrainTicket extends Ticket {}

// The kinds of train tickets
sig SingleTrainTicket, DailyTrainTicket, MonthlyTrainTicket extends TrainTicket
{}

        // ...facts

// Every ticket in the system belongs to one user
fact ticketsForUsers {
        all t: Ticket | one u: Registered | t in u.tickets
}

// A ticket can be bought only if there's an external actor providing it
fact availableTickets {
        all t: TrainTicket | one s: TrainService | t in s.issued
        all t: PublicTransportTicket | one s: PublicTransports | t in s.issued
}

// A user who bought a monthly ticket or a daily ticket can't buy other train
tickets of the same type
fact oneTrainTicket {
        all u: Registered | all d: DailyTrainTicket | all t: TrainTicket | t!=d and d
in u.tickets => t not in u.tickets
        all u: Registered | all m: MonthlyTrainTicket | all t: TrainTicket | t!=m and
```

```
m in u.tickets => t not in u.tickets
}

                        // COMMANDS AND PREDICATES

// There isn't any event in the system
pred noEvents {
        #Event = 0
}

// There are events in the system
pred eventsExisting {
        #Event > 2
}

// There isn't any registered user in the system
pred noRegisteredUsers {
        #Registered = 0
}

// There isn't any external acotr in the system
pred noExternalActors {
        #PublicTransports = 0 and #BikeSharingService = 0 and
#CarSharingService = 0 and #TrainService = 0 and #TaxiService = 0
}

// The system suggests acceptable alternative paths for some events
pred alternativesExisting {
        some e: Event | #e.alternative = 1
        all p: Path | all e: Event | p in e.alternative => p.accepted = True and
p.inTime = True
}

// Some users bought some tickets
pred ticketsExisting {
        #TrainTicket > 0 and #PublicTransportTicket > 0
        #Actor = 2
        #Registered = 1
        #Path > 0
```

```
}

//      run noEvents
//      run eventsExisting
//      run noRegisteredUsers
//      run noExternalActors
//      run alternativesExisting
//      run ticketsExisting
        run {}
```

5.2 Instances

5.2.1 Instance 1

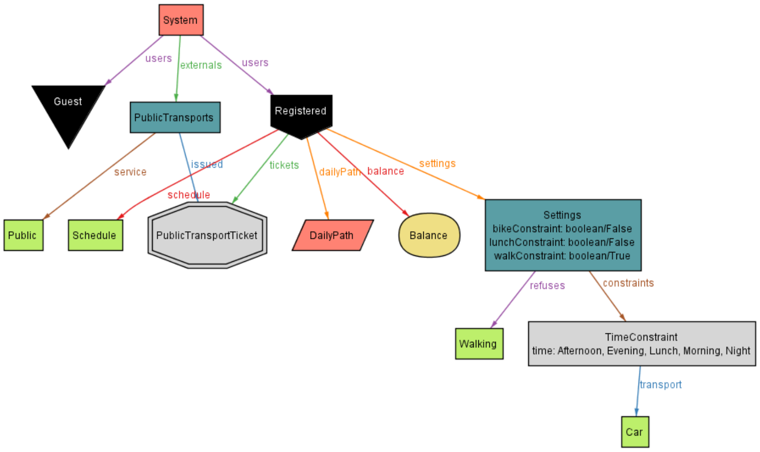


Figure 28: First Instance

Instance 1

In this instance there are no users events, so the system doesnt generate any suggested path. However, there exist users settings, schedules, daily paths and systems external actors. The system is ready to add an event in the registered users schedule.

5.2.2 Instance 2

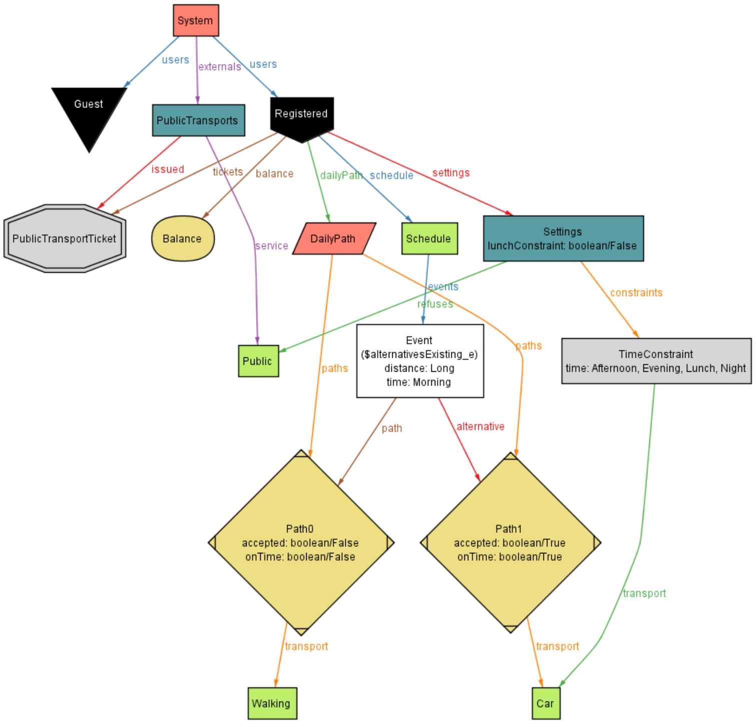


Figure 29: Second Instance

Instance 2

In this instance, the registered user has one event in his schedule. The system suggests a main path for this event, but it is neither in time or accepted by the user. So, the system suggests an alternative path for the same event, which allows the user to be in time.

5.2.3 Instance 3

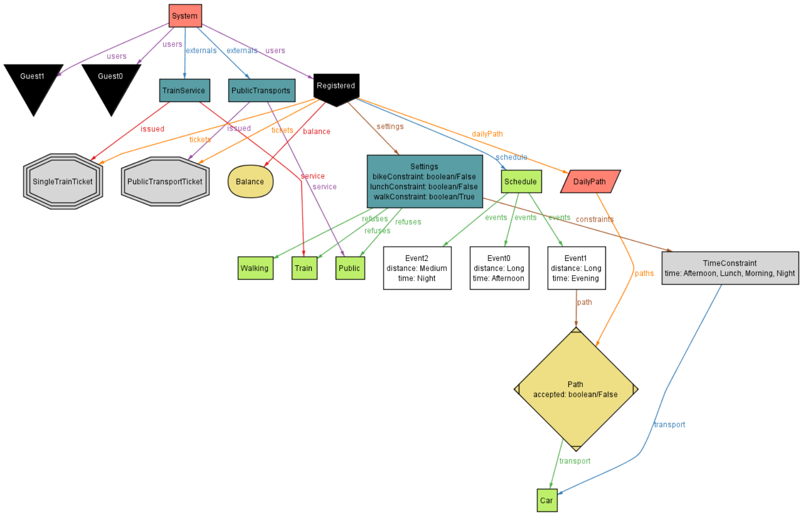


Figure 30: Third Instance

Instance 3 In this instance, both Public Transports and Train Service exist as external actors in the system. The registered user owns in his account a single train ticket and a public transports ticket. The link between the tickets and the actors which allowed their purchase is highlighted by the relation issued.

6 Effort Spent

- Fumagalli Paolo: 25 hours.
- Grotti Pietro: 25 hours.
- Gullo Marco: 25 hours.

7 References

7.1 Tools Used

- Google Doc
- Signavio
- Alloy Analyzer 4.2
- StarUML 5.3
- TeXworks