**Politecnico di Milano**

Software Engineering 2

# Requirements Analysis and Specifications Document

Version 1.0 - 29/10/2017

*Pietro Melzi, Alessandro Pina, Matteo Salvadore*

AA 2017-2018

# Contents

**1 INTRODUCTION 3**
1.1 Purpose 3
1.2 Scope 3
1.2.1 World and shared phenomena 4
1.2.2 Goals 4
1.3 Definitions, Acronyms, Abbreviations 5
1.3.1 Definitions 5
1.3.2 Acronyms 6
1.3.3 Abbreviations 6
1.4 Revision history 6
1.5 Reference Documents 7
1.6 Document Structure 7

**2 OVERALL DESCRIPTION 8**
2.1 Product perspective 8
2.1.1 Class Diagram 9
2.1.2 Statechart Diagram 10
2.2 Product functions 10
2.2.1 Registration and login 10
2.2.2 Event insertion 10
2.2.3 Schedule management 11
2.2.4 Trips arrangement 11
2.2.5 Travel means 11
2.2.6 Sharing vehicle location 11
2.3 User characteristics 11
2.4 Assumptions, dependencies and constraints 11

**3 SPECIFIC REQUIREMENTS 13**
3.1 External Interface Requirements 13
3.1.1 User Interfaces 13
3.1.2 Hardware Interfaces 16
3.1.3 Software Interfaces 16
3.1.4 Communication Interfaces 16
3.2 Functional Requirements 16
3.2.1 Detailed list 16
3.2.2 Scenarios 19
3.2.3 Use case descriptions 22
3.2.4 Traceability matrix 32
3.2.5 Use case diagrams 33
3.2.6 Sequence diagrams 35

3.3 Performance Requirements . . . . . 39
3.4 Design Constraints . . . . . 40
3.4.1 Standards compliance . . . . . 40
3.4.2 Hardware limitations . . . . . 40
3.4.3 Any other constraint . . . . . 40
3.5 Software System Attributes . . . . . 40
3.5.1 Reliability . . . . . 40
3.5.2 Availability . . . . . 40
3.5.3 Security . . . . . 41
3.5.4 Maintainability . . . . . 41
3.5.5 Portability . . . . . 41

**4 FORMAL ANALYSIS USING ALLOY 42**
4.1 Model . . . . . 42
4.2 Proof of Consistency . . . . . 49
4.3 Worlds obtained . . . . . 50
4.3.1 World 1 . . . . . 50
4.3.2 World 2 . . . . . 50
4.3.3 World 3 . . . . . 50

**5 EFFORT SPENT 52**
5.1 Pietro Melzi . . . . . 52
5.2 Alessandro Pina . . . . . 53
5.3 Matteo Salvadore . . . . . 53

**6 REFERENCES 54**
6.1 Bibliography . . . . . 54
6.2 Software used . . . . . 54

# Chapter 1

# INTRODUCTION

## 1.1 Purpose

This document provides a baseline for the project planning of Travlendar+, the system we want to develop. It analyses the environment in which the system is intended to be used and contains information about the offered functions.

The application wants to be a useful tool that can be used both in work time and in free time, helping people to organize daily appointments and travels between them.

Travlendar+ is a sort of online calendar: it substitutes the standard old agenda, where in the past you were used to circle important days and write down events' information in order to remember them, and enhances it, bringing together various useful utilities that otherwise you would be forced to use separately.
People currently use different services, provided by calendar and maps applications, in order to organize meetings and events. Travlendar+ includes all these useful functionalities in the same application, helping the user planning the day.

## 1.2 Scope

Travlendar+ is a service based on a mobile application and a web application.

The user simply has to create and add events into his schedule; the application will then, in order to reach locations in the most efficient way, organize travels, correctly inserting them within the daily schedule and notifying the user about eventual overlappings.
The user can modify the schedule in the way he prefers and can specify different types of preferences on travel means, so that the system can plan the trip according to his personal needs.

The principal goal of Travlendar+ is to save time, both in the construction of the schedule and in the traveling between events.
The main functionality of the system is to control the user's daily flow of events, helping him optimizing his time and making sure that all his preferences and constraints are respected.
Integration with external transport means providers is also offered, allowing the user to buy tickets and locate travel means without exiting the app.

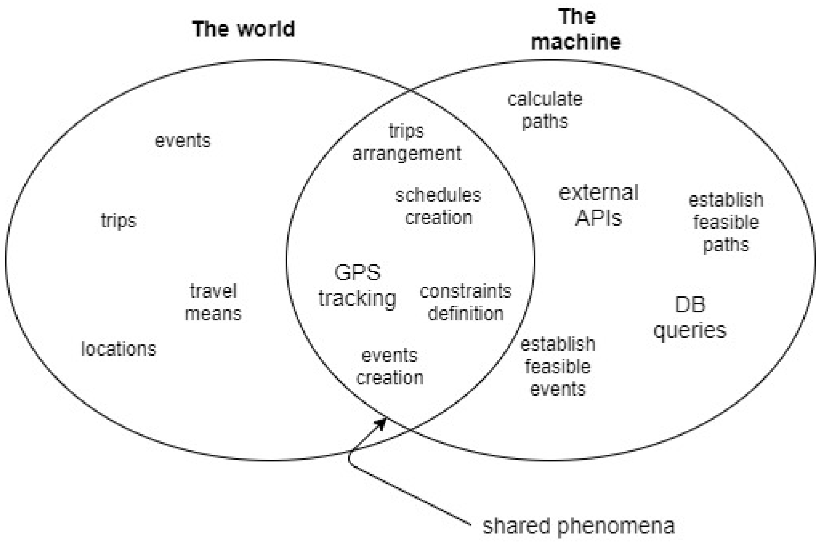
### 1.2.1 World and shared phenomena

This is an overview of the world in which Travlendar+ is intended to exist.
The system aims to plan events and appointments of the user in a way that allows him to attend every single one of them. Therefore, the world is made of entities related to travel.

Shared phenomena allow the user to achieve his goals and affect both user and system behaviors:

- Trips arrangement and schedules creation are phenomena controlled by the machine and observed by the world;
- Constraints definition and events creation are phenomena controlled by the world and observed by the machine.

The machine encloses the set of functions used to perform the different tasks. These items are not visible to the world.



### 1.2.2 Goals

A *visitor* should be able to:

**[G1** ] sign up into the system;

A *user* should be able to:

**[G2** ] log into the system;

**[G3** ] create an event specifying its location, date, starting and ending time;

**[G4** ] obtain the best travel path (according to his preferences) and the list of eventual feasible alternatives to reach a location;

**[G5** ] change the selected path with an alternative one;

**[G6** ] obtain a daily schedule that allows him to attend every event in program;

**[G7** ] apply constraints on travel means;

  **[G7.1** ] related to the length of travel;

  **[G7.2** ] related to the period of day;

**[G8** ] deactivate one or more travel means;

**[G9** ] select combinations of transportation means that minimize carbon footprint;

**[G10** ] reserve time for lunch or break events;

**[G11** ] arrange trips;

  **[G11.1** ] buy needed tickets;

  **[G11.2** ] find available sharing vehicles.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- *Arrange trip*: the system provides all the information available about a travel to the user. If the travel is related to one or more public travel means, the system helps the user to organize it. It is indicated if the user already holds a valid ticket for a certain travel or if he has to buy a required one. Purchase of a ticket is handled on the websites of the transport service providers. Trip and travel are synonymous.

- *Best path*: it is the preferred travel option proposed to reach the event location among all the feasible paths, according to parameters specified by the user. The best path can be chosen according to one of these features: length, cost or environmental sustainability. Best path is the path taken into account and showed in the daily schedule; the user can substitute it at any moment with an alternative feasible path.

- *Break event*: it is an optional event whose starting and ending time are flexible. The user can define this kind of event when he wants to reserve a certain amount of time in the schedule and he doesn't need to specify a starting time. For instance, a user could be able to specify that lunch must be possible every day between 11:30- 2:30, and it must be at least half an hour long, but the specific timing is flexible. The app would then be sure to reserve, if possible, at least 30 minutes for lunch each day.

- *Constraint*: it is a rule defined on travel means by the user. When the system calculates feasible paths, each constraint must be taken into account: travel options not respectful of existing constraints are ignored. A constraint can be associated to a type of event.
  There are different types of constraints: min/max distance allowed for a travel mean, interval of hours in which it is possible to take a travel mean and possibility to deactivate a travel mean.

- *Customized type of event*: it is a set of rules related to a particular type of event that can be used several time by the user. The user can define a customized type of event in hs preferences, starting from the default type of event.

- *Default type of event*: it is a general set of rules defined usually the first time that the user exploits the system, it can be modified. Default type of event is related to each event that doesn't need a particular type of event. New types of event are defined starting from the constraints enclosed in the default one.

- *Feasible path*: a path that allows the user to reach a specified location before the starting time of the event to attend. It observes the constraints defined for that event.
- *Google Maps API*: a set of functions offered by Google Maps to decide the travel path between two locations.
- *Overlapping event*: an event (or a part of it) that happens in the same time slot of another event, added previously in the schedule. Because the schedule must be feasible, an event is considered as ”overlapping” also if only its related travel overlaps an event present in the schedule.
- *Periodicity*: it is related to events that occur more than one time. It can be defined as weekly, monthly or in which days of the week the event happen.
- *Schedule*: a daily plan containing a set of events inserted by the user that allows him to travel and attend all the events. If an event overlaps other events, it cannot be inserted into the schedule.
- *Type of event*: a set of rules (constraints) that is defined by the user and can be associated to multiple events.
- *Transport service provider*: a public or private company that controls and supplies the transport with a travel mean.
- *Travel*: it is used to indicate the path that the user has to follows in order to reach a location. It can be composed by different travel components.
- *Travel component*: each single path that can be traveled with a travel mean. It has a starting time, a ending time, a departure location, an arrival location and a length. The union of one or more travel components creates a travel.

### 1.3.2 Acronyms

- *API*: Application Programming Interface;
- *DBMS*: Data Base Management System;
- *GTFS*: General Transit Feed Specification;
- *GDPR*: General Data Protection Regulation (EU privacy regulations);
- *RASD*: Requirement Analysis and Specification Document.

### 1.3.3 Abbreviations

- *Gn*: n-goal.
- *Dn*: n-domain assumption.
- *Rn*: n-functional requirement.
- *UCn*: n-use case.

## 1.4 Revision history

2017/10/29 - *Version 1.0* - First delivery of RASD.

## 1.5 Reference Documents

- Specification Document: "Mandatory Project Assignments.pdf";
- ISO/IEC/IEEE 29148:2011 Systems and software engineering - Life cycle processes - Requirements engineering (Document provided on beep course).

## 1.6 Document Structure

This RASD is composed by six parts:

1. The first part of the document contains an introduction to the system-to-be, where goals are identified and basic information is provided in order to better understand the rest of the document;
2. The second part of the document describes generally the system, identifying its boundaries and the actors involved in the life-cycle of the system. Boundaries are defined providing all the necessary assumptions: both the ones required in order to adapt to the user's specifications and the ones that will hold and from now on will be considered as true;
3. The third part of the document is composed by:
   - a definition of functional and non functional specific requirements;
   - a list of scenarios is provided: each of them describes a particular situation that the system might have to cope with;
   - UML diagrams that model in detail the system behavior.
4. The fourth part of the document contains the Alloy model of the system, including all the relevant details. It also provides a proof of consistency and some examples of the world generated;
5. The fifth part of the document contains a report of the hours spent to write this document;
6. The sixth part of the document contains references to external documents used in this document.

# Chapter 2

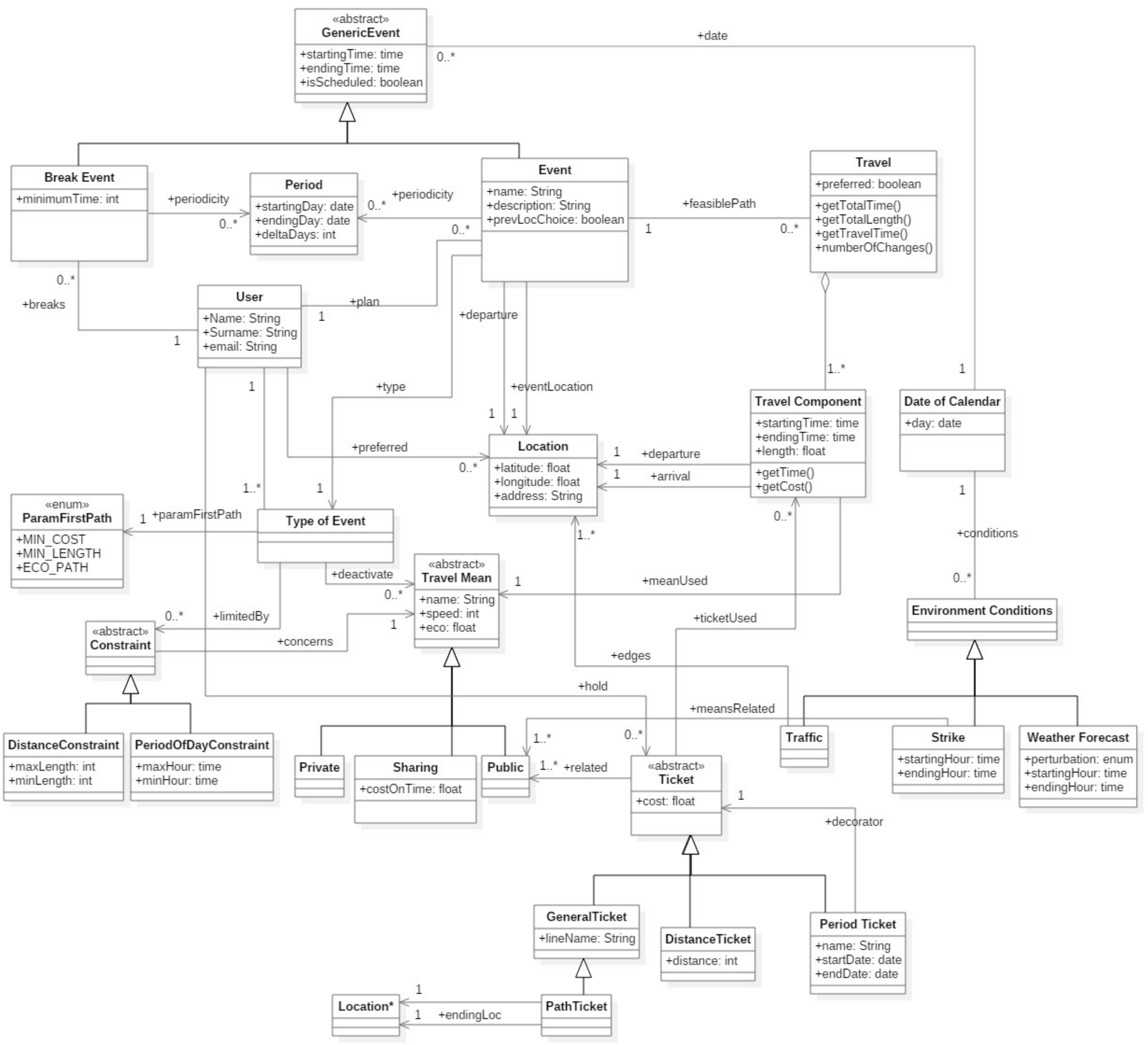
# OVERALL DESCRIPTION

## 2.1 Product perspective

The system will be developed from scratch and will implement these external services:

- feasible travels will be selected and elaborated according to the user preferences using the services provided by Google Maps APIs, eventually also using google traffic models to provide the most possible accurate travel times;
- the acquisition of tickets will be permitted by giving access to the websites of transport means providers;
- travel means will be proposed taking into account also the weather forecast provided by external weather APIs;
- all feasible travels that comprehend using public travel means will take into account GTFS and GTFS-Realtime feeds, provided by Google Transit APIs and eventually by Transport Service Providers proprietary APIs.

### 2.1.1 Class Diagram



An *Event* is associated to one or more *Travels*, a *Travel* proposed requires the use of one or more *Travel means*. The path of a *Travel* that include different *Travel Means* needs a structure where every *Travel Component* (the path that can be performed with the same *Travel Mean*), is memorized. Through a vector, an object of the *Travel* class contains each *Travel Component*.

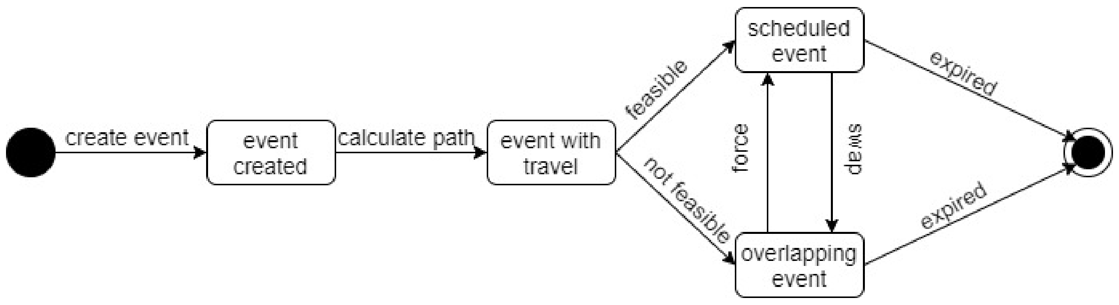
*Travel* class has the attribute *preferred*: a Boolean value that identifies the path showed in the daily schedule for an inserted *Event*. It must be checked that, for each *Event*, only one *Travel* is marked as preferred.

If a proposed path has different *Travel Components*, information about time and length are provided by methods of the *Travel* class that combine the values found in each related object of the *Travel Component* class.

To represent *Tickets* we use a hierarchy of classes: *Distance Ticket* is used for tickets whose validity is indicated with a length, *General Ticket* is used for particular tickets that allow the user to travel without constraints in a specified area, *Path Ticket* is used when departure and arrival locations are indicated on the ticket.
A *Ticket* can be valid into specified dates or after the validation, for this reason the management of the time of validity is done with *pattern Decorator*: it is used to specify the validity of an existing ticket.

NB: visibility of methods and attributes is not considered in this class diagram.

### 2.1.2 Statechart Diagram



## 2.2 Product functions

In order to better define clear boundaries for the development of the system, the main functions that will or will not be offered by the system will be explained clearly and synthetically in this section.

### 2.2.1 Registration and login

The system will allow visitors to register and perform a login in order to utilize the system functionalities.
The system will not allow visitors not yet signed in to utilize any functionality of the system, except for registration and login.

### 2.2.2 Event insertion

The system will allow the user to insert events, letting him define details (like, for example, type of event, locations, departure and ending time) and constraints concerning travel means.
If two events overlap with each other, the system will ask to the user which one does he want to give priority to.
The system will allow the user to create personal types of event. A type of event can indicate constraints and preferences specifically related to it.

### 2.2.3 Schedule management

The system will automatically manage the user's schedule respecting all his preferences and constraints added.
Events inserted by the user must always be visible, even those that do not show in the regular schedule because of overlaps, allowing the user to view and manage them.
If an event is added, modified or deleted, the system must update the schedule to assure that maximum efficiency is maintained.
The system will allow the user to reserve a time-slot in the schedule for lunches and breaks.
The system will try to compose the schedule maintaining a space between inserted events to let the user have his predetermined breaks.
The system will notify the user if he's trying to add an event that will prevent the presence of the break event.

### 2.2.4 Trips arrangement

The system will let the user check his travel arrangements.
For each trip the system will show the tickets needed to travel the selected path, specifying if the user needs to buy them. In that case, the option to acquire them on the website of the selected travel mean provider will be presented.

### 2.2.5 Travel means

The system will allow the user to apply constraints on travel means.
Travel means not desired by the user can be ignored during the definitions of a type of event.
The system will allow to add tickets and passes for travel means already in possession of the user, in order to avoid sustaining unnecessary costs.

### 2.2.6 Sharing vehicle location

The system will help the user locating available sharing vehicles, such as bikes and cars.
The system will not manage the vehicle sharing functionalities, such as reservations and payments. To perform these actions, the external application of the desired service is required.

## 2.3 User characteristics

- *Visitor*: a person using Travlendar+ without being registered or logged. The only actions that he can perform are registration and login.
- *User*: a registered person that successfully performed a login, able to fully utilize Travlendar+.

## 2.4 Assumptions, dependencies and constraints

Domain assumptions are:

**[D1** ] every inserted e-mail must be unique;

**[D2** ] every event is related to a location;

**[D3** ] an event must happen in an existing place;

**[D4** ] an event to attend cannot be in the past;

[**D5** ] the starting time of an event is always specified;

[**D6** ] the ending time of an event is not mandatory;

[**D7** ] a person can travel only on one travel mean at once;

[**D8** ] allowed travel means are cars, trains, metro, on foot, trams, bicycles, taxis, car sharing, bike sharing;

[**D9** ] every travel is programmed with a combination of one or more travel means;

[**D10** ] a person can be only in one place at once;

[**D11** ] every travel mean is related to information about its average carbon footprints;

[**D12** ] a flexible time-slot can have a daily or periodical validity;

[**D13** ] a flexible time-slot has a minimum amount of time that must be reserved;

[**D14** ] a ticket is required to use a public transport;

[**D15** ] a ticket may have a daily validity or a different periodicity;

[**D16** ] a user can own day/week/season passes;

[**D17** ] a payment is required to use a sharing vehicle.

# Chapter 3

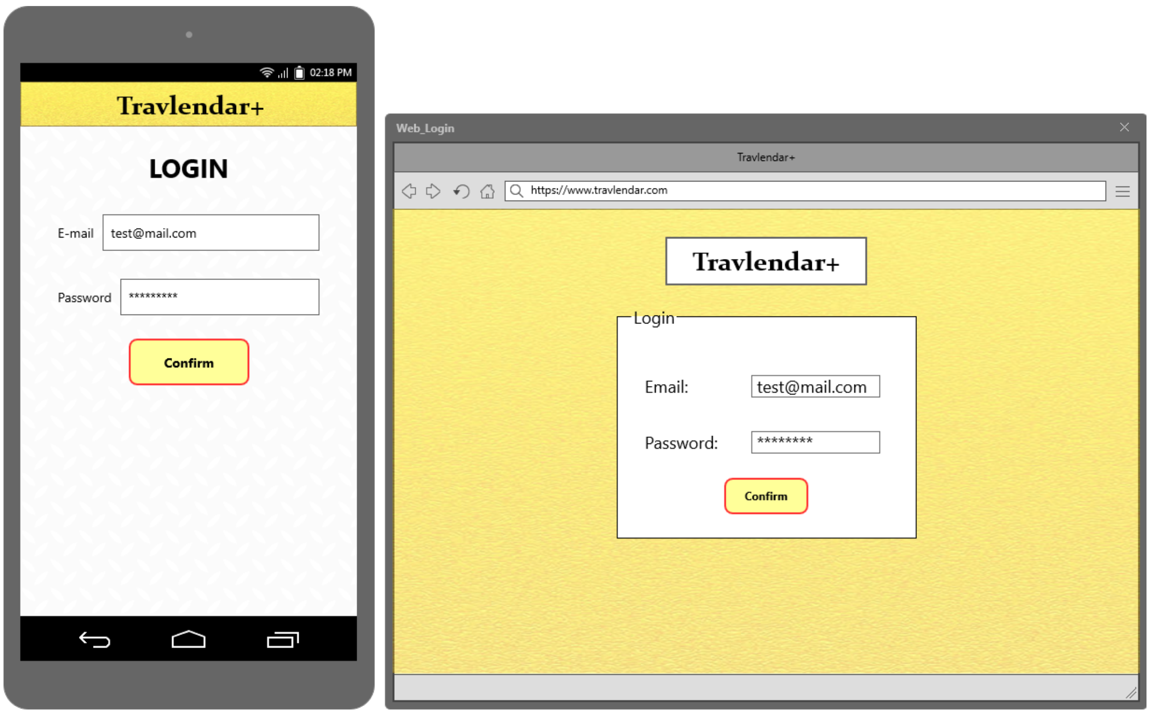
# SPECIFIC REQUIREMENTS

## 3.1 External Interface Requirements

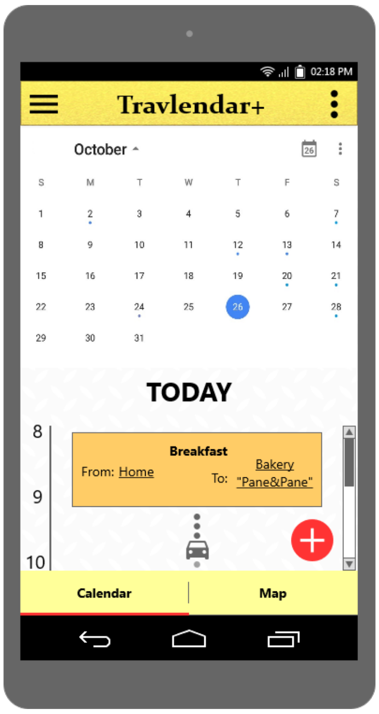
### 3.1.1 User Interfaces

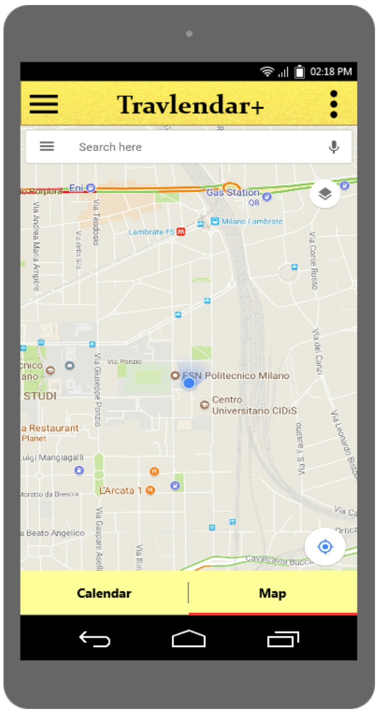
The following mockups are an overall representation of the look of the app and the website in their first release.

#### Login

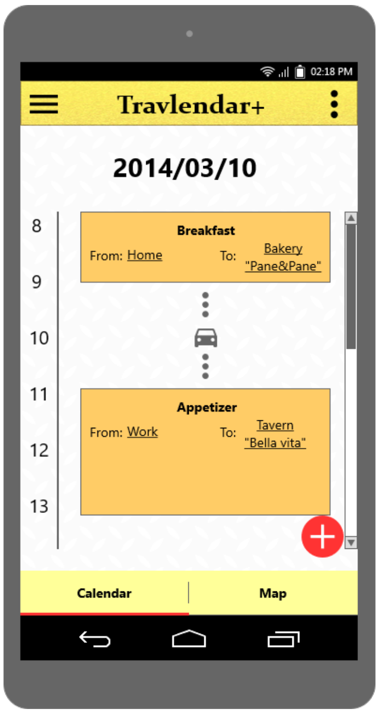


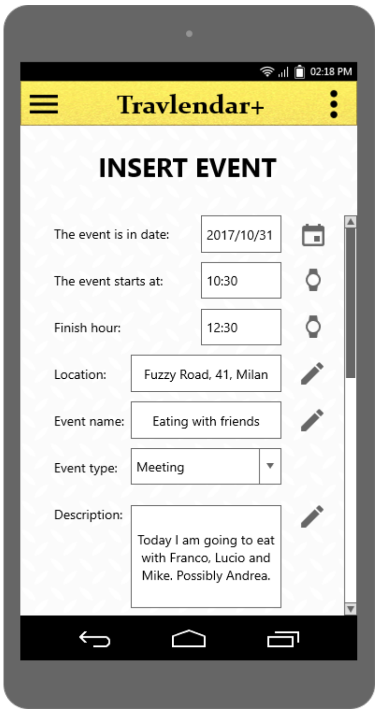
### Calendar and Map views





### Day Viewer and Event Creation

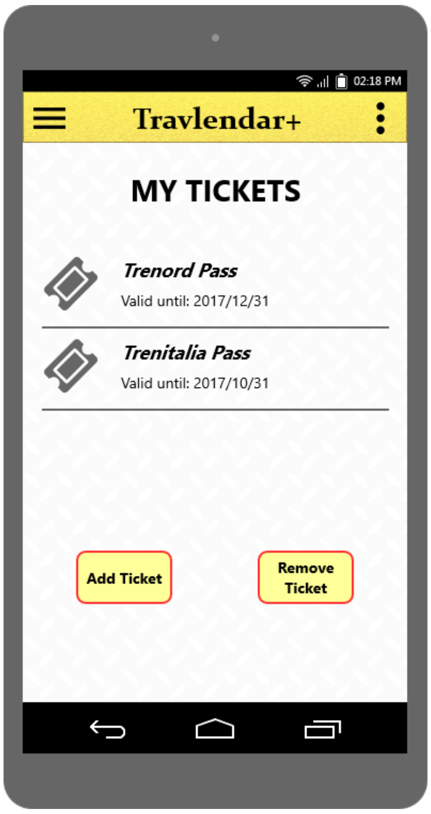




INSERT EVENT

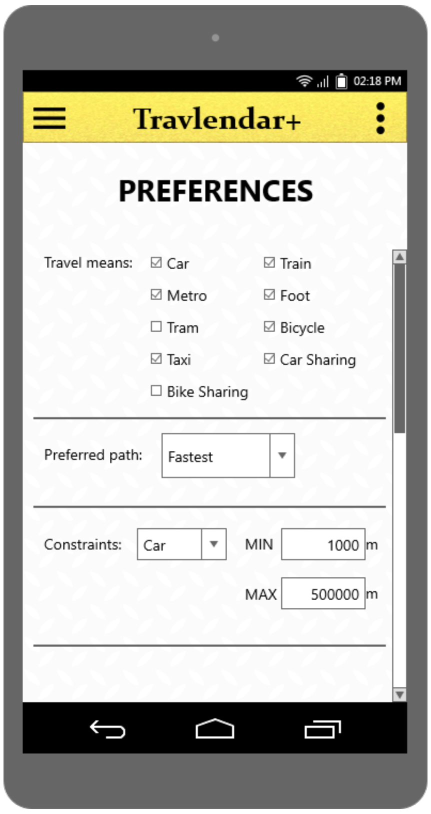
| | |
|---|---|
| The event is in date: | 2017/10/31 |
| The event starts at: | 10:30 |
| Finish hour: | 12:30 |
| Location: | Fuzzy Road, 41, Milan |
| Event name: | Eating with friends |
| Event type: | Meeting |
| Description: | Today I am going to eat with Franco, Lucio and Mike. Possibly Andrea. |

### My Tickets and Preferences



*Trenord Pass*
Valid until: 2017/12/31

*Trenitalia Pass*
Valid until: 2017/10/31

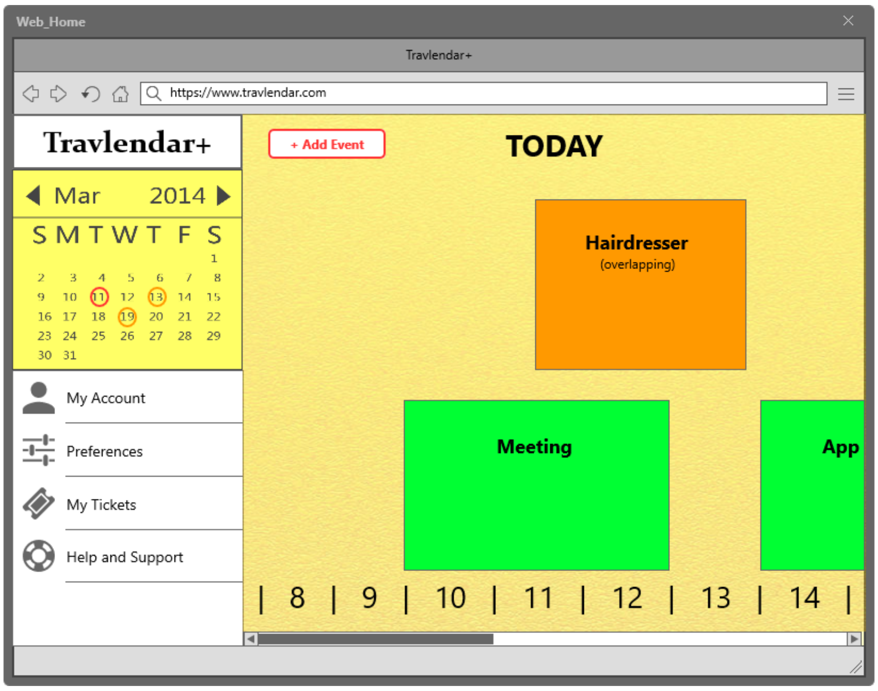


Travel means: ☑ Car ☑ Train ☑ Metro ☑ Foot ☐ Tram ☑ Bicycle ☑ Taxi ☑ Car Sharing ☐ Bike Sharing

Preferred path: Fastest

Constraints: Car MIN 1000 m MAX 500000 m

### WebApp View



### 3.1.2 Hardware Interfaces

The mobile app is supported on:

- Android 6.0 and superior;
- iOS 8 and superior;

To utilize the mobile app, the phone must be able to connect to the Internet and have a working GPS sensor to identify its position.

The web app is supported on these browsers:

- Google Chrome;
- Mozilla Firefox;
- Microsoft Edge;
- Safari;

Other browsers may be utilized to access the web app, but full compatibility is not guaranteed.

### 3.1.3 Software Interfaces

This system implements Google Maps APIs to calculate the optimal travel path to reach a destination.
Interaction with external websites of travel means providers is required in order to allow the user to buy tickets.
External weather APIs will be used to gather weather forecast data in order to propose travels taking into account the weather in the computation process.
The system implements Google Transit APIs in order to use GTFS and GTFS-realtime feeds to improve path computations when public travel means are involved. The system back-end (server side) will be implemented using Java EE 8 and we'll use MySQL 5.7 as DBMS.

### 3.1.4 Communication Interfaces

During the registration phase, the system will automatically send an email to the email address inserted by the user.
This email will contain a recap of the data inserted by the user during the registration, along with a linked URL address that needs to be opened by the user in order to verify that that email is currently valid and active. The client will communicate with HTTP protocol by using TCP (default port 443).

## 3.2 Functional Requirements

### 3.2.1 Detailed list

A *visitor* should be able to:

[**G1** ] sign up into the system:

[**D1** ] every inserted e-mail must be unique;

[**R1** ] the system checks if the e-mail inserted is real;

**[R2** ] a user cannot sign up with the same e-mail twice.

A *user* should be able to:

**[G2** ] log into the system:

**[D1** ] every inserted e-mail must be unique;

**[R3** ] the e-mail and password inserted must be correct;

**[R4** ] incorrect credentials prevent the user from logging in.

**[G3** ] create an event specifying its location, date, starting and ending time:

**[D2** ] every event is related to a location;

**[D3** ] an event must happen in an existing place;

**[D4** ] an event to attend cannot be in the past;

**[D5** ] the starting time of an event is always specified;

**[D6** ] the ending time of an event is not mandatory;

**[R5** ] a user must specify all mandatory fields to add the new event;

**[R6** ] the system reserves the specified time-slot for the event;

**[R7** ] the system warns the user if the inserted event overlaps with an already existing one;

**[R8** ] if the ending time of an event is not specified, the systems considers as ending time the hour of departure for the successive event;

**[R9** ] when an event is inserted after an event without a specified ending time, the ending time of the first event is anticipated as stated in [R8].

**[G4** ] obtain the best travel path (according to his preferences) and the list of eventual feasible alternatives to reach a location:

**[D7** ] a person can travel only on one travel mean at once;

**[D8** ] allowed travel means are cars, trains, metro, on foot, trams, bicycles, taxis, car sharing, bike sharing;

**[D9** ] every travel is programmed with a combination of one or more travel means;

**[R10** ] every travel path proposed must be feasible in the available time (the interval between two consecutive events);

**[R11** ] if the travel involves two or more travel means, the starting location of the first proposed travel path and the ending location of the last proposed travel path must coincide respectively with the starting location and the ending location of the whole planned travel;

**[R12** ] the system does not consider paths that violate constraints on travel means defined by the user;

**[R13** ] the system checks user preferences to decide which feasible travel path is the best;

**[R14** ] the system warns the user if it is not possible to arrive at an event location before its starting time;

- **[R15** ] appropriate travel means must be suggested according to the type of event that they are related to;
- **[R16** ] if a strike occurs, the system will not consider travel means involved in it;
- **[R17** ] if the weather forecasts rain or other adverse conditions, the system will not consider paths involving the bicycle.

**[G5** ] change the selected path with an alternative one:

- **[D7** ] a person can travel only on one travel mean at once;
- **[D8** ] allowed travel means are cars, trains, metro, on foot, trams, bicycles, taxis, car sharing, bike sharing;
- **[D9** ] every travel is programmed with a combination of one or more travel means;

- **[R18** ] the system must show to the user all possibilities to reach a location in according with the requirements of [[G4]];
- **[R19** ] alternative feasible travel paths must not generate overlappings with other events of the schedule.

**[G6** ] obtain a daily schedule that allows him to attend every event in program:

- **[D10** ] a person can be only in one place at once;

- **[R20** ] the combination of the travel paths proposed for the day must be feasible in the allotted time;
- **[R21** ] if there are multiple events at the same time the system will propose in the schedule only the first event added.
- **[R22** ] if the user forces into the schedule an event that overlaps with events already present in the schedule, these are removed from the schedule.

**[G7** ] apply constraints on travel means:

- **[D7** ] a person can travel only on one travel mean at once;
- **[D8** ] allowed travel means are cars, trains, metro, on foot, trams, bicycles, taxis, car sharing, bike sharing;
- **[D9** ] every travel is programmed with a combination of one or more travel means;

- **[R23** ] the system requires minimum and maximum length allowed for a path to impose a constraint on a travel mean;
- **[R24** ] the system requires an interval of time allowed to impose a constraint on a travel mean;
- **[R25** ] the system does not consider solutions that violate constraints.

**[G8** ] deactivate one or more travel means:

- **[D7** ] a person can travel only on one travel mean at once;
- **[D8** ] allowed travel means are cars, trains, metro, on foot, trams, bicycles, taxis, car sharing, bike sharing;
- **[D9** ] every travel is programmed with a combination of one or more travel means;

- **[R26** ] the system allows the user to specify one or more travel means that cannot be used.
- **[R27** ] the system does not consider solutions that include deactivated travel means.

**[G9** ] select combinations of transportation means that minimize carbon footprint:

- **[D11** ] every travel mean is related to information about its average carbon footprints;

- **[R28** ] for each travel path, the system estimates its carbon footprint produced.

**[G10** ] reserve time for lunch or break events:

- **[D12** ] a flexible time-slot can have a daily or periodical validity;
- **[D13** ] a flexible time-slot has a minimum amount of time that must be reserved;

- **[R29** ] the system allows the user to specify a flexible interval and a minimum amount of time to schedule a break;
- **[R30** ] if there is enough time for a break, the system reserves it within the specified flexible interval;
- **[R31** ] if there is not enough time into the flexible interval specified, a warning is thrown.

**[G11.1** ] arrange trips (buy needed tickets):

- **[D14** ] a ticket is required to use a public transport;
- **[D15** ] a ticket may have a daily validity or a different periodicity;
- **[D16** ] a user can own day/week/season passes;

- **[R32** ] the system allows the user to specify all the ticket he already owns;
- **[R33** ] the system shows to the user if he already holds a ticket for a proposed travel;
- **[R34** ] the system allows the user to buy public transportation tickets according to proposed travels;
- **[R35** ] the system provides information about time of departure and arrival of the proposed travels.

**[G11.2** ] arrange trips (find available sharing vehicles):

- **[D17** ] a payment is required to use a sharing vehicle;

- **[R36** ] the system shows to the user where sharing vehicles are located;
- **[R37** ] the system provides information about travel time with shared vehicles.

### 3.2.2 Scenarios

#### Scenario 1

Oscar is a businessman who travels a lot and he would like to organize his travels quickly and precisely. A friend advises him to try Travlendar+. Oscar accesses the Travlendar+ website and loads the registration page by clicking on the 'Sign Up' button. Then he fills all the mandatory fields, clicks on the 'Confirm' button and he receives a confirmation e-mail. He clicks on the confirmation link inside. Finally Oscar logs in and starts using Travlendar+.

**Scenario 2**

Nigel lives in Milan and tomorrow he's going to have an appointment in Lecco, so he has to decide how to reach his appointment location; Nigel accesses the login page from his personal computer by using his browser, fills the username and password fields, clicks on the 'Confirm' button and logs in. After that, he clicks on the dedicated button to add a new event, he fills all the requested fields, setting the event type as 'work meeting' in order to obtain proper suggested travel means, then he confirms the event creation. On the next day, Nigel travels to Lecco, following the travel schedule proposed by his Travlendar+ app and reaching his appointment right on time.

**Scenario 3**

Jasper inserts an event into his Travlendar+ calendar, but the location inserted is too far away from his previous event, therefore a feasible path to reach the location of the next event is not available in the allotted time. Jasper is notified by a warning that he will not be able to reach his appointment in time.

**Scenario 4**

Ophelia inserts a new event into his Travlendar+ calendar, but that event overlaps with another already existing event. Ophelia is notified by her app and the system asks Ophelia to choose which one of the overlapping events she wants to attend. She chooses the last event inserted, then she reads the travel means proposed to respect her new time schedule.

**Scenario 5**

Henrietta wants to personalize her Travlendar+ experience, so she clicks on the 'Menu' icon and then the voice 'Preferences'. She selects her preferred travel means, she chooses to always obtain the fastest path and then, since she walks with a limp, she inserts a maximum walking distance of 500 meters. Before clicking on the 'Confirm' button, she sees a polluting truck outside her window, so she decides to always follow travel paths that minimize her carbon footprint. She selects the relative option in her preferences. Finally she clicks on the 'Confirm' button and she observes that her suggested travels have changed according to her new preferences.

**Scenario 6**

Arthur can't concentrate on his work if he doesn't eat his lunch between 12.00 and 13.00. He also needs at least 25 minutes to consume a proper lunch. For this reason Arthur inserts a flexible break event into his Travlendar+ calendar, in order to be sure that every day his app will reserve a time-slot for his lunch. A week later, Arthur inserts a new event whose travel overlaps with his flexible lunch event and his app notifies him. Arthur decides to ignore the warning, since he'll s going to eat during his travel.

**Scenario 7**

Gwendolyn looks at her Travlendar+ app and discovers that on the next day she has to travel first by train to Milan and then take a bike of MoBikes sharing system in Milan. She clicks on the train travel slot in order to buy the proper ticket and the apps redirects her in the right Trenords online shop webpage. When she is about to buy the ticket she suddenly remembers that she has a weekly Trenord pass, therefore she cancels the transaction and, in order to avoid making the same mistake twice, she adds her pass information into her Travlendar+ app. The next day Gwendolyn takes the train to Milan and when she arrives she opens her Travlendar+ app in order to find the nearest bike of MoBikes. Gwendoline arrives in time at her appointment.

### Scenario 8

Harvey has to reach an appointment near his home next week. He adds an event in his Travlendar+ calendar. He observes that the app suggests him to use a bike to reach the location of this event. The day before his appointment the weather forecasts rain for the successive day, so Harvey is notified by his app that, due to the forecast, he should avoid taking the bike, suggesting instead to reach his appointment by car.

### Scenario 9

Sara inserts an event in her Travlendar+ calendar, then she looks at the proposed travel path. Since she did not like the proposed travel, she clicks on the proposed path and selects another feasible alternative. The next day Sara travels according to the travel path she likes more.

### 3.2.3 Use case descriptions

#### [UC1] Registration

| | |
|---|---|
| Participating actors | Generic visitor. |
| Entry Condition | There are no entry conditions. |
| Event Flow | 1. The visitor clicks on the ”Register” button displayed onto the homepage;<br>2. The visitor fills all the mandatory fields shown, including his email, his password (twice), his name, his surname and a captcha;<br>3. The visitor clicks on ”Confirm” button;<br>4. The visitor receives a confirmation email and clicks on the confirmation link;<br>5. The system saves all user data inserted. |
| Exit Condition | The visitor’s registration is completed successfully, so the visitor is registered as an user of Travlendar+ and he can log in the system as a registered user. |
| Exception | If:<br>• The visitor inserts an email already connected to an existing account;<br>• The visitor inserts invalid info in at least one of the mandatory fields;<br>• The visitor leaves empty at least one of the mandatory fields;<br>Then the system will request the visitor to complete/revise all uncorrected field, highlighting them. If the visitor does not activate the account, the activation link will expire after a month and all the user’s data will be deleted. |

Table 3.1: Registration use-case

### [UC2] Login

| | |
|---|---|
| Participating actors | Unauthenticated User. |
| Entry Condition | There are no entry conditions. |
| Event Flow | 1. The visitor clicks on the "Login" button displayed on the homepage;<br>2. The visitor inserts the email and the password previously used for registration;<br>3. The visitor clicks on the "Confirm" button;<br>4. The system redirects the user to the main view of Travlendar+. |
| Exit Condition | The login of the visitor is completed successfully, so the visitor can use all the Travlendar+ functions. |
| Exception | If:<br>• The email inserted is not one of the emails previously used by an user to sign up;<br>• The password inserted by the visitor is not the one associated with the email inserted;<br>• At least one of the field is left empty;<br>Then the system will notify the visitor to complete/revise all the uncorrected fields, highlighting them. |

Table 3.2: Login use-case

### [UC3] View calendar

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+. |
| Event Flow | 1. The user clicks on the "Calendar" tab;<br>2. The system shows a calendar where days with events are marked and a preview of the current day's events;<br>3. The user clicks on a day in order to obtain all the inserted events and the travel paths related to them. |
| Exit Condition | The system lets the user check his calendar. |
| Exception | There are no exceptions. |

Table 3.3: View calendar use-case

### [UC4] Create event

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+ |
| Event Flow | 1. The user clicks on the dedicated button to add a new event;<br>2. The user inserts all the info related to the event: date, starting time, ending time, location, name of the event, type of event (predefined or personalized), description, location starting from to reach the event's location;<br>3. The user confirms the creation of the event;<br>4. The system computes the best possible path according to the user's preferences. |
| Exit Condition | The system redirects the user to the calendar and adds the travel time slot required to reach that event (comprehensive of travel description). |
| Exception | If the inserted event overlaps with one or more previously added events (the travel is also considered in the eventual overlap), then the user is notified with a warning message and the overlapping event is not considered in the user travel planning schedule (but it remains saved in the calendar). The user has to choose which one of the overlapped events does he want to attend. |

Table 3.4: Create event use-case

### [UC5] Create personalized event profiles

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+. |
| Event Flow | 1. The user opens the preferences menu;<br>2. The system shows the default type of event;<br>3. The user selects the tab which create a personalized type of event;<br>4. The system shows a page containing the default constraints and some fields to fill;<br>5. The user inserts the name of the personalized type of event that he wants to create;<br>6. The user inserts the constraints on travel means related to that type of event and can modify the existing constraints;<br>7. The user clicks on the "Save" button;<br>8. The system adds the new type of events to those already existing. |
| Exit Condition | The user has successfully added a new personalized type of event in the system. |
| Exception | If the user exits the page without clicking the "Save" button, then the system will not save the new personalized type of event created by the user. |

Table 3.5: Create personalized event profiles use-case

**[UC6] Define flexible breaks**

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+. |
| Event Flow | 1. The user clicks on the dedicated button to add breaks into the schedule;<br>2. The user inserts a flexible period of time (specifying starting and ending time) that will contain the break, along with the minimum amount of time that will be dedicated to it;<br>3. The user also specifies the periodicity of the break (daily, weekly, monthly or until a specified date) or if it refers only to certain days of the week (until a specified date);<br>4. The user presses the "Save" button. |
| Exit Condition | The system notifies the user that the info inserted about breaks are correctly saved. |
| Exception | If it is not possible to dedicate the allotted time to the breaks because of previously added events, than the system asks the user to change the length of the break. |

Table 3.6: Define flexible breaks use-case

**[UC7] Arrange trips**

| | |
|---|---|
| Participating actors | User, Transport service provider. |
| Entry Condition | The user must be registered, logged in Travlendar+ and he must have opened his calendar and selected a specific day. |
| Event Flow | 1. The user clicks on the trip he wants to arrange;<br>2. The system shows all the tickets that need to be bought and the tickets already bought;<br>3. If the user wants to buy a ticket he clicks on that ticket and the system redirects the user to the website of the right Transport service provider in order to buy the ticket. |
| Exit Condition | The user has successfully arranged his travel. |
| Exception | The system fails to redirect the user to the website of the right Transport service provider, an error message is displayed. |

Table 3.7: Arrange trips use-case

### [UC8] Locate the nearest sharing vehicle

| | |
|---|---|
| Participating actors | User, Transport service provider. |
| Entry Condition | The user has opened the ”Arrange trip” tab and is about to travel. |
| Event Flow | 1. The user opens the map;<br>2. The system shows the nearest sharing vehicle in the map, according to the chosen path and the info provided by the transport service provider; |
| Exit Condition | The user reaches and uses the suggested vehicle. |
| Exception | • If no vehicle is found near the user, the system re-computes another path and shows it to the user;<br>• If the user takes a shared vehicle, but not the suggested one, nothing happens, because the user is not obligated to take the suggested vehicle; |

Table 3.8: Locate nearest sharing vehicle use-case

### [UC9] Add ticket possessed

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user has opened the ”Arrange trip” tab. |
| Event Flow | 1. The user selects the tab ’My tickets’; 2. The system shows a page containing all the tickets and passes possessed by the user; 3. The user clicks on the ’Add ticket’ button; 4. The system shows a page containing fields to fill; 5. The user inserts info regarding the ticket/pass already in his possession; 6. The user clicks on the ”Save” button; 7. The system adds the ticket/pass to those already present in the user account. |
| Exit Condition | The user has successfully inserted his ticket/pass in the system. |
| Exception | If the user exits the page without clicking the ”Save” button, then the system will not save the ticket added by the user. |

Table 3.9: Add ticket possessed use-case

### [UC10] Obtain feasible travel paths

| | |
|---|---|
| Participating actors | User, Google Maps APIs. |
| Entry Condition | The user must be registered and logged in Travlendar+. |
| Event Flow | 1. The user opens the ”Calendar” tab; 2. The user selects a day in the calendar; 3. The system shows the proposed feasible paths (shown as travel events) between the events; 4. If the user wants to select an alternative travel path, he can clicks on the travel event in order to choose among the proposed feasible alternatives. |
| Exit Condition | The user has seen the possible travel paths that can be used to reach his meetings. |
| Exception | If no feasible travel path exists between two events, the system shows a warning in the calendar. |

Table 3.10: Obtain feasible travel paths use-case

### [UC11] Choose between overlapping events

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+, at least two events are overlapped and the user must have opened. |
| Event Flow | 1. The user clicks on a specific day;<br>2. The system shows a schedule including all inserted events and the travel paths related. The overlapping events are displayed in a separate way in respect to the actual day schedule;<br>3. The user choose an overlapping event into his day schedule;<br>4. The system removes the precedent event (in conflict), putting it into the overlapping event list. |
| Exit Condition | The system shows the new day schedule, with updated travel paths. |
| Exception | There are no exceptions. |

Table 3.11: Choose between overlapping events use-case

### [UC12] Edit event

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+, at least one event must have been added. |
| Event Flow | 1. The user clicks on a specific day of the calendar;<br>2. The user selects the "Edit" option on the event he wants to modify;<br>3. The system shows the fields where the user can change the information previously inserted;<br>4. The user changes the event information and clicks on the "Confirm" button;<br>5. The system calculates the new travel according to the specifications inserted by the user. The modified event remains in the schedule if it does not overlap with another event. |
| Exit Condition | The system shows the new daily schedule, eventually with updated travel paths. |
| Exception | The user edits the location or the starting/ending time of the event in a way that attend to event is no more feasible. The user is notified with a warning message, the event is removed from the schedule. |

Table 3.12: Edit event use-case

**[UC13] Delete event**

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+, at least one event must have been added. |
| Event Flow | 1. The user clicks on a specific day of the calendar;<br>2. The user selects the "Delete" option on the event he wants to delete;<br>3. The user clicks on the "Confirm" button;<br>4. If an overlapping event becomes a feasible one, the system calculates the travel and the event is added to the schedule. |
| Exit Condition | The system shows the new daily schedule, eventually with updated travel paths. |
| Exception | There are no exceptions. |

Table 3.13: Delete event use-case

**[UC14] Edit personalized event profiles**

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+, at least one type of event must have been added. |
| Event Flow | 1. The user opens his preferences;<br>2. The user selects the "Edit" button on the type of event he wants to modify;<br>3. The system shows the settings associated to the type that the user wants to edit;<br>4. The user can add, edit or delete single constraints;<br>5. The user clicks on the "Confirm" button;<br>6. If there is at least one event related to that type of event, the system asks to the user if he wants to apply changes on preferences to the existing events of that type. |
| Exit Condition | The system saves the new set of constraints for the type of event modified and, eventually, updates the related events. |
| Exception | If an event is no more a feasible one, the system behaves like it is an exception for edit event. |

Table 3.14: Edit type of event use-case

### [UC15] Delete personalized event profiles

| | |
|---|---|
| Participating actors | User. |
| Entry Condition | The user must be registered and logged in Travlendar+, at least one type of event must have been added. |
| Event Flow | 1. The user opens his preferences;<br>2. The user selects the "Delete" option on the type of event he wants to delete;<br>3. The user clicks on the "Confirm" button;<br>4. If there is at least one event related to that type of event, the system asks the user if he wants to maintain the old preferences for the existing events of that type. |
| Exit Condition | The system deletes that type of event from the preferences and, eventually, removes constraints from the related events. |
| Exception | There are no exceptions. |

Table 3.15: Delete type of event use-case

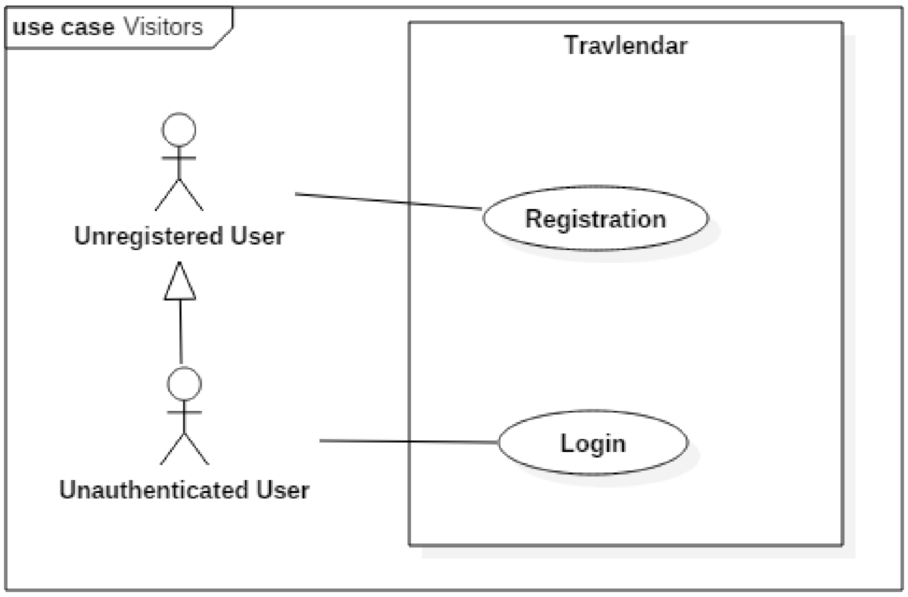
### 3.2.4 Traceability matrix

| **GoalID** | **ReqID** | **UseCaseID** |
|---|---|---|
| G1 | R1-R2 | UC1 |
| G2 | R3-R4 | UC2 |
| G6 | R20-R21 | UC3 |
| G3 | R5-R6-R7 | UC4 |
| G4 | R10-R11-R12-R13-R14-R15 | UC4 |
| G7 | R25 | UC4 |
| G8 | R27 | UC4 |
| G7 | R23-R24 | UC5 |
| G8 | R26 | UC5 |
| G10 | R29 | UC6 |
| G11.1 | R33-R34-R35 | UC7 |
| G11.2 | R36 | UC8 |
| G11.1 | R32 | UC9 |
| G4 | R10-R11-R12-R14-R15 | UC10 |
| G5 | R18 | UC10 |
| G7 | R25 | UC10 |
| G8 | R27 | UC10 |
| G6 | R21-R22 | UC11 |
| G3 | R6-R7 | UC12 |
| G4 | R10-R11-R12-R13-R14-R15 | UC12 |
| G6 | R20 | UC12 |
| G7 | R25 | UC12 |
| G8 | R27 | UC12 |
| G6 | R20 | UC13 |
| G7 | R23-R24 | UC14 |
| G8 | R26 | UC14 |

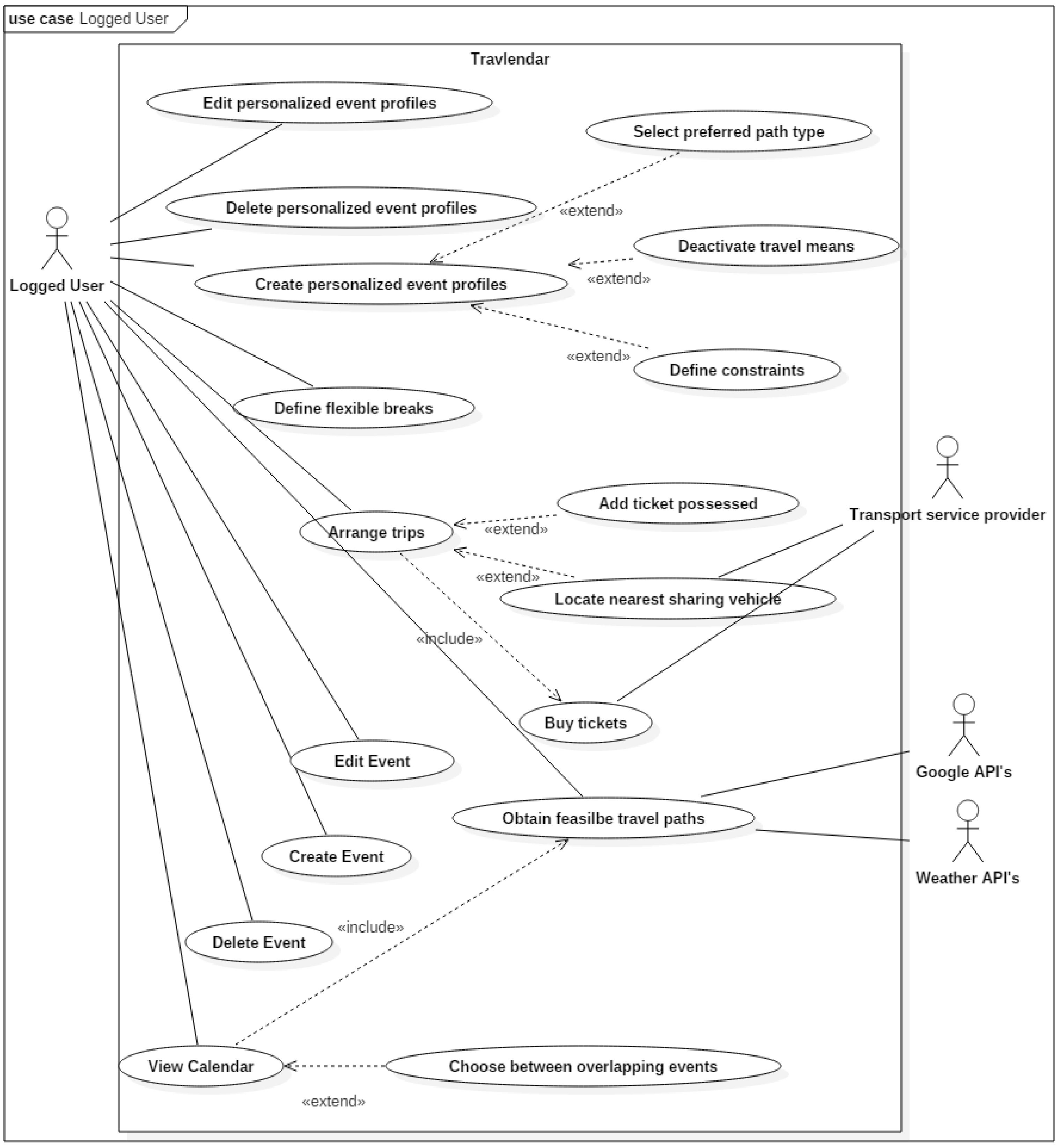
Table 3.16: Traceability matrix

### 3.2.5 Use case diagrams

#### Visitors

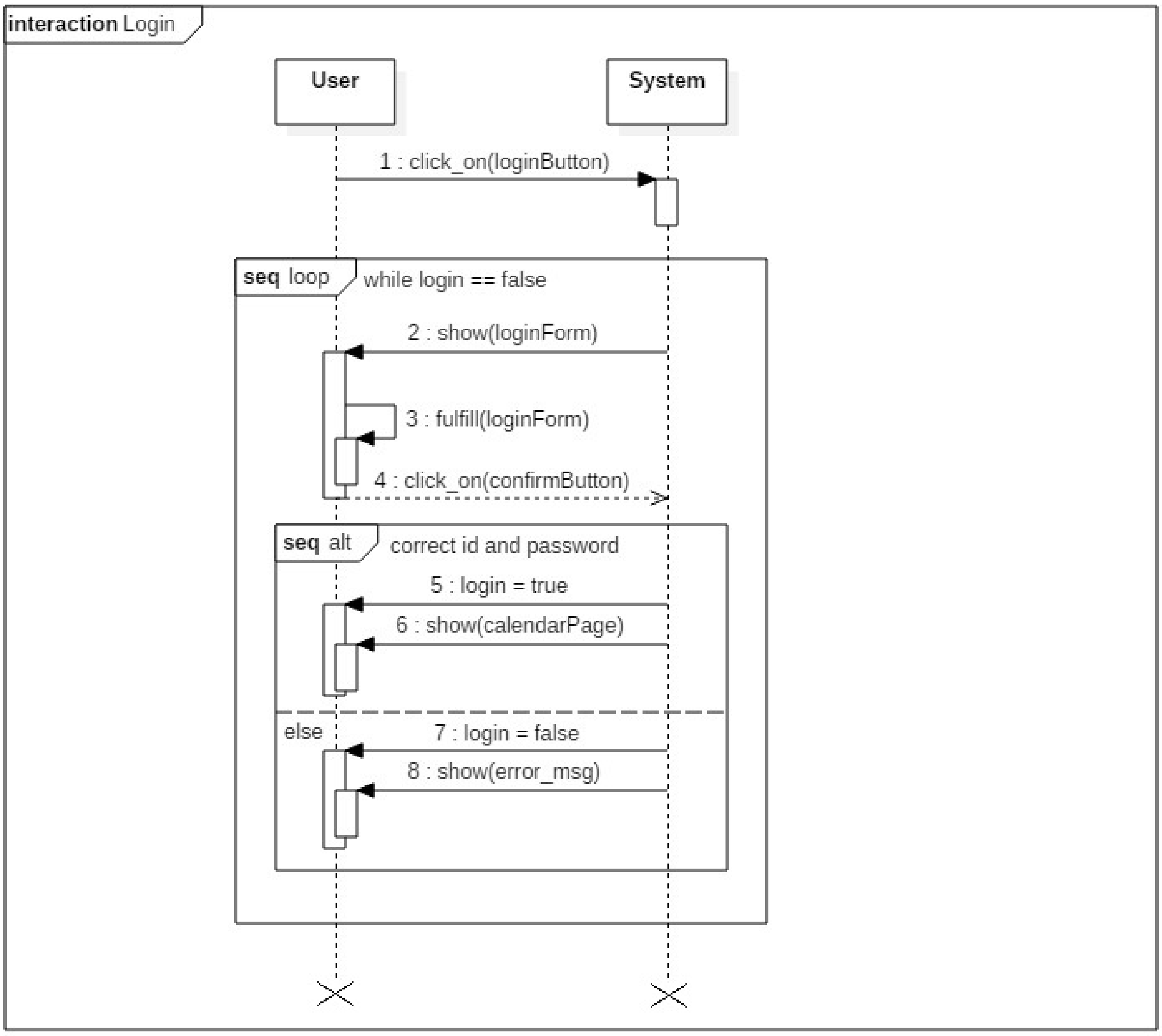


### User

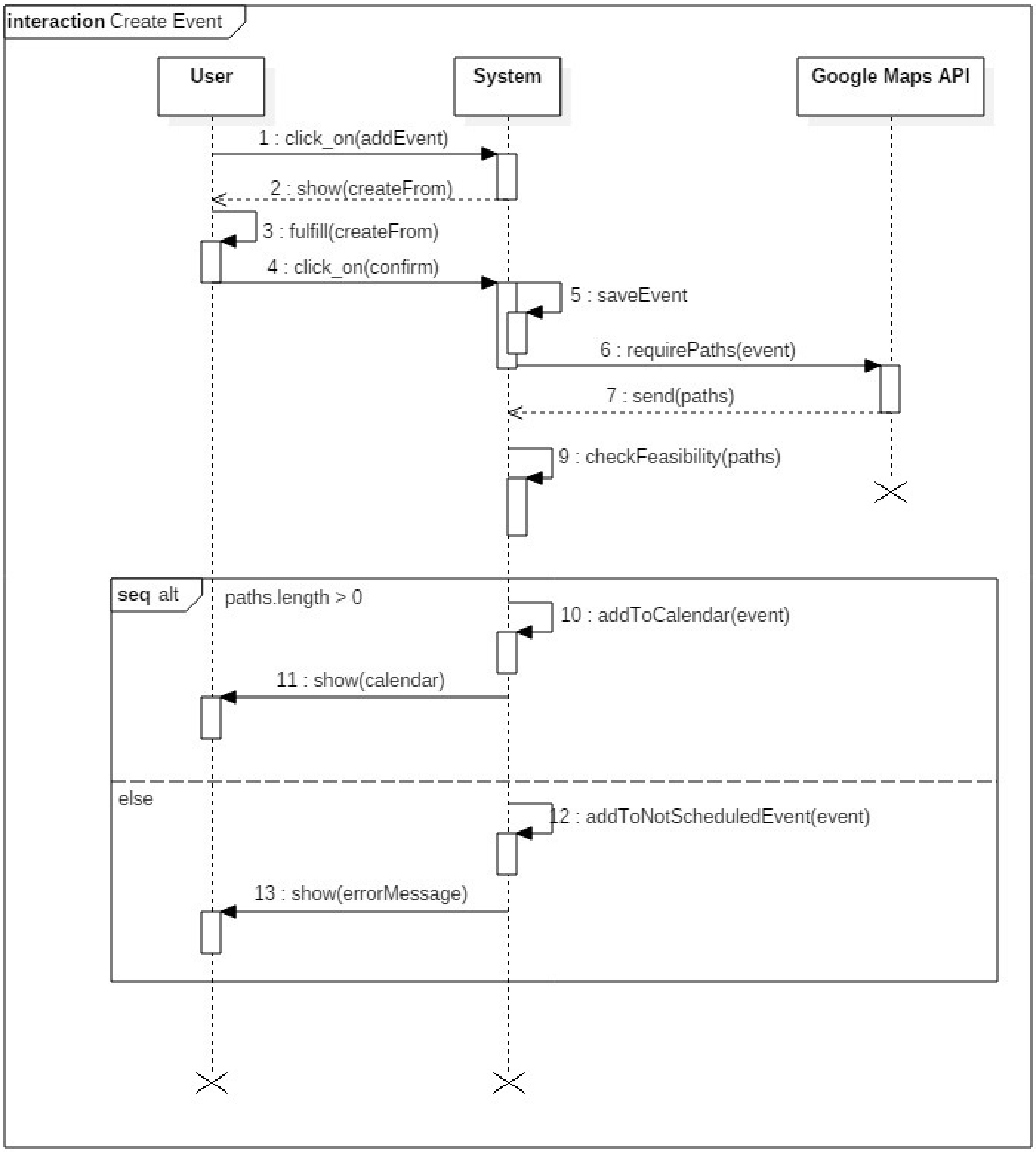

use case Logged User
Travlendar
Logged User
Edit personalized event profiles
Select preferred path type
Delete personalized event profiles
«extend»
Deactivate travel means
Create personalized event profiles
«extend»
«extend»
Define constraints
Define flexible breaks
Add ticket possessed
Transport service provider
Arrange trips
«extend»
«extend»
Locate nearest sharing vehicle
«include»
Buy tickets
Edit Event
Google API's
Obtain feasilbe travel paths
Weather API's
Create Event
Delete Event
«include»
View Calendar
Choose between overlapping events
«extend»

### 3.2.6 Sequence diagrams

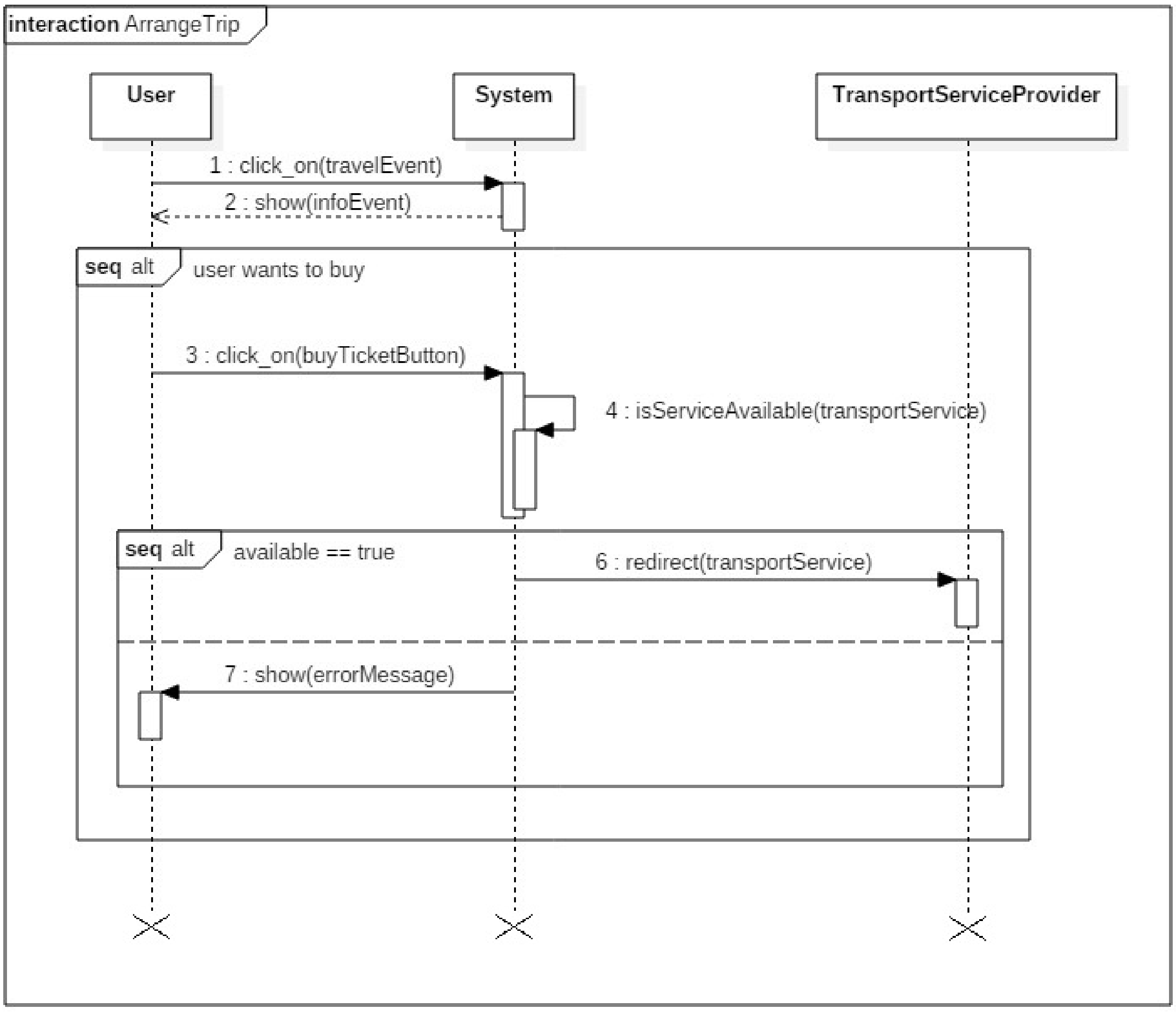
#### Login



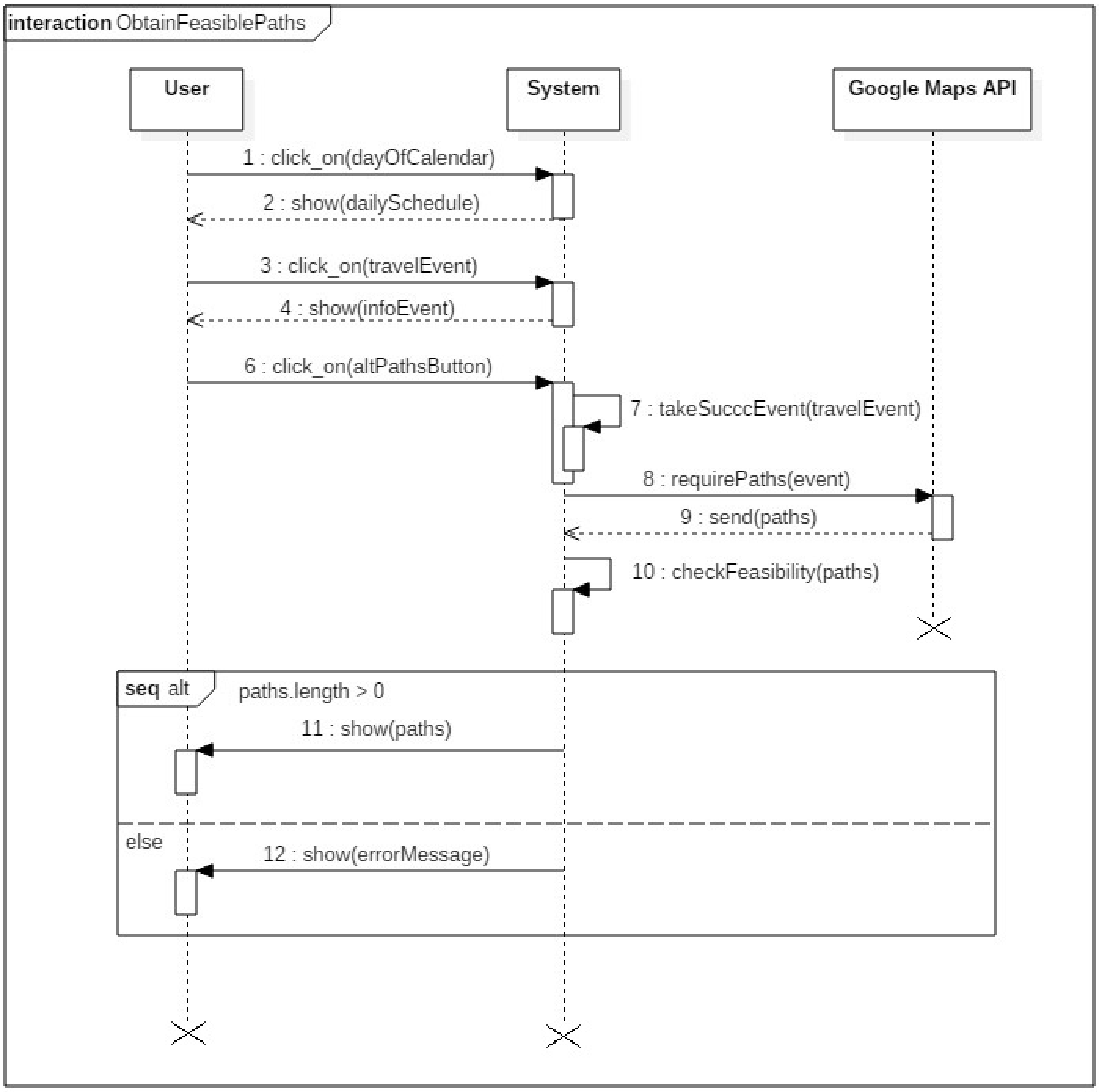
### Create event



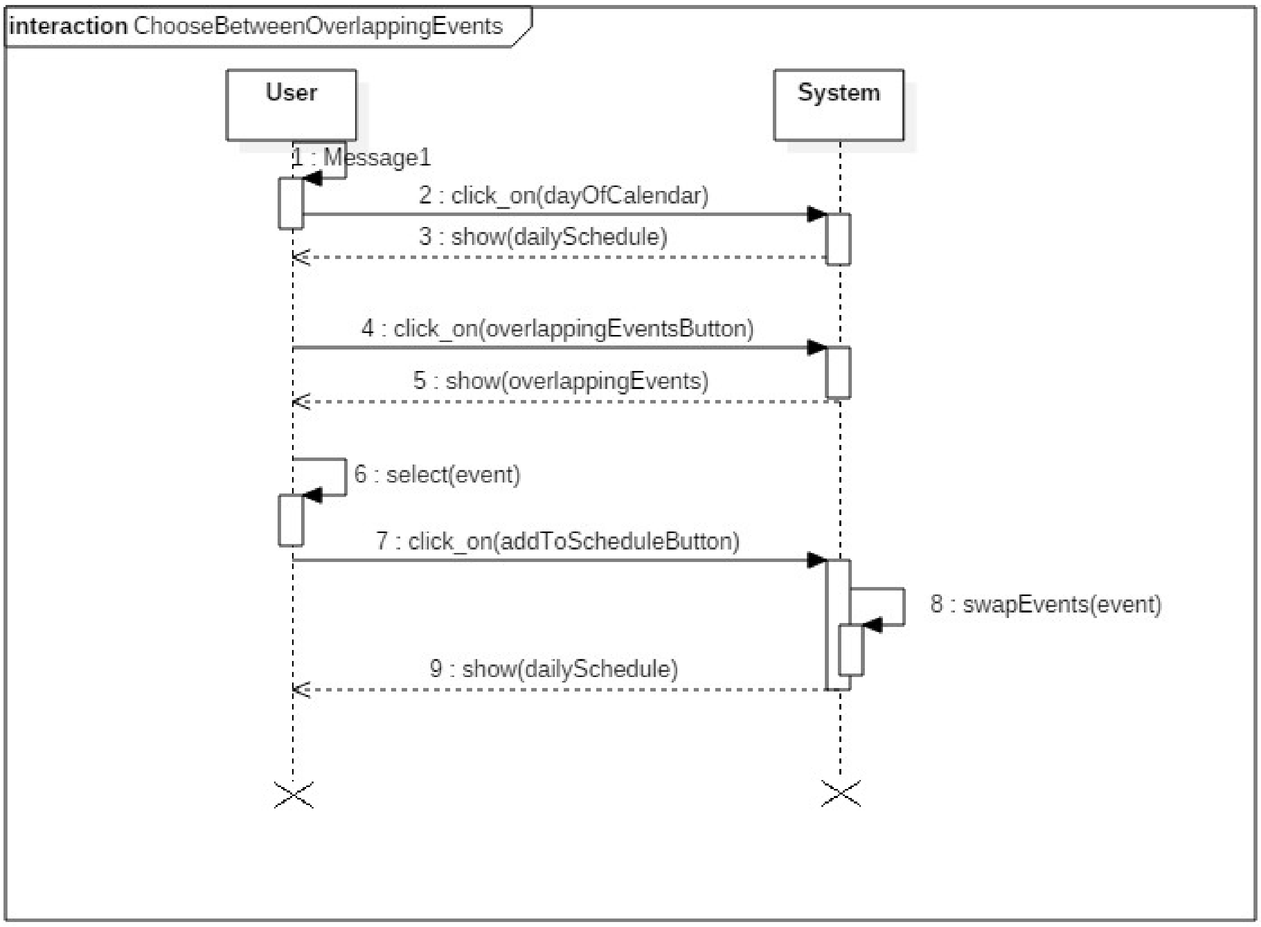
### Arrange trip



### Obtain feasible paths



**Choose between overlapping events**


interaction ChooseBetweenOverlappingEvents
User
System
1 : Message1
2 : click_on(dayOfCalendar)
3 : show(dailySchedule)
4 : click_on(overlappingEventsButton)
5 : show(overlappingEvents)
6 : select(event)
7 : click_on(addToScheduleButton)
8 : swapEvents(event)
9 : show(dailySchedule)

## 3.3 Performance Requirements

The task that should take up most of the computational time is the management of the schedule, which happens every time an event gets added, modified or deleted. To avoid slowing down the phone of the user, this computation should be done server side, sending only the result to the user.
Changing screens should occur quickly because of the light weight of the computation required to do so.
Barring eventual malfunctions of the external services implemented in the system and depending on the signal strength of the user's phone, 95% of the server updates should take at most 2 seconds, since the internet connection used by the users is not part of the system-to-be, network delays are not considered into our performance constraints. Travlendar+ will not impose any limit to the number of registered users. The system must support simultaneously at least 1000 users (logged into the system).

## 3.4 Design Constraints

### 3.4.1 Standards compliance

The system should produce a log containing all the action performed in the system, in order to able to perform a recovery in case of failures.
This recovery functionality is not implemented in this version of the system.
Since the system memorize user's personal data it will have to respect privacy laws, in particular during the entire specification and development process we'll use as reference the General Data Protection Regulation (GDPR) (Regulation (EU) 2016/679), adopted in April 2016 by the European Union ( It becomes enforceable from 25 May 2018).

### 3.4.2 Hardware limitations

In order to run on mobile devices, the application will need at least:

- 3G mobile internet connectivity;
- enough space in memory to install the application and store personal data;
- GPS module.

Since the application is primarily designed to be used on mobile phones while been away from home, low resolution and limited display size will be huge design constraints.
This also means that developing an effective user interface that can effectively show quickly and efficiently all the information necessary to the user will be a challenging task.
Also depending on the phone hardware, low processing power and small amount of memory are a matter of concern.

### 3.4.3 Any other constraint

Travlendar+ is dependent on external services to locate sharing vehicles, acquire tickets and produce travel paths. Because of this reason, it is critical to perform an update every time that one of these external functionalities changes, with the intent to avoid failures of the system.
Possible malfunctions of these external services should be accounted for during the development of Travlendar+, making sure that the system doesn't get hung up or crashes while trying to reach them.

## 3.5 Software System Attributes

### 3.5.1 Reliability

The reliability (Nb = 1 - Probability of failure) requested is related to many factors: the software developed must be stable and efficient in both server and client side, the user must satisfy the minimum requirement specifications specified in this document otherwise the reliability of the client app or the browser interface are not guaranteed; The reliability of the servers must be also taken into account and guaranteed by a proper maintenance .

### 3.5.2 Availability

The system must offer an availability of 99.9% (an overall 24/7 service). The remaining 0.1% (at maximum) shall take into account the time spent for ordinary maintenance sessions. A backup server will be provided in order to maintain the availability of the service even after the main server failure.

### 3.5.3 Security

All user's data will be stored and encrypted using a proper hashing mechanism, even system administrators must not been able to access personal user's data. The GPS location of the users, detected by the mobile app, during the user's travels, is not to be memorized. All the connections established between users and server must use the HTTPS protocol. Every communication between server and client will be encrypted.

### 3.5.4 Maintainability

A version control system will be used in order to manage and organize all the different code revisions. In the development phase the entire source code will be properly documented and commented in order to ease the effort of possible future developers of understanding how the system has been designed and how it works. The documentation will also facilitate the system's maintenance.

### 3.5.5 Portability

The web application must support main browsers (such as Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) latest version.
The mobile application will be developed at first for Android architecture and then for iOS architecture.
The server-side of the application is written in Java, therefore it can be executed on any machine that runs a Java Virtual Machine.

# Chapter 4

# FORMAL ANALYSIS USING ALLOY

This chapter includes the Alloy code, used to describe and check the consistency of our model (see also the class diagram at 2.1.1)using a formal notation and to provide examples of the possible worlds generated by the Alloy analyzer tool.
We've described the domain of our model and his main properties (all modeling assumptions are described directly into the alloy code below) and we've also modeled some main operations to be provided by the system-to-be. Therefore we've proven the consistency of our model and we've used the results to refine our specifications contained into this document; during the alloy's modeling effort we've point out some flaws of our assumptions and of our UML models and so we've fixed it.

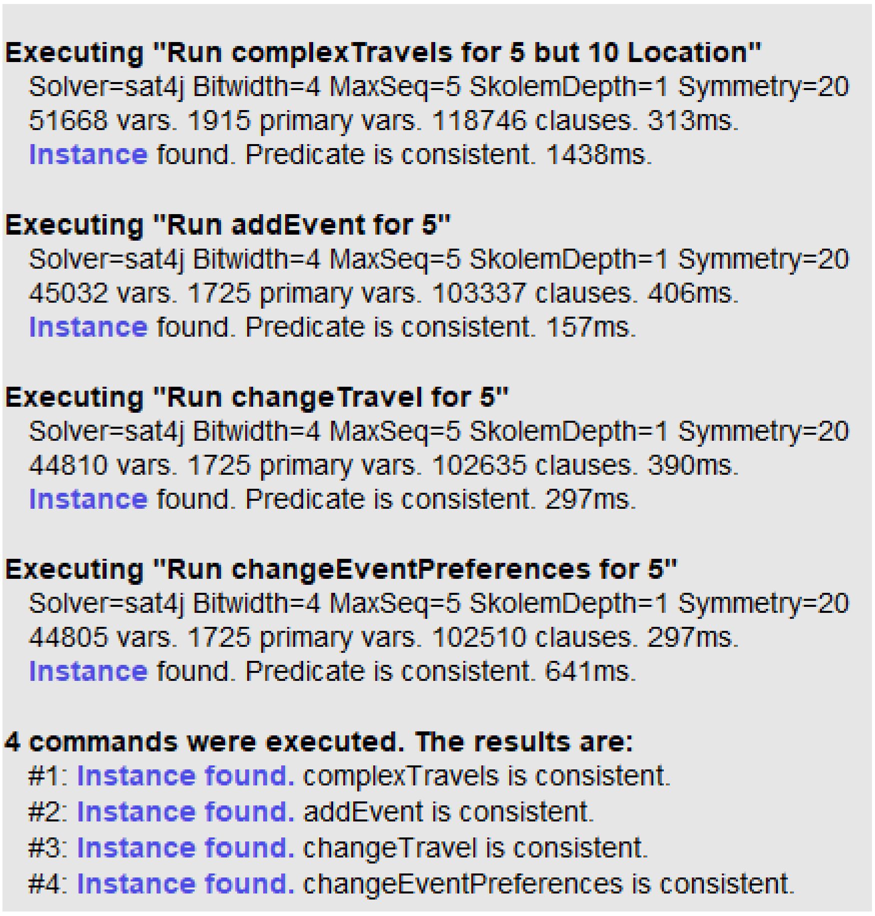
## 4.1 Model

```
open util/integer
open util/boolean
/* Modeling Notes
1 - Period class has not been modelled cause its aim is to avoid
    ↪ the user to insert
a periodic event again and again, but for modelling issues it's
    ↪ the same as have many
 different events at different times
2- Enviroment conditions are not modeled
3- The time is modeled as incremental integers that rapresent
    ↪ consecutive points in the timeline, even across multiple
    ↪ days
*/

// Strings sets have to be specified, so I've created an alias
    ↪ just for simplicity
sig StringModel{ }

sig User{
  name : one StringModel,
  surname: one StringModel,
  email: one Email,
  preferences: some TypeOfEvent,
  breaks: set BreakEvent,
  hold: set Ticket, //tickets owned
  plan: set Event, //Planned Events
  preferredLocations: set Location,

}

sig TypeOfEvent{
  /*Param first path not modeled since it's used only
  to decide which path the system suggests first to the user*/
  deactivate: set TravelMean,
  isLimitedBy: set Constraint,
}

abstract sig Constraint{
  concerns: one TravelMean,
}

sig DistanceConstraint extends Constraint{
  maxLenght: one Int,
  minLenght: one Int
} {
  maxLenght > 0
  minLenght > 0
  maxLenght > minLenght
}

sig PeriodOfDayConstraint extends Constraint{
  /*Min and max Hour are modeled as an integer, as all other time
     ↪  variables int this model to allow easier comparisons,
  the concept modeled is just the same, but we avoid useless
     ↪ complexity.
  Doing so min and max hour becomes min and max time*/
  maxTime: one Int,
  minTime: one Int,
}{
  maxTime > minTime
}

/* In this model we keep only a generic ticket*/
sig Ticket{
  cost: one Float,
  ticketUsed: set TravelComponent,
  relatedTo: some PublicTravelMean,
}{
  all component: ticketUsed| (component.meanUsed in relatedTo)
}

abstract sig TravelMean{
  name: one StringModel,
  //speed and eco consumption are useless to modelling purpose
}
sig PrivateTravelMean extends TravelMean{ }
sig SharingTravelMean extends TravelMean{ }
sig PublicTravelMean extends TravelMean{ }

abstract sig GenericEvent{
  startingTime: one Int,
  endingTime: one Int,
  isScheduled: one Bool,
}{
  startingTime <  endingTime
}

sig BreakEvent extends GenericEvent{
  minimum: one Int, //NB = minimum time required to make a break
}{
  minus[endingTime, startingTime]≥minimum
  minimum>0
}

sig Event extends GenericEvent{
  type: one TypeOfEvent,
  feasiblePaths: set Travel,
  departureLocation: one Location,
  eventLocation: one Location,
  travelTime: one Int, //Rapresent the minimum travel time
     ↪ reqired
  /* descriptive variables are omitted, the variable
     ↪ prevLocChoice is
  omitted cause it's only an operative variable and it would not
     ↪ enrich the model */
} {
   #feasiblePaths≥0
  (eventLocation= departureLocation or isScheduled=False)implies
     ↪ (#feasiblePaths=0 and travelTime=0 )else (#feasiblePaths>0
     ↪  and travelTime>0)
}

sig Travel{
  composed: seq TravelComponent,
}{
  #composed≥0
  not composed.hasDups
}

sig TravelComponent{
  departureLocation: one Location,
  arrivalLocation: one Location,
  meanUsed: one TravelMean,
  travelDistance: one Int, //Distances are modeled as integers (
     ↪ should be float)
}{
  departureLocation ≠ arrivalLocation
  travelDistance > 0
}

sig Location{
  latitude: one Float,
  longitude: one Float,
  address: one StringModel,
}{
  not latitude=longitude
}

//Float abstraction
sig Float {}

sig Email{}

/*****************FACTS*****************/

// any user's email must be univocal
fact email_Is_Unique{
  no disjoint u, u' : User | u.email = u'.email
}

// There must exists only a location corresponding to a latitude
   ↪ and a longitude
fact location_Is_Unique{
  no disjoint l, l' : Location | l.latitude=l'.latitude and l.
     ↪ longitude=l'.longitude and l.address=l'.address
}

//All travels must have a connected path and lead to the
   ↪ destination
fact travelsAlwaysLeadToDestination{
  all e: Event, travel: Travel  | ( travel in e.feasiblePaths)
     ↪ implies (let sequence=travel.composed |
    (sequence.first.departureLocation=e.departureLocation
      and sequence.last.arrivalLocation= e.eventLocation
      and ( #sequence=2 implies sequence[1].departureLocation=
         ↪ sequence[0].arrivalLocation)
      and (no element: sequence.rest.butlast.elems | element.
         ↪ departureLocation=e.departureLocation)
      and (no element: sequence.rest.butlast.elems | element.
         ↪ departureLocation=e.eventLocation)
      and (no element: sequence.rest.butlast.elems | element.
         ↪ arrivalLocation=e.departureLocation)
```

```
          and (no element: sequence.rest.butlast.elems | element.
             ↪ arrivalLocation=e.eventLocation)
          and (all element: sequence.rest.butlast.elems | let i=
             ↪ sequence.indsOf[element]|
             (sequence[minus[i,1]].arrivalLocation=element.
                ↪ departureLocation)and (sequence[plus[i,1]].
                ↪ departureLocation=element.arrivalLocation))
        )  )
}

// Into an event two proposed paths must not be identical
fact noTravelsWithSameTravelComponents{
  all e: Event, travel: e.feasiblePaths  | (no duplicate:(e.
      ↪ feasiblePaths-travel) | (travel.composed = duplicate.
      ↪ composed) )
}

fact noTypeOfEventWithoutUser{
  no type: TypeOfEvent | (all u: User | !(type in u.preferences))
}

fact noEventWithoutUser{
  no event: Event | (all u: User | !(event in u.plan))
}

fact noBreakEventWithoutUser{
  no break: BreakEvent | (all u: User | !(break in u.breaks))
}

fact noTravelsWithoutEvent{
  no t:Travel | ( all e:Event | !( t in (e.feasiblePaths)) )
}

fact noTravelsComponentWithoutTravel{
  no c:TravelComponent | ( all t:Travel | !( c in univ.( t.
      ↪ composed)) )
}
fact noConstraintWithoutTypeOfEvent{
  no c:Constraint | ( all t:TypeOfEvent | !( c in  t.isLimitedBy)
      ↪  )
}

//No two coinciding but distinct locations
fact noLocationOverlapping {
  no disj l, l1: Location | ( l.longitude = l1.longitude and l.
      ↪ latitude = l1.latitude )
}

//returns the maximum time at which the user must start
    ↪ travelling
```

```
fun relativeStart[e:Event]: one Int{
  minus[e.startingTime, e.travelTime]
}
//events and their travels times doesn't overlap (if they are
   ↪ scheduled) with other events.
fact noScheduledEventsOverlapping{
  all u:User |(all e1:u.plan | (no e2:(u.plan-e1)|(e1.isScheduled
     ↪ =True and e2.isScheduled=True and
    ((relativeStart[e1]≥relativeStart[e2] and relativeStart[e1]=<
       ↪ e2.endingTime)or(e1.endingTime≥relativeStart[e2] and e1.
       ↪ endingTime≤e2.endingTime)))))
}

//Forall scheduled breaks is always granted the minumum break
   ↪ time
fact breakEventAlwaysGranted{
  all u:User | (all break: BreakEvent | ( ((break in u.breaks)
     ↪ and break.isScheduled=True)
    implies( no event1: u.plan | (event1.isScheduled=True and
    no event2: u.plan | ( event2.isScheduled=True and
      let precedent= event1.endingTime |  all travel2: Travel | (
         ↪ travel2 in event2.feasiblePaths) and let successor=
         ↪ event2.startingTime |
      (precedent>break.startingTime or successor>break.endingTime
         ↪ ) implies
         (minus[successor,precedent] < break.minimum)
         )  )   )     )       )
}

fact allTravelsRespectDeactivateConstraints{
  all event: Event | ( no travelComponent:event.feasiblePaths.
     ↪ composed.elems|( travelComponent.meanUsed in event.type.
     ↪ deactivate ) )
}


fact allTravelsRespectDistanceConstraints{
  all event: Event | all travelComponent:event.feasiblePaths.
     ↪ composed.elems |
       ( all constraint: (event.type.isLimitedBy &
          ↪ DistanceConstraint) | (
         (constraint.concerns=travelComponent.meanUsed) implies
           (travelComponent.travelDistance≥constraint.minLenght
              ↪ and travelComponent.travelDistance≤constraint.
              ↪ maxLenght)
      ))
}

/*travel slot times, if they involve travel means with a
   ↪ constraint relative to a specific period of time,
 must be included into that period*/
fact allTravelsRespectPeriodConstraints{
  all event: Event | all travelComponent:event.feasiblePaths.
     ↪ composed.elems |
       (all constraint: (event.type.isLimitedBy &
          ↪ PeriodOfDayConstraint) | (
          (constraint.concerns=travelComponent.meanUsed) implies(
            (event.startingTime≤constraint.maxTime) and
            (relativeStart[event]≥constraint.minTime) ) )
      )
}



/*****************PREDICATES*****************/

pred addEvent[u:User, e:Event, u':User]{
  //preCondition
  not(e in u.plan)
  //postCondition
  u.name=u'.name
  u.surname=u'.surname
  //u.email=u'.email
  u.preferences=u'.preferences
  u.breaks=u'.breaks
  u.hold=u'.hold
  (u.plan + e)=u'.plan
  u.preferredLocations=u'.preferredLocations

}

pred changeTravel[e:Event,travel:Travel, e':Event]{
  //precondition
  not ( travel in e.feasiblePaths)
  //postconditions
  e.type = e'.type
  (e.feasiblePaths +travel) = e'.feasiblePaths
  e.departureLocation = e'.departureLocation
  e.eventLocation = e'.eventLocation
}

pred changeEventPreferences[event:Event, type2:TypeOfEvent, event
   ↪ ':Event ]{
  //precondition
  event.type ≠ type2
  //postconditions
  event'.type =type2
  (event'.feasiblePaths & event.feasiblePaths) = none // if
     ↪ preferences are totally change I expect others paths
  event'.departureLocation = event.departureLocation
  event'.eventLocation = event.eventLocation
}


pred complexTravels{
#Ticket >2
#User =1 // different users, different events and no shared
    ↪ constraints => we'll observe instances for just one user
all event: Event | (#event.feasiblePaths>1)
#Constraint>2
#BreakEvent=1
}

run complexTravels for 5 but 10 Location
run addEvent for 5
run changeTravel for 5
run changeEventPreferences for 5
```

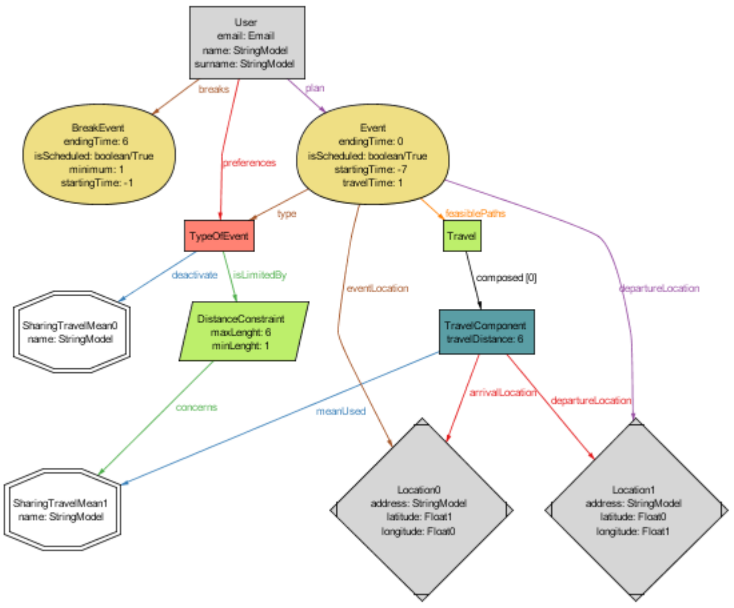
## 4.2 Proof of Consistency



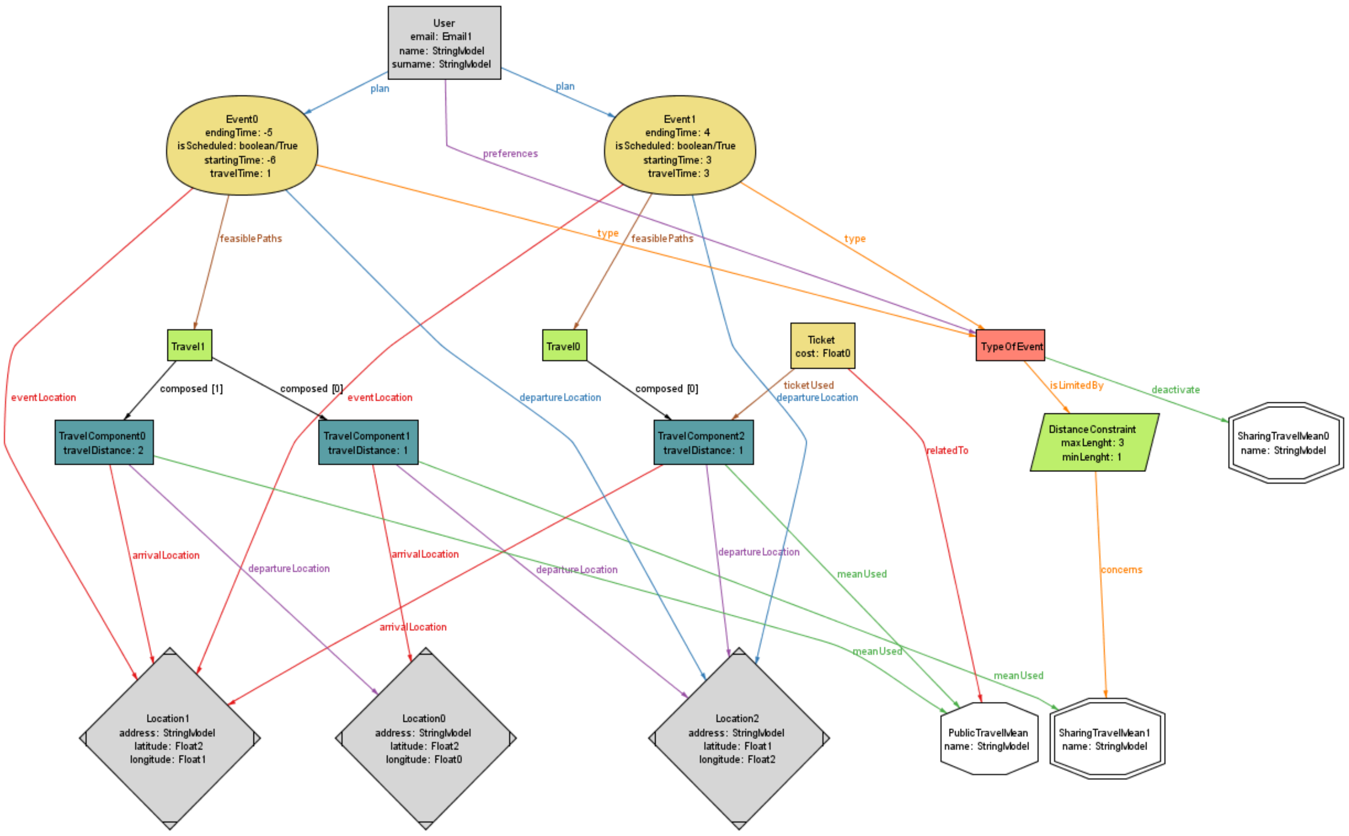
```
Executing "Run complexTravels for 5 but 10 Location"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   51668 vars. 1915 primary vars. 118746 clauses. 313ms.
   Instance found. Predicate is consistent. 1438ms.

Executing "Run addEvent for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   45032 vars. 1725 primary vars. 103337 clauses. 406ms.
   Instance found. Predicate is consistent. 157ms.

Executing "Run changeTravel for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   44810 vars. 1725 primary vars. 102635 clauses. 390ms.
   Instance found. Predicate is consistent. 297ms.

Executing "Run changeEventPreferences for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   44805 vars. 1725 primary vars. 102510 clauses. 297ms.
   Instance found. Predicate is consistent. 641ms.

4 commands were executed. The results are:
   #1: Instance found. complexTravels is consistent.
   #2: Instance found. addEvent is consistent.
   #3: Instance found. changeTravel is consistent.
   #4: Instance found. changeEventPreferences is consistent.
```

## 4.3 Worlds obtained

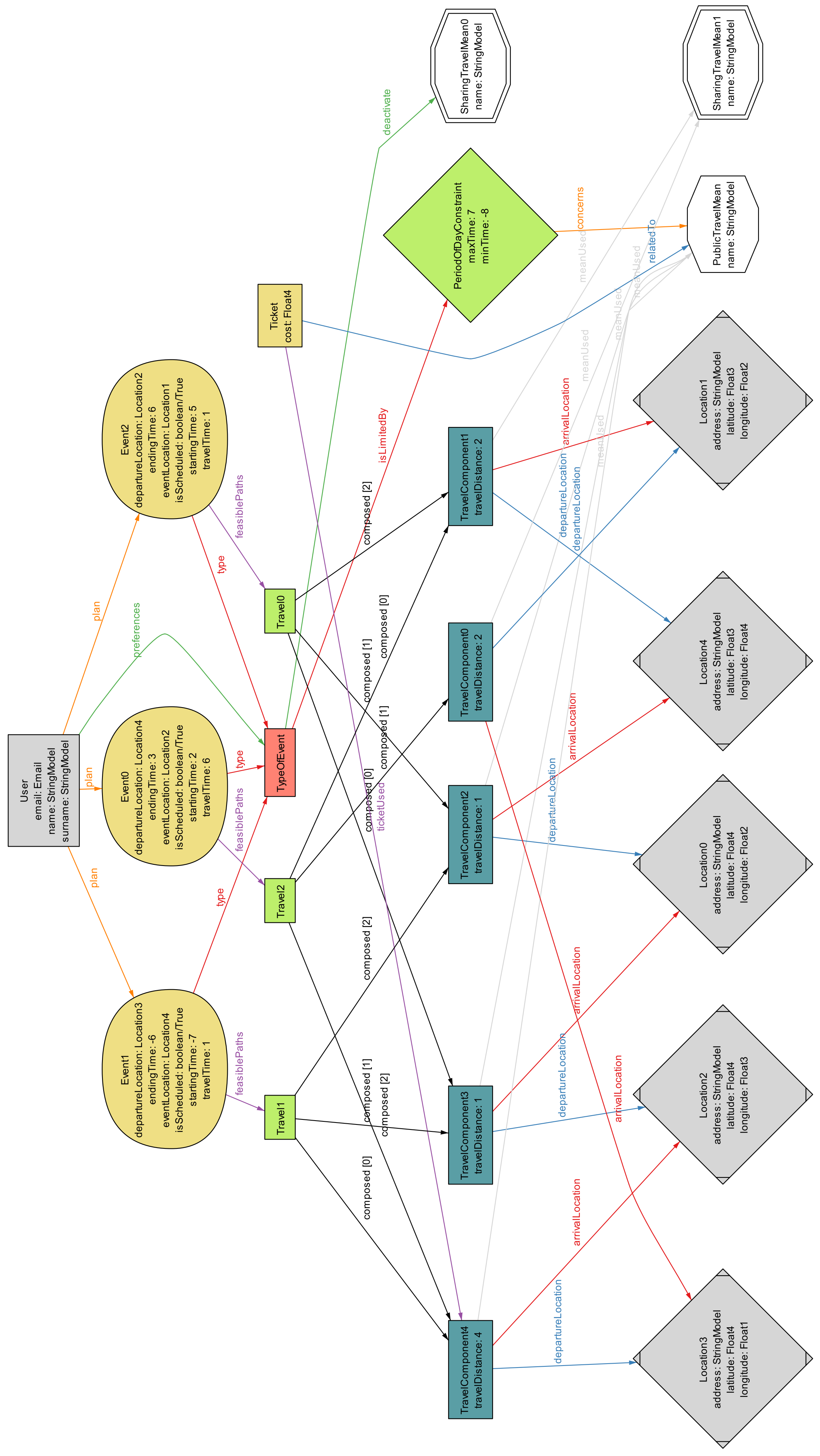
### 4.3.1 World 1



### 4.3.2 World 2



### 4.3.3 World 3


User
email: Email
name: StringModel
surname: StringModel
Event2
departureLocation: Location2
endingTime: 6
eventLocation: Location1
isScheduled: boolean/True
startingTime: 5
travelTime: 1
Event0
departureLocation: Location4
endingTime: 3
eventLocation: Location2
isScheduled: boolean/True
startingTime: 2
travelTime: 6
Event1
departureLocation: Location3
endingTime: -6
eventLocation: Location4
isScheduled: boolean/True
startingTime: -7
travelTime: 1
Ticket
cost: Float4
TypeOfEvent
Travel0
Travel2
Travel1
PeriodOfDayConstraint
maxTime: 7
minTime: -8
SharingTravelMean0
name: StringModel
SharingTravelMean1
name: StringModel
PublicTravelMean
name: StringModel
TravelComponent1
travelDistance: 2
TravelComponent0
travelDistance: 2
TravelComponent2
travelDistance: 1
TravelComponent3
travelDistance: 1
TravelComponent4
travelDistance: 4
Location1
address: StringModel
latitude: Float3
longitude: Float2
Location4
address: StringModel
latitude: Float3
longitude: Float4
Location0
address: StringModel
latitude: Float4
longitude: Float2
Location2
address: StringModel
latitude: Float4
longitude: Float3
Location3
address: StringModel
latitude: Float4
longitude: Float1
plan
preferences
type
feasiblePaths
composed [0]
composed [1]
composed [2]
ticketUsed
isLimitedBy
deactivate
concerns
relatedTo
meanUsed
arrivalLocation
departureLocation

# Chapter 5

# EFFORT SPENT

## 5.1 Pietro Melzi

| Date | Task | Hours of work |
|---|---|---|
| 06/10/17 | Group session - Assumptions and goals, identification of classes | 3 |
| 07/10/17 | Group session - Goal, requirements and domain assumptions | 4 |
| 09/10/17 | Group session - Use cases definition | 2 |
| 10/10/17 | Use case | 0:30 |
| 11/10/17 | Considerations on class diagram | 0:30 |
| 12/10/17 | Group session - Analysis UML model proposed by Melzi | 1 |
| 13/10/17 | Corrections on use cases, reading examples about RASD | 1h30 |
| 14/10/17 | Group session - Redefined goals, requirements, domain assumptions with relation among them | 8 |
| 15/10/17 | LaTeX syntax and class diagram | 1 |
| 16/10/17 | Considerations on class diagram, creation of class diagram with StarUML | 1:30 |
| 18/10/17 | General corrections and init of sequence diagrams | 1 |
| 19/10/17 | Group session - Corrections and fixes | 1 |
| 20/10/17 | Sequence diagrams | 1 |
| 21/10/17 | Group session - Glossary, sequence diagrams, general fixes | 6 |
| 22/10/17 | Sequence diagrams in StarUML, world and shared phenomena, statechart diagram | 4 |
| 23/10/17 | Use cases about edit and delete (event, type of event) | 2:30 |
| 25/10/17 | Use cases revised, glossary updated, init traceability matrix | 2 |
| 26/10/17 | Group session - Check and corrections | 1 |
| 27/10/17 | Traceability matrix and correction on sequence diagrams | 1 |
| 28/10/17 | Group session - Final | 3 |

## 5.2 Alessandro Pina

| Date | Task | Hours of work |
|---|---|---|
| 6/10/17 | Group session - Assumptions and goals | 3 |
| 7/10/17 | Group session - Goal, requirements and domain assumption | 4 |
| 9/10/17 | Group session - Use cases definition | 2 |
| 12/10/17 | Group session - Analysis UML model proposed by Melzi | 1 |
| 14/10/17 | Group session - Mockups | 8 |
| 15/10/17 | Mockups improvement | 3 |
| 16/10/17 | Scenarios review | 2 |
| 17/10/17 | Document structure | 1 |
| 18/10/17 | Use cases review | 2 |
| 19/10/17 | Group session - correction and fixes | 1 |
| 20/10/17 | Purpose/goals clarification | 1 |
| 21/10/17 | Group session - Missing sections | 6 |
| 23/10/17 | Product perspective/User characteristic | 2 |
| 24/10/17 | Mockups change | 3 |
| 25/10/17 | General review | 3 |
| 26/10/17 | Group session - Correction and fixes | 1 |
| 27/10/17 | Final overall check | 2 |
| 28/10/17 | Group session - Final | 3 |

## 5.3 Matteo Salvadore

| Date | Task | Hours of work |
|---|---|---|
| 6/10/17 | Group session - Assumptions and goals | 3 |
| 7/10/17 | Group session - Goal, requirements and domain assumption | 4 |
| 8/10/17 | Goals and shared phenomena analysis | 3 |
| 9/10/17 | Group session - Use cases definition | 2 |
| 10/10/17 | Use cases tables | 1 |
| 12/10/17 | Group session - Analysis UML model proposed by Melzi | 1 |
| 14/10/17 | Group session - Functional requirements | 8 |
| 15/10/17 | Scenarios and software system attributes | 3 |
| 16/10/17 | Software system attributes | 1 |
| 17/10/17 | Alloy | 4:30 |
| 18/10/17 | Alloy | 2 |
| 19/10/17 | Group session - correction and fixes | 1 |
| 20/10/17 | Alloy | 2 |
| 21/10/17 | Group session - Alloy | 6 |
| 23/10/17 | Alloy | 2 |
| 24/10/17 | Alloy inserted into RASD and latex revision | 2 |
| 25/10/17 | RASD revision | 1 |
| 26/10/17 | Group session - Correction and fixes | 1 |
| 27/10/17 | Alloy worlds + RASD revision | 1:30 |
| 28/10/17 | Group session - Final | 3 |

# Chapter 6

# REFERENCES

## 6.1 Bibliography

(I) Hans van Vliet - Software Engineering: Principles and Practice;

(II) Daniel Jackson - Software Abstractions Logic, Language, and Analysis;

(III) Slides about requirement engineering and Alloy on beep course (AA 2017-2018 Software Engineering II ).

## 6.2 Software used

This document is written in LaTex, a markup language used for text editing, and includes sections realized with different software tools. Here we show the list of used software:

- Texmaker 5.0.2 (compiled with Qt 5.8.0) for writing the document;
- StarUML 2.8.0 for class diagram, use cases diagrams and sequence diagrams;
- draw.io for model of shared phenomena and statechart diagram;
- Mockplus 3.2.6 for mockups;
- Alloy Analyzer 4.2 for the creation of the model.