POLITECNICO
MILANO 1863

REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

Travlendar+

Alessio CANTINA	*895395*
Simone CRIPPA	*898304*
Nicola CUCCHI	*893748*

Version 1.1 – November 24th

Contents

1 INTRODUCTION

1.A Purpose

How many times have you arrived late for a meeting or chosen a bad transport solution losing a lot of time and causing useless pollution? Have you ever been in trouble due to transport strikes? There can be a long list of problems like these in our everyday life, both professional and private.

Everyone can easily agree that our everyday lives have become more and more complex, full of activities in the last years, making us run from one place to the other being always in a hurry. It seems that we need something powerful to manage all our schedules, technology can help us achieving these goals easier and faster.

Another important problem, more and more serious every year, is pollution. The situation was well described in an article by Legambiente Lombardia, based on Arpa Lombardia data few months ago, concerning the increasing presence of CO2 gases, especially in the urban or metropolitan areas (see Reference Documents for further details).

The aim of this document is to outline a computer system which will hopefully help citizens have a better activity schedule for any purpose. Once the project is approved, this document, along with other ones that will follow, will constitute a solid, non-technical guide to developers' work.

1.B Scope

To be more specific, many users want to access to the service in order to have time schedule or activities planning managed by the system, including the basic reminder function of a calendar.

Furthermore this document wants to describe an application which should have many advantages such as: reducing the travel time between appointments according to the preferred means of transport or to the weather; preventing users from being late for appointments; giving the possibility of buying public or private transport tickets and locating car/bike sharing systems.

1.B.1 Goals

Here is the list of goals, in a numbered list, in order to easily refer to a specific goal in the following sections:

- **[G.1]**: allow a Guest to create an account and log into it
- **[G.2]:** allow the system to update the stored info (timetables and weather forecasts)
- **[G.3]**: allow the user to buy a specific public transport ticket
    - **[G.3.1]**: allow the user to buy day/week/season pass
    - **[G.3.2]**: allow the user to buy the ticket when the journey has just been computed or when the journey is about to start (a warning can be shown)
- **[G.4]**: allow the user to locate the nearest bike or car of a bike/car sharing system
- **[G.5]**: allow the user to add an activity to his calendar or modify an existing one
    - **[G.5.1]**: if the system detects an added activity whose location is unreachable in the allotted time the user will be warned

- **[G.6]**: allow the user to calculate the best route from an activity to the next one
    - **[G.6.1]**: allow the system to support multiple means of transport
    - **[G.6.2]**: if the system detects that weather conditions aren't suitable for some means of travel, those one will not be suggested for any route until weather conditions will be adequate
    - **[G.6.3]**: if the system detects that the selected mean of transport is unsuitable for the distance/purpose, notifies the user and changes it
- **[G.7]**: allow the user to be in time for his appointments
    - **[G.7.1]**: if the system detects that selected public transport is liable to strikes, the system will avoid this public transport and it will notify the user
- **[G.8]**: allow the user to select different kind of preferences
    - **[G.8.1]**: allow the user to have a list of banned means of travel that will be never suggested for any route
    - **[G.8.2]:** allow the user to select preferred means of transport depending on the time of the day
    - **[G.8.3]**: allow the user to have a lunch time interval with a custom duration and flexible interval
    - **[G.8.4]**: allow the user to minimize carbon footprint
- **[G.9]**: allow the user to consult his calendar

In the next parts of this document we will refer only to [G.n] goals including also the sub-goals [G.n.m] correlated.

This computer system will benefit not only the users, by managing their activities in a smart way, but also all the people thank to pollution decrease and quality life improvement. Users will be able to access the services through the application.

1.B.2 Phenomena Distinction

In order to have a better comprehension of the situation we are going to face with, here there is a useful list of the different phenomena we are taking into account from now on in the project, divided by area of competence.

Phenomena entirely located in the world

- Effective journey from a place to another
- Facts happening during an activity
- Taking various means of transport
- Deciding preferences
- Buying a ticket (if the user buys tickets by himself without using the feature of the application)

Shared Phenomena

- Account creation
- Login

- Showing the Calendar
- Changing Preferences
- Addition/modification/deletion of an event
- Buying a ticket (through the application)

Phenomena entirely located in the machine

- Best journey computation
- Retrieve and store data (weather and transport)
- Retrieve user account preference
- Checking overlapping of events
- Locate the nearest bike/car of a sharing system
- Activity Reachability Checks

1.C Definitions, Acronyms, Abbreviations

1.C.1 Definitions

- ***Customer/User***: the person who makes use of services available through the application, gaining access to a personal account in order to be able to use the functions of the system. In any case the customer is also the person who actually benefits from the system functionalities as the system has a one-to-one interaction with every different customer.
- ***Application***: a program designed to work on every device such as smartphones and tablets. In this specific context a customer may have more than one device, connected with the same account. We will refer to the application without highlighting any detail concerning the typology of the implementation of it, since this will be analysed in the following documents.
- ***System***: in this document we will refer to the system as the actual computer-based services provider where the application is the way to have access to it, the system is made of many components which will be analyzed further on.

1.C.2 Acronyms

- ***"RASD"***: Requirement and Analysis Specification Document
- ***"RACS"***: Reliable Array of Cloned Services
- "TLS": Transport Layer Security protocol
- "SSL": Secure Sockets Layer protocol

1.C.3 Abbreviations

- [G.n]: is the goal number n;
- [D.n]: is the domain assumption number n;
- [RE.n]: is the functional requirement number n;

1.D Revision history

- version 1.0: first release;

- version 1.1: Deletion of "mobile application" and "installing" references replacing them with abstract concept of application (see Application definition in 1.C.1 section) + scenario 1 and 2 modification and scenario 4 refactoring + Use Case Diagram modification;

1.E Reference Documents

We will refer to some external documents, data, facts, through this document and they are all listed below:

- "Mandatory Project"
  - https://beep.metid.polimi.it/documents/121843524/7b4699a6-8343-4c13-b9eb-8254828cb4af
- LegAmbiente Lombardia article concerning pollution in Lombardy, 17th March 2017
  - http://lombardia.legambiente.it/contenuti/comunicati/smog-7-province-superano-i-35-giorni-di-sforamento-di-pm10-milano-media-di-60mi
- Tickets legislation (few examples)
  - http://www.trenitalia.com/tcom/Informazioni/Condizioni-Generali-di-trasporto/Condizioni-Generali-di-trasporto
  - http://www.trenord.it/it/assistenza/condizioni-di-trasporto.aspx
  - https://www.atm.it/it/Pagine/CondizioniDUso.aspx

1.F Document Structure

This document develops as follows. In chapter 2 we give a general description of the factors that affect both Travlendar+ system and its requirements. In particular, this section will focus on the functionalities provided by the system, the constraints we have to impose, the assumptions we make. Requirements are analysed more in detail in chapter 3. Formal and informal graphical representations are given as a support.

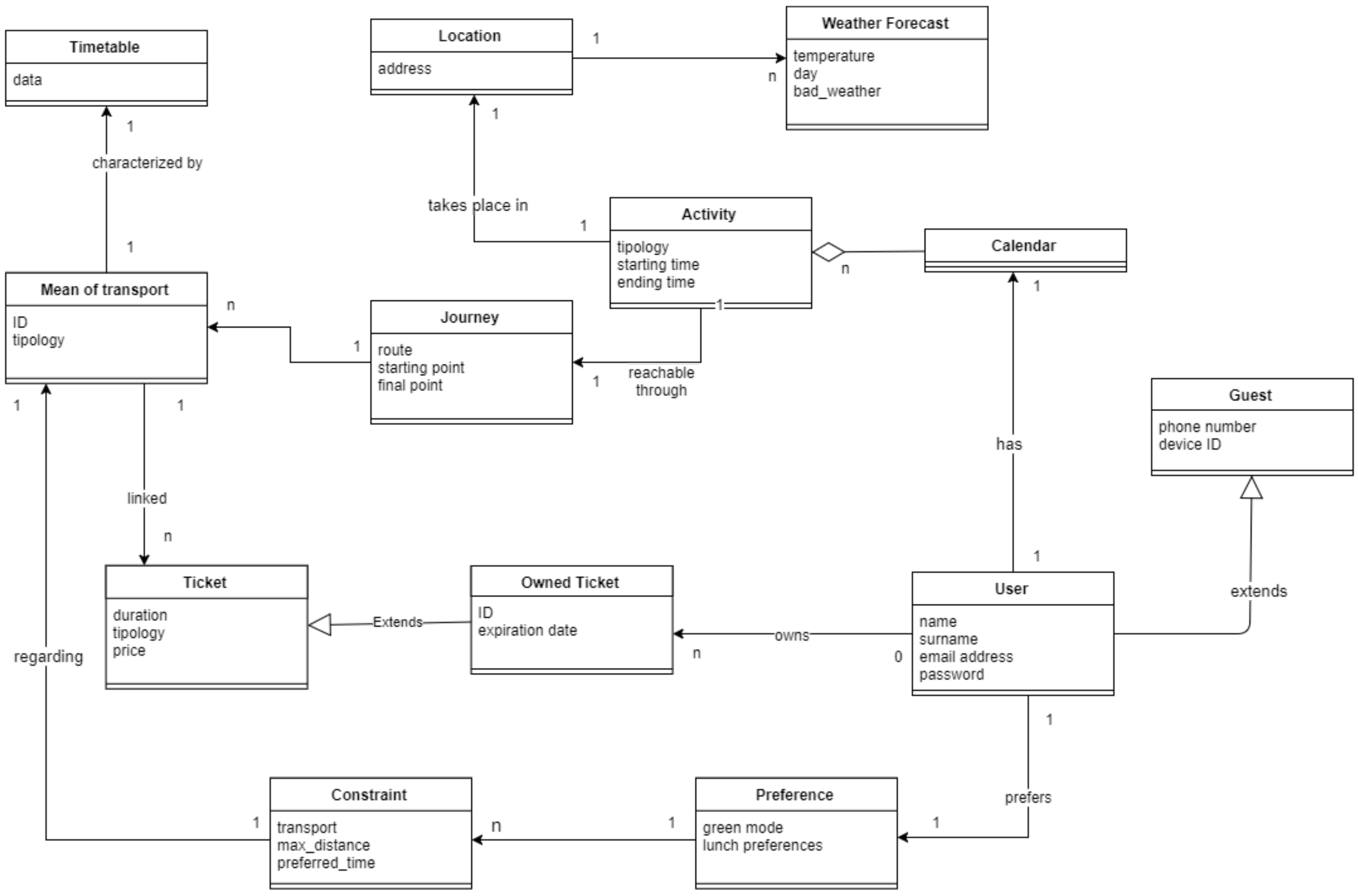
2 OVERALL DESCRIPTION

2.A Product Perspective

Before analyzing in detail all the aspects that will define the system we are proposing, we find appropriate to state its structure, and give mnemonic names to the different components which are related to our project.

The system all the other components rely on is called ***Travlendar+,*** it will be accessed by people in search of service, named ***Users/Customers***, who will administer their own account (for example by declaring/modifying their preferences…). Customers will be able to use an application, ***MyTravlendar+***, running on a device to access the system services.

Here a graphical description of the whole system, through an high level class diagram, follows:



2.B Product Functions

Before introducing analysis in a more detailed way, a brief summary of the major functions that the software will perform can be very useful.

Users can request the main function of the system which consists of the management and schedule of any kind of activity (professional and private) in order to have a clever distribution of the activities during the day, week or month. Another function consists of a user who wants to add an activity to the system; data asked by the system are provided by the user and then the procedure starts. The system executes the procedure analyzing also the current state of the user time-table in order to detect if the new activity is acceptable or it is overlapping with something (there can be many other reasons to reject the addition of an activity such as the impossibility to reach the following destination in time), in this situation the system will provide a warning, telling the user to change the time-slot for the activity he/she wanted to add.

Every user is connected to an account, which has a section used to specify some criteria or preferences stated by the system to compute the best choice of travel (computing also the time needed for transport). These main functions are supported by “other functionalities” such as the time reservation for lunch, the location of the nearest car/bike of a sharing system and the possibility to buy tickets of supported public transport.

All these functions will be discussed in more details further in this document and in the following ones, according to domain-specific assumptions, and dependently on many other factors such as contracts with transport companies etc...

2.C User Characteristics

We are not able to distinguish different kinds of users in a restricted way, because we have just one type of user which interfaces with the system: the customer. Our system goal is to provide services in a personal way through the interaction with many other external systems but we cannot classify a service dealer, a service customer or any other kind of user of our system...

There can be a basic distinction over different types of users, according to the registered activities such as: users with only private life activities, with only professional events or a mixed usage of Travlendar+. In more detail users will also be classified according to their use of other software capabilities (buying transport tickets, bike/car sharing) generally described as “user preferences”.

Users can be of any age, therefore no specific technological knowledge but a very basic one is needed in order to benefit from the minimum services available through the system.

2.D Assumptions, dependencies and constraints

Since the environment in which the system is supposed to be operating is not totally defined, some assumptions are to be made:

- **[D.1]** no user movement during an activity (it starts and finishes in the same place). We must make this assumption, otherwise the user should have explicitly calculated again the journey to the next appointment through the application;
- **[D.2]** activities inserted in the system reflect the truth, there are no unregistered activities and if a user adds an event or an activity, he is supposed to participate in it;
- **[D.3]** the area covered by the system is limited to Lombardy;

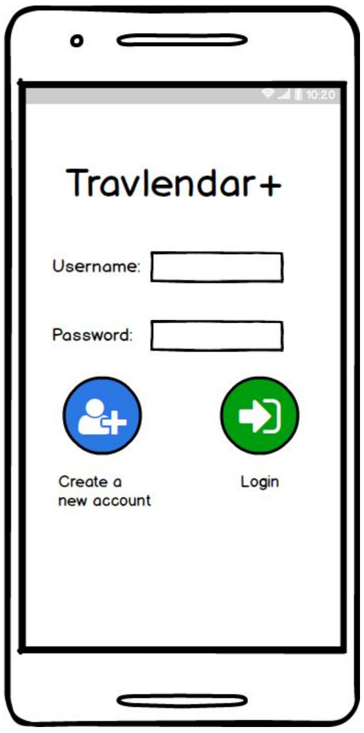
- **[D.4]** users have at least a basic knowledge of their own devices, internet connection and anything else which might be needed such as credentials;
- **[D.5]** a deadline (as preference) will be instituted. Activity journeys whose deadline has finished are fixed and can’t be recalculated (the activity can be deleted in any case). The application guarantee the journey consistency of activities, whose deadline has finished, taking into account the weather forecasts and transport timetables collected at that time;
- **[D.6]** bike/car sharing data will be provided by bike/car sharing companies when requested;
- **[D.7]** the computed transport time is based on an average speed for bikes, travel times for cars (including departure time, distance...) and related info provided by public transport companies (every time a fixed interval occurs data will be updated);
- **[D.8]** weather data are collected by an external provider, with updates sent according to company contracts (every time a fixed interval occurs data will be updated);
- **[D.9]** transport tickets already bought are under the legislation of the provider company (see Reference Documents section for further details);
- **[D.10]** public transport timetable is recorded through official websites of the corresponding companies (strikes, timetable modifications etc);
- **[D.11]** buying a ticket can be done only when the activity enters in the deadline period (a warning will be shown);
- **[D.12]** the location of the nearest bike/car will be computed after having shown a warning before starting the relative journey, during the journey computation instead time travelling will be calculated by the system through statistic data;
- **[D.13]** the booking of bikes/cars of sharing systems is ruled by companies legislations.

3 SPECIFIC REQUIREMENTS

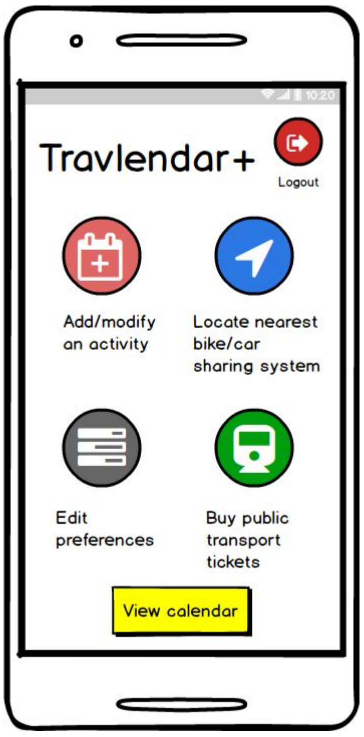
3.A External Interface Requirements

3.A.1 User Interfaces

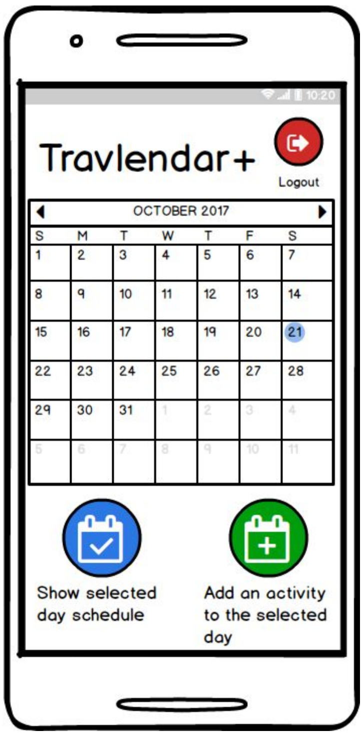
In this section we specify the characteristics of each interface interposed between the software product and its user. These are just mock-ups of the application interface and can be used to have an idea on how the user interface will look like, in the continuation of the project some changes may be done.



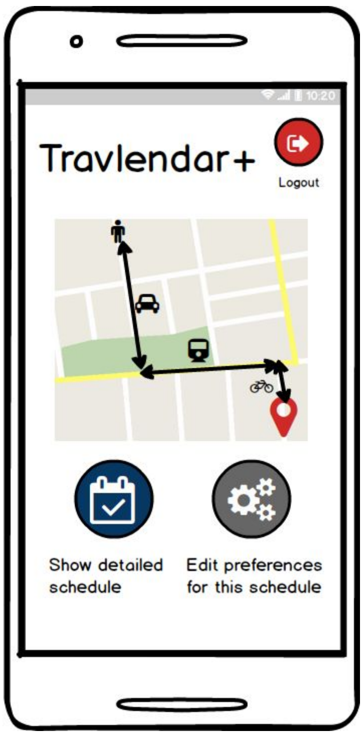
Homepage



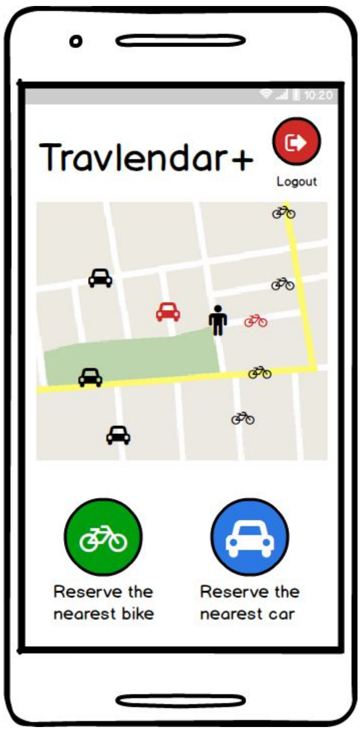
Account Homepage


Travlendar+
Logout
OCTOBER 2017
S M T W T F S
Show selected day schedule
Add an activity to the selected day

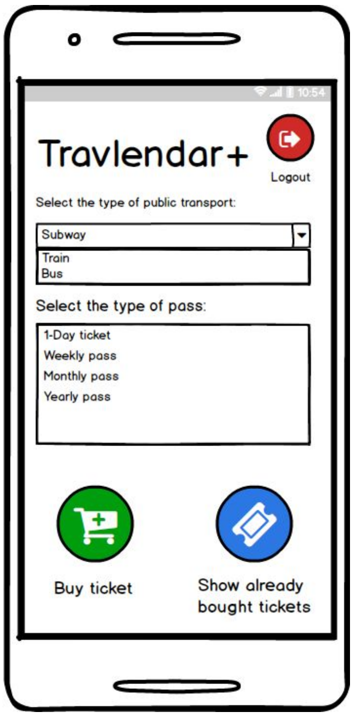
Calendar



Journey Computation



Locate the nearest bike/car of a sharing system



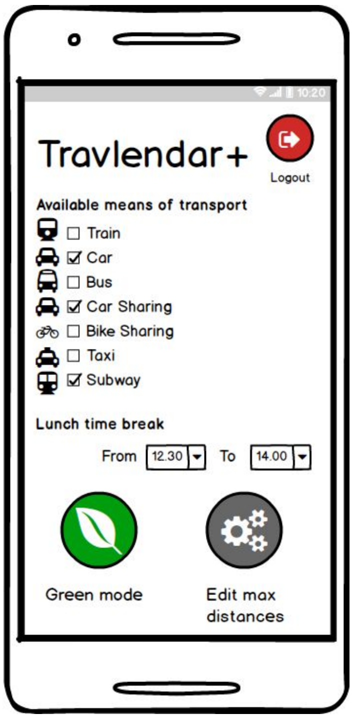
Select the type of public transport:

Subway
Train
Bus

Select the type of pass:

1-Day ticket
Weekly pass
Monthly pass
Yearly pass

Buy Tickets



Available means of transport

- ☐ Train
- ☑ Car
- ☐ Bus
- ☑ Car Sharing
- ☐ Bike Sharing
- ☐ Taxi
- ☑ Subway

Lunch time break

From 12.30 To 14.00

Manage preferences (2)

3.A.2 Hardware Interfaces

The project does not require any additional specific hardware interface over the users' devices in order to fulfill the provided goals, so the reader is redirected to 3.D.2 section to find more details on user hardware limitations or constraints.

3.A.3 Software Interfaces

However there can be many operating systems on different devices, which can bring to potentially different configurations, the application should work on the main operating systems, independently from its implementation. There will be an interface whose aim is to connect the system with the single device or user. In this sense we are building a system able to respond to user requests independently from their device operating system.

3.A.4 Communication Interfaces

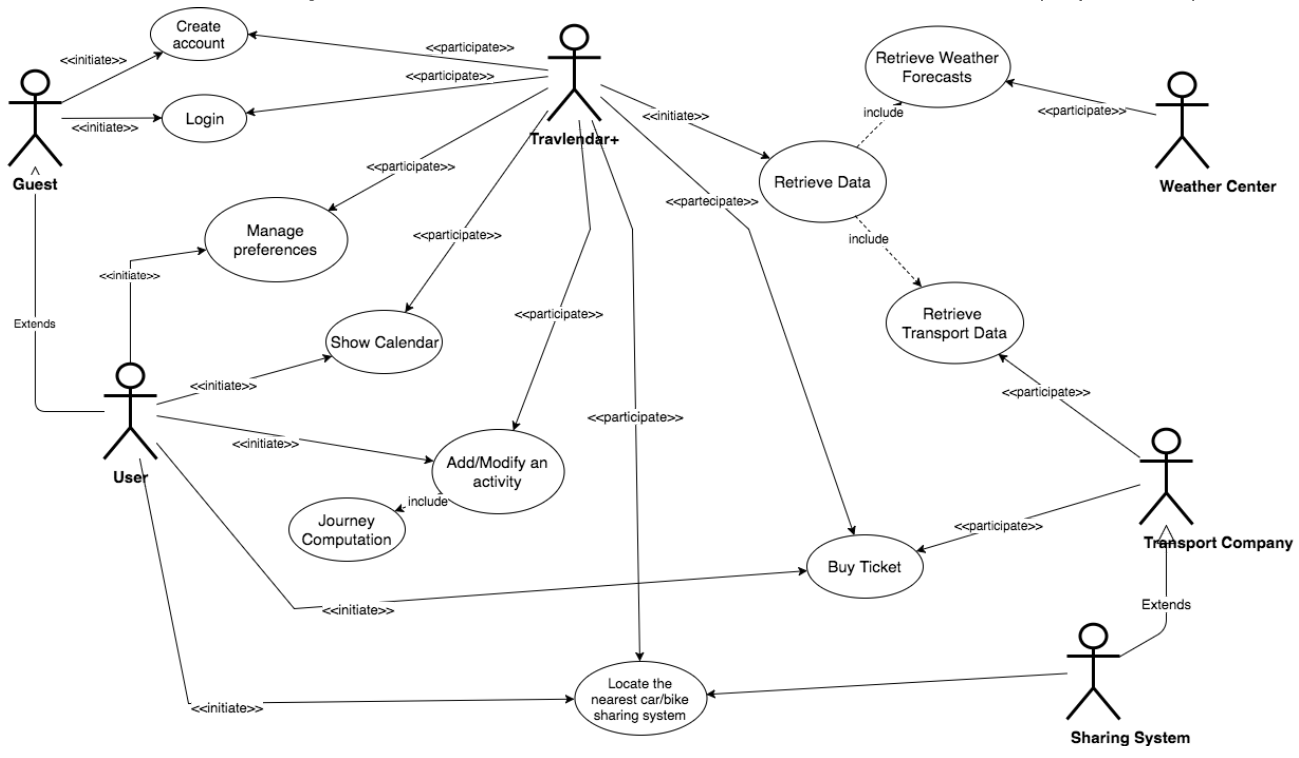
Few communication interfaces are required in order to satisfy the connection between components of our system:

- Connection between the application and the system uses the HTTPS protocol to send and receive data.
- Connection between the system and external interfaces is also provided through HTTPS when available, otherwise HTTP protocol will be used to retrieve updated data for weather forecasts and transport timetables, strikes etc...

3.B Functional Requirements

In this section we will focus on the functional requirements of the project, that will be declared in a numbered list in order to refer to them in an easier way later on.

Here is a Use Case Diagram to better understand the whole situation that the project will provide.



3.B.1 Requirements list

First of all, we show an overview of all the requirements through the Use Case Diagram, and then they will be analyzed individually in more detail.

Now it follows the numbered list of requirements using the schema:

- Goal that has to be reached
  - **Requirements needed**
    - Correlated assumptions

- [G.1]: allow a Guest to create an account and log into it
  - **[RE.1]: A visitor is able to begin the registration process. During the process the system will ask him/her to provide credentials.**
  - **[RE.2]: The system is able to check if credentials are correct or not and also if there's already an account linked to the inserted mail for registration procedure.**
  - **[RE.3]: The system is able to store the data inserted during the registration.**
    - [D.4]
- [G.2]: allow the system to update stored info (timetables and weather forecasts)
  - **[RE.4]: The system is able to retrieve data and info from Transport Companies and Weather Center.**
  - **[RE.5]: The system is able to retrieve data from external providers.**
    - [D.8], [D.10]
- [G.3]: allow the user to buy a specific public transport ticket
  - **[RE.6]: The user is able to select tickets and the relative amount he wants to buy.**
  - **[RE.7]: The user receives a receipt of payment within the tickets he purchased.**
  - **[RE.8]: The user is able to buy online through most common payment methods such as PayPal and credit cards.**
    - [D.3], [D.4], [D.9]
- [G.4]: allow the user to locate the nearest bike or car of a bike/car sharing system
  - **[RE.9] The system is able to provide the list of available bikes within a certain area.**
  - **[RE.10]: The system is able to collect the location from the calendar regarding the previous registered activity.**
    - [D.1], [D.2], [D.3], [D.4], [D.6], [D.13]
- [G.5]: allow the user to add an activity to its calendar or modify an existing one
  - **[RE.11]: The application must prevent data loss.**
  - **[RE.12]: The system is able to verify whether the additional event is reachable and the journey is feasible.**
    - [D.2], [D.3], [D.4], [D,5], [D.7]
- [G.6]: allow the user to calculate the best route from an activity to the next one
  - **[RE.13] The system is able to retrieve activities data from the calendar.**
  - **[RE.14] The system is able to retrieve weather and transport data.**
  - **[RE.15] The system takes into account user preferences during the journey computation.**
    - [D.2], [D.3], [D.5], [D.7], [D.8], [D.10], [D.11], [D.12], [D.13]
- [G.7]: allow the user to be in time for his appointments
  - the same for [G.6]

- [G.8]: allow the user to select different kind of preferences
    - **[RE.16]: The system is able to store account values regarding different preferences.**
        - [D.4]
- [G.9]: allow the user to consult his calendar
    - **[RE.17]: The system is able to provide to the user all the list of registered events.**
        - [D.4]

3.B.2 Scenarios and Diagrams

In this section it follows a sequence of possible scenarios, each of them regarding a specific Use Case present in the Use Case Diagram. For each scenario is given a relative table, for most of them a sequence diagram and in some specific cases also other kinds of diagram such as activity or state chart.

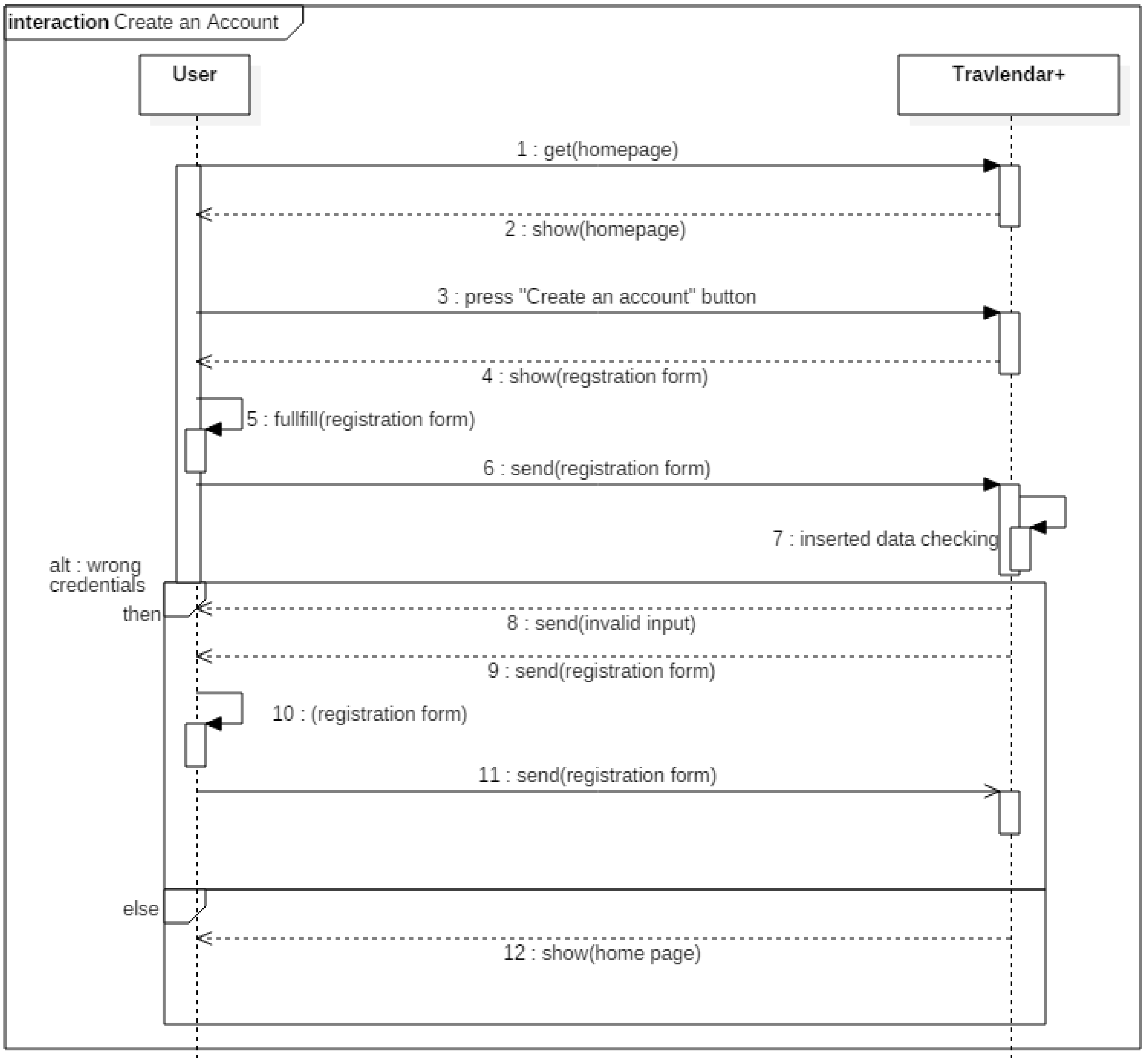
In order to have a better readability of the document we divided every scenario and its relative sequence diagram in single pages. A table like the following will be used in order to describe every single Use Case.

Name of the Use Case	
ACTORS	Components that are in relationship within the use case
GOALS	Aims satisfied by the use case
REQUIREMENTS	Functionalities required in order to achieve the goals
ASSUMPTIONS	Domain assumption dependencies that must be taken in order to achieve the goals through the given requirements
INPUT CONDITIONS	The entry point of the given use case in order to start the process
EVENT FLOW	Description of the single relations that intercur between the actors in a numbered list (sequential point of view). This describes the normal event flow, without unexpected events, errors etc.
OUTPUT CONDITIONS	The exit point that the process achieves in the normal event flow
EXCEPTIONS	This section describes what can go wrong and the relative exit/restarting point

Scenario 1

Theresa is new on Travlendar+ and since she never used this application she wants to create an account. Opening the application, the login page is shown, she presses the "Create an account" button and then she is redirected to the registration page. She inserts all the required data and press the "Done" button. She receives an email to confirm the successful registration.

<u>Create an Account</u>	
ACTORS	User, Travlendar+
GOALS	[G.1]
REQUIREMENTS	[RE.1], [RE.2], [RE.4]
ASSUMPTIONS	[D.4]
INPUT CONDITIONS	The application is correctly available.
EVENT FLOW	1. The user opens the application from his device for the first time. 2. Travlendar+ opens the home page and asks credentials for "email" and "password" or shows the "Create an Account" button. 3. The user presses the button. 4. The system redirects the user to the registration page. 5. The user inserts the required data and press the "Done" button.
OUTPUT CONDITIONS	The user is redirected to the application homepage and he receives a confirmation email.
EXCEPTIONS	● The user inserts an already used email ● The user doesn't insert all the required data. All exceptions are handled by Travlendar+ which notifies the user with the correct message and redirects him to the application homepage.

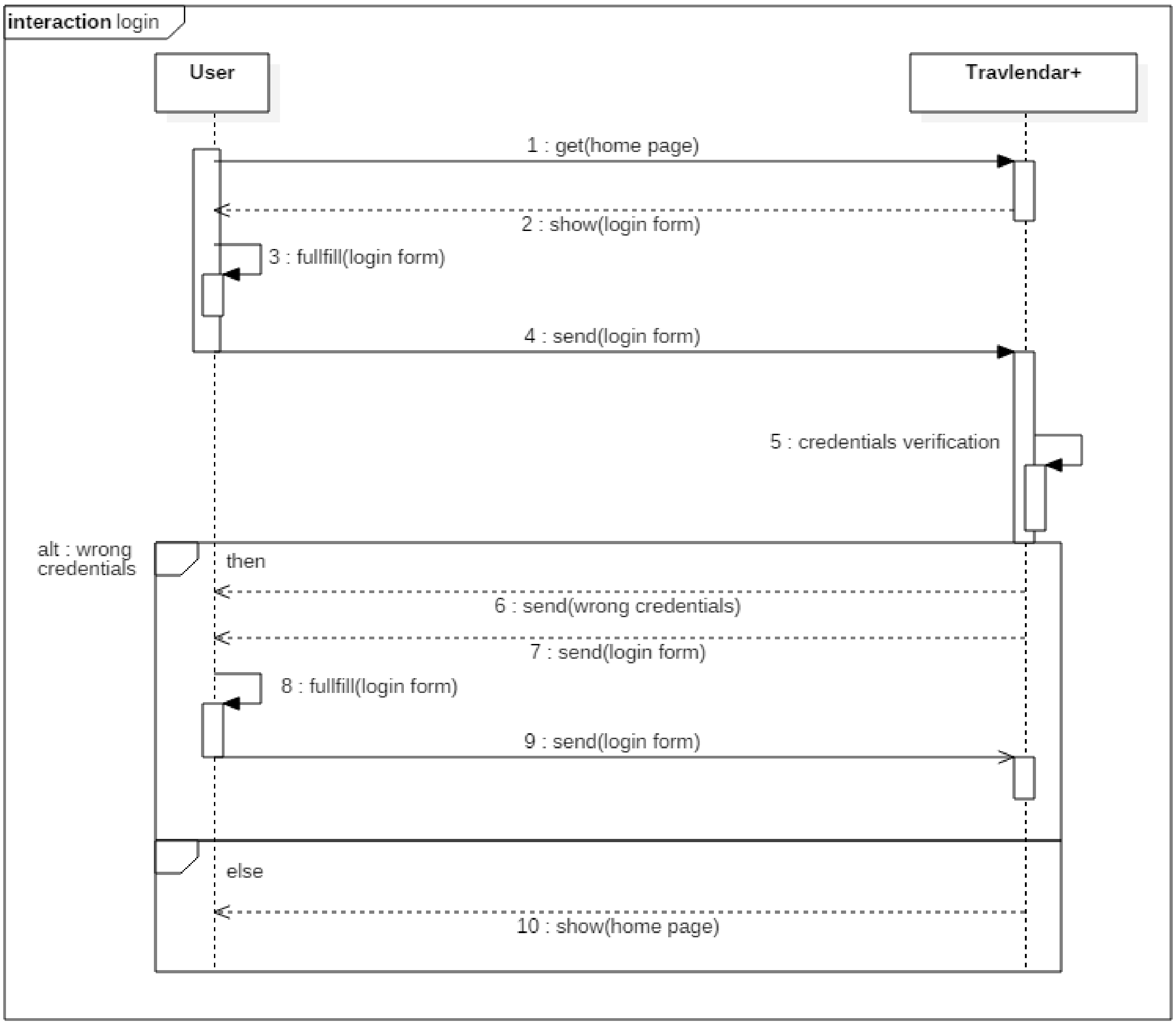

interaction Create an Account
User
Travlendar+
1 : get(homepage)
2 : show(homepage)
3 : press "Create an account" button
4 : show(regstration form)
5 : fullfill(registration form)
6 : send(registration form)
7 : inserted data checking
alt : wrong credentials
then
8 : send(invalid input)
9 : send(registration form)
10 : (registration form)
11 : send(registration form)
else
12 : show(home page)

Sequence Diagram related to "Create an Account" Use Case

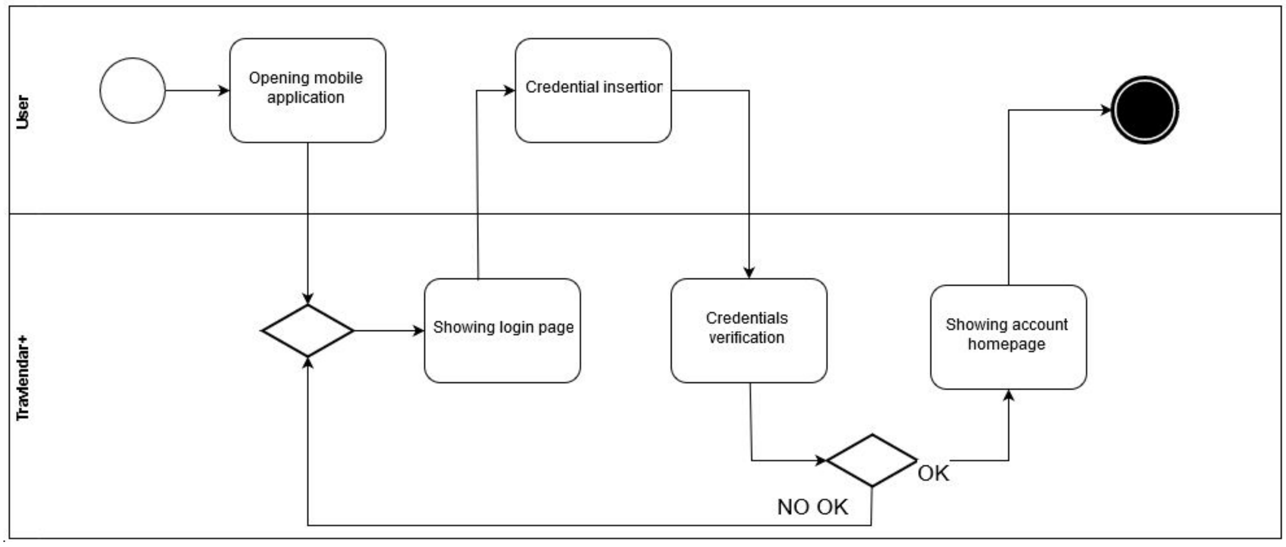
Scenario 2

Paul has just changed the smartphone but he has an already registered account on Travlendar+ so he opens the application. He inserts the credentials already used in the previous device. The system checks the inserted email and password and shows his account homepage.

Login	
ACTORS	User, Travlendar+
GOALS	[G.1]
REQUIREMENTS	[RE.1], [RE.2], [RE.4]
ASSUMPTIONS	[D.4]
INPUT CONDITIONS	The user has already an account.
EVENT FLOW	1. The user opens the application from his device. 2. Travlendar+ shows the home page and asks credentials for “email” and “password”. 3. The user inserts the required data and press the “Login” button. 4. The system redirects the user to the home page of his account.
OUTPUT CONDITIONS	The user sees his own account home page with menu, preferences etc...
EXCEPTIONS	● The user inserts a non valid username. ● The user inserts a non valid password. All exceptions are handled by Travlendar+ which notifies the user that credentials are wrong and taking back the Event Flow to the point number 2.



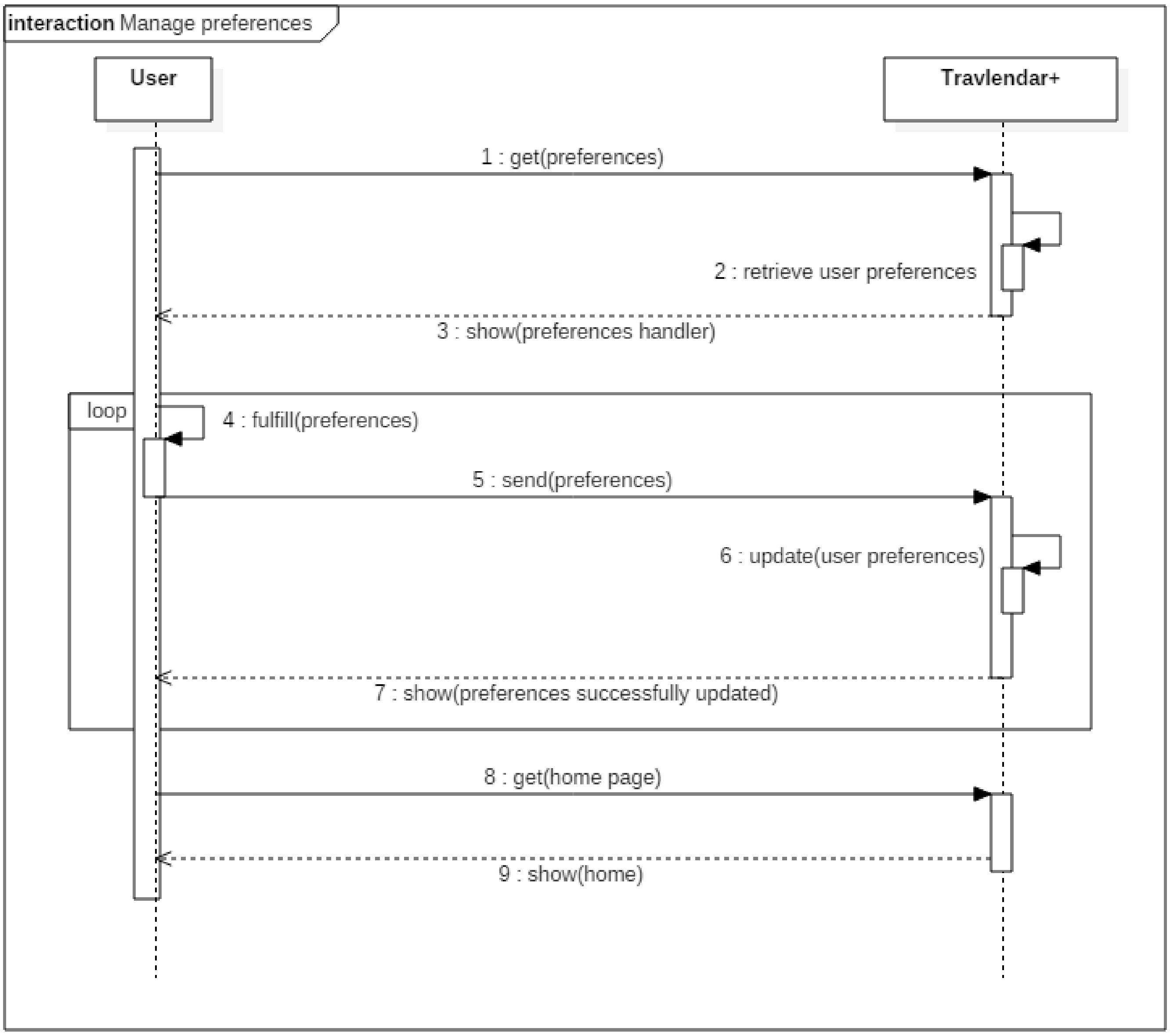
Sequence Diagram related to "Login" Use Case (up) and Activity Diagram (down)



Scenario 3

Jack has seen a documentary last night and he was impressed by things he heard about environment and human health. Now he wants to modify his account preferences in order to become more "green" and to do more motor activity. He goes in the Preference Settings section of the account and checks the "Green Mode" option that minimizes the carbon footprint of his journeys then he increases the coverable distance on foot and by bike.

Manage Preferences	
ACTORS	User, Travlendar+
GOALS	[G.8]
REQUIREMENTS	[RE.16]
ASSUMPTIONS	[D.4]
INPUT CONDITIONS	The user is already logged and he is on the homepage of his account.
EVENT FLOW	1. The user press the "Preferences" button on the application. 2. Travlendar+ retrieves data from the system 3. Travlendar+ shows the preferences management page we are provided all possible mentioned options. 4. The user select the desired option and choose his preferred value. 5. Travlendar+ retrieves the inserted data and updates the preferences on the system. 6. The user exits from the preferences management page.
OUTPUT CONDITIONS	All the changes are updated in the account
EXCEPTIONS	/


interaction Manage preferences
User
Travlendar+
1 : get(preferences)
2 : retrieve user preferences
3 : show(preferences handler)
loop
4 : fulfill(preferences)
5 : send(preferences)
6 : update(user preferences)
7 : show(preferences successfully updated)
8 : get(home page)
9 : show(home)

Sequence Diagram related to “Manage Preferences” Use Case

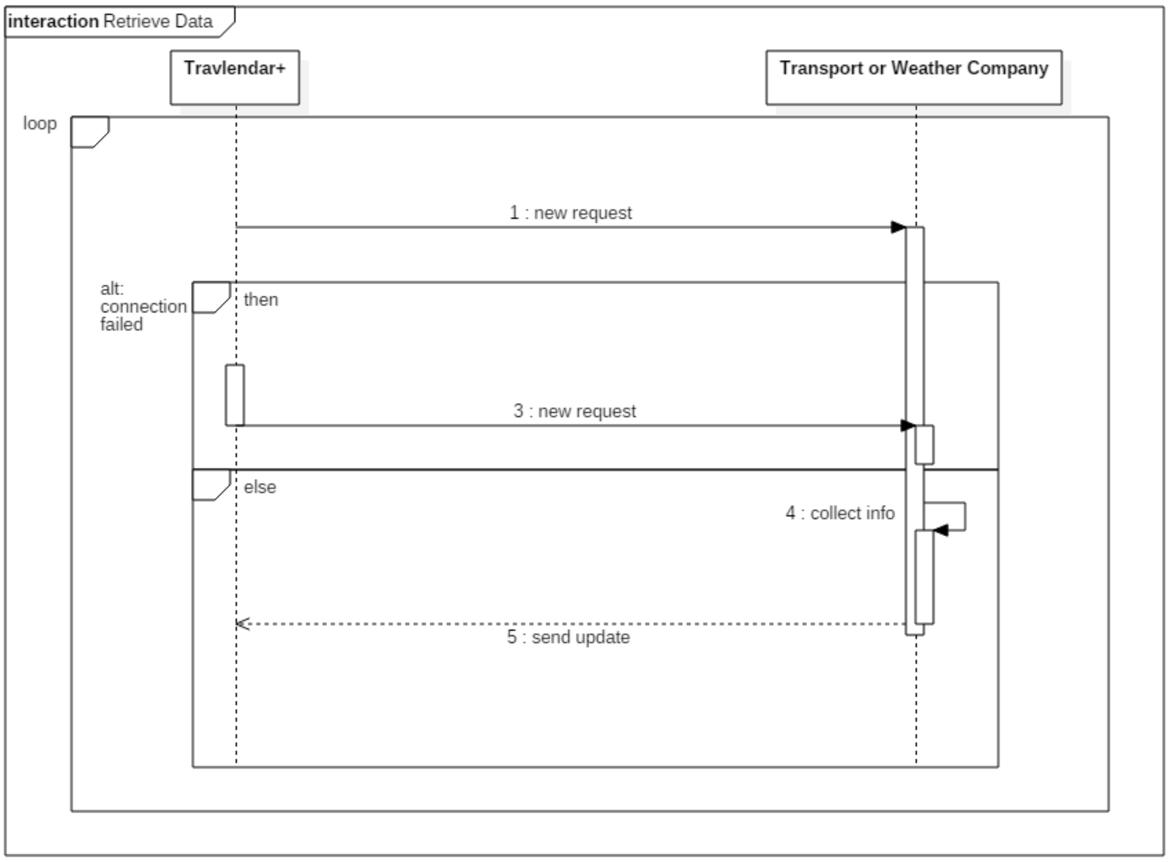
Scenario 4

Travlendar+ is asked to retrieve data for a specific request to a specific external agent has just finished. The system sends a request and waits for new upcoming data. The Transport Company/Weather Center receives the request, collects all the needed data included in the signed contract with Travlendar+ and sends them.

NB With this scenario we'd like to describe not only the "Retrieve and Store Data" use case, but also the use cases that are directly included in this functionality (Update Weather Forecasts and Update Transport Data). Only one Sequence Diagram will be shown for these use cases.

Retrieve Data	
ACTORS	Travlendar+, Weather Center, Transport Companies
GOALS	[G.2]
REQUIREMENTS	[RE.4], [RE.5]
ASSUMPTIONS	[D.10], [D.8]
INPUT CONDITIONS	The system has already retrieved data from external system at least once and a new request arrives..
EVENT FLOW	1. The system asks to the related external system for updated info. 2. Procedure of "Update Weather forecasts" or "Update Transport Data" are executed. (see related Use Cases) 3. The retrieved data are given to the process was asking for.
OUTPUT CONDITIONS	Tthe system waits for the next input conditions verification.
EXCEPTIONS	The Company from which the system wants to retrieve updated data may be unreachable in instruction number 2, in this case refer to exception of the related Use Cases.

Sequence Diagram related to "Retrieve and Store Data" Use Case



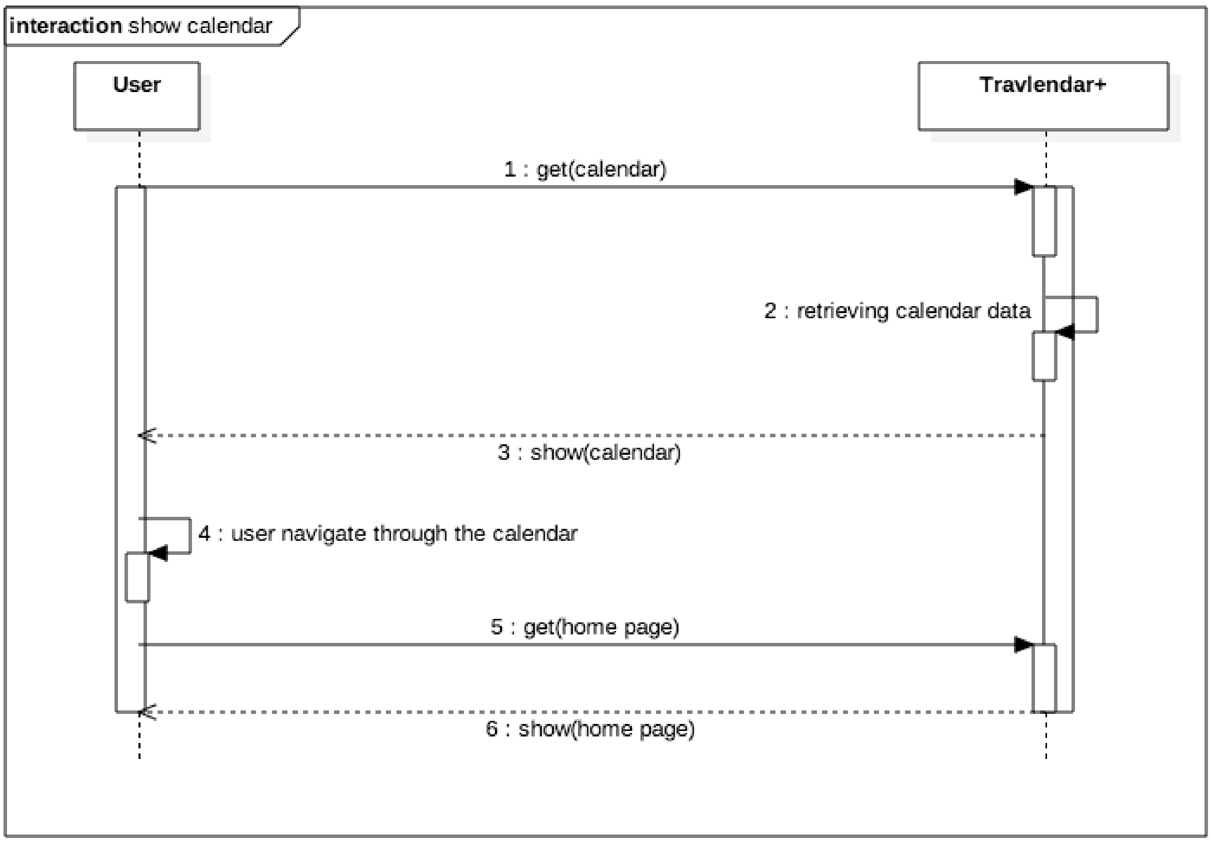
Retrieve Weather Forecasts	
ACTORS	Travlendar+, Weather Center
GOALS	[G.2]
REQUIREMENTS	[RE.4], [RE.5]
ASSUMPTIONS	[D.10], [D.8]
INPUT CONDITIONS	The contract between the actors has been already signed and initialized.
EVENT FLOW	1. Travlendar+ sends the request for an update. 2. Weather Center takes data from its own system and sends back them to Travlendar+ 3. Travlendar+ receives new data
OUTPUT CONDITIONS	See output conditions of “Retrieve and Store Data” Use Case
EXCEPTIONS	Weather Center system is not available / there are problems with connection, so instruction number 1 is repeated until it is rightly executed.

Retrieve Transport Data	
ACTORS	Travlendar+, Transport Company
GOALS	[G.2]
REQUIREMENTS	[RE.4], [RE.5]
ASSUMPTIONS	[D.10], [D.8]
INPUT CONDITIONS	The contract between the actors has been already signed and initialized.
EVENT FLOW	1. Travlendar+ sends the request for an update. 2. Transport Company takes data from its own system and send back to Travlendar+ 3. Travlendar+ receives new data
OUTPUT CONDITIONS	See output conditions of “Retrieve and Store Data” Use Case
EXCEPTIONS	Transport Company system is not available / there are problems with connection, so the System will wait 5s then instruction number 1 is repeated until it is rightly executed.

Scenario 5

Jamie, a registered and logged user, wants to check the appointments of tomorrow. From the account homepage he presses the "Calendar" button and is redirected to his calendar, then he selects the right day/week/month he wants to check, all the events are shown and he surfs through the calendar to check all his activities.

Show Calendar	
ACTORS	User, Travlendar+
GOALS	[G.9]
REQUIREMENTS	[RE.17]
ASSUMPTIONS	[D.4]
INPUT CONDITIONS	The user is already logged and the account homepage is shown.
EVENT FLOW	1. The user presses the "Calendar" button. 2. Travlendar+ retrieves data from the system. 3. Travlendar+ shows the calendar of the user with all his activities. 4. The user is able to move from a month/week/day to another.
OUTPUT CONDITIONS	The user returns to his account homepage.
EXCEPTIONS	There might be a loop between instruction 3 and 4 , depending on the user behavior.

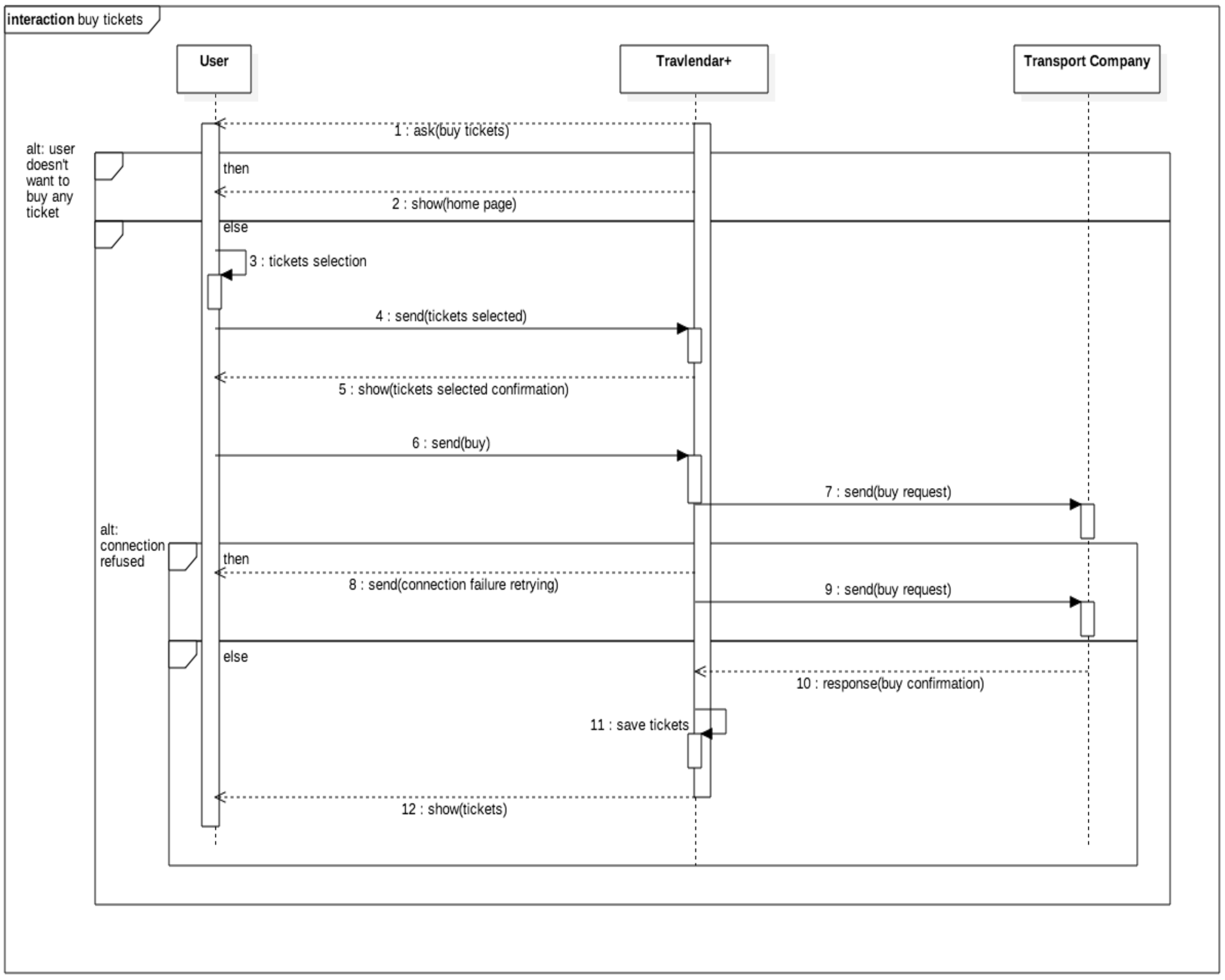

interaction show calendar
User
Travlendar+
1 : get(calendar)
2 : retrieving calendar data
3 : show(calendar)
4 : user navigate through the calendar
5 : get(home page)
6 : show(home page)

Sequence Diagram related to "Show Calendar" Use Case

Scenario 6

Frank receives a notification from Travlendar+ that asks him if he wants buy a public transport ticket for a next journey on his calendar. Frank decides what tickets he wants to buy, the tickets are bought and can be found under the tickets section.

Buy a ticket	
ACTORS	User, Travlendar+, Transport Companies
GOALS	[G.3]
REQUIREMENTS	[RE.6], [RE.7], [RE.8]
ASSUMPTIONS	[D.3], [D.4], [D.9]
INPUT CONDITIONS	The journey computation has already been executed and a warning due to deadline expiration is being shown
EVENT FLOW	1. Travlendar+ asks the user to buy the tickets associated to the selected journey. 2. The users checks the tickets he wants to buy. 3. The system shows a recap of the transaction. 4. The user select the “Buy” button. 5. The system communicates with all the related Transport Companies, signs the related journey as covered (bought tickets) and ends the process showing to the user the “tickets” section of the account.
OUTPUT CONDITIONS	The user receives a confirmation of the correct end of the process.
EXCEPTIONS	● The user might skip this process because he doesn’t want to buy any ticket right now. ● There might be a problem while connecting to Transport Companies

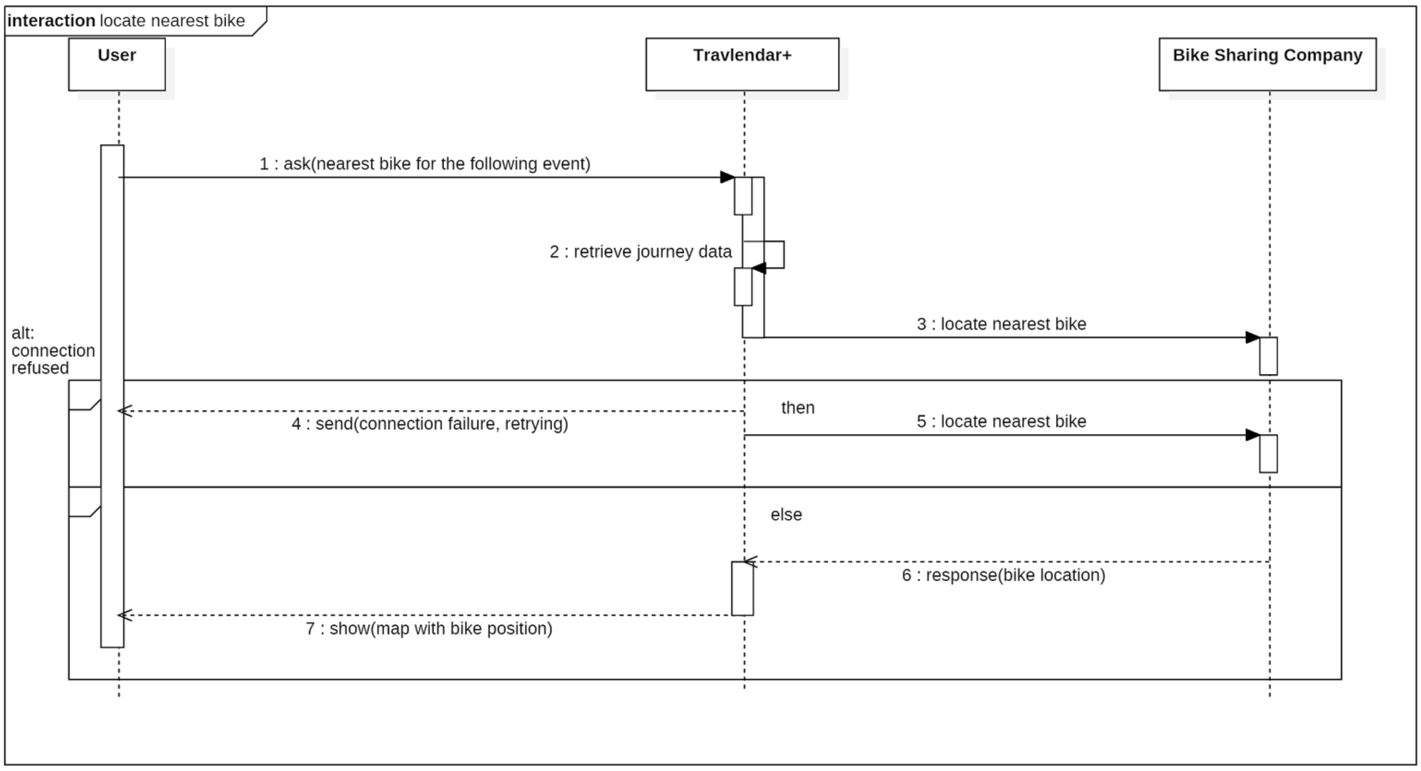


Sequence Diagram related to "Buy Tickets" Use Case

Scenario 7

Paul's device vibrates and shows a warning. The next journey is composed also (or only) of a phase in which a bike/car of a sharing system is needed. He is asked if he wants to locate the nearest car or bike, he presses the "locate nearest car/bike sharing system " and a page is shown. He is now able to see the nearest car or bike depending on what was calculated for the imminent journey.

Locate the nearest car/bike sharing system	
ACTORS	User, Travlendar+, Sharing System
GOALS	[G.4]
REQUIREMENTS	[RE.9], [RE.10]
ASSUMPTIONS	[D.1], [D.2], [D.3], [D.4], [D.6], [D.13]
INPUT CONDITIONS	The user is already logged.
EVENT FLOW	1. The user wants to know how to reach the nearest bike due to a journey by bike in the next future so he presses the relative button. 2. The system retrieves data from his account and select the specific journey. 3. The system asks to the Bike Sharing System to locate the nearest bike to the user position which is a data stored in the calendar. 4. The external system computes the request and sends back the position of the needed means of transport.
OUTPUT CONDITIONS	The system shows a map on the device of the user indicating the position of the destination.
EXCEPTIONS	The Sharing System might be unreachable, in this case the system warns the user until a new request is accepted and computed by the external system.


interaction locate nearest bike
User
Travlendar+
Bike Sharing Company
1 : ask(nearest bike for the following event)
2 : retrieve journey data
3 : locate nearest bike
alt:
connection
refused
then
4 : send(connection failure, retrying)
5 : locate nearest bike
else
6 : response(bike location)
7 : show(map with bike position)

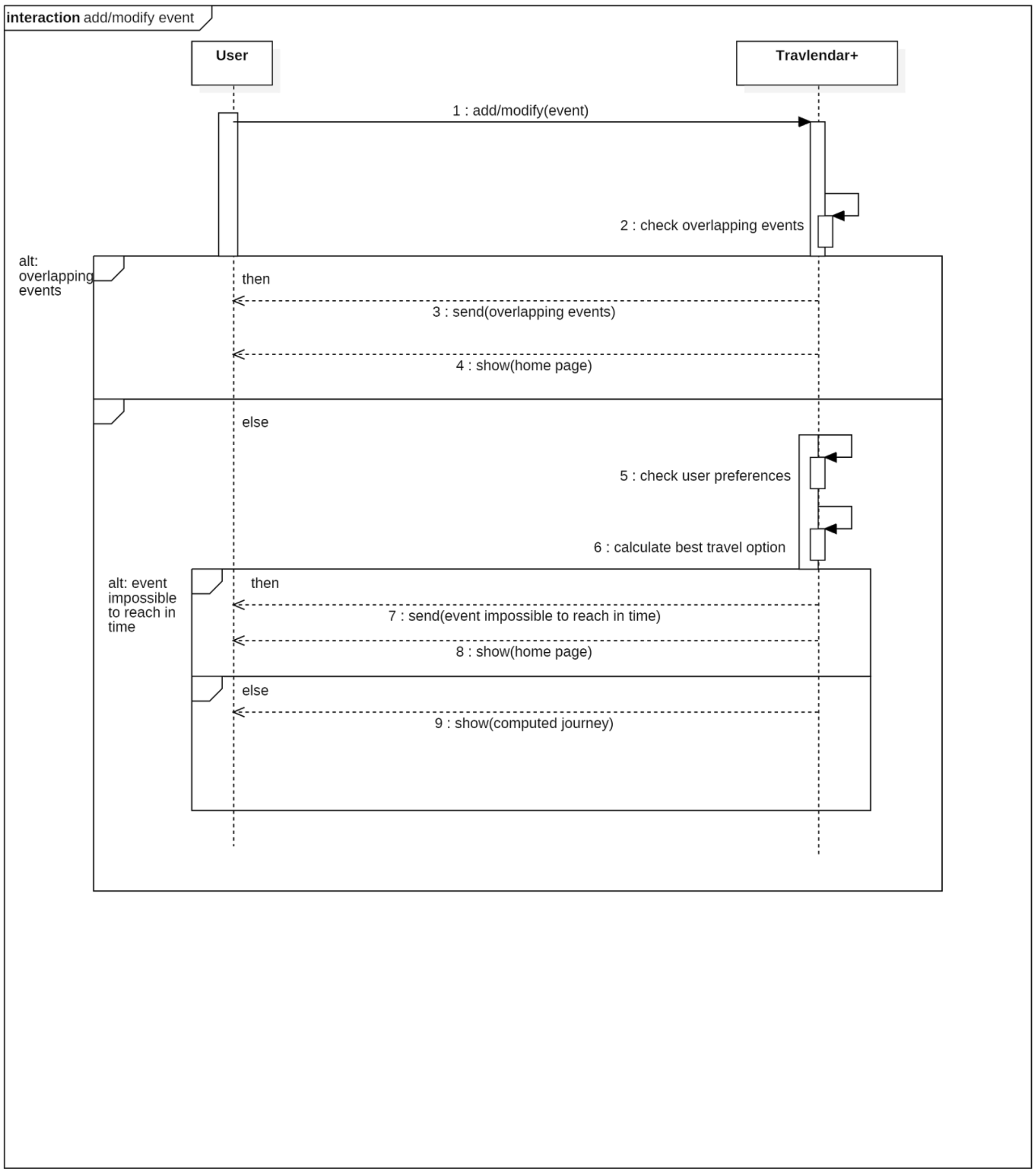
Sequence Diagram related to "Locate the nearest bike of a sharing system" Use Case

Scenario 8

Morgan wants to add an event on his calendar, he fulfills the event informations then presses "done" button, at this moment the application will show him different journeys according to his preferences, Morgan will choose one of those.

Add/Modify** an event	
ACTORS	User, Travlendar+
GOALS	[G.3], [G.5], [G.6], [G.7]
REQUIREMENTS	[RE.6], [RE.7], [RE.8], [RE.11], [RE.12], [RE.13], [RE.14], [RE.15]
ASSUMPTIONS	[D.2], [D.3], [D.4], [D.5], [D.7], [D.8], [D.9], [D.10], [D.11], [D.12], [D.13]
INPUT CONDITIONS	The user selects the "Add/Modify an event" option from the homepage.
EVENT FLOW	1. The system asks to user if he wants to add a new one or modify/cancel an existing one. 2. The user selects to add a new event. 3. The system shows the input form to retrieve all the needed data for a new activity such as location, time, duration etc… 4. The user fills the form and select the "Done" button. 5. The system retrieves the calendar of the user and checks if input is acceptable and if the additional event is overlapping with the existing ones. 6. The system retrieves the account preferences and computes the best travel options (see Journey Computation Use Case for more detail). 7. The user selects the preferred option.
OUTPUT CONDITIONS	The system shows the added event into the calendar, a recap of the journey for that activity and shows the possibility to start the process described in Buy Ticket Use Case.
EXCEPTIONS	● At instruction 5 the check might fail for input incompleteness or due to activity overlapping, in this case the process restarts from instruction 1. ● other exceptions might arise from instruction 6, see the related Use Case for more info.

**** modify an event is also intended as the deletion of an event that follows a different event flow, not described in this document because it’s easy to think about it.**

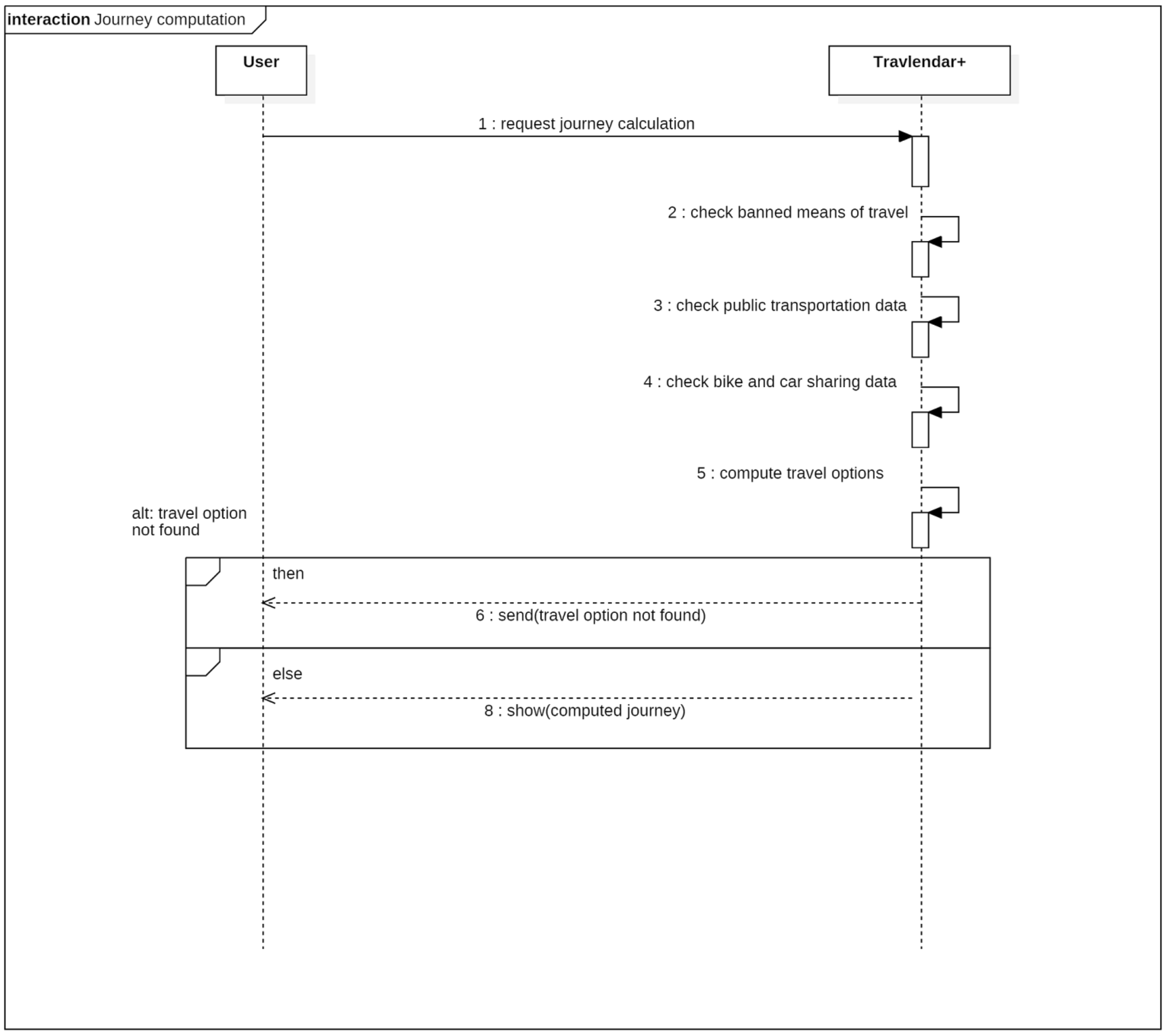

interaction add/modify event
User
Travlendar+
1 : add/modify(event)
2 : check overlapping events
alt: overlapping events
then
3 : send(overlapping events)
4 : show(home page)
else
5 : check user preferences
6 : calculate best travel option
alt: event impossible to reach in time
then
7 : send(event impossible to reach in time)
8 : show(home page)
else
9 : show(computed journey)

Sequence Diagram related to “Add/Modify an event” Use Case

NB since the following Use Case is strictly related to "Add an event" and is included in it we decided not to describe it through a scenario but we think is enough to report the relative Use Case Table.

Journey Computation	
ACTORS	User, Travlendar+
GOALS	[G.6]
REQUIREMENTS	[RE.13], [RE.14], [RE.15]
ASSUMPTIONS	[D.2], [D.3], [D.5], [D.7], [D.8], [D.10], [D.11], [D.12], [D.13]
INPUT CONDITIONS	The user has started the procedure to add/modify an event on his calendar, see related Use Case (event and its info are already inserted).
EVENT FLOW *	1. Travlendar+ verifies that the additional event doesn't overlap with all the events present in the current calendar. 2. Travlendar+ searches in the system for specific weather and transport data processing the inserted info and searches also for user's preferences. 3. Travlendar+ computes all the possible options which satisfy the user's preferences and constraints such as lunch time reservation.
OUTPUT CONDITIONS	Travlendar+ shows the computed options and waits for user's choice.
EXCEPTIONS	● If the check in instruction 1 fails a warning is shown and the process ends. ● If instruction 3 doesn't find any solution, due to a too big travel duration that causes impossibility to reach the next event in time a warning is shown and the process ends.

*** this scenario can also be triggered when the activity enters the time period covered by the deadline (see [D.5]).**


interaction Journey computation
User
Travlendar+
1 : request journey calculation
2 : check banned means of travel
3 : check public transportation data
4 : check bike and car sharing data
5 : compute travel options
alt: travel option not found
then
6 : send(travel option not found)
else
8 : show(computed journey)

Sequence Diagram related to "Journey Computation" Use Case

3.C Performances Requirements

The system has to support the simultaneous connection of all users, who are estimated to be 800’000, estimating that no more than 8% of Lombardy population will use our system. We expect to have on average no more than 5% of customers connected at the same time so we need to support at least 40’000 users, without making any difference about the purpose of the user connection (journey computation, calendar, preference management etc…). This means that our system should have an high bandwidth in order to be able to satisfy multiple requests.

Since there are many functionalities that the user can require and it is impossible to detect any “peak workload interval”, we cannot estimate anything about the performances of a single user procedure because it depends on the weight of its process (what he is looking for) and on the other procedures that are being executed at the same time.

3.D Design Constraints

3.D.1 Standards compliance

Since the application is designed for devices, users will interact with the system through their own devices. In this sense the system will be able to provide interfaces in order to be connected with all other components needed such as customers interfaces and external interfaces of other actors connected with our system.

3.D.2 Hardware limitations

In this section are described the hardware constraints needed by our application to compute its tasks:

- Since the application will mainly request procedures to the system a specific hardware is not required, every kind of device in commerce is able to fulfill hardware minimum requirements such as memory storage, RAM and CPU properties.
- As written in [D.4] assumption we require at least the capability of the device to have a stable internet connection, either wifi or 3G

3.D.3 Any other constraint

Since this document is the first of others that will follow, other constraints will be discussed in more detail further on.

3.E Software System Attributes

3.E.1 Reliability

The reliability of the system is strictly related to the uptime of the system, we can improve reliability using RACS which increase performance during heavy loads and provide fault tolerance in case of hardware failure.

3.E.2 Availability

We expect no more than 3 sessions of 2 hours of downtime during each month. To be more precise, we aim to have 99.2% of uptime at least, considering possible technical and maintenance activities, already included in the 3 sessions per month.

We expect also that users are able to have a stable internet connection, otherwise we cannot guarantee any kind of service.

Since our system is able to retrieve and store data from external systems, the availability of updated data is related to the availability of that external systems, nevertheless our system is able to execute users' requests even when updated data are not available thanks to the storing ability.

3.E.3 Security

Considering that Travlendar+ stores many sensitive data about users such as their appointment times and locations, we want to ensure that data are stored safely in our system.
Usernames and passwords are hashed and salted[1] to mitigate an eventual data leak, while all users stored data are encrypted on the system.
Every connection between the system and the application uses SSL/TLS protocol, to prevent man in the middle attacks[2].

3.E.4 Maintainability

The most important factor of maintainability is without any doubt the well documentation of every part of the project. With this idea we want to have a coherent and well-described documentation, in order to facilitate interventions and function additions in the future.

Another characteristic of maintainability is the adoption of architectural patterns, depending on the language of the implementation, such as a Model-View-Controller pattern if using an OO programming language.

3.E.5 Portability

Travlendar+ has no need of portability, since it will provide interfaces itself to adapt the connection, the execution and the procedures with different operative systems on the different devices. For the application portability please refer to 3.A.3 .

[1]*In cryptography, a salt is random data that is used as an additional input to a one-way function that "hashes" a password or passphrase, salts are used to safeguard passwords in storage preventing rainbow tables attacks.*

[2] *a man-in-the-middle attack (MITM) is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.*

4 FORMAL ANALYSIS USING ALLOY

4.1 Signatures

```
open util/time
open util/integer
open util/boolean

sig User {
   calendar: one Calendar,
   preferences: one Preference
}

sig Preference{
   bannedTransports: set MeanOfTransport,
   greenMode: one Bool
}

sig Calendar {
   activities: some Activity,
   journeys: some Journey
}

sig Activity {
   start_time: one Time,
   final_time: one Time,
   location: one Location,
} {
   gt[final_time, start_time]
}

sig Location {
}

sig Weather {
   location: one Location,
   time: one Time,
   bad_weather: one Bool
}

sig Journey {
   start_time: one Time,
   end_time: one Time,
   start_point: one Location,
   final_point: one Location,
   transports: some MeanOfTransport
} {
   gt[end_time, start_time]
}

sig MeanOfTransport{
   green: one Bool,
   weather_dipendent: one Bool
}
```

4.2 Facts

```
fact noMoreCalendars{
   #User = #Calendar
}

fact noAloneJourney{
   all j: Journey | one u: User | j in u.calendar.journeys
}

fact noAlonePreference{
   all p: Preference | one u: User | p in u.preferences
}

fact noAloneActivity{
   all a: Activity | one u: User | a in u.calendar.activities
}

fact noDifferentWeather{
   no disjoint t1, t2: Weather | t1.location = t2.location and t1.time = t2.time and
      t1.bad_weather = t2.bad_weather
}

fact noTransportWithBadWeather{
   all u: User | all j: Journey, w: Weather | hasJourney[j,u] and
      (((j.start_point = w.location and j.start_time = w.time and w.bad_weather = True) or
      (j.final_point = w.location and j.end_time = w.time and w.bad_weather = True)) implies
         no m: MeanOfTransport | m in j.transports and m.weather_dipendent = True)
}

fact noDoubleCalendarOwner{
   all c: Calendar | no disjoint u1, u2: User | c in u1.calendar and c in u2.calendar
}

fact noDoubleActivityOwner{
   all a: Activity | no disjoint c1, c2: Calendar | a in c1.activities and a in c2.activities
}
```

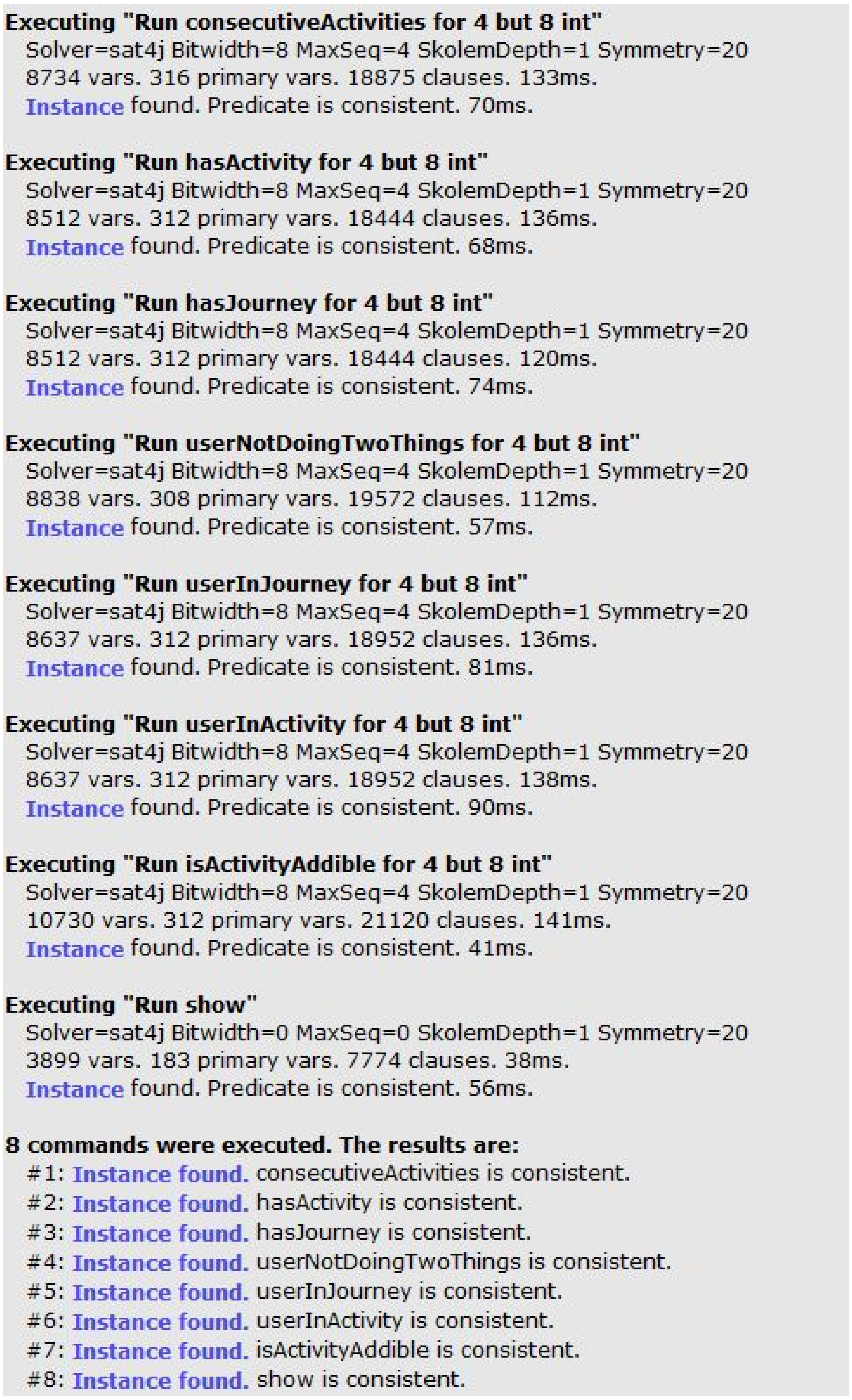
```
fact noOverlappingActivities{
   all u: User |
      all disjoint a1, a2: Activity | hasActivity[a1, u] and hasActivity[a2, u]
         implies (gte[a1.start_time, a2. final_time] or gte[a2.start_time, a1.final_time])
}

fact reachableActivitiesInCalendar{
   all u:User |
      all disjoint a1, a2: Activity | hasActivity[a1, u] and hasActivity[a2, u]
         and consecutiveActivities[a1, a2, u] <=>
         (one j: Journey | hasJourney[j, u] and
            a1.location = j.start_point and a2.location = j.final_point
               and gte[j.start_time, a1. final_time] and
               gte[a2.start_time, j.end_time]    )
}

fact noBannedTransport{
   all u: User |
      all j: Journey | hasJourney[j, u] and
         no m: MeanOfTransport | m in j.transports and
            m in u.preferences.bannedTransports
}

fact onlyGreenTransportsWhenRequired{
   all u: User | u.preferences.greenMode = True =>
      all j: Journey | hasJourney[j, u] and
         all m: MeanOfTransport | m in j.transports and
            m.green = True
}

fact lessActivitiesThanJourneys{
   all u: User | #u.calendar.activities < #u.calendar.journeys
}
```

4.3 Dynamic Model

```
pred consecutiveActivities[a1, a2: Activity, u: User]{
   no a3: Activity | hasActivity[a3, u] and hasActivity[a1, u] and hasActivity[a2, u] and
      ((lte[a1.final_time, a3.start_time] and gte[a2.start_time, a3.final_time]))
}

pred hasActivity[a: Activity, u: User]{
   a in u.calendar.activities
}

pred hasJourney[j: Journey, u: User]{
   j in u.calendar.journeys
}

pred userNotDoingTwoThings[t: Time]{
   all u: User | not (userInJourney[t, u] and userInActivity[t, u])
}

pred userInJourney[t: Time, u: User]{
   one j: Journey | hasJourney[j, u] and
      gte[j.end_time, t] and gte[j.start_time, t]
}

pred userInActivity[t: Time, u: User]{
   one a: Activity | hasActivity[a, u] and
      gte[a.final_time, t] and gte[a.start_time, t]
}

//activity addible in the middle of other 2 activities
pred isActivityAddible[u: User, a: Activity]{
   no disjoint a1, a2: Activity | hasActivity[a1, u] and hasActivity[a2, u] and
      consecutiveActivities[a1, a2, u] and gte[a.start_time, a1.final_time]
         and gte [a2.start_time, a.final_time] and
         (some j1, j2: Journey | j1.start_point = a1.location and j1.final_point = a.location
            and gte[j1.start_time, a1.final_time] and gte[a.start_time, j1.end_time] and
             j2.start_point = a.location and j2.final_point = a2.location
            and gte[j2.start_time, a.start_time] and gte[a2.start_time, j2.end_time])
}

pred show{}


 run consecutiveActivities for 4 but 8 Int

 run hasActivity for 4 but 8 Int

 run hasJourney for 4 but 8 Int

 run  userNotDoingTwoThings for 4 but 8 Int

 run userInJourney for 4 but 8 Int

 run userInActivity for 4 but 8 Int

 run isActivityAddible for 4 but 8 Int

 run show{}
```

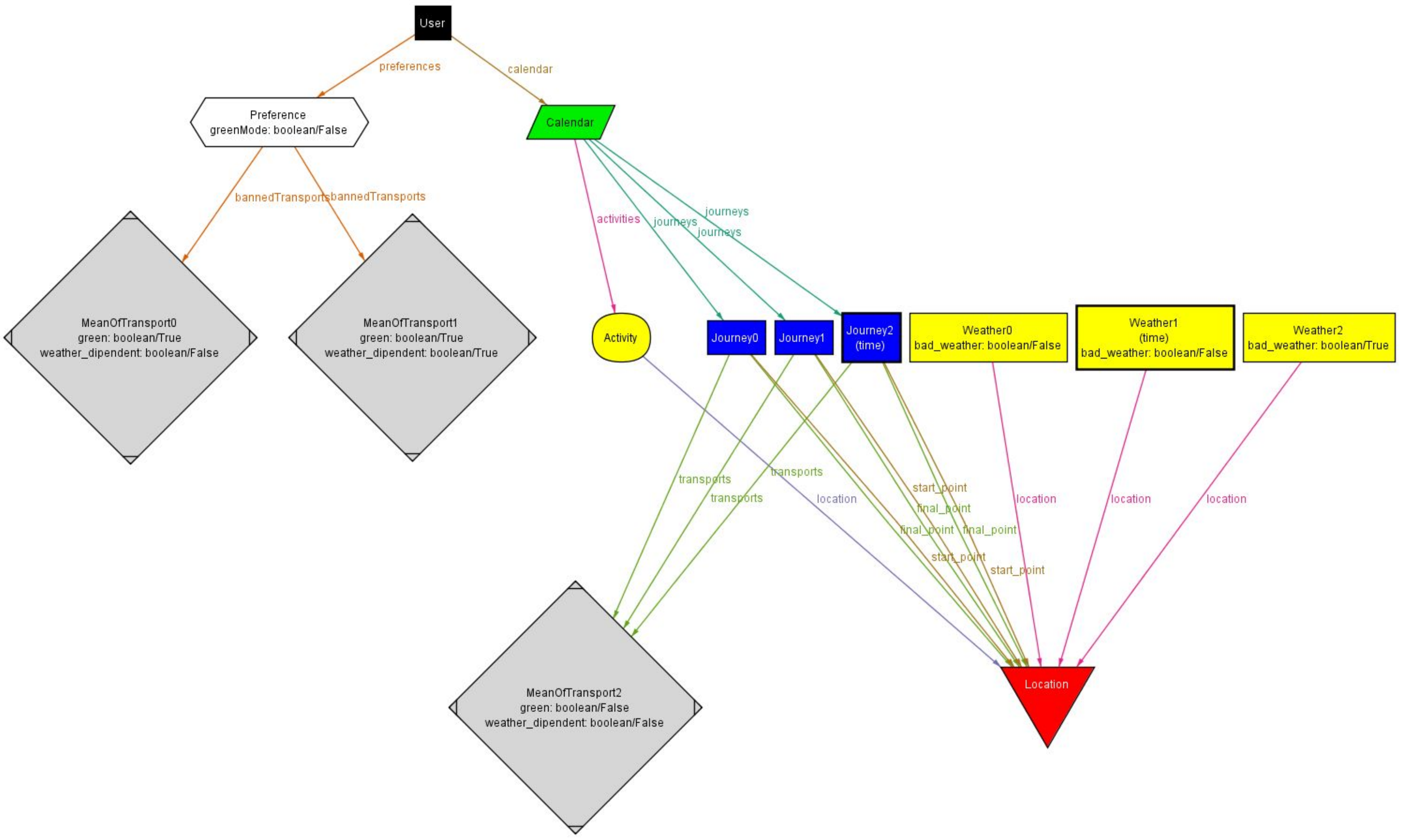
4.4 Results

4.4.1 Proof of Consistency



```
Executing "Run consecutiveActivities for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8734 vars. 316 primary vars. 18875 clauses. 133ms.
   Instance found. Predicate is consistent. 70ms.

Executing "Run hasActivity for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8512 vars. 312 primary vars. 18444 clauses. 136ms.
   Instance found. Predicate is consistent. 68ms.

Executing "Run hasJourney for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8512 vars. 312 primary vars. 18444 clauses. 120ms.
   Instance found. Predicate is consistent. 74ms.

Executing "Run userNotDoingTwoThings for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8838 vars. 308 primary vars. 19572 clauses. 112ms.
   Instance found. Predicate is consistent. 57ms.

Executing "Run userInJourney for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8637 vars. 312 primary vars. 18952 clauses. 136ms.
   Instance found. Predicate is consistent. 81ms.

Executing "Run userInActivity for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   8637 vars. 312 primary vars. 18952 clauses. 138ms.
   Instance found. Predicate is consistent. 90ms.

Executing "Run isActivityAddible for 4 but 8 int"
   Solver=sat4j Bitwidth=8 MaxSeq=4 SkolemDepth=1 Symmetry=20
   10730 vars. 312 primary vars. 21120 clauses. 141ms.
   Instance found. Predicate is consistent. 41ms.

Executing "Run show"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   3899 vars. 183 primary vars. 7774 clauses. 38ms.
   Instance found. Predicate is consistent. 56ms.

8 commands were executed. The results are:
   #1: Instance found. consecutiveActivities is consistent.
   #2: Instance found. hasActivity is consistent.
   #3: Instance found. hasJourney is consistent.
   #4: Instance found. userNotDoingTwoThings is consistent.
   #5: Instance found. userInJourney is consistent.
   #6: Instance found. userInActivity is consistent.
   #7: Instance found. isActivityAddible is consistent.
   #8: Instance found. show is consistent.
```

4.4.2 Generated World



5 EFFORT SPENT

Engineer	**Student ID**	**Effort Time**
Alessio CANTINA	***895395***	***31h 50m***
Simone CRIPPA	***898304***	***29h 55m***
Nicola CUCCHI	***893748***	***33h 50m***

6 REFERENCES

During the drawing up of the document the following tools have been used:

- Draw.io 7.5.6
- Google Docs
- StarUML version 2.8.0
- Balsamiq Mockups 3.5.15
- Alloy Analyzer 4.2