SOFTWARE ENGINEERING 2

TRAVLENDAR RASD
REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

Version 1.0 - 27/10/2017

Plamen Pasliev – 898793

Victor Álvaro Gonzalez - 897700

TABLE OF CONTENTS

1. INTRODUCTION

1.1 OBJECTIVES

This document represents the Requirement Analysis and Specification Document (RASD). The main goals of this document are to analyze the users in order to model a system that meets their needs, describe the system in terms of functional and non-functional requirements, specify the constraints and the limits of the software and define the main typical use cases and user's behaviors. This document is addressed to the developers who have to implement the requirements and could be used as a contractual basis.

1.2 SCOPE

1.2.1 DESCRIPTION OF THE GIVEN PROBLEM

In this project we are going to develop and implement an application called Travlendar. It is a calendar-based application which allows you to create a calendar according to the events you have such as appointments related to work or personal reasons. On top of calculating the time that the user has between meetings so that he does not arrive late, the application will suggest the best mobility option between events and will alert him when it is impossible to reach a specific event on time. Users could define their transportation preferences, they can activate or deactivate any kind of transportation (including walking). The application will also take into account the weather in the location of the user. If it is raining at the time the user has to move to another event, the system will take this into account and will change the way of transport if it is necessary. The application will also allow the user to define breaks to eat or to develop other types of activities. In this way the system will organize the meetings of the user according to their breaks and the time they need to do these activities. Finally, users should also be able, if they wish to, to select combinations of transportation means that minimize carbon footprint.

1.2.2 CURRENT SYSTEM

Even though there are already applications in the market in charge of the planning of meetings, none of these have the functionalities of Travlendar. These are limited to organize the events and notify the participants through the application and that these participants can be put in contact with other users. By this we can say that there is no application in the market with the qualities of Travlendar and therefore it has no competitors until now.

1.2.3 GOALS

- [G1] Allow a User to manage appointments.
- [G2] Allow a User to specify their own travel preferences.
- [G3] Allow a User to introduce the breaks that he requires during the day and the temporal range in which he wants to do the rest.
- [G4] User should receive alerts.
- [G5] The user should receive directions from his current location to the location of the event.
- [G6] The system must consider days in which the public transport or the transport chosen by the user is not available or delayed.
- [G7] The system must consider the weather when an event takes place.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1.3.1 DEFINITIONS

- *Meeting:* personal or work-related appointment. In some parts of the text, meetings are referred as appointments.
- *User*: a user of the Travlendar system.
- *Calendar:* a timetable containing meetings sorted by date.
- *Alert:* A notification or a pop-up screen.

1.3.2 ACRONYMS

- *RASD*: Requirement Analysis and Specification Document.
- *API*: Application Programming Interface.
- *MoSCoW:* is a prioritization technique used in management, business analysis, project management, and software development to reach a common

understanding with stakeholders on the importance they place on the delivery of each requirement.

1.3.3 ABBREVIATIONS

- [Gn]: n-goal.
- [Dn]: n-domain assumption.
- [Rn]: n-functional requirement.
- [Pn]: n-performance requirement.

1.4 REFERENCE DOCUMENTS

1. http://alloy.mit.edu/alloy/documentation.html
2. http://score-contest.org/2018/projects/travlendar.php
3. https://www.cs.umd.edu/~mvz/cmsc630/alloy-manual.pdf
4. https://www.doc.ic.ac.uk/project/examples/2007/271j/suprema_on_alloy/Final%20Report/LaTeX/report.pdf
5. https://beep.metid.polimi.it
6. https://www.websequencediagrams.com
7. https://www.draw.io
8. https://www.gliffy.com/uses/uml-software/
9. https://developer.apple.com/xcode/ *(User Interface)*
10. https://en.wikipedia.org/wiki/MoSCoW_method

1.5 DOCUMENT STRUCTURE

1. In the first part of the document the objectives as well as the main goals of the project are defined. In the same way is explained, without going into much detail, how the application works. Finally, is given some information about definitions and abbreviations to better understand the rest of the document.
2. In the second part it is given an overall description of the system including the functions of the application, clarifying some concepts of the system. Also list the

actors who are going to take part of the system. In the same way the constraints and limitations of the system will be defined. Finally, are specified text and domain assumptions to resolve certain types of doubts that may arise by reading the document.

3. The third part of the document refers to specific requirements. We have defined both functional and non-functional requirements. In this part of the document we will go into more detail in the aspects mentioned in section 2.

4. The fourth part of the document defines the main use cases referring to the goals specified above.

5. In the fifth are exposed the UML diagrams with class and sequence diagrams.

6. Sixth part is embodied with the Alloy model of the system and includes all the relevant details, a proof of consistency and an example of the generated world are also provided. [L SEP]

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

In present times, our lives are full of events. Work, family and friend events fill our daily schedules and it is hard to keep up with everything. Everyone needs to remember where, at what time and with who they need to be at a certain time of the day. To solve this problem the mobile app Travlendar comes in place. It helps the user to manage his events, how to go reach them and how to pay for them.

2.2 PRODUCT FUNCTIONS

The full list of functional and non-functional requirements as well as constraints, assumptions and dependencies will be discussed in the section 3. The main requirements are that the user can add events to his agenda. Additionally, the app must provide optimal suggestions of the different ways the user can reach his destination including travelling time, minimizing carbon emissions, event overlaps, etc. The app should assist the user in paying for tickets, bike and car rent subscriptions.

2.3 USER CHARACTERISTICS

The only user that is affiliated with the application is the smartphone user himself.

User: A person who has downloaded, installed and authorized the app to use the phone's native functions such as geolocation, calendar, etc.

2.4 CONSTRAINTS

The system shall adhere to certain requirements regarding reliability and performance. Those will be discussed in greater detail in section 3.

2.4.1. REGULATORY POLICIES

Travlendar does not use a database and customers do not need to deliver sensitive information. Travlendar simply provides the means customers use to manage their events locally on their device.

2.4.2 HARDWARE CONSTRAINTS

- The app should run on an Android 4.4+ or iOS 9.0+ smartphone.

- Internet connection should be enabled.

- Geolocation should be enabled.

- The app should run in portrait mode.

- The app should run in landscape mode.

2.5 ASSUMPTIONS AND DEPENDENCIES

For Travlendar to function in a correct manner, these assumptions and dependencies must be met:

[D1] The smartphone device has constant internet connectivity.

[D2] The smartphone device has geolocation turned on at all times.

[D3] The phone is synchronized correctly with the current time of the user's region via mobile link or manually.

[D4] If for any reason the user has not attended a meeting, the system will give indications to the next meeting from the current location of the user.

[D5] User correctly enters meeting address when created.

3. SPECIFIC REQUIREMENTS

3.1 EXTERNAL INTERFACE REQUIREMENTS

USER INTERFACES

This part of the document presents the application interface. The application is divided mainly in two views by a tap bar controller. In the main one the user activates or deactivates the means of transport they want. The second view presents the list with the events we have on the selected day. The user can change the day by browsing the calendar at the top of the view. Also in this view there is a button in the top right to add a meeting and another one to edit the events on that selected day. If the user wants to add an event, all they have to do is press the + on the top bar. Once pressed, the screen will change to the default form to fill in to add a new meeting. If they want to add a break they will have to change it. In the same way to add a break they have to fill out the form and press save. If the user wants to edit or delete an event you will have to press the edit button on the top bar. Press once to delete the event by pressing the red circle to the left of the event. If you want to change some field of the event, you will have to press the event and the form will reappear to change the data of the selected event.

Figure 1: Main Travlendar screen

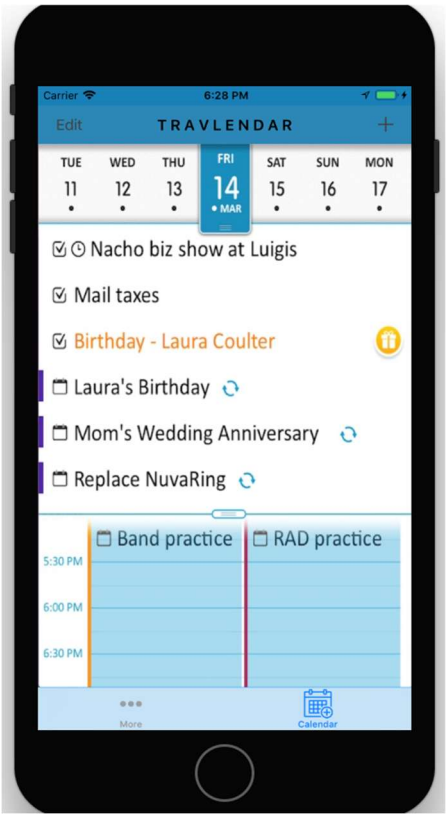


Figure 2: A list of events at a specific date

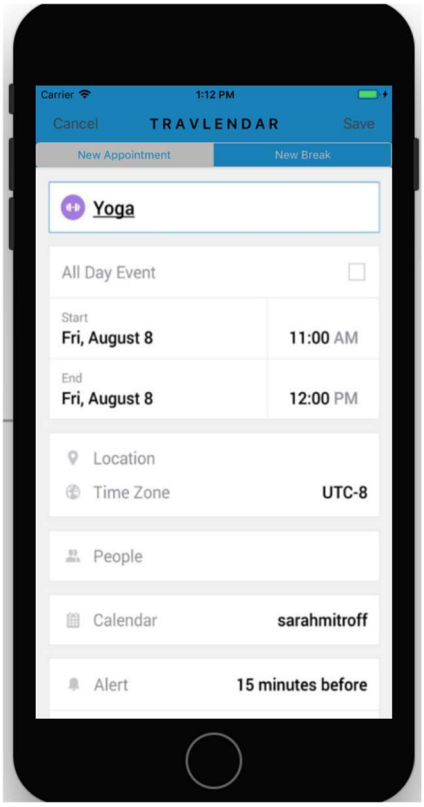


Figure 3: Adding an event

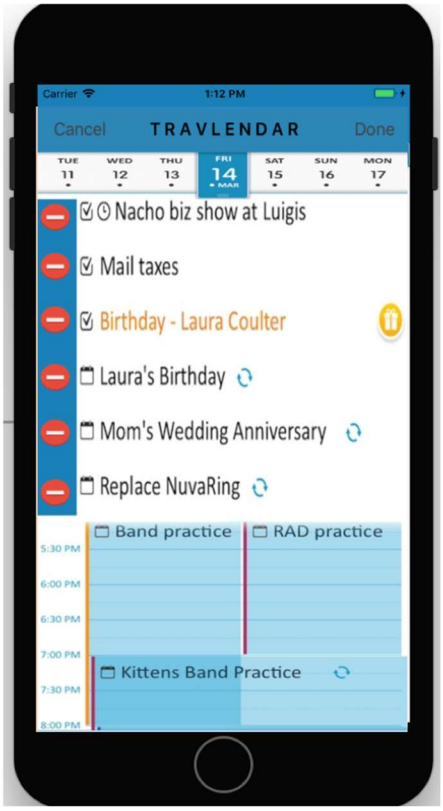


Figure 4: Deleting/Editing events

3.2 FUNCTIONAL REQUIREMENTS

This section specifies the requirements of each goal previously named.

We are using the MoSCoW model for the priority of the requirements. According to this model, requirements can have four diferent priorities, namely: must have, should have, could have and won't have.

- ✓ [G1] Allow a User to manage appointments:

  [R1] The system must allow the user to place an event in his agenda.
  [R2] The system must allow the user to add a title to his meeting.
  [R3] The system must allow the user to add a start and an end location to his meeting.
  [R4] The system must allow the user to add the start and the end time of his meeting.

[R5] The system must detect the current location of the mobile app user.
[R6] The system must calculate the estimated travel time to reach a meeting.
[R7] The system must notify the user if there is an event (including travel time) overlap.
[R9] The system must be able to obtain different routes between the events using different means of transport in such a way that the user has several options to choose from.

- ✓ [G2] Allow a User to specify their own travel preferences:

  [R8] The system must allow the user to activate and deactivate preferred means of transport.

  [R9] The system must be able to obtain different routes between the events using different means of transport in such a way that the user has several options to choose from.

- ✓ [G3] Allow a User to introduce the breaks that he requires during the day and the temporal range in which he wants to do the rest:

  [R10] The system must allow the user to place a break in his agenda.
  [R11] The system must allow the user to add the time interval when the break should take place.
  [R12] The system must allow the user to add the duration of his break.
  [R7] The system must notify the user if there is an event (including travel time) overlap.

- ✓ [G4] Users should receive alerts

  [R7] The system must notify the user if there is an event (including travel time) overlap.
  [R13] The system should be able to check the weather at given location.
  [R14] The system should be able to check the availability of transport at given location.
  [R15] Users must receive an alert if the weather conditions are not pleasant when using means of transport such as walking or cycling.
  [R16] Users must receive an alert if a certain way of transport to their appointment is not available.
  [R17] Users must receive an alert 15 minutes before they have to leave for their next meeting.

- ✓ [G5] The user should receive directions from his current location to the location of the event:

  [R5] The system must detect the current location of the mobile app user.

  [R6] The system must calculate the estimated travel time to reach a meeting.

  [R18] The system must assist the user in his travels and display his current and target location while he is travelling towards an event.

- ✓ [G6] The system must consider days in which the public transport or the transport chosen by the user is not available or delayed:

  [R14] The system should be able to check the availability of transport at given location.

- ✓ [G7] The system must consider the weather when an event takes place

  [R13] The system should be able to check the weather at given location.

3.3 PERFORMANCE REQUIREMENTS

[P1] The content of a page in the mobile app must load within 5 seconds.
[P2] The content of a page in the mobile app should load within 2 seconds.
[P3] The content of a page in the mobile app could load within 1 second.
[P4] An external input to the system must result in visual feedback within 1 second.
[P5] An external input to the system should result in visual feedback within 0.5 seconds.
[P6] An external input to the system could result in visual feedback within 0.2 seconds.

3.4 DESIGN CONSTRAINTS

These are discussed in detail in section 2.4 of this document.

3.5 SOFTWARE SYSTEM ATTRIBUTES

RELIABILITY

The probability of failure of the system should be close to 0 at all times. This will require updates to improve the services of the application and correct possible errors of the early versions.

AVAILABILITY

Availability depends on the reliability of the system. As reliability increases, so does availability. The system should provide its services at all times.

SECURITY

The security does not take a very important value since the data is stored in the application itself, so it is very difficult for information theft to occur.

MAINTAINABILITY

It is one of the most important points that represents the majority of the expenses of the project. There must be continuous maintenance of the application to make it work in the best possible way.

PORTABILITY

Most of the functionality of the app is provided by different APIs (Google maps, Google calendar, etc.). Since the app will be running both on Android and iOS, some of the software or the logic behind it will be reused.

4. USE CASES

1. User creating a new appointment

ACTORS	User
GOALS	[G1]
INPUT CONDITIONS	There are no entry conditions.
EVENTS FLOW	• The user should press the “Calendar” bottom. • The user should press the “+” button. • The user should enter a title for the event. • The user should enter a start time for the event. • The user should enter an end time for the event. • The user should enter his location before the event (start location). • The user should enter the location of the event (end location). • The user should press the “Save” button.
OUTPUT CONDITIONS	The appointment is added to the calendar.
EXCEPTIONS	1. The user inserts a non-existing location. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 6. 2. The user inserts an event that overlaps with another one. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4.

2. User editing an appointment

ACTORS	User
GOALS	[G1]
INPUT CONDITIONS	The user must have created an event.
EVENTS FLOW	• The user should press the “Calendar” bottom. • The user should press the “Edit” bottom. • The user should press on the event he wishes to edit. • The user should edit the event. • The user should press the “Save” button.
OUTPUT CONDITIONS	The appointment is added to the calendar.
EXCEPTIONS	1. The user inserts a non-existing location. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4. 2. The user inserts an event that overlaps with another one. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4.

3. User deleting an appointment

ACTORS	User
GOALS	[G1]
INPUT CONDITIONS	The user must have created an event.
EVENTS FLOW	• The user should press the “Calendar” bottom. • The user should press the “Edit” bottom. • The user should press on the red circle at the left of the event he wishes to delete.
OUTPUT CONDITIONS	The appointment is removed from the calendar.
EXCEPTIONS	

4. User creating a new break

ACTORS	User
GOALS	[G3]
INPUT CONDITIONS	There are no entry conditions.
EVENTS FLOW	• The user should press the “Calendar”” bottom. • The user should press the “+” button. • The user should press the “New break” button • The user should enter a title of the break. • The user should enter the interval time where he/she wants to do the break. • The user should enter the duration of the break. • The user should press the “Save” button.
OUTPUT CONDITIONS	The break is added to the calendar.
EXCEPTIONS	1. The user inserts a break that overlaps with another event. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 5.

5. User editing a break

ACTORS	User
GOALS	[G3]
INPUT CONDITIONS	The user should have created a break.
EVENTS FLOW	• The user should press the “Calendar”” bottom. • The user should press the “Edit” button. • The user should press on a break. • The user should edit the input fields of the break. • The user should press the “Save” button.
OUTPUT CONDITIONS	The break is added to the calendar.
EXCEPTIONS	1. The user inserts a break that overlaps with another event. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4.

6. User deleting a break

ACTORS	User
GOALS	[G3]
INPUT CONDITIONS	The user should have created a break.
EVENTS FLOW	• The user should press the “Calendar”” bottom. • The user should press the “Edit” button. • The user should press on the red circle at the left of the event he wishes to delete.
OUTPUT CONDITIONS	The break is deleted from the calendar.
EXCEPTIONS	

7. User specifies travel preferences

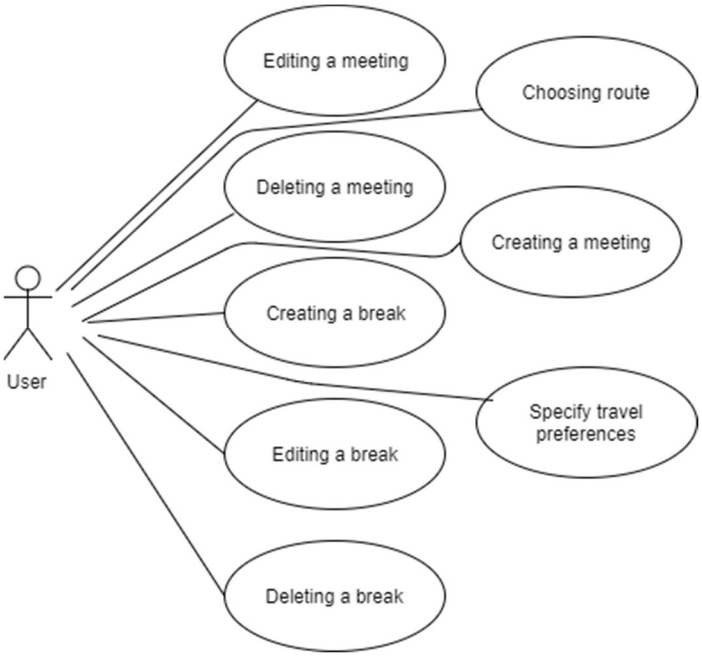
ACTORS	User
GOALS	[G2]
INPUT CONDITIONS	There are no entry conditions.
EVENTS FLOW	• The user should press the “More” button. • The user should choose his preferences by using the radio buttons mapped to the different means of travel.
OUTPUT CONDITIONS	The travel preferences have been chosen.
EXCEPTIONS	1. The user deselects all travel preferences. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 2.

8. User chooses route

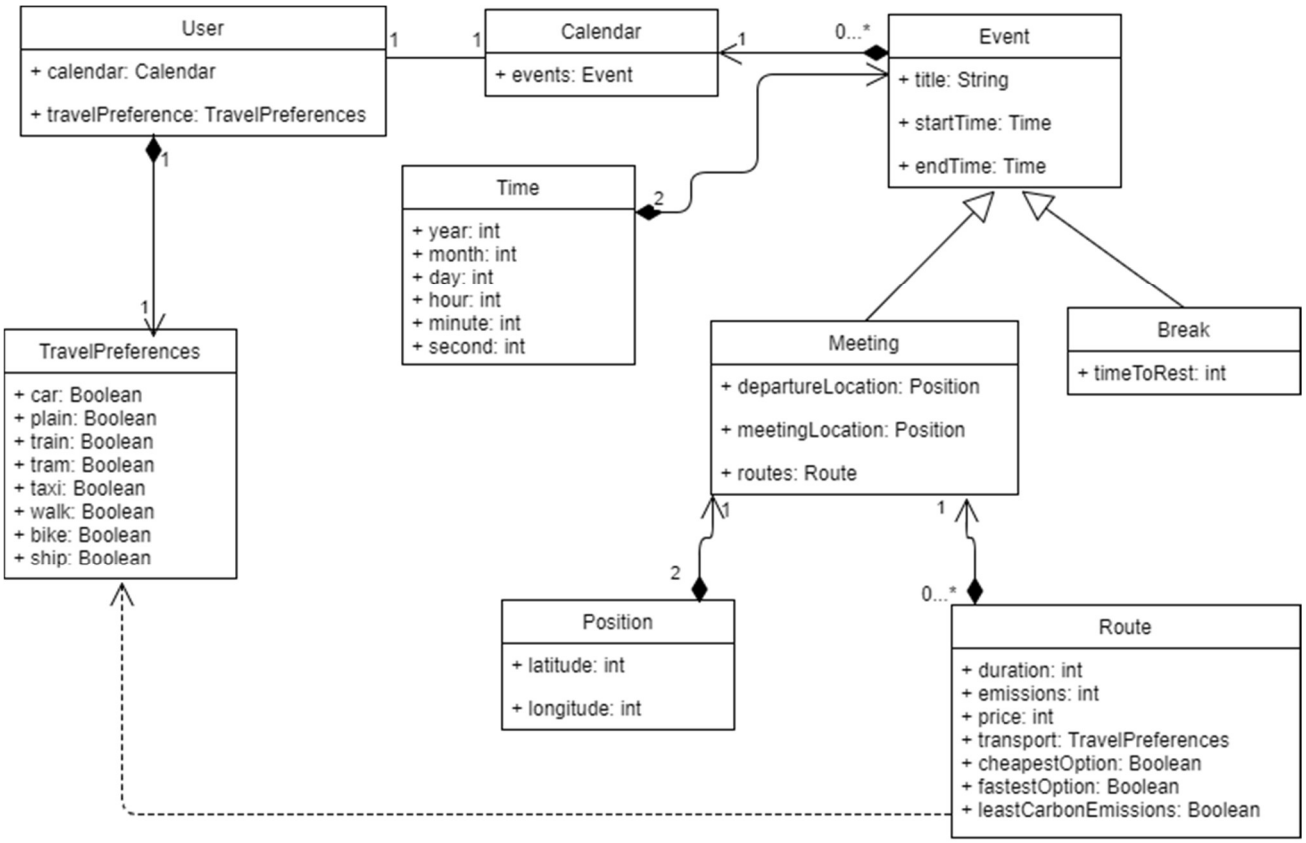
ACTORS	User
GOALS	[G2] [G6] [G7]
INPUT CONDITIONS	The user should have created an appointment.
EVENTS FLOW	• The user should press the “Calendar”” bottom. • The user should press on an appointment. • The user should press the “Travel” button. • The user should choose one of the suggested routes. • The system will display a map with a marker which represents the user while he reaches his destination.
OUTPUT CONDITIONS	The route has been chosen.
EXCEPTIONS	1. There is no route available. This exception is handled notifying the issue to the User and taking back the Event Flow to the point 3.

5. UML DIAGRAMS

Use case diagram:

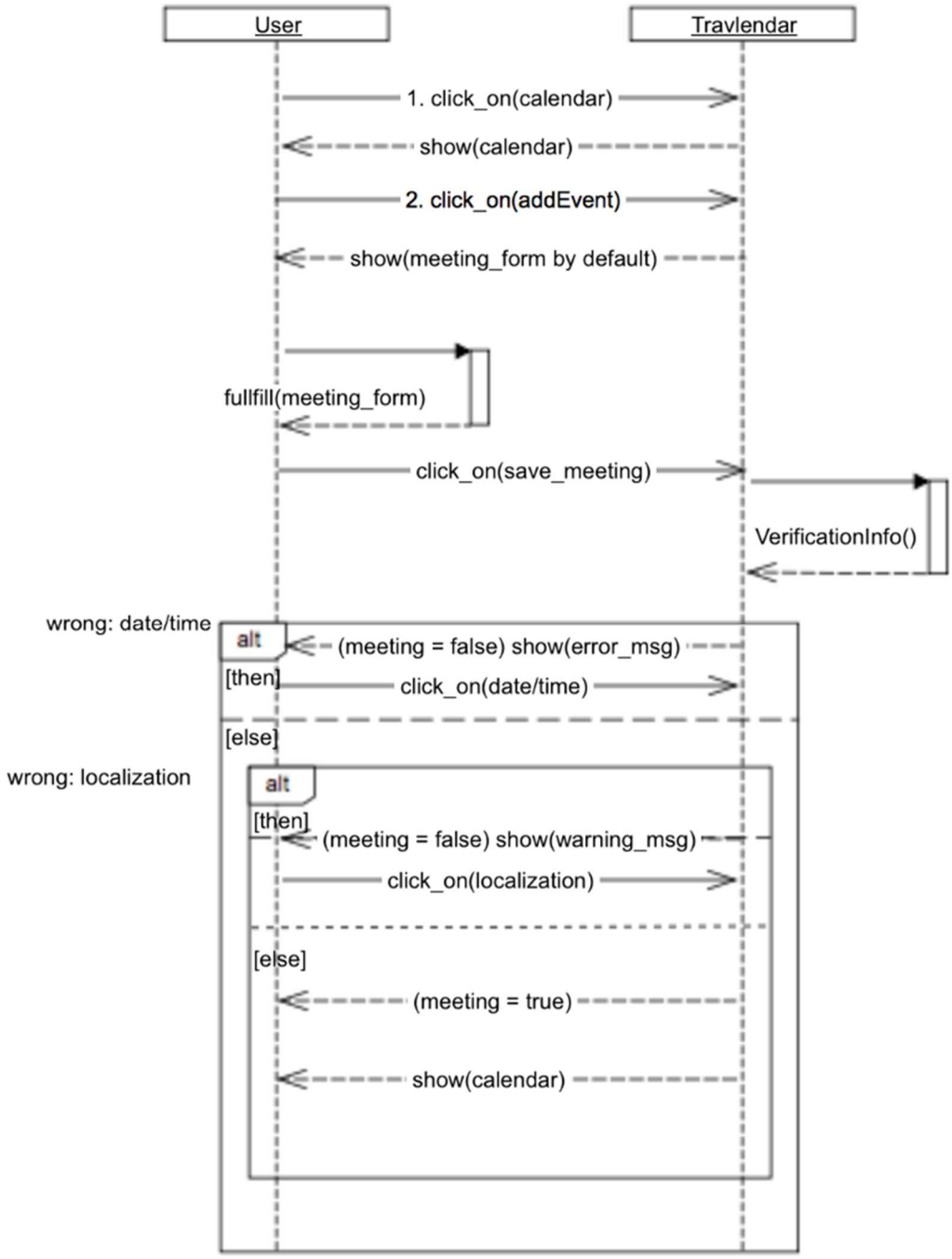


Class diagram:

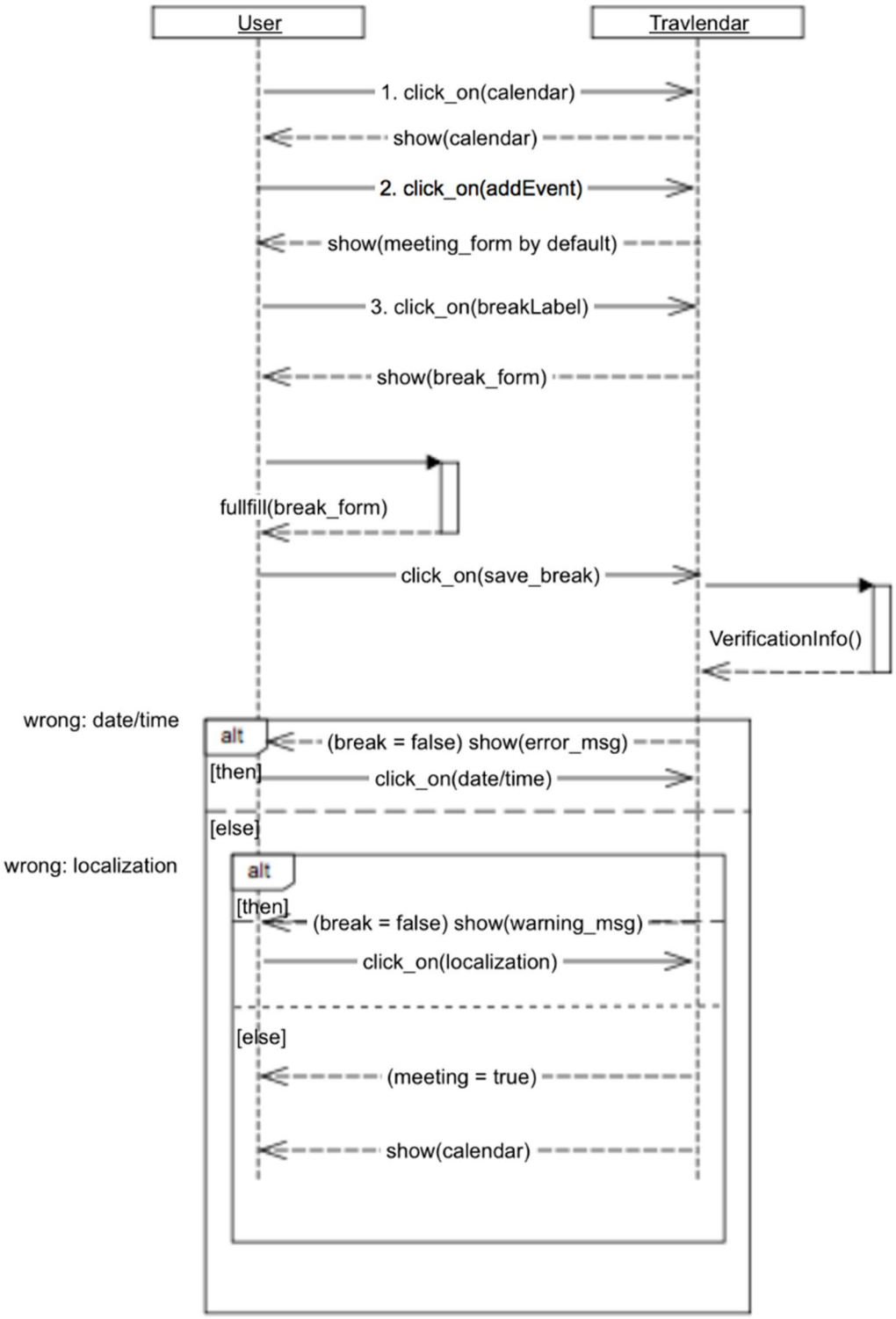


Sequence diagrams:

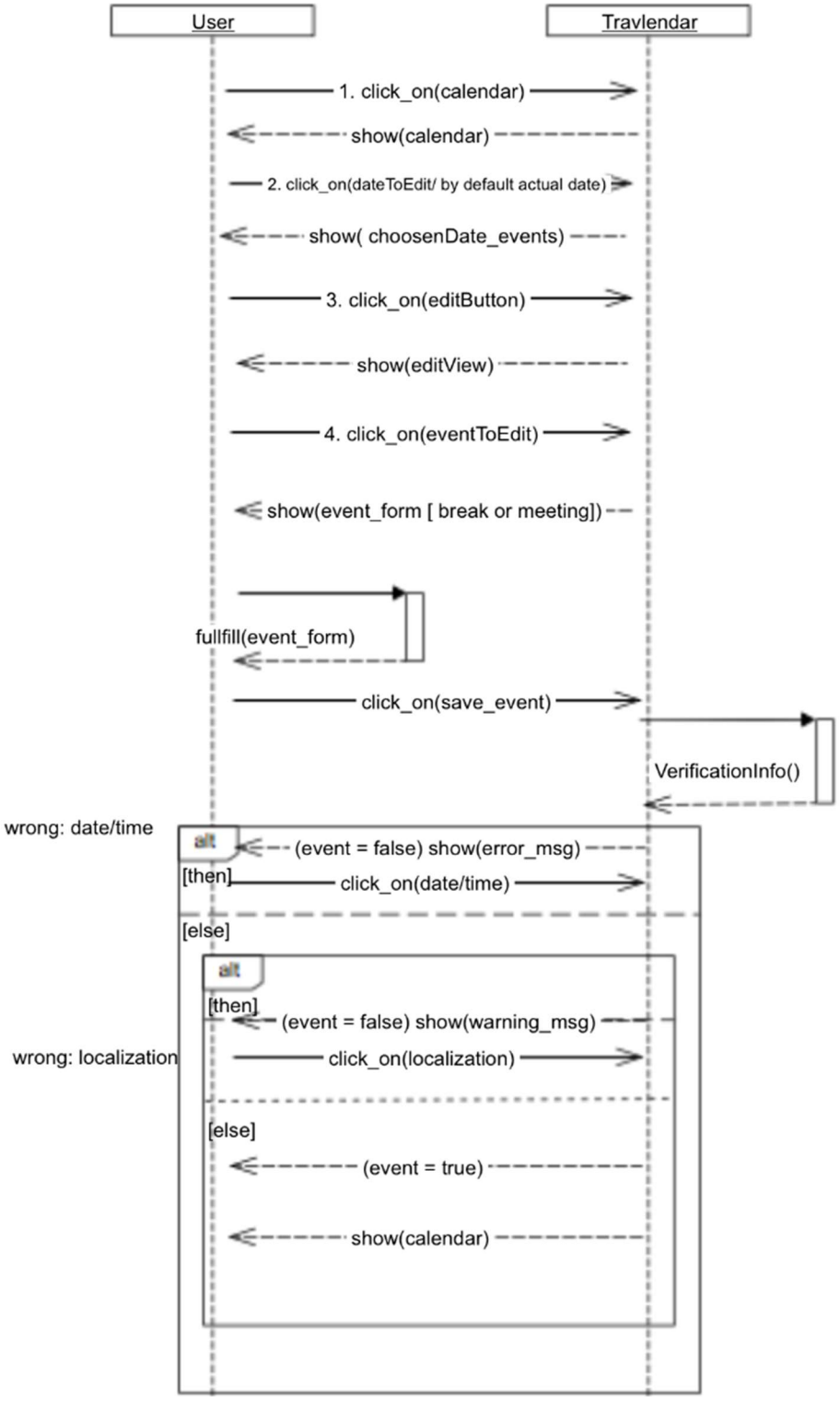
1. *Creating a meeting*



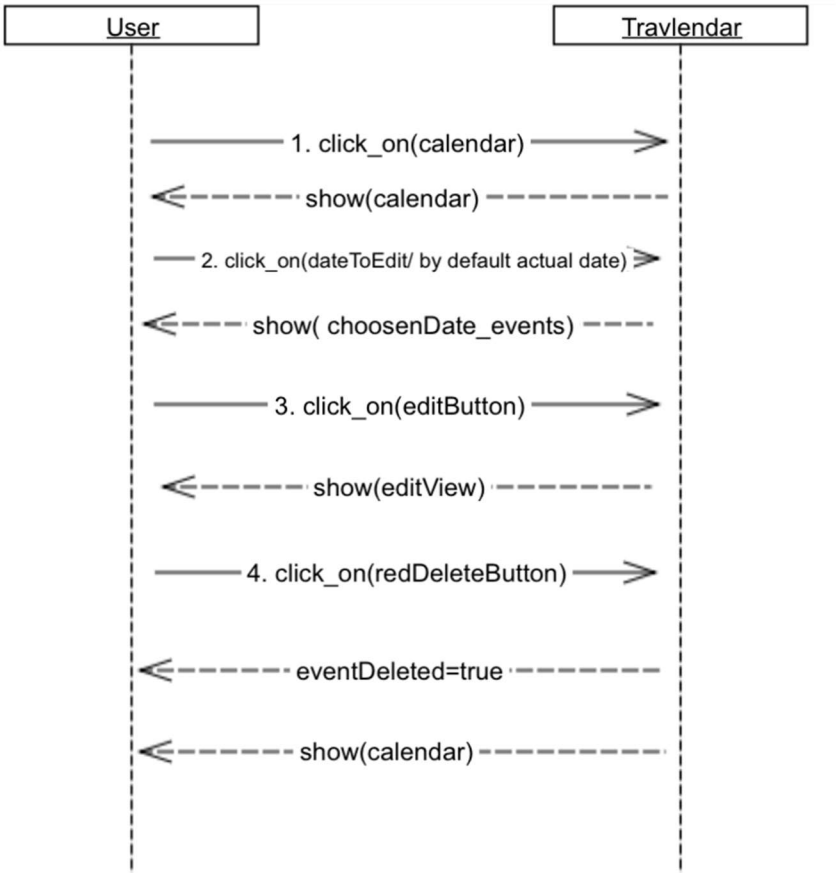
2. Creating a break



3. Editing event



4. Deleting an event



6. FORMAL ANALYSIS USING ALLOY

6.1 PURPOSE

We are using Alloy to model a User creating and deleting events. We can prove that creating an event does not overlap with other events and the data of the created event is complete. We can also show that removing an event successfully removes it from the calendar of the user.

6.2 CODE

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


sig Calendar{
	events:  set Event
}

sig User{
	travellingPreferences: one TravelPreferences,
	calendar: one Calendar
}

sig Route{
	duration: one Int,
	transport: one TravelPreferences,
	cheapestOption: one Bool,
	fastestOption: one Bool,
	leastCarbonEmissions: one Bool,
	emissions: one Int,
	price: one Int
}
{
	duration>0
	price>0
	emissions>0
}
```

```
sig TravelPreferences{
	car: one  Bool,
	plain: one Bool,
	train: one Bool,
	tram: one Bool,
	taxi: one Bool,
	walk: one Bool,
	bike: one Bool,
	ship: one Bool
}

abstract sig Event{
	title: one String,
	startTime: one Time,
	endTime: one Time
}

sig Meeting extends Event{
	departureLocation: one Position,
	meetingLocation: one Position,
	routes: set Route
}

sig Break extends Event{
	timeToRest: one Int
}{
	timeToRest>0
}

sig Position{
	latitude: one Int,
	longitude: one Int
}

--Each route has only one type of transport. It is true that they might have more but it is assumed for simplicity.
fact justOneTransportRoute{
	all r: Route, u: User | r.transport.car.isTrue && u.travellingPreferences.car.isTrue => not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue
	all r: Route, u: User | r.transport.ship.isTrue && u.travellingPreferences.ship.isTrue=> not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.car.isTrue
	all r: Route, u: User | r.transport.train.isTrue && u.travellingPreferences.train.isTrue => not r.transport.bike.isTrue && not r.transport.car.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue
```

```
&& not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not
r.transport.ship.isTrue
	all r: Route, u: User | r.transport.bike.isTrue &&
u.travellingPreferences.bike.isTrue => not r.transport.car.isTrue && not
r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue
&& not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not
r.transport.ship.isTrue
	all r: Route, u: User | r.transport.plain.isTrue &&
u.travellingPreferences.plain.isTrue => not r.transport.bike.isTrue && not
r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.car.isTrue
&& not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not
r.transport.ship.isTrue
	all r: Route, u: User | r.transport.walk.isTrue &&
u.travellingPreferences.walk.isTrue => not r.transport.bike.isTrue && not
r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue
&& not r.transport.taxi.isTrue && not r.transport.car.isTrue && not
r.transport.ship.isTrue
	all r: Route, u: User | r.transport.taxi.isTrue &&
u.travellingPreferences.taxi.isTrue => not r.transport.bike.isTrue && not
r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue
&& not r.transport.car.isTrue && not r.transport.walk.isTrue && not
r.transport.ship.isTrue
	all r: Route, u: User | r.transport.tram.isTrue &&
u.travellingPreferences.tram.isTrue=> not r.transport.bike.isTrue && not
r.transport.train.isTrue && not r.transport.car.isTrue && not r.transport.plain.isTrue
&& not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not
r.transport.ship.isTrue
}

--Just one option of cheapest, fastest and least carbon emissions for each set of routes.
fact oneRouteSelected{
	all r1,r2: Route | r1.duration < r2.duration =>r1.fastestOption.isTrue && not
r2.fastestOption.isTrue
	all r1,r2: Route | r1.emissions < r2.emissions =>r1.leastCarbonEmissions.isTrue
&& not r2.leastCarbonEmissions.isTrue
	all r1,r2: Route | r1.price < r2.price =>r1.cheapestOption.isTrue && not
r2.cheapestOption.isTrue
}

--Each event has his own title.
fact titleIsUnique{
	all disj e1,e2: Event | disj[e1.title,e2.title]
}

--The latitude and longitude must be inside the valid values.
fact validPosition{
	all p:Position | gt[90, p.latitude] && gt[p.latitude, -90] && gt[180, p.longitude]
&& gt[p.longitude, -180]
}
```

```
--At any Time there must be just one event. Also the startTime and the endTime of the event must be different
fact oneEventAtOneTime{
	all disj e1,e2: Event | (gt[e1.startTime,e2.endTime] or gt[e2.startTime,e1.endTime])
	all disj e3,e4: Event | disj [e3.endTime, e4.endTime] and disj[e3.startTime, e4.startTime]
	all disj e: Event | disj[e.startTime, e.endTime]
}

--Each User must have his own calendar
fact unicCalendar{
	all disj u1,u2: User | disj[u1.calendar, u2.calendar]
}

--There must be at leat one travel preference enable because if not, is not possible to show any route to the user
fact atLeastOneTravelPreferenceEnabled{
	all u: User | (u.travellingPreferences.car.isTrue or u.travellingPreferences.bike.isTrue or u.travellingPreferences.car.isTrue or u.travellingPreferences.ship.isTrue or u.travellingPreferences.train.isTrue or u.travellingPreferences.walk.isTrue or u.travellingPreferences.tram.isTrue or u.travellingPreferences.taxi.isTrue or u.travellingPreferences.train.isTrue or u.travellingPreferences.plain.isTrue)
}

-- Checks if the data of the event is not null.
pred isDataCompleted[e: Event]{
	e.title != none
	e.startTime != none
	e.endTime != none
}

--Checks if in the interval of time t1:startTime, t2:endTime of the event there is not any --events in the calendar of the user.
pred freeTimeWindow[t1,t2: Time, c: Calendar]{
	all e: c.events | gt[t1, e.endTime] or gt[e.startTime, t2]
}

--Adding an event to the set of events of the calendar of the user u. After that there is ---created a new user u' with the new event included. Fist of all, checks that the event is --not in the set. After that is added and finally checks that is inside the set after adding ---it.
pred addEvent[u,u': User, e: Event]{
	not e in u.calendar.events
	u'.calendar.events = u.calendar.events + e
	e in u'.calendar.events

}
```

```
--Removing an event to the set of events of the calendar of the user u. After that there is
--created a new user u' without the event e. Fist of all, checks that the event is in the set.
--After that is removed and finally checks that is not inside the set after removing it.
pred removeEvent[u,u': User, e: Event]{
        e in u.calendar.events
        u'.calendar.events = u.calendar.events - e
        not e in u'.calendar.events
}

--Before creating the new event, it is checked if the event has a title and also if there is a
--free window in the interval of time of the event in the calendar of the user.
pred createNewEvent[e: Event, u,u': User]{
        isDataCompleted[e]
        freeTimeWindow[e.startTime, e.endTime, u.calendar]
        addEvent[u,u',e]
}

--After removing the event, it is checked if there is a free window in the calendar of the
--user in the interval time of the event removed.
pred deleteEvent[e: Event, u,u': User]{
        removeEvent[u,u',e]
        freeTimeWindow[e.startTime, e.endTime, u'.calendar]
}

run createNewEvent for 6 but exactly 5 String, 5 Position, 2 User, 8 int
run deleteEvent for 6 but 8 int, exactly 5 String
```

6.3 RESULTS

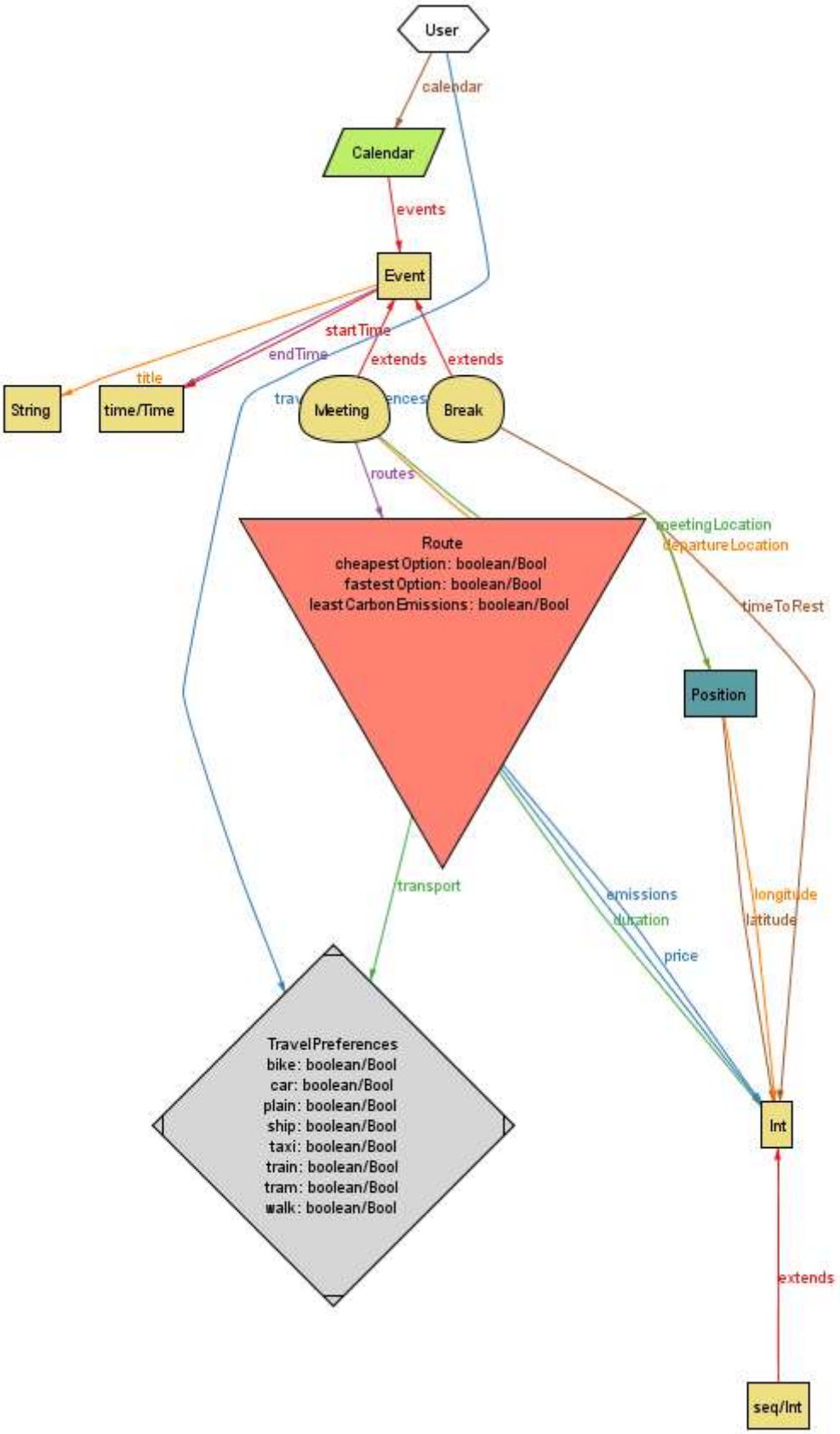
```
run createNewEvent for 6 but exactly 5 String, 5 Position, 2 User, 8 int
run deleteEvent for 6 but 8 int, exactly 5 String
```

6.3.1 PROOF OF CONSISTENCY

```
Executing "Run createNewEvent for 6 but 8 int, exactly 5 String, 5 Position, 2 User"
  Solver=sat4j Bitwidth=8 MaxSeq=6 SkolemDepth=1 Symmetry=20
  197477 vars. 9177 primary vars. 525175 clauses. 491ms.
  Instance found. Predicate is consistent. 815ms.

Executing "Run deleteEvent for 6 but 8 int, exactly 5 String"
  Solver=sat4j Bitwidth=8 MaxSeq=6 SkolemDepth=1 Symmetry=20
  210625 vars. 9762 primary vars. 533028 clauses. 487ms.
  Instance found. Predicate is consistent. 451ms.
```

6.3.2 GENERATED WORLD



7. EFFORT SPENT

Name	Date	Hours spend
Plamen	12/10/17	2
Plamen	15/10/17	5.5
Plamen	16/10/17	4
Plamen	17/10/17	1.5
Plamen	19/10/17	4
Plamen	21/10/17	5
Plamen	22/10/17	2
Plamen	24/10/17	6
Plamen	27/10/17	1
Victor	12/10/17	2
Victor	14/10/17	7
Victor	15/10/17	2
Victor	16/10/17	4
Victor	19/10/17	7
Victor	21/10/17	3
Victor	22/10/17	3.5
Victor	24/10/17	3

Total: **Plamen: 31 hours**

Victor:31.5 hours