POLITECNICO
MILANO 1863

Travlendar+ project

Requirement Analysis and Specification Document

RICCARDO FACCHINI

ANDREA GUGLIELMETTI

October 25, 2017

Deliverable specific information

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Riccardo Facchini - Andrea Guglielmetti
Version:	1.1
Date:	October 25, 2017
Download page:	https://github.com/Riccardo95Facchini/FacchiniGuglielmetti.git
Copyright:	Copyright © 2017, Riccardo Facchini - Andrea Guglielmetti – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This Requirement Analysis and Specification Document (RASD for short) has the purpose of fully describing to a wide range of potential readers the system and to function as a base for legal agreements between developers and other parties.
In the following pages there will be precise descriptions of all the functional and non-functional requirements, the different scenarios and cases of interaction between the system and the users, with attention to what the users need from it, the domain of the system and the constraints implied.
The readers of this document are the developers of the system and its applications, agents from the local public transportation agencies and independent company in similar professions such as taxi businesses or bike/car sharing companies.

1.2 Scope

The scope of this project is to develop a system called Travlendar+ that will allow in the most efficient way possible the paring of daily commutes and the management of scheduled appointments and meetings, by providing for each situation the best alternatives of moving throughout the city both for work related reasons and for personal motives.
Users of Travlendar+ can create a calendar with every appointment paired with time and place, the system will then compute the best way of reaching each location in time by choosing between every commuting option available at the moment and taking into account the preferences expressed by the user in the customization settings, possible strikes of the local transportation services and the weather, if the location is too far and cannot be reached in time a warning is going to pop up on the screen of the user.

1.3 Stakeholders

Here are listed all the potential stakeholders with a brief description and how they may be affected by the system:

- **User**: All the individuals that will use the system to schedule their daily commute.
- **Public transportation companies**: Local and national companies that handle public transportation may have an advantage in giving an easy way to integrate their schedules with Travlendar+ as it could mean a higher number of sold tickets.
- **Taxi and Car/Bike sharing companies**: Given that is not always possible for each type of user to walk long distances and public transport does not reach every possible destination they may be interested in a partnership between their service and the system.
- **Mobile network carriers**: They have an interest in providing a network and contracts to connect devices to the service.

1.4 Definitions, acronyms and abbreviations

What follows is the list of all the main terms and abbreviations used in the document.

1.4.1 Definitions

- **User**: who is using the system to schedule their calendar.
- **Trip**: the plan to move from point A to point B done using one or more means.
- **Travel**: synonymous of trip.
- **System**: All the software needed to deliver all the functionalities desired, often used as a synonymous of Travlendar+.

1.4.2 Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **SRS**: Software Requirement Specification. Synonymous of RASD
- **ETA**: Estimated Time of Arrival
- **GPS**: Global Positioning System
- **W3C**: World Wide Web Consortium
- **HTTP**: HyperText Transfer Protocol
- **HTTPS**: HyperText Transfer Protocol over Secure Socket Layer
- **SDK**: Software Development Kit
- **TCP**: Transfer Control Protocol
- **API**: Application Programming Interface
- **RAM**: Random Access Memory
- **UMTS**: Universal Mobile Telecommunications System
- **DB**: Database
- **DBMS**: Database Management System

1.4.3 Abbreviations

No other abbreviations aside from acronyms were used.

1.4.4 Revision History

- 25/10/2017 Small changes and fixes to grammar and pages structure.

1.5 Reference documents

- Document of the assignment: Mandatory Project Assignments.pdf

2 Overall Description

2.1 Product Perspective

The system is going to be divided in three main components:

1. Mobile application version for phones and tablets.
2. Web browser version.
3. Backend structure to support the functioning of the service.

While the backend structure is needed for the functioning of the service provided, only one of the two applications is required to interact with the system.
APIs for each component must be provided in order to allow future development and introduction of new functionalities like an automated system for buying tickets of public transportation vehicles.

Class Diagram

In Figure 1 is represented the class diagram of the basic components of the system to be, a more detailed one will be discussed later on.

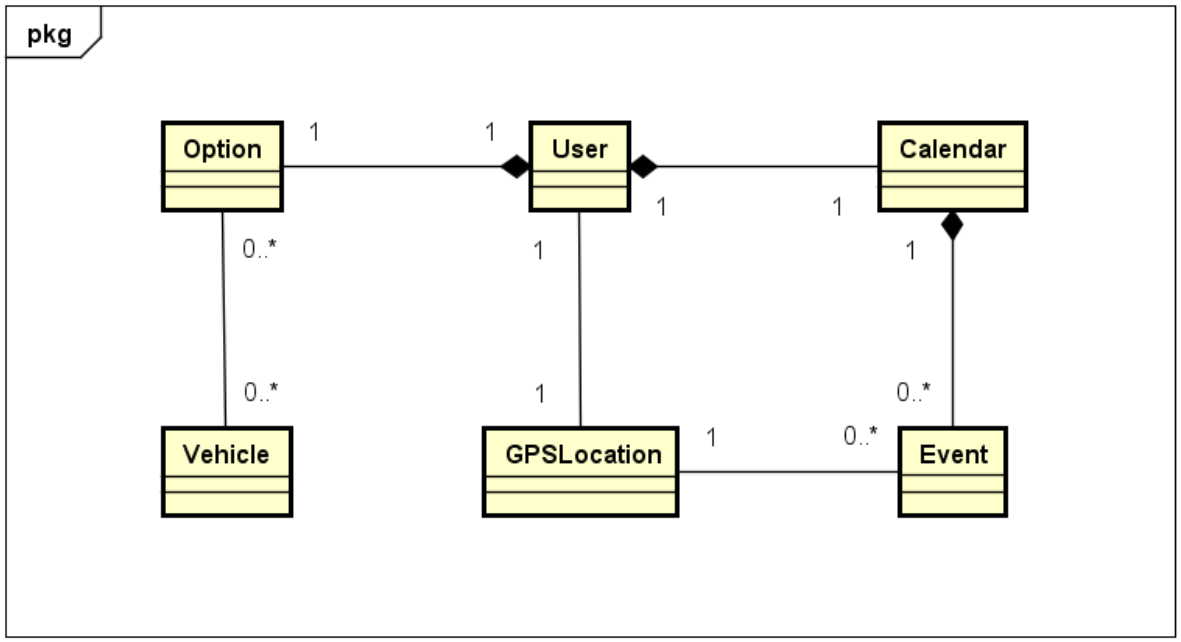

Figure 1: Basic Class Diagram

Difference between Machine and World

The following is a distinction between the **Machine** and the **World**.

1. Machine: the Machine is the *Product-to-be* (often also referred to as *System-to-be* or *System* for short).
   The *Machine Domain* on the other hand is everything that the Machine can operate onto, meaning that it can manipulate it or more in general it has control over.

2. World: the World is the physical world that interacts with the Machine or that it can be observed by it, it's the environment in which the System will gather information and will be affected by the actions performed.

Machine and World are connected by *Shared Phenomena*, the set of events of the World that are observable by the Machine and the ones the Machine can directly cause with its actions.
An example of a Shared Phenomena may be something as mundane as the rain, since it's an event that happens in the World and is observable by the Machine via a forecast or a weather report.

2.2 Product Functions

The system allows each user to create their personal calendar by specifying place and time of each appointment and then view the proposed solution, to be more specific the user can:

1. Register to the system with username and password.
2. Logging to the service.
3. Manage the information of an account and delete it.
4. Specify means of travel preferences.
5. Create an appointment in the calendar.
6. Change appointment information.
7. Delete an appointment.
8. Schedule flexible breaks (like lunch) of specific length in a given interval.

2.3 User Characteristics

The users interested in using the system should be at least familiar with the concept of navigating a web page and using a smartphone in the day to day routine without needing any technical competence regarding the topic, they must be aware of the laws regarding the public circulation on the streets of the country they wish to use the services provided and need a valid driver licence if they want to use a car, they also must possess an electronic payment method to use third party car/bike sharing options.

2.4 Domain Assumptions

We assume that the following statements are true:

1. There is always at least one possible means of transportation.
2. The GPS location are always within 1m of the correct position.
3. Weather forecasts have a 100% accuracy.
4. Public transportations always arrive on time.
5. Internet connection is never lost.
6. Is possible to integrate the system with already existing application from third party companies and public transportations.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The main way the users can interact with the system is via the mobile application for their smartphone, the interface should be user-friendly and in particular easy to read, with large and high contrast text to minimize reading problems in direct sunlight; the second way of connecting to the services provided is to use the web application their personal computer. In both cases the user interfaces must satisfy the following constrains:

- The first page must always ask the user to login or register to service.
- After the login, the system redirects the user to his/her home page.
- (Web) A toolbar must be present in every page, except login and registration page.
- (Mobile) A toolbar must be present in the homepage.
- The *Create a meeting* page must provide a guided process to set-up a meeting and clearly show if the created meeting is not reachable in time from the location of a previous appointment.
- The *Manage meetings* page must show a list of the user's meetings and allow the user to select a meeting to obtain further information.
- The interface must offer the possibility to choose between a set of different languages.
- The user interface must dynamically adapt to the screen size.
- The Mobile and Web application must use the same graphical elements.

In addition to these constraints, other platform-dependent constraints are provided:

- Web Application:

  All the pages must submit to W3C standards.

- Mobile Application:

  All mobile versions must follow the design guidelines provided by the respective platform manufacturer (Android, iOS, Windows . . .).

3.1.2 Hardware Interfaces

For the use of the web application is needed a personal computer connected to the internet and equipped with a web browser capable of rendering, while the mobile application must be able to connect to the network in order to exchange information with the server, such as destination and location of the user retrieved via the GPS of the mobile device.
Hardware requirements for both applications are later specified in subsubsection 3.4.2

3.1.3 Software Interfaces

Supported browsers for the web application should include Google Chrome, Mozilla Firefox, Opera, Safari, Internet Explorer and Edge, while the mobile application must be available on Android, iOS, Windows Phone and Blackberry OS.
The server side of the application requires:

- Java EE, to write the server application that perform the travel computation and the database access.
- MySQL, to memorize user information and meetings inside a relational database.

The client side of the application requires the latest version of the platform SDK.

3.1.4 Communication Interfaces

The client communicates to the server via HTTPS protocol using TCP.
In addition, the system must be able to use the API of other application in order to retrieve weather forecasts and news about strikes.

3.2 Functional Requirements

3.2.1 Registration

Purpose
The main purpose of the *Registration* use case is to provide the user a service which permits the subscription to the system. The user must fill a registration form with his/her personal information and accept the Terms and Conditions of use. After that a confirmation e-mail is sent to the specified e-mail.

Functional Requirements

R.1: The system must not accept an already registered e-mail.

R.2: The user must provide the following information:

- name
- surname
- e-mail
- password

R.3: The system cannot allow the user to proceed in the registration process if he/she does not accept the Terms and Conditions of use.

R.4: The system must send an e-mail to the user after he/she submits the form.

R.5: The system must generate a unique link for the registration e-mail.

R.6: The user must be able to exit the form any time.

Scenario
Alice is searching an app to track down her appointments. Searching on the Play Store app of her phone, she finds out an app called Travlendar+. She downloads it, compiles the registration form, clicks on the link inside the confirmation app and the she is ready to use Travlendar+.

Use Case
The *Registration* use case is analysed in Table 1

Activity Diagram
The activity diagram of the *Registration* use case in showed in Figure 2

Sequence Diagram
The sequence diagram of the *Registration* use case in showed in Figure 3

Name	**Registration**
Actors	Non registered User
Entry conditions	_
Flow of events	1. The user asks to the system to register to its service. 2. The system shows the appropriate form to fill. 3. The user fills the form inserting its own information. 4. The user submits the form. 5. The system checks if the e-mail is unique. 6. The system sends to the specified e-mail address a confirmation e-mail with a unique link. 7. The user must open the e-mail and click on the confirmation link. 8. The system receives the confirmation, saves the data inside a database and notifies to the user.
Exit conditions	The user is now registered and he/she can login and start to use the service.
Exceptions	Exceptions can occur when requirements R.1:, R.2:, R.3: are violated, in this case the system reloads the registration form and goes back to step 2. Registration process is also aborted when the user decides to exit from it.

Table 1: *Registration* use case description

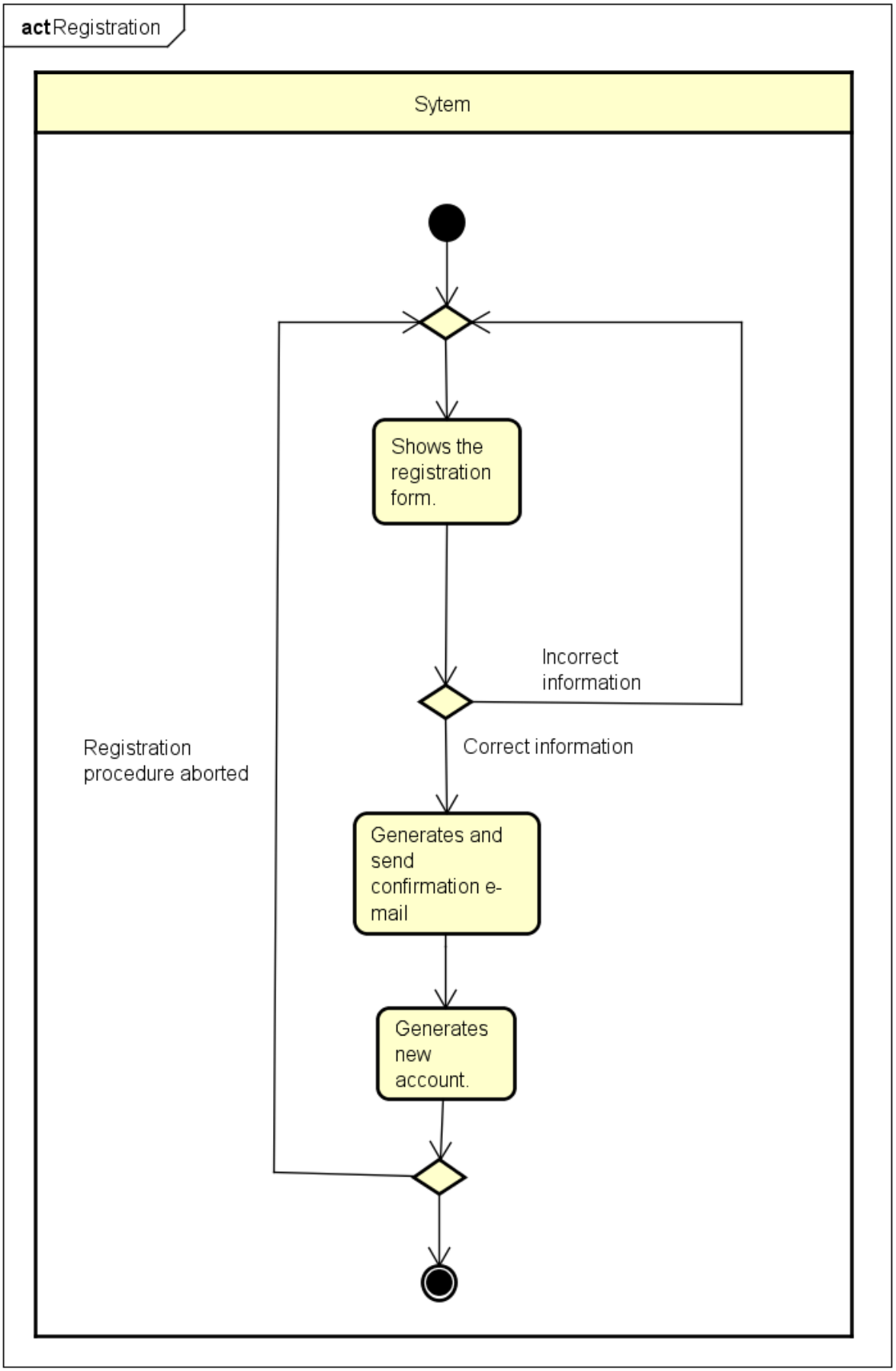

Figure 2: *Registration* activity diagram

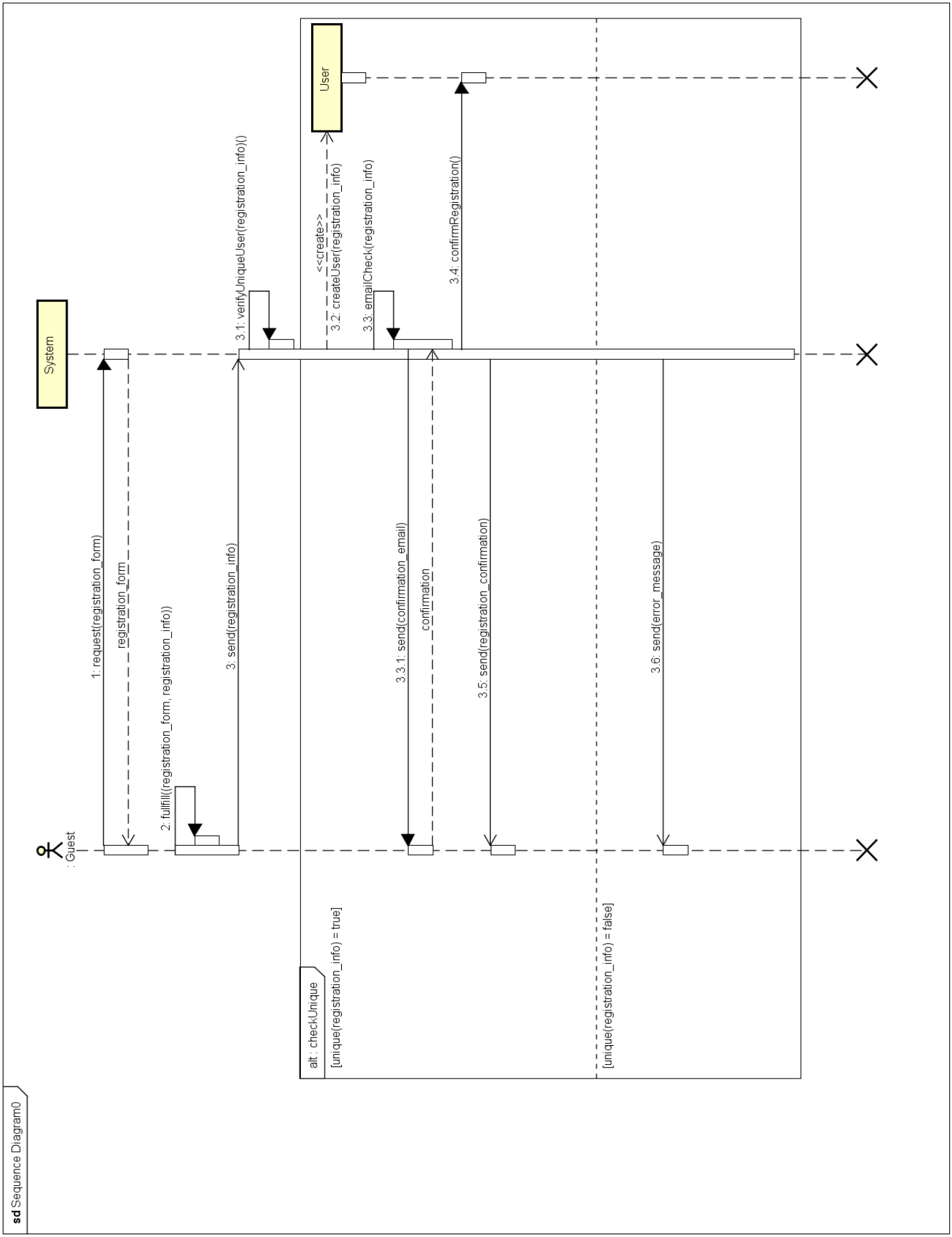

Figure 3: *Registration* sequence diagram

3.2.2 Login

Purpose
The purpose of *Login* functionality is grant access to *Travlendar+* to any registered user. The system requires an already inserted e-mail and corresponding password in order to access the service.

Functional Requirements

R.7: The user must be already registered to the system to perform a successful login.

R.8: The system must require the user to insert his/her e-mail and password to perform a login.

R.9: The system must check if the password corresponds to a registered e-mail.

R.10: The system grants access to its service only to authenticated users.

R.11: The user can insert a new password if and only if he/she is a registered user and clicks on "*Forgot my password*".

R.12: The password must correspond to the most recent one associated to the user's e-mail.

Scenario
Bob wants to login on Travlendar+. He opens the Travlendar+'s web site, then clicks on Login and eventually inserts his e-mail and password, but the system does not accept his data and prevents the login. Bob checks the e-mail inserted and finds a typo, he corrects it and submits again the data. Now all the information required are correct and the system accept the login.

Use Case
The *Login* use case is analysed in Table 2.

Activity Diagram
The activity diagram of the *Login* use case in showed in Figure 4.

Sequence Diagram
The sequence diagram of the *Login* use case in showed in Figure 5.

Name	**Login**
Actors	Non registered User
Entry conditions	The user is already registered into the system and wants to login.
Flow of events	1. The user asks to the system to perform a login. 2. The system shows the *Login* page. 3. The user inserts his/her e-mail and password and submits. 4. The system checks if the e-mail corresponds to a registered e-mail. 5. The system checks if the password is associated to the entered e-mail.
Exit conditions	The user is now logged in.
Exceptions	Exceptions occurs when the e-mail has never been registered to the system or when the password does not corresponds to the e-mail.

Table 2: *Login* use case description

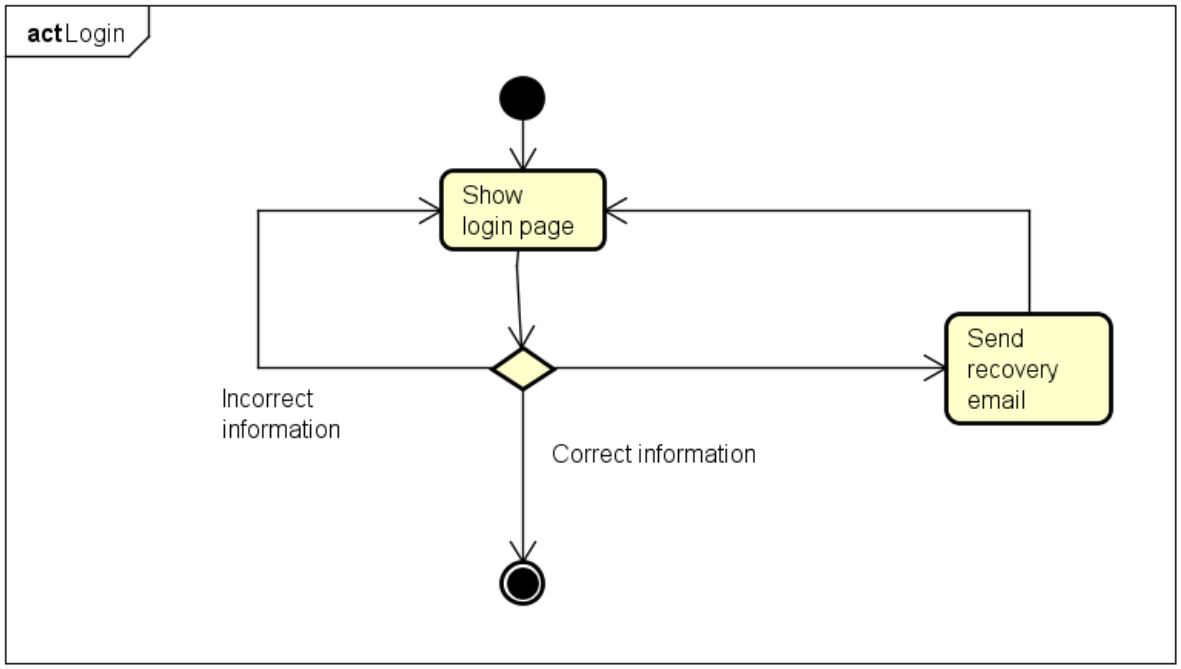

Figure 4: *Login* activity diagram

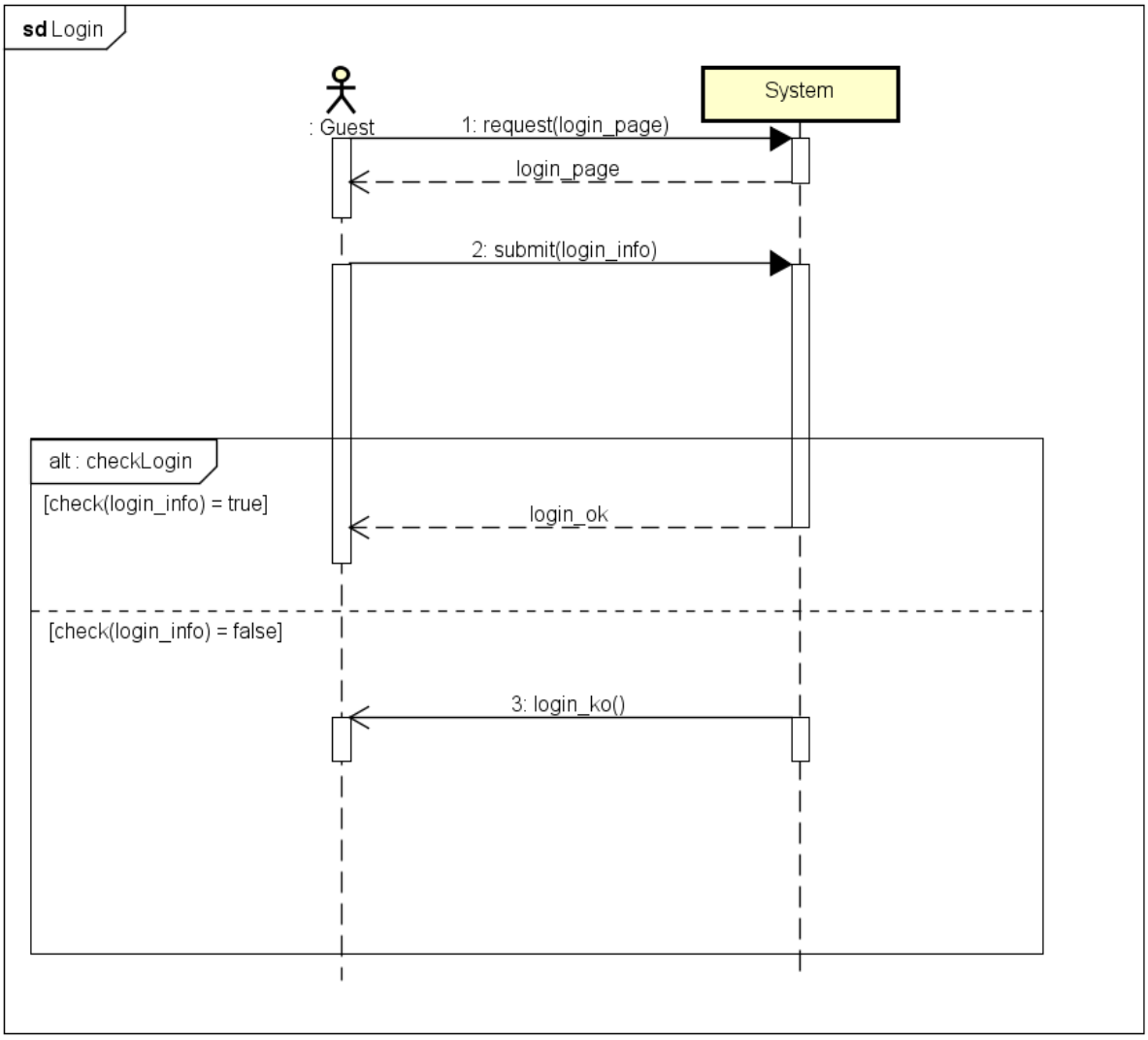

Figure 5: *Login* sequence diagram

3.2.3 Specify Means of Travel

Purpose
The purpose of this functionality is to allow the user to select an order for his/her favourite means of travel and what vehicles he/she owns, the system will then take this data in consideration once it has to calculate a trip plan by giving precedence to the vehicles higher in the hierarchy. Both preferred and owned vehicles are optional information, if the user doesn't insert them the system will simply not be influenced when computing travels.

Functional Requirements

R.13: The user must be logged in.

R.14: A vehicle already selected in an option must not appear again in the list when the user is selecting one.

R.15: The user must be able to change the inserted data at any time.

R.16: The system must take the preferences into account each time it computes a travel.

Scenario
Bill loves to move using his bicycle, and wishes to use it each time he can, so he decides to place it as his favourite vehicle by opening the Travlendar + app and after logging in he selects the "Options" icon, then he proceeds to open the "Select favourite vehicles" section and presses the "+" sign to add one, he then chooses "Bicycle" from the list he is presented with.
Bill then remembers to also add the Bicycle to his owned vehicles, so the app won't suggest him to use a bike sharing company, to do that he selects the "Owned vehicles" option and just like before he presses the "+" sign and then picks "Bicycle" from the list to add it.

Use Case
The *Specify Means of Travel* use case is analysed in Table 3 and in Figure 6

Activity Diagram
The activity diagram of the *Specify Means of Travel* use case in showed in Figure 7

Sequence Diagram
The sequence diagram of the *Specify Means of Travel* use case in showed in Figure 8

Name	Specify Means of Travel
Actors	User
Entry conditions	The user must be logged in
Flow of events	1. The user opens the app's options. 2. The user selects either the option to order the vehicles preferences or the one to add an owned one. 3. The user selects the "+" sign to add a vehicle. 4. The system provides a list of vehicles not already selected to the user. 5. The user chooses a vehicle from the list. 6. The system stores the choice in the database.
Exit conditions	The user has selected one or more vehicles in his/her preference and/or in the owned section.
Exceptions	If the use already selected all possible vehicles the system won't allow another insertion.

Table 3: *Specify Means of Travel* use case description

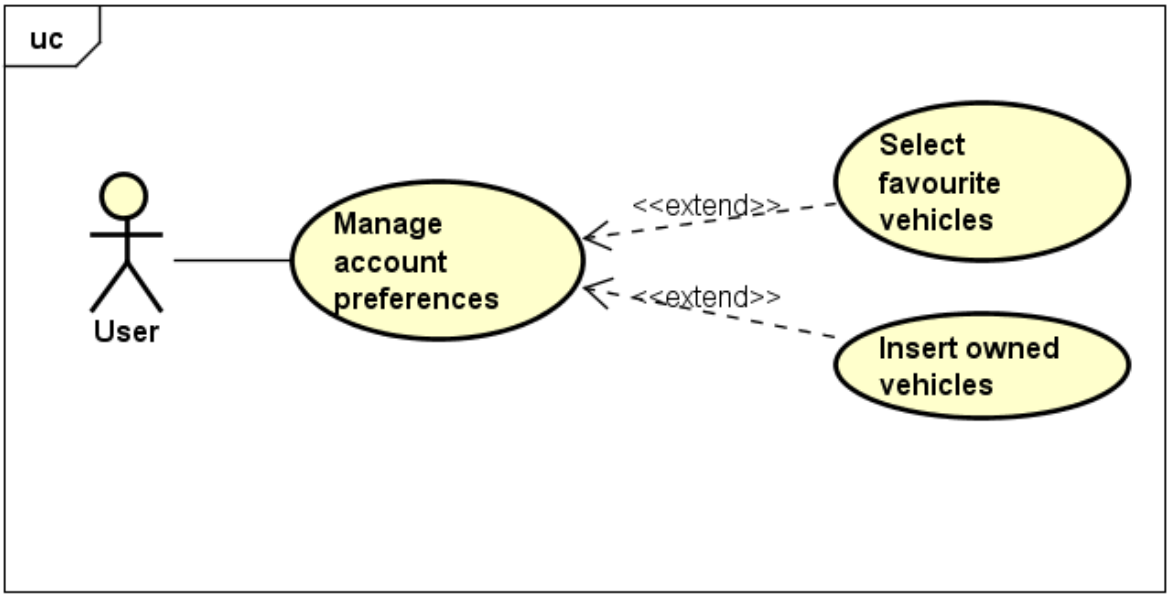

Figure 6: *Specify Means of Travel* use case

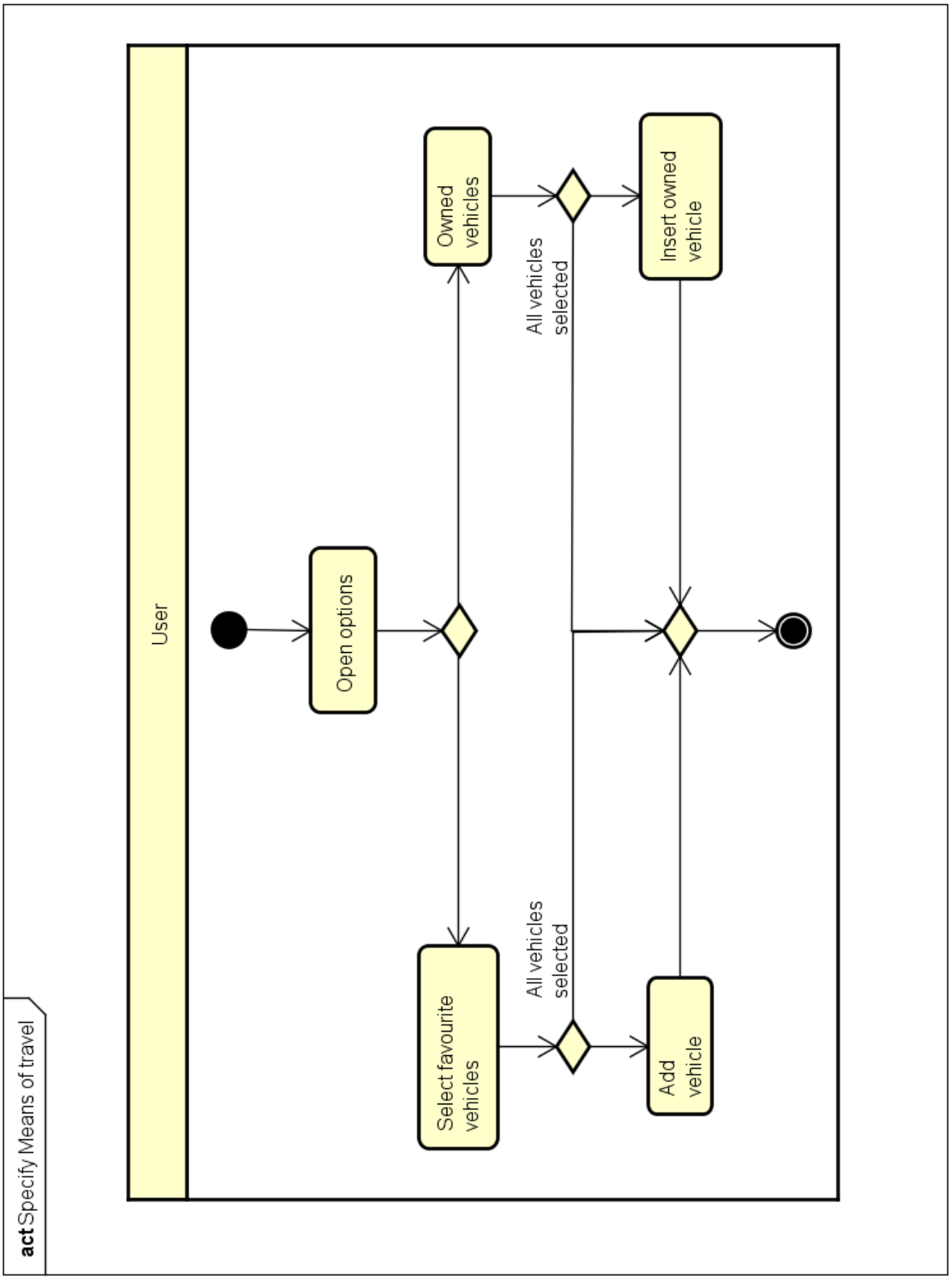

Figure 7: *Specify Means of Travel* activity diagram

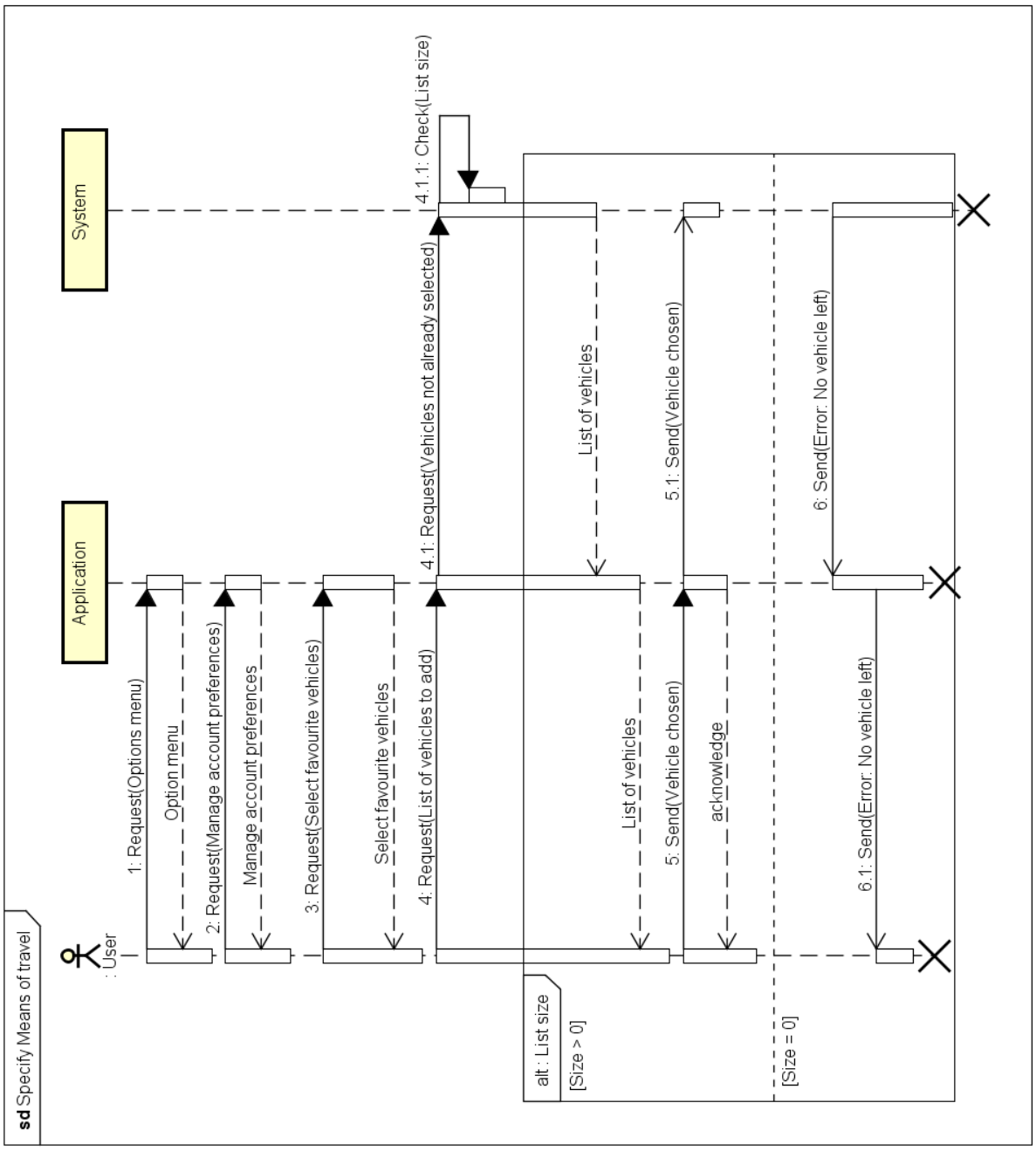

Figure 8: *Specify Means of Travel* sequence diagram

3.2.4 Create Appointment

Purpose
This function allows the user to create and register an appointment in the calendar. The user inserts the appointment's name, the time and location in which occurs and the time and location from where the system must calculate the path. The system then calculates the best path based on the user's preferences, the weather of appointment day and possible strikes. If the appointment place is not reachable in time the system warns the user.

Functional Requirements

R.17: The user must be logged in.

R.18: The user must be able to abort the operation any time.

R.19: The user must insert:

- Appointment name.
- Appointment time and location.
- Starting time and location.

R.20: The starting and destination locations must exists.

R.21: The appointment time must be after the starting time.

R.22: Both the appointment time and the starting time must not be in the past.

R.23: If the appointment is not reachable in time a warning must alert the user.

R.24: If the appointment duration overlaps with a previous created appointment, a warning must alert the user.

R.25: If between the starting time and the appointment time there are one or more appointments, a warning must alert the user.

R.26: The user must be able to adjust appointment information after a warning.

R.27: The user must be able to ignore the warning if and only if the appointment does not overlaps with others previous created appointments.

R.28: The system must calculate the best path every time the appointment information changes.

Scenario
Jessica has taken an appointment with her dentist on 27th October at 8:00 am. She opens her *Travlendar+* app, selects "*Create an appointment*" and inserts the name of the event, the time and the address of the dentist's studio. She also inserts from where and when the app must calculate the path, she sets the starting time at 7:30 am and, since she is at home, the starting location via GPS.

Use Case
The *Create Appointment* use case is analysed in Table 4 and in Figure 9.

Activity Diagram
The *Create Appointment* activity diagram is showed in Figure 10.

Sequence Diagram
The *Create Appointment* sequence diagram is showed in Figure 11.

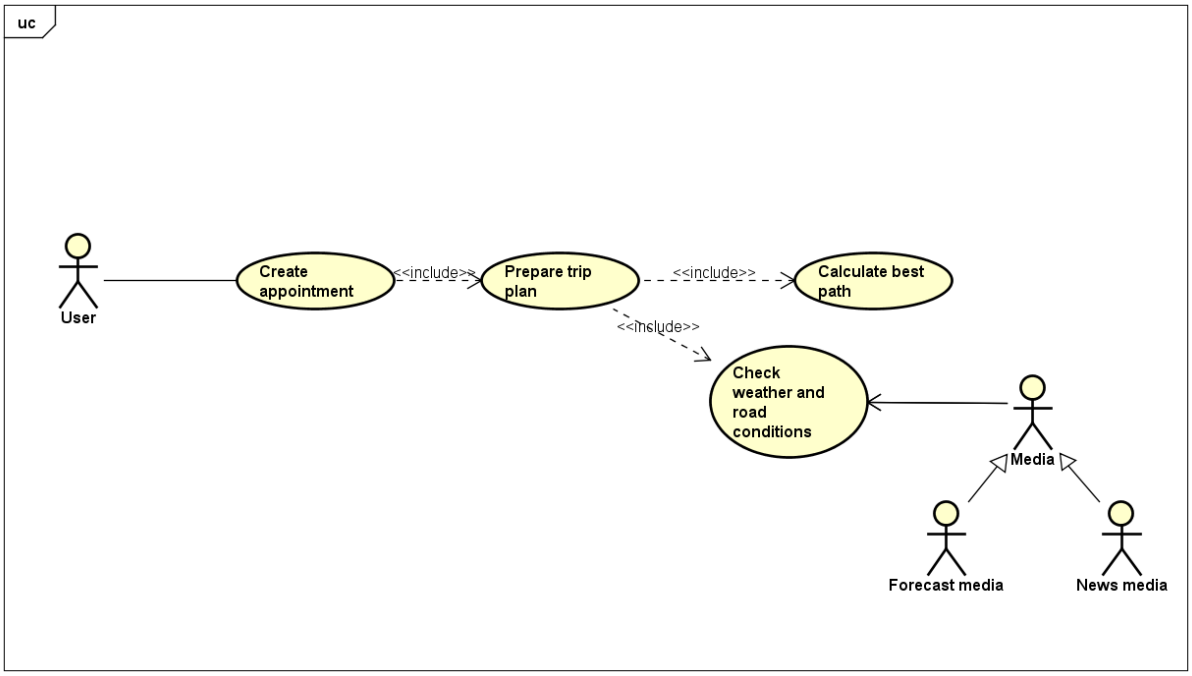

Figure 9: *Create Appointment* use case

Name	**Create appointment**
Actors	User
Entry conditions	The user must be logged in
Flow of events	1. The user select "*Create an appointment*. 2. The user fills the form inserting the data required by the requirement R.19:. 3. The system calculates the best path.
Exit conditions	The system saves the appointment informations.
Exceptions	Exceptions can occur when information about time and location of the appointment does not follow the requirements, in this case a warning is generated. If the warning is about the non-reachability of the appointment in time the system asks to the user if he/she wants to modify appointment information or to ignore the warning. Instead, if the created appointment overlaps with another appointment the system warns the user asking to modify the appointment information. If the created appointment does not overlap with other appointments, but is not reachable in time the user can choose to ignore the warning and let the system to create the appointment. If the user chooses to abort the operation the appointment will not be saved.

Table 4: *Create Appointment* use case description

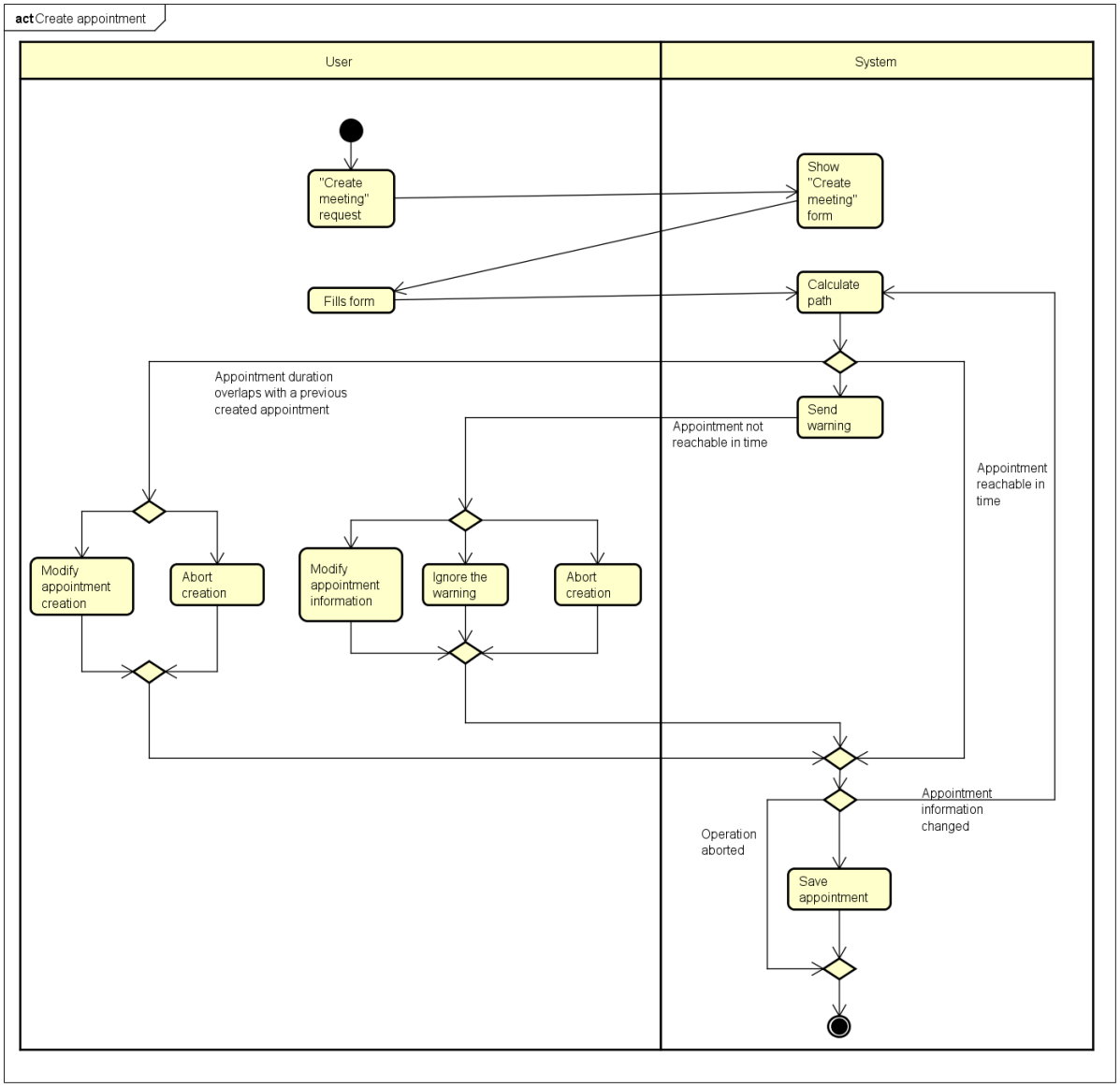

Figure 10: *Create Appointment* activity diagram

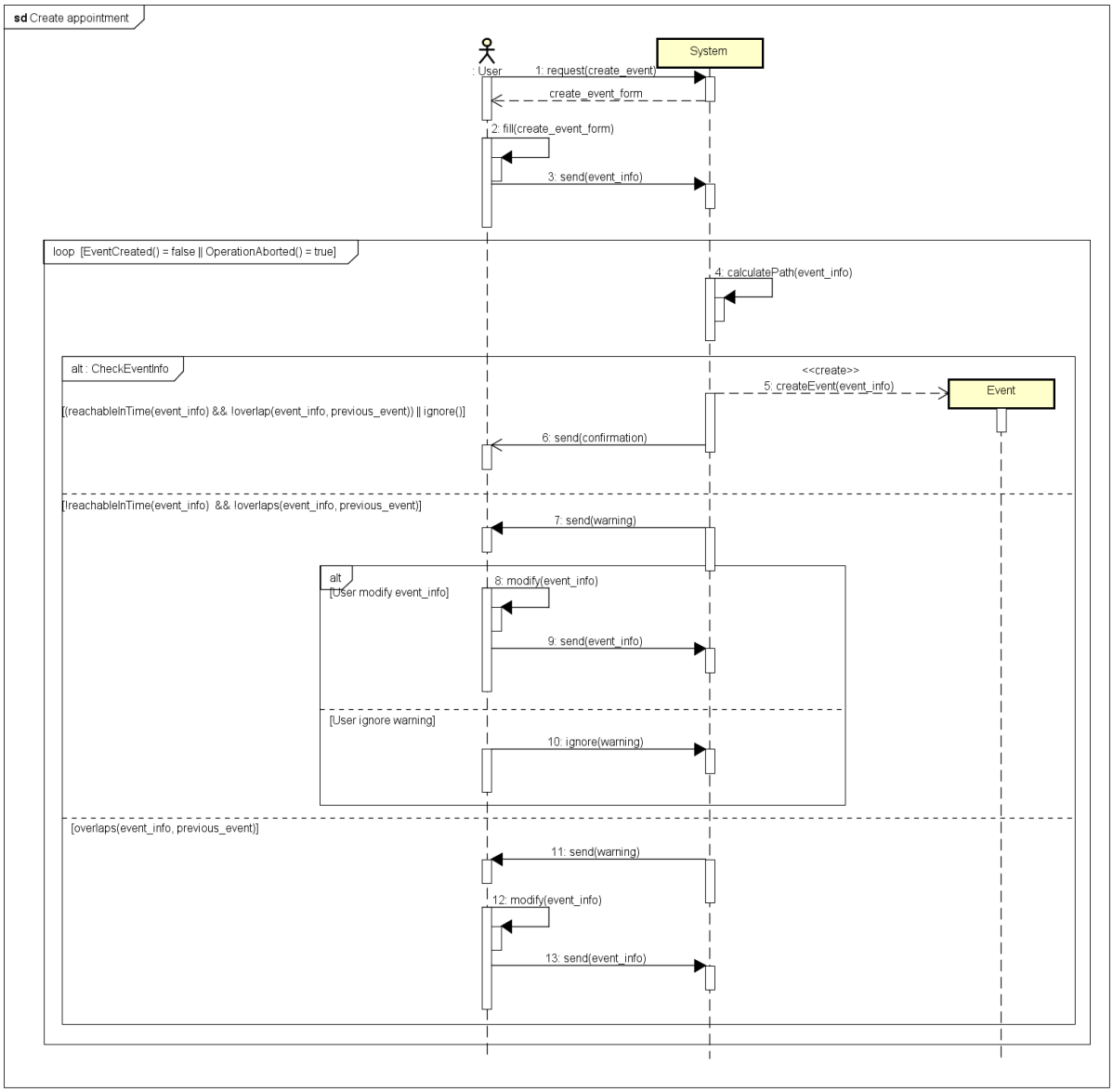

Figure 11: *Create Appointment* sequence diagram

3.2.5 Change Appointment Information

Purpose
This function allows the user to select an appointment already registered in the calendar and then change parameters as he/she wishes, the system will then compute again the trip and store the changes made in the database.

Functional Requirements

R.29: The user must be logged in.

R.30: The user must have at least one upcoming event saved.

R.31: The user must be able to change information as many times as he/she wishes.

R.32: Past events cannot be changed.

Scenario
Anna has taken an appointment with her doctor for next week at 3:00 pm and she already recorded it using Travlendar+, but she remembers that she has to bring her son to football practice before 3:15 pm, she decides then to call her doctor and re-schedules the appointment for 2:00 pm, once the call is finished she opens Travlendar+ and proceeds to open "Manage meetings", and then after selecting the appointment selects "Change meeting details", where she can change the time of the appointment and then saves it.

Use Case
The *Change Appointment Information* use case is analysed in Table 5 and in Figure 12

Activity Diagram
The activity diagram of the *Change Appointment Information* use case in showed in Figure 13

Sequence Diagram
The sequence diagram of the *Change Appointment Information* use case in showed in Figure 14

Name	**Change Appointment Information**
Actors	User
Entry conditions	The user must be logged in and must have at least one upcoming event.
Flow of events	1. The user opens the app. 2. The user selects the "Manage meetings" section. 3. The user selects the meeting he/she wants to change. 4. The user selects "Change meeting details". 5. The user changes the meeting as he/she wishes by providing at least one of the following: (a) Date of the meeting. (b) Time of the meeting. (c) Location of the meeting. (d) Name of the meeting. 6. The user saves the changes. 7. The system updates the meeting in the database and computes again the trip.
Exit conditions	The user changed a meeting.
Exceptions	If the meeting has expired and the user tries to change it, the application will avoid it.

Table 5: *Change Appointment Information* use case description

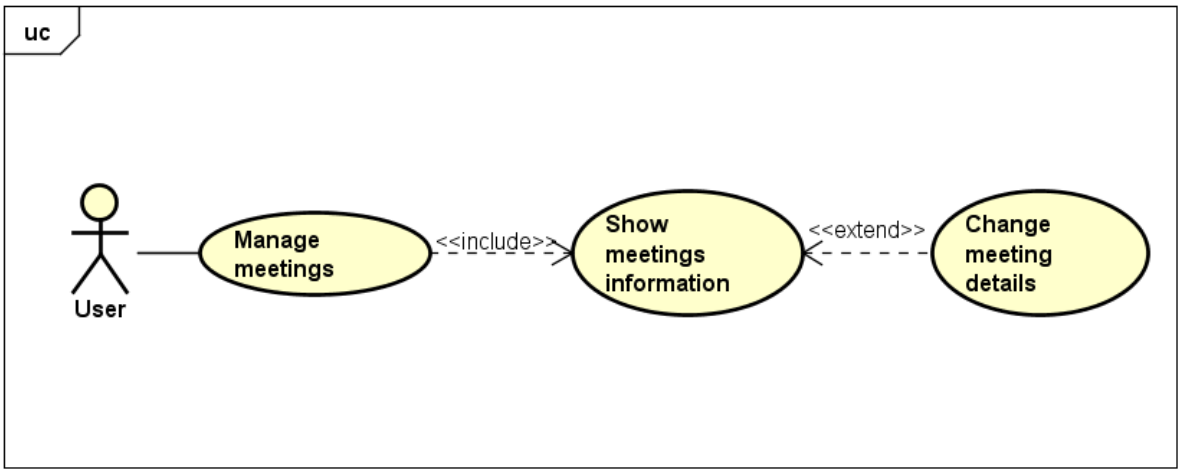

Figure 12: *Change Appointment Information* use case

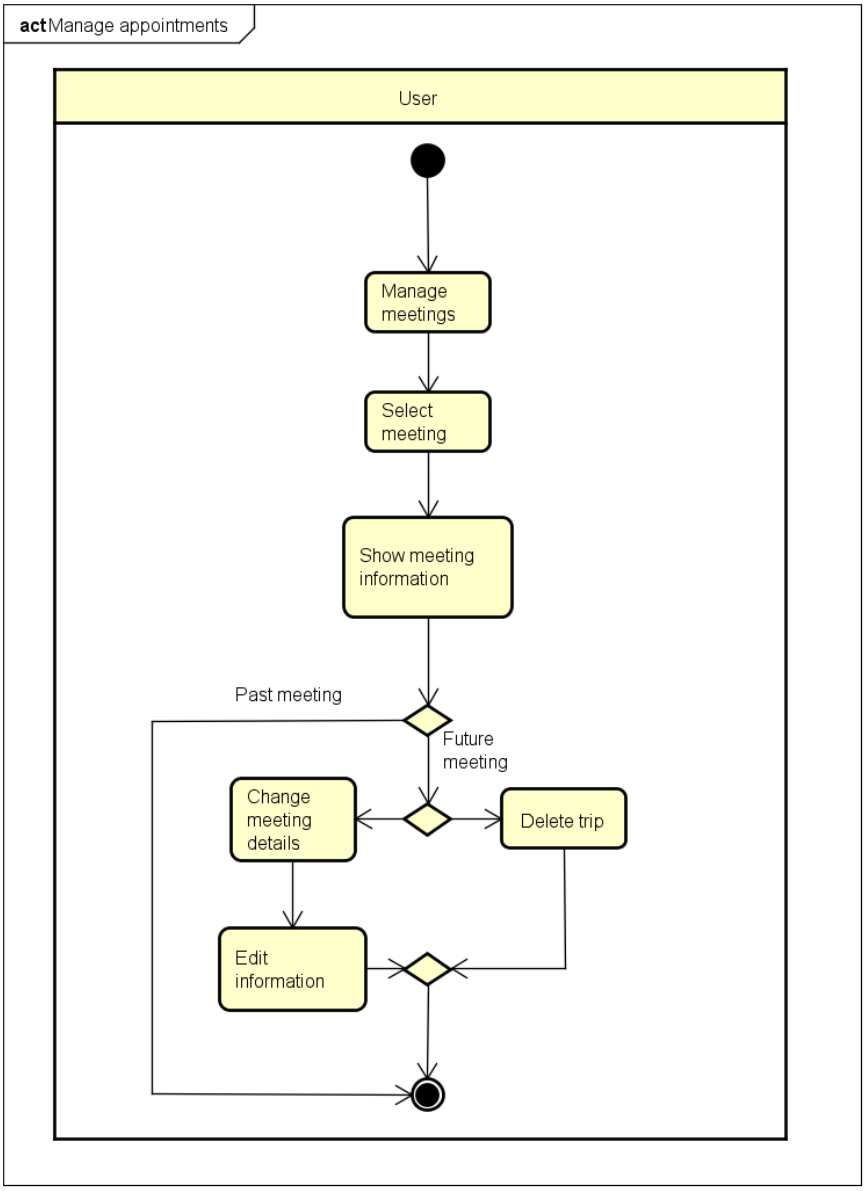

Figure 13: *Change Appointment Information & Delete Appointment* activity diagram

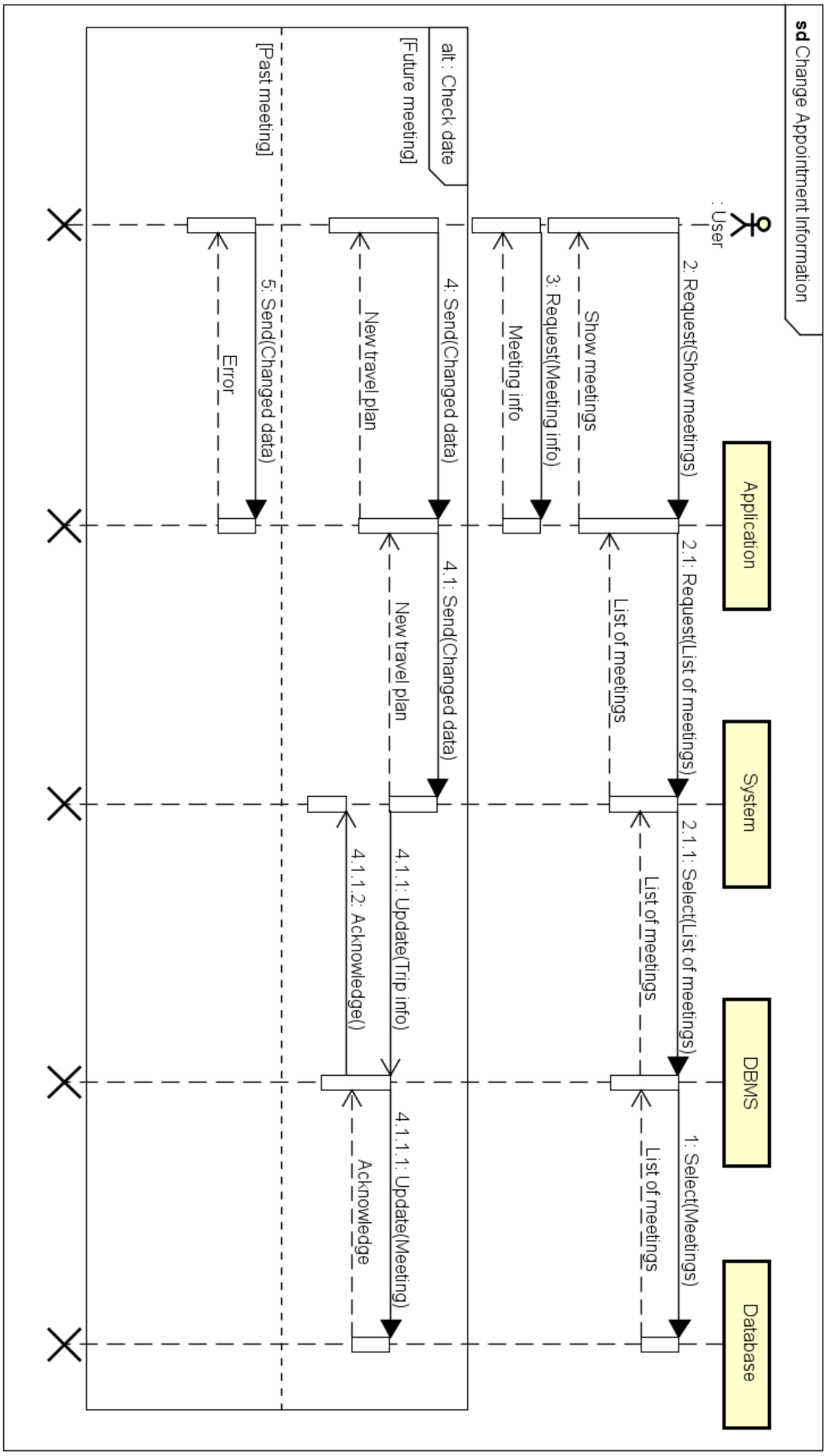

Figure 14: *Change Appointment Information* sequence diagram

3.2.6 Delete Appointment

Purpose
This function allows the user to select an appointment already registered in the calendar and removing it, the system will then notify the DBMS to remove it from the database.

Functional Requirements

R.33: The user must be logged in.

R.34: The user must have at least one upcoming event saved.

R.35: Once a meeting is deleted all data regarding it is lost.

R.36: Past events cannot be deleted.

Scenario
Emily decided with her friends to go out for dinner Saturday and inserted the place and time in a meeting using Travlendar+, but one of the other girls later proposed to have dinner at home, and since Emily has the biggest dining room she invited all the others to her place, since she doesn't need any more a travel planned she deletes the meeting previously created by selecting it in the "Manage meetings" section, she then proceeds to remove it by pressing the "Delete trip" button.

Use Case
The *Delete Appointment* use case is analysed in Table 6 and in Figure 15

Activity Diagram
The activity diagram of the *Delete Appointment* use case in showed in Figure 13 alongside *Change Appointment Information* from subsubsection 3.2.5.

Sequence Diagram
The sequence diagram of the *Delete Appointment* use case in showed in Figure 16

Name	**Delete Appointment**
Actors	User
Entry conditions	The user must be logged in and must have at least one upcoming event.
Flow of events	1. The user opens the app. 2. The user selects the "Manage meetings" section. 3. The user selects the meeting he/she wants to delete. 4. The user presses the "Delete trip" button. 5. The system sends the delete request to the DBMS. 6. The DBMS deletes the entry selected by the user from the database.
Exit conditions	A meeting was deleted.
Exceptions	If the meeting has expired and the user tries to delete it, the application will avoid it.

Table 6: *Delete Appointment* use case description

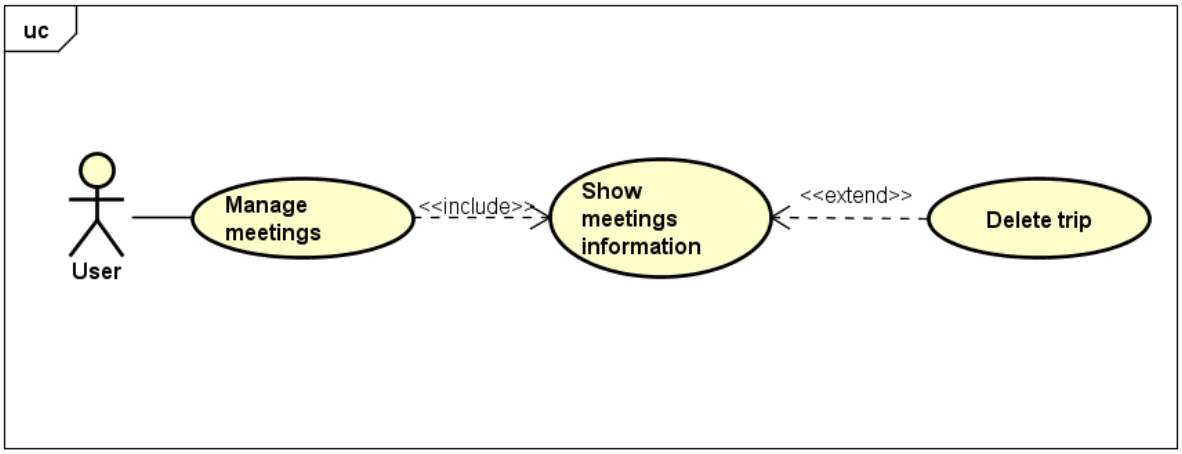

Figure 15: *Delete Appointment* use case

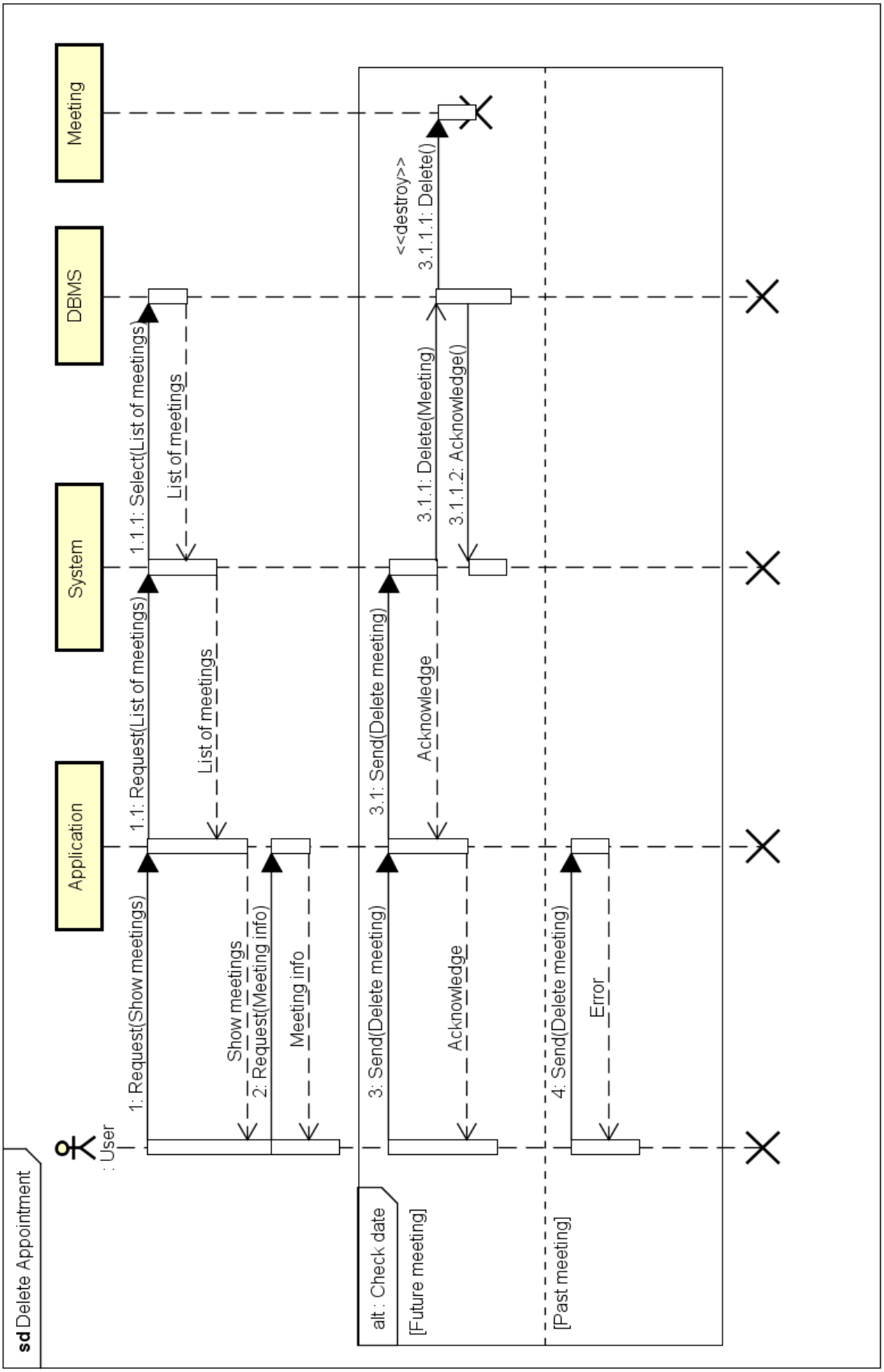

Figure 16: *Delete Appointment* sequence diagram

3.2.7 Manage Breaks

Purpose
The functionality delivered by *Manage Breaks* refers to the possibility of scheduling a flexible time in which the system will reserve a given amount of minutes to any kind of break, with the option of changing or deleting said break in the future.
Note that we will focus on the creating aspect, since treating also changing and deleting would offer no additional insight.

Functional Requirements

R.37: The user must be logged in.

R.38: The user must insert the following parameters:

(a) Starting time (From).

(b) Duration of the break.

(c) Days of the week.

(d) End time (To).

Scenario
Jimmy has a 3 hours window on Monday between school and soccer practice, from 12:30 am to 15:30 am in which he wants to have lunch and then study the remaining time.
He decides to use Travlendar+ to schedule a flexible break, first he opens the app on his smartphone, then after going into "Manage account preferences" he adds a break of 30 minutes in the spare time he has by using the "Create break" option and filling the necessary fields.

Use Case
The *Create Breaks* use case is analysed in Table 7, in Figure 17 are also represented the "Delete break" and "Change break" functions.

Activity Diagram
The activity diagram of the *Manage Breaks* use case in showed in Figure 18

Sequence Diagram
The sequence diagram of the *Create Breaks* use case in showed in Figure 19

Name	Create Breaks
Actors	User
Entry conditions	The user must be logged in.
Flow of events	1. The user opens the app. 2. The user opens the options menu. 3. The user enters in "Manage account preferences". 4. The user selects "Create break". 5. The user fills the requested fields with the desired information. 6. The user saves the changes. 7. The system sends the data to the DBMS. 8. The DBMS stores the data about the break.
Exit conditions	The user created a break
Exceptions	If the user doesn't choose a day of the week the system will act like if every day was selected.

Table 7: *Create Breaks* use case description

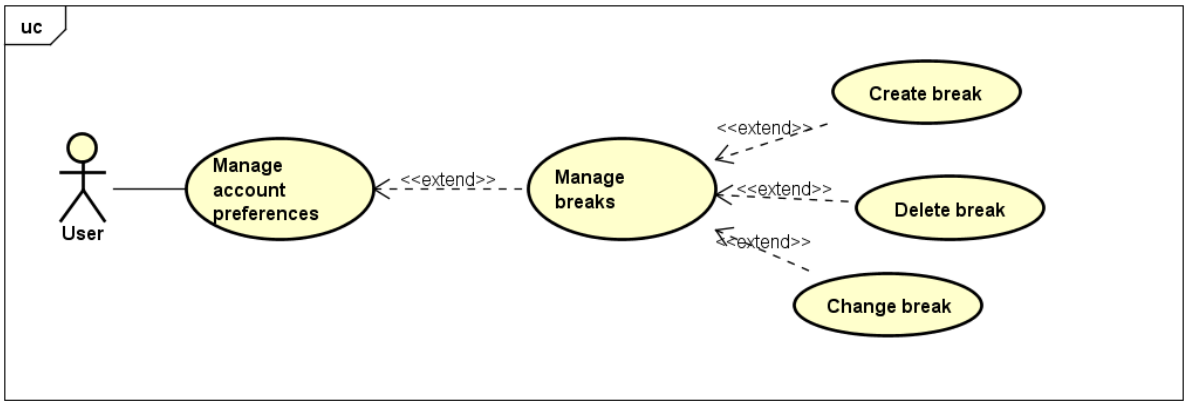

Figure 17: *Create Breaks* use case

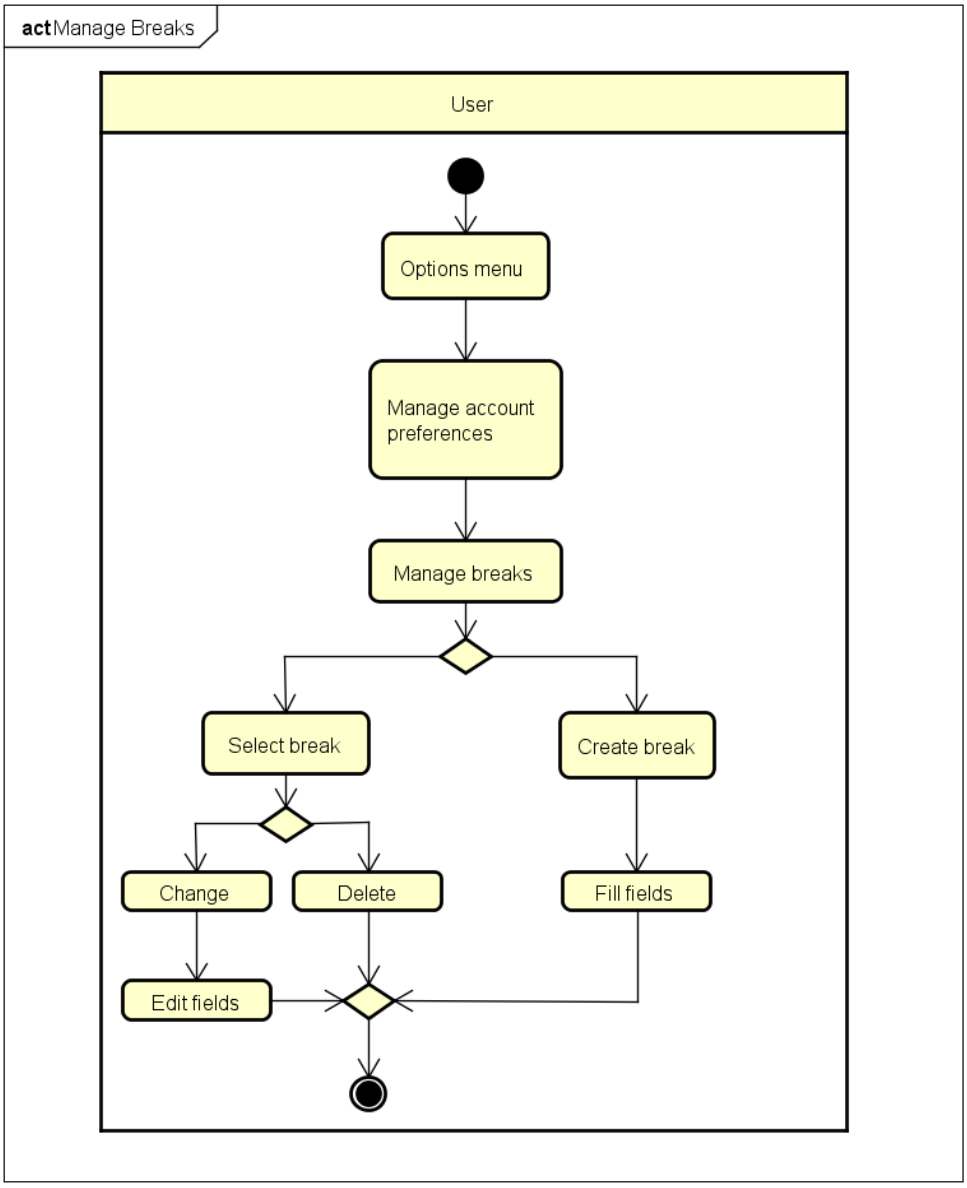

Figure 18: *Manage Breaks* activity diagram

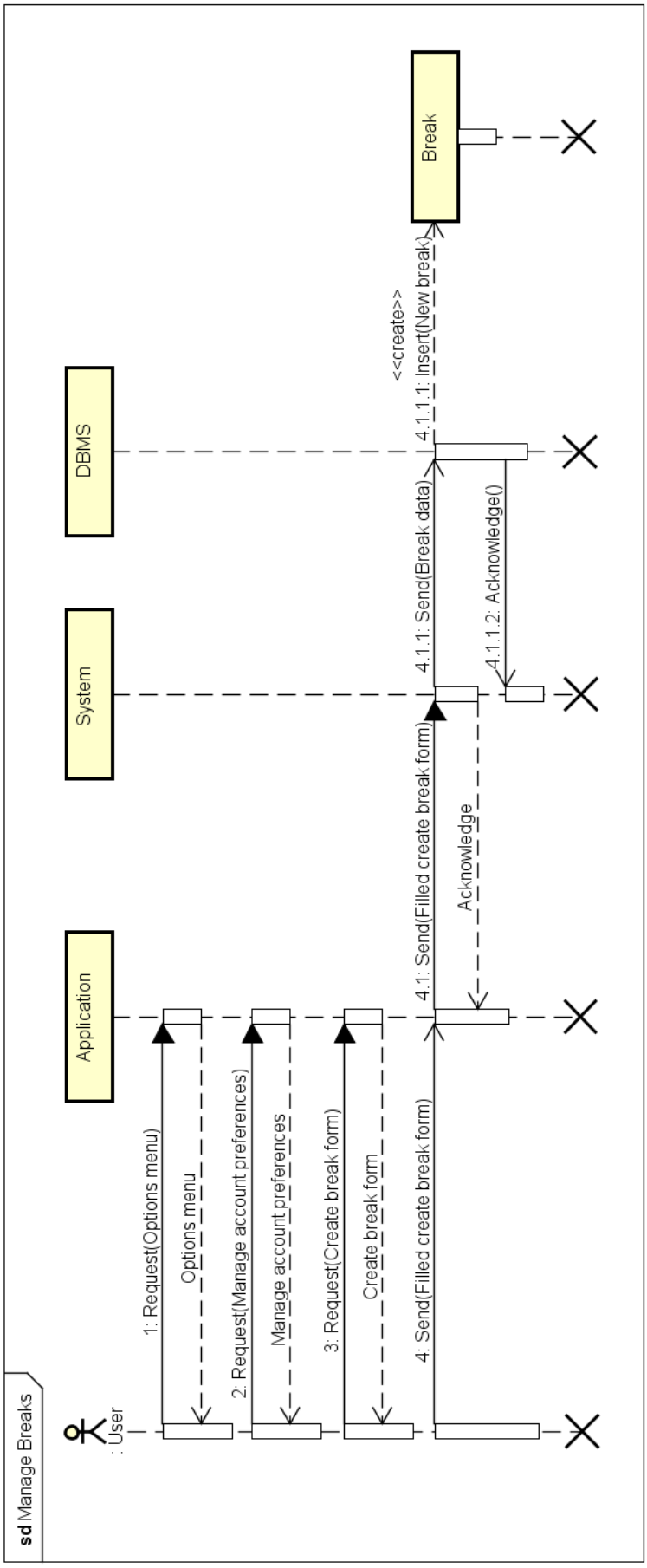

Figure 19: *Create Breaks* sequence diagram

3.2.8 Manage Account

Purpose
The main purpose of this use case is to grant the user the possibility to view and update his/her personal information and even delete the account.

Functional Requirements

R.39: The user must be logged in.

R.40: The system must shows all the information inserted during the registration process, except the password.

R.41: The system allows the user to update every information except the e-mail used to login.

R.42: The user must insert the old password once and the new one twice in order to update his/her password.

R.43: The system allows to change the password if and only if the old one has been inserted correctly.

R.44: The system does not allow the user to update the password if the new one has not been inserted twice.

R.45: The system makes every changes permanent as soon as the user confirms the operation.

R.46: The user can delete his/her account if and only if he/she inserts the current password.

R.47: The system removes from the database the user's personal information if and only if the user deletes his/her account.

Scenario 1
Carlo wants to change his password because it does not seem quite strong for him. He opens his *Travlendar+* app, selects "*Show my information*", "*Modify my information*" and then "*Modify password*". He enters his old password and the new password twice. The old password is written correctly, so the system accepts the new password.

Scenario 2
Bob does not use very often *Travlendar+*, so he decides to uninstall the app, but first he wants to delete his account. He opens the app, select "*Show my information*", "*Modify my information*" and then "*Delete my account*" and, in order to confirm his choice, inserts his password, the system then deletes all Bob's personal information from the database. Now Bob must register again if he wants to start using *Travlendar+*'s services once more.

Use Case
The use case is analysed in Figure 20 and in Table 8, Table 9 and in Table 10.

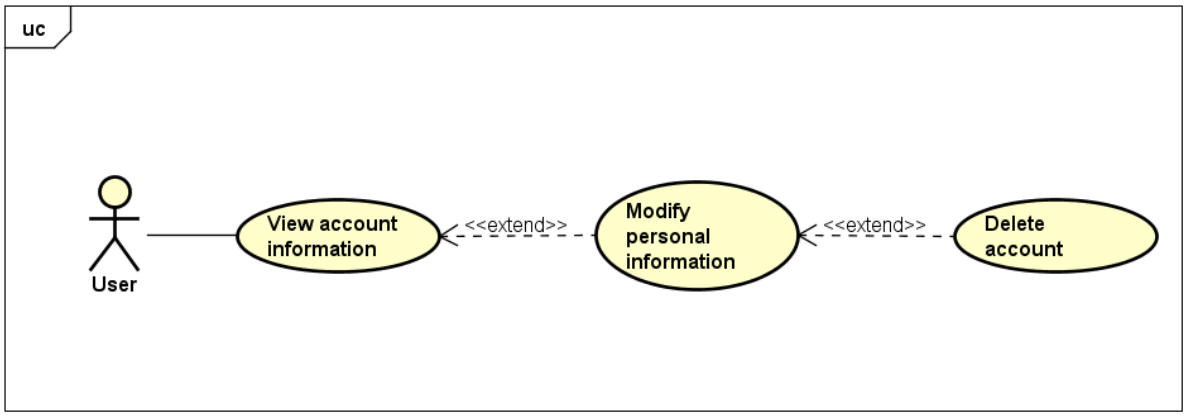

Figure 20: *Manage Account* overall use case

Name	**View account information**
Actors	User
Entry conditions	The user must be logged in.
Flow of events	1. The user opens the app. 2. The user opens the options menu. 3. The user enters in "*Show personal information*". 4. The system shows the user's personal information.
Exit conditions	The system lets the user checks his/her information.
Exceptions	

Table 8: *View Account Information* use case description

Name	**Modify personal information**
Actors	User
Entry conditions	The user must be logged in.
Flow of events	1. The user opens the app. 2. The user opens the options menu. 3. The user enters in "*Show personal information*". 4. The user selects "*Modify my information*". 5. The user enters the updated information. 6. The user confirms the changes.
Exit conditions	The system saves the information.
Exceptions	If one of the requirements from R.41: to R.44: is not satisfied the system ignores the changes and reload the "*Modify my information*" page.

Table 9: *Modify personal information* use case description

Name	**Delete account**
Actors	User
Entry conditions	The user must be logged in.
Flow of events	1. The user opens the app. 2. The user opens the options menu. 3. The user enters in "*Show personal information*". 4. The user selects "*Modify my information*". 5. The user selects "*Delete my account*". 6. The user inserts his/her password. 7. The user confirms the operation.
Exit conditions	The system deletes the user's account and the user is not registered anymore.
Exceptions	If the user enters the wrong password the system prevents the account cancellation.

Table 10: *Delete account* use case description

3.3 Performance Requirements

Without taking into consideration the speed of the internet connection, in order to guarantee an acceptable user experience the following requirements must be satisfied:

- Navigation between pages of the system must happen in 0.5s or less.
- The best travel plan must be computed in 5s or less.
- No limit of registered users in the database.
- No limit of schedulable appointments.
- At least 1000 users must be able to use the service at the same time.

3.4 Design Constraints

3.4.1 Standards Compliance

The web application must comply with the standards dictated by W3C, while the mobile application must follow the Oracle guidelines for Java programming.

3.4.2 Hardware Limitations

Minimum system requirements for the two applications:

- Web application
  - 512Mb of RAM.
  - 2Mb/s internet connections.
  - 800X600 screen resolution.
- Mobile application
  - 1Gb of RAM.
  - 3G UMTS internet connections.
  - 100Mb of free space.

The system should also be able to process operations in parallel.

3.5 Software System Attributes

3.5.1 Reliability

Each trip plan computed given the preferences expressed by the user and the weather forecast and strikes must not differ more than 5% from the optimal travel distance or ETA.

3.5.2 Availability

The system to be must guarantee an availability of no less than 99%.

3.5.3 Safety and Privacy Constraints

The user oversees his/her own security while travelling and must grant access to the current location, information about former trips and personal data are stored but only the user itself has access to them, the password itself is not saved in the database, instead the result of an encryption algorithm on the password is stored to compare it to the result of the same operation once the user inserts it.

3.5.4 Maintainability

The system must be developed in such a way that future implementation of new features and changes to existing ones can be done seamlessly, in other words the system has to be modular and scalable.

3.5.5 Portability

As already mentioned in subsubsection 3.1.3 the software must be available on different configurations, it must be as environment independent as possible, meaning that it has to work on different platforms with the minimum amount of changes to the software itself.

4 Alloy

In the following section we present the complete *Class Diagram* in Figure 21 and the Alloy code written using the *Alloy Analyzer 4.2* tool given by MIT.
At the end there is also an image of an instance found in Figure 23.

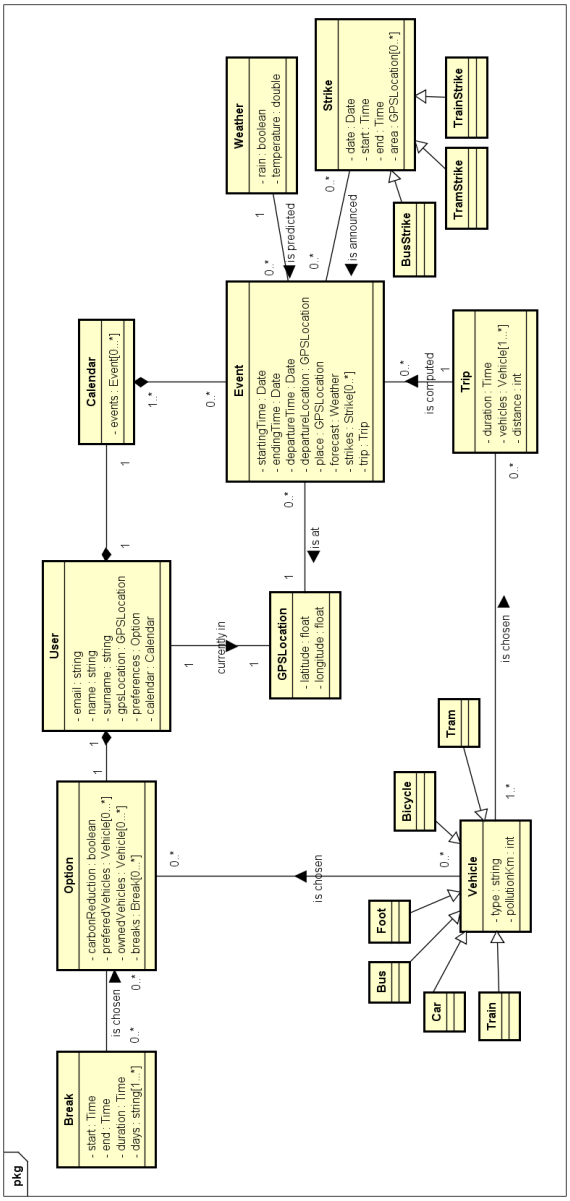

Figure 21: Class diagram illustrating the structure of the system-to-be.


```
//Dates are calculated as the number of seconds from the
//1st January 1970 following the UNIX standard

//————————————————————————————————————SIGNATURES————————————————————————————————————
open util/boolean

//The GPS position is represented as two integers for semplicity
sig GPSLocation{
    latitude: one Int,
    longitude: one Int
}

//A single event recorded inside the user's calendar
sig Event{
    startingTime: one Int,
    endingTime: one Int,
    place: one GPSLocation,
    weather : one Weather,
    strike : set Strike,
    trip: one Trip
}{
    endingTime > startingTime
}

//The flexible break the user can set during the day
sig Break{
    start : one Int,
    end: one Int,
    duration: one Int
}{
    start < end  and
    minus[end, start] > duration and
    duration > 0
}

//The calendar of a user, containing all the events.
sig Calendar{
    events: set Event
}{
    no disj e1, e2 : Event | !noOverlap[e1, e2]
}

//A model of the user
sig User{
    email: one Int,
    gpsLocation : one GPSLocation,
    preferences: one Preference,
    compiles: one Calendar
}

//The preferences specified by the user (A.K.A. options/settings)
sig Preference{
    preferedVehicle : set Vehicle,
    ownedVehicle : set Vehicle,
    breaks : set Break
}{
    no disj v1, v2 : Vehicle | v1 in preferedVehicle and
```



```
58      v2 in preferedVehicle and  v1 in ownedVehicle and
        v2 in ownedVehicle
60  }

62  //The weather forecast for a trip, we consider it accurate all the time
    sig Weather{
64      rain : Bool
    }
66
    //The trip itself from point A to point B
68  sig Trip{
        duration : one Int,
70      vehicles : some Vehicle
    }{
72      all t : Trip | some e : Event | t in e.trip
    }
74  //------------------------------ABSTRACT SIGNATURES-------------------------------------

76  //A vehicle/means of transport, either owned, public transport or shared.
    abstract sig Vehicle{
78      type : one Int
    }
80
    //For semplicity "Foot" is modeled as a vehicle
82  sig Foot extends Vehicle{}
    sig Bicycle extends Vehicle {}
84  sig Car extends Vehicle {}
    sig Bus extends Vehicle {}
86  sig Tram extends Vehicle {}
    sig Train extends Vehicle {}
88
    //Public transportation unavailability
90  abstract sig Strike{
        startingTime: one Int,
92      endingTime: one Int,
        area: some GPSLocation
94  }

96  sig BusStrike extends Strike{}
    sig TramStrike extends Strike{}
98  sig TrainStrike extends Strike{}

100 //-------------------------------------------FACTS-------------------------------------------

102 //No two users have the same email
    fact allUniqueUsers{
104     no disj u1, u2 : User | u1.email = u2.email
    }
106
    //A calendar belongs to only one user
108 fact userHasOneCalendar{
        no disj u1, u2 : User | some c : Calendar |
110         c in u1.compiles and c in u2.compiles
    }
112
    //A calendar cannot exists without its user
114 fact noUserNoCalendar{
```



```
    all c : Calendar | some u : User |
116         c in u.compiles
}
118
//An event cannot exists without being in a calendar
120 fact noCalendarNoEvent{
    all e : Event | some c : Calendar |
122         e in c.events
}
124
//A preference cannot exists without a user that chose it
126 fact noUserNoPreferences{
    all p : Preference | some u : User |
128         p in u.preferences
}
130
//Given the interval of a break it must be assured that at
132 //least the specified duration is free, in other words there
//always is an amount of free time equals to the break
134 //duration during a break interval.
fact MinimumBreakDuration{
136     all e1, e2 : Event, b : Break |
    e1. endingTime < e2.startingTime and e1.endingTime > b.start
138     and e2.startingTime < b.end
    implies minus[e2.startingTime, e1.endingTime] >= b.duration
140 }

142 //When it rains the trip must not contain a bike as transportation mean
fact noBicycleWhenRains{
144     all e : Event | e.weather.rain = True implies
    (no b: Bicycle | b in e.trip.vehicles)
146 }

148 //During a bus strike the trip must not contain a bus as transportation mean
fact noBusWhenStriked{
150     all e: Event | some bs : BusStrike | eventDuringStrike[e, bs] implies
        (no b : Bus | b in e.trip.vehicles)
152 }

154 //During a tram strike the trip must not contain a tram as transportation mean
fact noTramWhenStriked{
156     all e: Event | some ts : TramStrike | eventDuringStrike[e, ts] implies
        (no t : Tram | t in e.trip.vehicles)
158 }

160 //During a train strike the trip must not contain a train as transportation mean
fact noTrainWhenStriked{
162     all e: Event | some ts : TrainStrike | eventDuringStrike[e, ts] implies
        (no t : Train | t in e.trip.vehicles)
164 }

166 //Breaks must not overlap with one another
fact noBreakOverlap{
168     all disj b1, b2 : Break |
        b1.start < b2.start and b1.end < b2.start or
170         b2.start < b1.start and b2.end < b1.start
}
```



```
172
//————————————————————————————————————————PREDICATES————————————————————————————————————————
174
//Addition of an event to a calendar
176 pred addEvent[c, c' : Calendar, e : Event]{
        c'.events = c.events + e
178 }

180 //Removal of an event from a calendar
    pred deleteEvent[c, c': Calendar, e : Event]{
182     c'.events = c.events - e
    }
184
    //Checking if an event is happening during a public transportation strike
186 pred eventDuringStrike[e : Event, s : Strike]{
        userStartingTime[e] > s.startingTime and
188     userStartingTime[e] < s.endingTime
    }
190
    //Checking that events do not overlap with each other
192 pred noOverlap[e1, e2: Event]{
        e1.startingTime < e2.startingTime and
194     e1.endingTime < e2.startingTime or
        e2.startingTime < e1.startingTime and
196     e2.endingTime < e1.startingTime
    }
198
    //Shows the addEvent predicate
200 pred showAdd[c, c' : Calendar, e : Event]{
        addEvent[c, c', e]
202     #Calendar.events > 2
    }
204
    //Shows the deleteEvent predicate
206 pred showDelete[c, c' : Calendar, e : Event]{
        deleteEvent[c, c', e]
208     #Calendar.events > 1
    }
210
    //————————————————————————————————————————FUCNTIONS————————————————————————————————————————
212
    //When the user has to leave in order to reach the meeting in time
214 fun userStartingTime[e : Event] : Int{
        minus[e.startingTime, e.trip.duration]
216 }

218 //————————————————————————————————————————ASSERTIONS————————————————————————————————————————

220 //Assertion to prove that no event overlap with another
    assert noOverlappingEvents{
222     all c, c': Calendar|
            all e1, e2, e3: Event | e1 in c.events and e2 in c.events
224         and !(e3 in c.events)
            and !noOverlap[e1, e2] and addEvent[c, c', e3]
226         implies !noOverlap[e1, e3] and !noOverlap[e2, e3]
    }check noOverlappingEvents for 6 but 1 Calendar
228
```



```
//Assertion to prove that during the span of a break there is time for the pause
assert minimumDurationGuaranteed{
    all c, c':Calendar, e : Event, p : Preference |
        addEvent[c, c', e] implies
            (all e1 : Event, b : Break |
                e1 in c'.events  and b in p.breaks and
                e.endingTime < e1.startingTime and
                e.endingTime > b.start and
                e1.startingTime < b.end implies
                minus[e1.startingTime, e.endingTime] >= b.duration)
}
check minimumDurationGuaranteed for 6

//Assertion to prove that a trip as at least one vehicle, makin it valid
assert validTrip{
    all e : Event |  #e.trip.vehicles > 0
}
check validTrip for 6

//Assertion to prove that during a strike of a vehicle that vehicle will not be
//inside the chosen ones
assert noStrikedVehiclesInTrip{
    all e : Event | some bs : BusStrike,  ts : TramStrike, trs : TrainStrike |
    (eventDuringStrike[e, bs] implies (no b : Bus | b in e.trip.vehicles) and
        eventDuringStrike[e, ts] implies (no t : Tram | t in e.trip.vehicles) and
        eventDuringStrike[e, trs] implies (no t : Train | t in e.trip.vehicles))
}
check noStrikedVehiclesInTrip for 6

pred show{}

run show for 5 but exactly 1 User, 3 Event, 2 Break
```


4.1 Alloy Execution Result

The following is the result of the execution of the checks:

Figure 22: Alloy's Execution Result

Instance Found

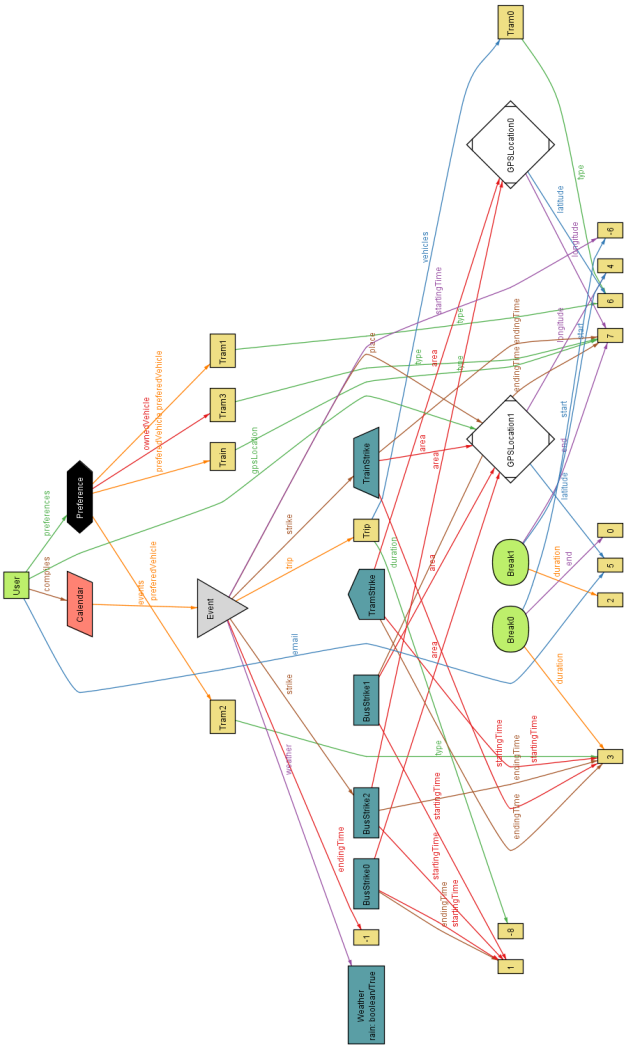

Figure 23: Instance of Alloy found.

5 Appendix

6 Software Used

1. Texmaker as an editor for LaTeX.
2. Astah for schemes drawing.
3. Git % GitKraken
4. Alloy Analyzer 4.2

7 Hours of Work

The hours listed are comprehensive of individual and group work time.

1. Riccardo Facchini: 22:30h
2. Andrea Guglielmetti: 21h

