Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

Travlendar+

R.A.S.D.

Requirements Analysis & Specifications Document

Alberto Floris

Claudio Montanari

Luca Napoletano

Contents

Chapter 1

Introduction

1.1 Purpose

This document represents the *RASD* (Requirements Analysis & Specifications Document) of the *Travlendar+* application. The main purpose is to describe the system with respect to the real world.

In the following chapters indeed, the user needs and functionalities will be analyzed to construct a useful model for a later implementation; this will be done taking into account real world constraints, to understand all the possible issues that may define and change the application structure.

1.2 Scope

Travlendar+ is aimed to be a 360 degree agenda for everyone's daily life. Firstly, the user will be able to define his appointments and a wide range of preferences. Preferences will include: type of transport, kind of appointment, eco-friendly profile and maximum walk range.

The System will be able to schedule the user appointments taking into account his preferences and making sure it will not be late. In addition, it will propose various mobility options considering time constraints (speed, traffic...), weather constraints (e.g. when it rains the user would avoid bikes trip), day constraints (strikes, festivity...).

Finally, the System will permit the user to buy public transport rides, rent or locate the nearest car or bike. This will be done considering potential discount and daily, weekly or season travel card.

Nowadays a system that integrates all these features doesn't exist, neither a supporting framework, this allow us to design the entire system from scratch.

1.3 Definitions, Acronyms, Abbreviations

In this part of the *RASD* Document there are some definitions, acronyms and abbreviations that will be used among the following chapters.

1.3.1 Definitions

- **The Software:** when referring to *The Software*, this document refers to the entire *Travlendar+* infrastructure, at implementation and design level.
- **Application:** in the document is used like a synonymous of Software.
- **The Service:** when referring to *The Service*, this document refers to the service provided by the *Travlendar+* Software, at market level.
- **User:** the final person who use the *Travlendar+* software.
- **Registered User:** a user who has registered himself within the *Travlendar+* System.
- **Logged In User:** a registered user who is logged in the *Travlendar+* System. This kind of user will be one of the main actors interacting with the Software.
- **Visitor:** a user which is not registered within the *Travlendar+* environment.
- **Time Slot:** period of continuous time when a task can be schedule.
- **Task:** an appointment the user has to do. A task always has associated a location, a time slot and a priority. Each task can have one or more of these behaviors:
  - **Fixed time:** a task with fixed time possesses a user specified time slot, that the System has to consider when producing the day's schedule.
  - **Flexible time:** a task with flexible time does not possess a fixed user specified time slot, instead it possesses a wider time frame so that the System could choose when to schedule the task within the given time frame constraint, which is user specified.
  - **Variable time:** a task with variable time does not possess a user specified time slot, so the System will instead schedule the task relating to the other time constraints.
  - **Fixed Day:** a task with fixed day possesses a user specified day within the week or the month, that the System has to consider when producing the day's schedule.
  - **Flexible Day:** a task with flexible day does not possess a fixed user specified day, instead it possesses a set of days within the week or the month so that the System could choose when to schedule the task within the given set, which is user specified.
  - **Variable Day:** a task with variable day does not possess a user specified day, so the System will instead schedule the task relating to the other time constraints within a user specified deadline.
  - **Fixed period:** the task repeat itself exactly after the given period.

  - **Flexible period:** the task repeat itself within the given flexible period, which is an interval of one or more days length.
  - **Not Periodic:** the task is not repeated.

- **Periodic task:** a task that possesses a fixed or flexible period.
- **Tasks Conflict:** given two user tasks, there will be a conflict if the two time slot associated to those tasks overlaps.
- **Task priority:** when referring to task priority we intend to refer to a user defined property which represent how much is important to follow that task constraints. The task priority can vary from 0 (lowest priority) to 5 (highest priority).
- **Road Path:** the path that the user has to do in order to reach a location from another.
- **Location:** the place where the user either is or wants to go.
- **Localization:** the way our application knows the user's position.
- **Residence:** the place where the user lives.
- **ZTL:** an area where some kind of private vehicles, such as cars or motorbikes, cannot enter. If the user has specific documentation, he can enter in the ZTL with his private vehicles.
- **Private Vehicle:** every kind of vehicle owned by the user, such as a car, a motorbike or a bike.
- **Shared based Vehicle:** vehicles that the user can book, and then rent, for a trip. To use these kind of vehicle the user needs a valid account of the vehicle sharing service he wants to use.
- **Public Transport Vehicle:** vehicles belonging to the public transport such as: bus, metro, tram and taxis.
- **Ticket:** a valid public transport document to use that service.
- **Break Time:** the time chosen by the user in which he doesn't want to have appointments.
- **User Preferences:** all the in-application preferences that user can choice. These preferences are:
  - Use of the private vehicle.
  - Use of the public transport services.
  - Use of either the Car-Sharing or the Bike-Sharing services.
  - Max range of walking.
  - Break time.
  - Eco profile.
  - Public transport allowed time slot.

These kind of preferences can be used either as global preferences or as task preferences. In the former case they are treated for all the tasks, in the latter they are treated only for the task which has them. If a task has some preferences associated, then the system will consider instead of the global ones.

- **Travel Solution:** the proposed solution for the user that allows him to reach the destination in the way the user has specified through the preferences, which includes: travel indications for each travel mean involved, starting and arrival time.
- **Scheduling:** when referring to schedule we mean the sequence of user tasks for each day, and the set of travel solutions between them. These travel solutions will be suggested by the *Travlendar+* software, as described under the **Goals** section.
- **Notifications:**
  - System Notification: when referring to system notification we mean a notification issued to the user by the *Travlendar+* environment, that serves as reminder for the user himself about when he needs to move from one location to another.
  - Digest: when dealing to Digest we mean a notification issued to the user by *Travlendar+* environment that regards the entire schedule for a specific day. This kind of notifications could, for example, be issued during early morning as a reminder of what the user has to do within the day.

  Either *System Notifications* or *Digests* can be issued as e-mails or simple notification using the *APIs* given by the deployment Operating System, depending on which platform the user is working on.
- **Extraordinary conditions:** when referring to extraordinary conditions we refer to situations that are not predictable by the Software: examples of extraordinary conditions are:
  - Not Announced Strikes.
  - Public transportation failures due to incidents or technical issues.
  - User's private vehicle failures.
  - User's unexpected appointment.
  - Tasks that are not being inserted within the *Travlendar+* environment.
  - Extremely uncommon weather conditions: hurricanes, snow storms, floods or landslides.
  - Restrictions to the normal traffic flow, either pedestrian or vehicular, issued by the local government authorities.
- **Working days:** we consider as working days: Monday, Tuesday, Wednesday, Thursday and Friday.
- **Weather conditions:** when referring to weather conditions we mean all those conditions dealing with weather that can affect the task scheduling. For example, *rainy* is a condition that affects the task scheduling, whether *cloudy* it's not.

1.3.2 Acronyms

- R.A.S.D: Requirements Analysis and Specifications Document
- A.P.I: Application Programming Interface
- Z.T.L: Limited Traffic Zone

1.3.3 Abbreviation

These abbreviations will be used both in this document and in the follows documents.

- [G k]: The k-th goal
- [D k]: The k-th Domain Assumption
- [R k]: The k-th Functional Requirement

1.4 Document Structure

The *RASD* Document is divided into 4 main parts:

- Introduction
- Overall Description
- Specific Requirements
- Formal Analysis

The main purpose of this document is to describe in depth the characteristic of the *Travlendar+* application.

In the first chapter is described the System at an high level of information, explaining the main purpose of the application and its scope. Moreover there is also a part that helps the reader in understanding the main definitions, acronyms and abbreviations used in the entire document.

In the second chapter, the document goes more in detail about the application, talking about the main goals the application needs to satisfy, and how the System interacts with the main actors, such as the final user, the public transport service, the car-sharing or the bike-sharing service. Finally, in this part there is a section devoted to explain the main domain assumptions the *Travlendar+* environment requires to reach the goals.

In the third chapter are reported some mock ups about our application, divided in mobile application and web application, as indeed our System can be used either on a mobile device or on a classical web browser. After that there is an explanation about the functional and non-functional requirements of *Travlendar+* and also there are some *UML Use Case* diagrams and *UML Sequence* diagrams that aim to describe better how the application will work.

In the chapter four are attached the *Alloy* models used to verify the correctness of the *Travlendar+* System. The code is reported and properly documented and, in addiction, there are some runs of the more relevant assertions and predicates.

Finally there is the chapter five, in which is reported the effort spent by each member of the team.

1.5 Revision History

This part of the *R.A.S.D.* Document contains the versions involved in the development of the final one.

Version 0.1: First written of the Index
Version 0.2: Starting analysis of the world
Version 0.3: Participating Actors added to document (referring to section 2.3)
Version 0.4: First writing of the Application's goals
Version 0.45: Starting with Alloy models
Version 0.6: Adding Mock ups to the Document
Version 0.7: Added Use Case & Sequence Diagram
Version 0.8: Finishing Alloy models
Version 0.9: Final Review of the Document
Version 1.0: Delivery of the *RASD* Document
Version 1.1: Application Icon changed
Version 1.2: Domain Assumption and Requirement Revision

1.6 Reference Documents

- Specification documents: *Assignments AA 2017-2018.pdf*
- 29148-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering
- Alloy Language Reference

Chapter 2

Overall Description

2.1 Product Perspective

The *Travlendar+* Software has as main goal to schedule all the user's appointments in a comfortable and useful way: planning the optimal travel solution to allow the user to reach his appointments in time.

To do that, the user has to give his tasks to our System, then the System creates a possible calendar considering user's preferences, tasks' priorities and roads status; asking directly to the user in case of conflicts.

The various trips between tasks may rely on public transport, in that case the System will suggest the user the best ticket option, trying to minimize the user expense considering the whole calendar's appointments. Otherwise, if is it the case of a shared based travel mean, the System will allow the user to book and later rent such vehicle through the appropriate site or application. Since it is unrealistic to guarantee a precise schedule when accounting for shared-based travel means, due to a series of problems out of the scope of the application[1], the System will check the availability of the vehicle before the trip starts and later notify the user, suggesting a reservation time for a close vehicle or another travel option.

The System provides a series of preferences settings, among the more relevant there are: the possibility to specify a repetition during the week of a certain task (also month-periodic or with user-defined periodic tasks) within a fixed, flexible or variable time slot; the possibility to define a maximum range for walking to reach a certain location; the possibility to avoid or prefer a certain travel mean, given the user vehicles' (eco-profile) or the time; finally a special task to reserve some breaks time to the user.

In addition, the user will be able to dynamically modify tasks' preferences and features, with different granularity options, receiving a new calendar schedule back.

[1] When the *Travlendar+* application has to schedule a trip with a shared based travel mean it cannot be sure that a shared-based vehicle will be available in that zone at that time

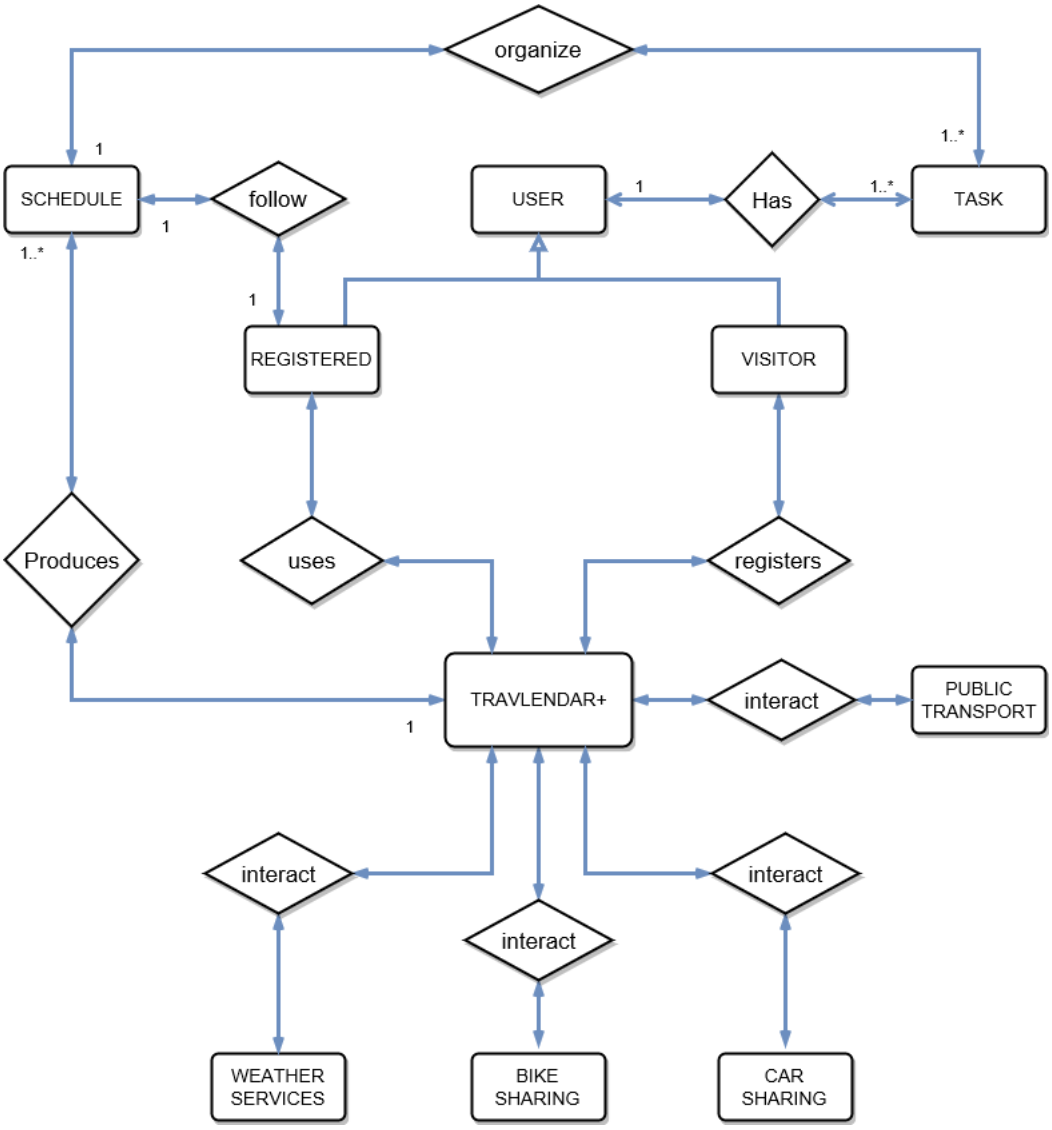

Figure 2.1: ER Diagram of the *Travlendar+* System

2.2 Product Functions

In this section of the chapter 2 we are going to talk about the main goals that *Travlendar+* should satisfy, according to the assigned specification document.

[G 1] Schedule the user's task

- [G 1.1] The System should allow the user to add tasks with all possible behaviors as described in the **Definitions** section.
- [G 1.2] The System should allow the user to add break tasks as defined in the **Definitions** section.
- [G 1.3] The System must decide the starting and ending time of each flexible and variable task, according to the given time slot and trying to avoid conflicts.

[G 1.4] The System will have to suggest a travel solution in order to reach each task location, according to the global and task related user's preferences.

[G 1.5] The travel solution proposed by the System should respect the time slot of each task.

[G 1.6] The System should handle impossible tasks in terms of time, reachability or conflicts by notifying the user of the situation.

[G 2] The System will have to allow the user to choose from a set of global and task dependent preferences in order to schedule the user's tasks.

[G 3] The System should allow the user to select from a set of different travel solutions the one he prefers.

[G 4] The application always tries to respect the user's preferences, giving more priority to the task based one. If it isn't possible, it tries to respect the user's preferences according to the priority given to each preference, otherwise the user will be notified.

[G 5] The System will have to interact with public transportation services in order to arrange trips for the user, allowing the user to buy rides.

[G 5.1] The purchase of season, weekly, daily tickets should be proposed when preferable in terms of money.

[G 6] The System will have to interact with Car-Sharing and Bike-Sharing services to guarantee the best travel solution according to the user's preferences.

[G 7] The System should compute a travel solution without any kind of transport vehicles.

[G 8] System should always allow user registrations.

[G 9] The System should compute a travel solution which allows the user to retake his private vehicles.

2.3 User Characteristics

The main actors who interact with the *Travlendar+* software will be presented under this section. Note that the contribute of each actor can vary consistently, depending mainly on implementation and design details. For example, it can be possible to obtain certain data from one actor instead of another, or there can be actors who provide only redundant information: that can be true assuming that some kind of data can be obtained from multiple sources, and it is easy to understand that at design and implementation level to interact with a smaller number of actors can be preferred.

- *Public Transport Services*: The software will have to interact with Public transport services in order to obtain ride's and pass' prices, traffic information, news about strikes, bus stops, metro stations and public transportation scheduling.

- *Map and Weather Services*: The software will have to interact with cloud services that provide weather information. Moreover, the software will have to interact with map services such as *Google Maps* in order to schedule tasks and suggest the best itinerary for each travel.
- *Visitor and Registered User*: The software will have to interact consistently with either visitors (that are, as described before, unregistered users) and registered users. This means that the software will have to recognize whether a user is registered and logged, and, if not, it will have to provide a sign up and sign in interface. In particular, *Travlendar+* services will be available only for Registered Users, whether visitors can only visit explanation pages about the service.
- *Bike and Car Sharing Services*: The software will have to interact consistently with Bike sharing and Car sharing services in order to provide consistent traveling alternatives to users who can and want to use such services. In particular, the software will have to refer to services that operates in the deployment area.
- *Local Administrations*: In the future, the software could interact with local administrations (e.g. municipalities, regional administrations) in order to have access to useful data that can be used to offer the best possible service. For example, the software could access historical traffic data in order to suggest whether the user should use his personal vehicle or the public transportation service.

2.4 Assumptions, dependencies and constraints

In this section of the *RASD* Document we describe assumptions and constraints (both from the given specification and from design constraints) to develop at the top the *Travlendar+* application.

2.4.1 Domain Assumptions

We suppose that these properties hold in the analyzed world:

[D 1] The user of *Travlendar+* can always give to the application his position correctly.

[D 2] Each user's task has one position and one time slot.

[D 3] Every task is associated to one or more transport mean preferences.

[D 4] Every task has: a unique time behaviour, a unique day behaviour and a unique period behaviour.

[D 5] The time granularity of a timeslot is 15 minutes.

[D 6] Every user has a unique username.

[D 7] The user must specify a residence.

[D 8] Every task has a location within the covered area.

[D 9] The covered area is Milan.

[D 10] To use the Car-Sharing service, the user must have installed the correct application.

[D 11] To use the Bike-Sharing service, the user must have installed the correct application.

[D 12] To buy the public transport ride, the user must have installed the correct application.

[D 13] For either the Bike-Sharing or the Car-Sharing services the user must have already an account to use them. If the user hasn't already an account, he can sign up to the service and then he has the possibility to add that travel mean preferences to our application.

[D 14] The Car-Sharing and the Bike-Sharing services give to our application always the correct positions of their vehicles.

[D 15] Both the Car-Sharing and the Bike-Sharing services always allow us to detect the free vehicles correctly.

[D 16] Public transport has at most 5 minutes of delay.

[D 17] Car-Sharing and Bike-Sharing services let the user to rent either a car or a bike, and once the vehicle is booked, no one is allowed to take that vehicle except for the user.

[D 18] The public transport society provide a method to know when strikes are scheduled.

[D 19] Public transport timetables are available.

[D 20] An external weather service will be used to know the week weather forecast.

[D 21] The travel path given by the external maps service is the optimal solution in terms of time or distance.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

In this section we are going to better explain the relation between our software and the main actors of *Travlendar+* System.

3.1.1 User Interfaces

Our application will be mainly developed as a Web Application, so the user can access to *Travlendar+* services with a Web Browser. Furthermore, we want to develop an application for Android and IOS for a better user experience on mobile devices. What follows are some mock up pictures related to both the Web Application and the Mobile Application. The aim is to provide a clear visualization of what the main functionalities of the System are and how the user will interact with them.

Log in

In the two images below there are the log in screens from the *Travlendar+* Web and Mobile applications. In both cases, the visitor user may easily insert his Username and Password to log in into the System.

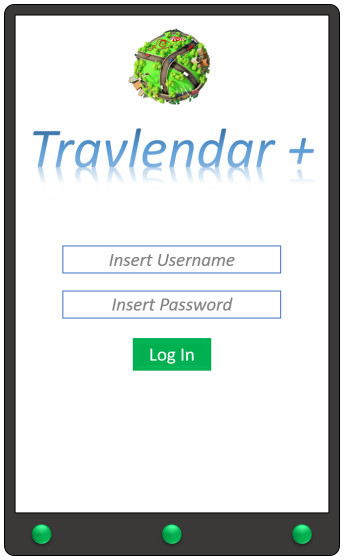

Figure 3.1: Login from the mobile application

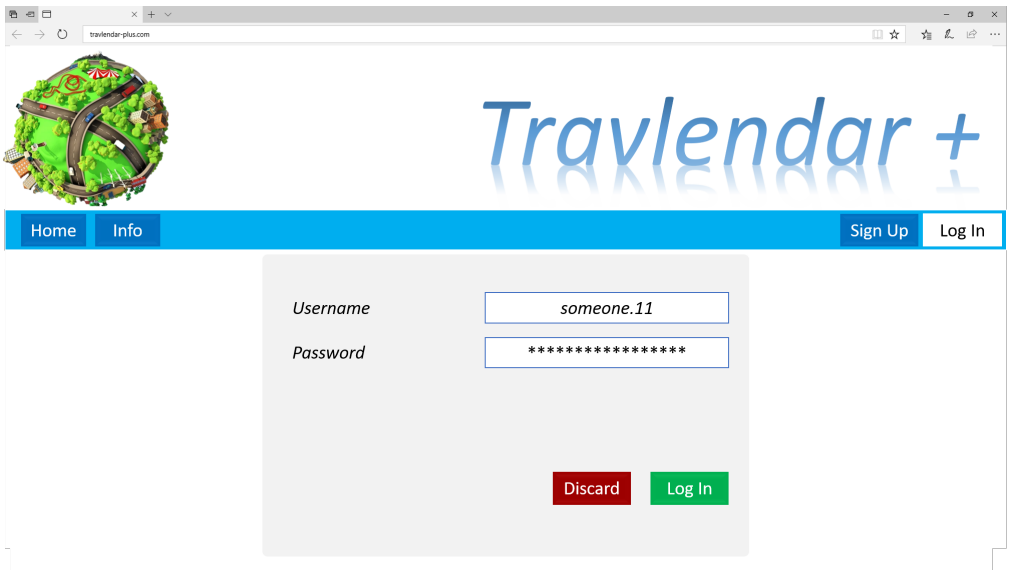

Figure 3.2: Login from the web page

Calendar

In the images below is represented a possible calendar rendering from both the *Travlendar+* Web and Mobile applications. In these pages the user can have a look at the scheduled tasks, changing if he prefers the time granularity, from weekly to daily or monthly. In these screens the user will be able to select a task simply by clicking on it.

In addition, on the Mobile application it will be possible to access a menu by swiping from the left to the right part of the screen.

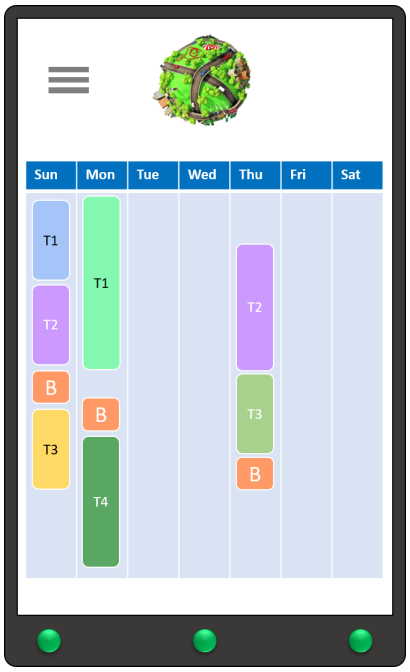

Figure 3.3: Calendar render on mobile device

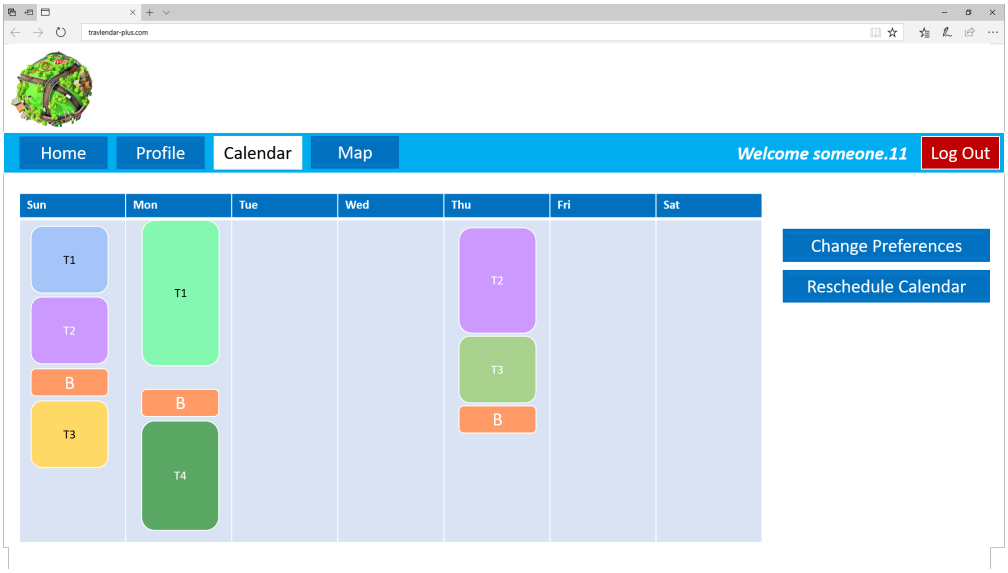

Figure 3.4: Calendar render on web page

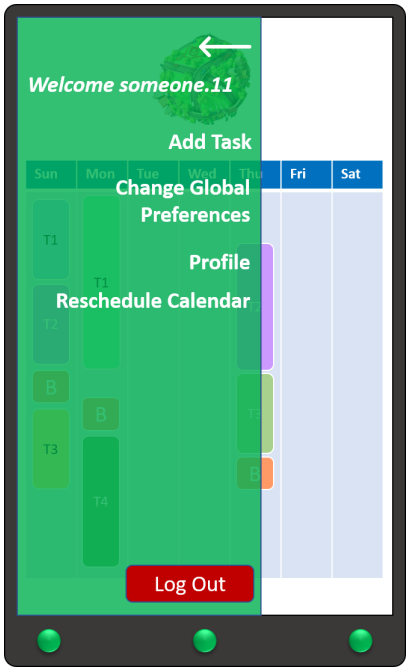

Figure 3.5: Main menu on mobile device

Task Modification

From the images below it's possible to see a rendering for the task modification screen. In these pages the user will be able to change task's preferences, task's time and date. It will also be possible to access the Travel Map pages which are better explained in the following subsection.

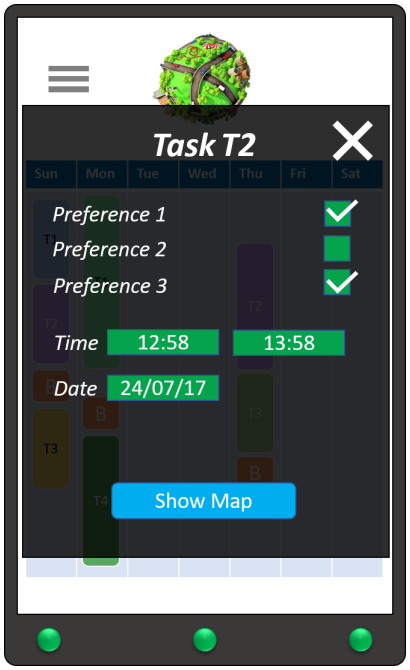

Figure 3.6: Task modification on mobile device

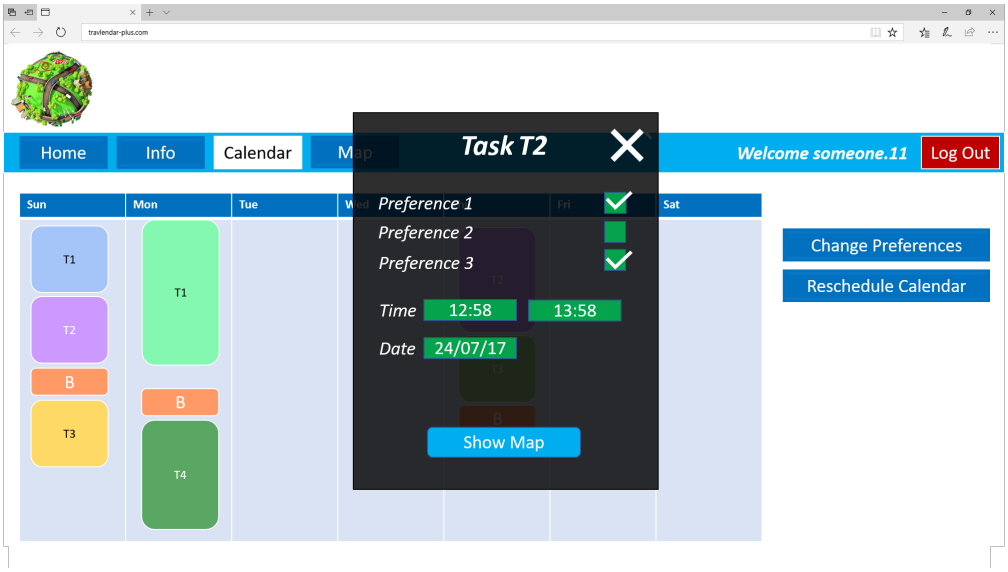

Figure 3.7: Task modification on web page

Task Travel Information

In these last two pictures it's possible to see the screens related to the Travel information, from these pages the user can have a look at the travel solution scheduled for each task. From this page it's also possible to be redirected to an appropriate site or application for finalizing the purchase of a ride or to rent a vehicle.

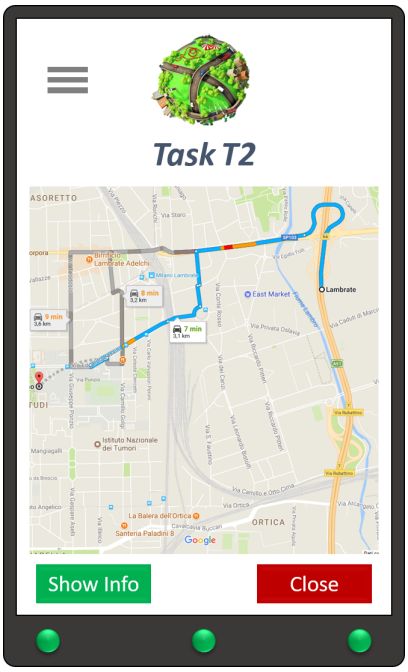

Figure 3.8: Map render and road information on mobile device

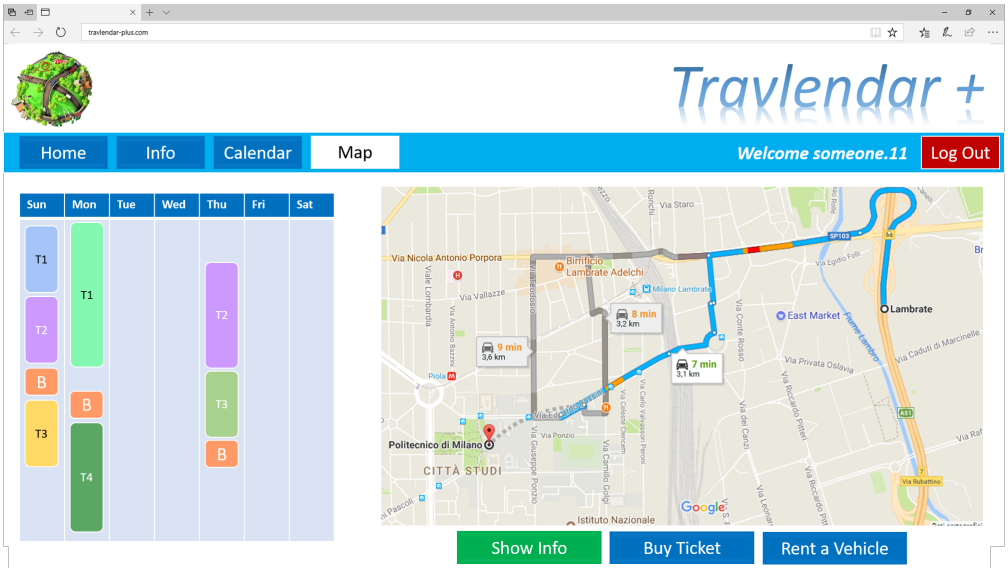

Figure 3.9: Map render and road information on web page

3.1.2 Software Interfaces

The *Travlendar+* system needs to know more information concerning, for instance, the traffic on the roads, the weather forecast or the status of the public transport service. So, in order to schedule the user's calendar, our application needs to interact with these external software:

1. External Map Service
2. External Weather Service
3. Public Transport System
4. Car-Sharing Service
5. Bike-Sharing Service

All these software will be accessed through specific *APIs*

3.1.3 Communication Interfaces

Travlendar+ software will be developed mainly with the idea of a web application but, there will be multiple way to access it, such as through a web browser or a mobile device. In both cases it is mandatory to have a working *Internet* connection, in order to send data to the main *Application Server*, to compute the necessary calculations about the user's tasks, and to resend the computed calendar.

3.2 Functional Requirements

To reach the main goals of the *Travlendar+* application we need to satisfy these functional requirements:

[R 1] The user will have to specify a residence at registration time. This home address can be updated lately, thus the Software will have to permit to modify it.

[R 2] The application must send a reminder to the user 30 minutes before he should begin the trip. Moreover, the application has to verify if a Sharing-Based vehicle is effectively available, if actually it is a user preference.

[R 3] If there isn't a Sharing based travel service available, the application must give a feasible alternative travel solution to the user according to his preferences.

[R 4] If two or more user's tasks have an overlapping, which cannot be schedule in a feasible way, the Application asks to the user to choose the task he prefers to do.

[R 5] The Software will have to permit the user to specify when he wants to return home. Thus, the software should have a special task representing it.

[R 6] The application has to reschedule a day if the user adds, deletes or updates a task during the day.

[R 7] If the user wants to add, delete or update a task during the day, the Application gives a travel solution that may not respect the time constraints.

[R 8] If the user adds a task that isn't reachable within the given time, the *Travlendar+* System sends a warning to the user.

[R 9] Each task can be repeated a finite or infinite number of times. In particular, a task can be repeated:

- [R 9.1]: the same day of every week
- [R 9.2]: the same day of every month
- [R 9.3]: the same day of every year
- [R 9.4]: two or more days during a single week

Thus, the application should take care of repeated tasks considering the same specific preferences as defined by the user when creating the task.

[R 10] The application should consider whether the user is using private vehicles or not throughout a day.

- [R 10.1]: The Software should not suggest to use a private vehicle if the user left his residence without it.
- [R 10.2]: The software should consider that if the user left home using a private vehicle, it will be necessary to return home using the same private vehicle.

[R 11] The Software should permit to add or modify either general preferences, which are always taken care of, or specific preferences, which applies on a single task.

- [R 11.1]: If a specific preference conflicts with a general preference, the software will have to consider the specific one, ignoring the general one.

[R 12] The application should allow the user to insert tasks which are time-variable during the week. Thus, the Software should take care of these tasks scheduling them in an appropriate way, choosing day and time.

[R 13] The application should permit the purchase of either public transport service or vehicle sharing services through the respective external applications.

[R 14] The application should show all user's schedule with different kind of visualization: by month, by week or by day.

[R 15] The application should permit the selection of a single task, in order to show the relative preferences and should give the possibility to change these ones.

[R 16] The system has to guarantee new user signing up with a proper sign up page.

[R 17] The system, in order to choose the best travel solution for the user, has to consider two main parameters: cost and time of the travel. To keep this kind of information, the system uses this formula: $y = a \cdot cost + b \cdot time$. Therefore, the system will minimize the y variable. If there are two travel solutions which have the same y values then the application will ask to the user which one to choose.

[R 18] If public transport is a user's preference, the system has to suggest the best ticket available with respect to the user's tasks.

[R 18.1]: If the user has already a season ticket, then this information will be taken into account when scheduling new and old user's tasks.

3.3 Use Case Diagram and Scenarios description

3.3.1 Login System

Every user who wants to use the *Travlendar+* application should be registered to the application System. After that, the user has to log in into our System in order to be recognized and to see his personal calendar.

Actors	• Visitor • Registered User • Logged-In User
Goals	[G 8]
Enter Condition	There is no enter condition for this Use Case
Events Flow	1. The *Visitor* access to the web site or application log in page 2. The *Visitor* fills all the mandatory information (personal data, password, email) 3. The *Travlendar+* System registers the user and send back a confirmation email 4. The *Visitor* user becomes a *Registered* user inserting his email and password in the log in page 5. The *Registered* user now can access the *Travlendar+* services through the log in page 6. After the insertion of username and password, the user becomes a *Logged in* user
Exit Condition	The user is registered in the *Travlendar+* System, and his account is created. Now the user is able to access to his calendar and his account through both the web site and the mobile application
Exception	The *Visitor* is not able to register himself because is already registered. The *Registered* user is not able to sign in the System because the inserted username or password are wrong or if he did not confirmed his email. All these kind of exceptions will be handled by the System showing to the user an error message either in the website or in the mobile application

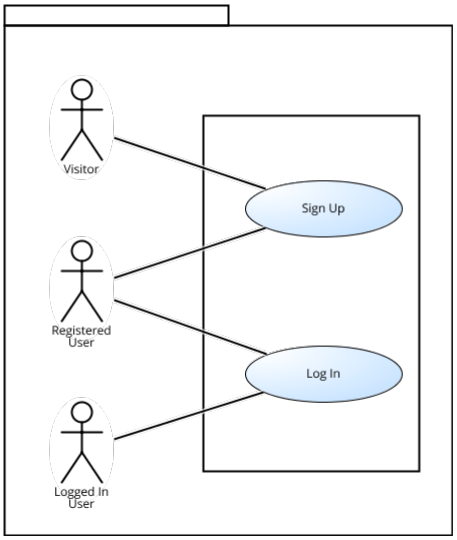

Figure 3.10: UML Use Case Diagram for Log In System

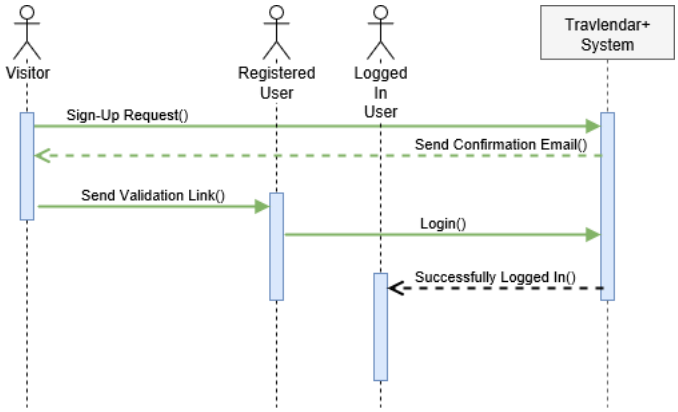

Figure 3.11: UML Sequence Diagram for Log In System

From this point among the next sections, we consider as the *User* a logged user.

3.3.2 Adding a new Task

In this section is reported the procedure used to create and add a new task to the user's calendar. The user can select a variety of preferences to adapt the task to his needs, indeed he can select when and where the task will take place, how he wants to reach the location of that task, the task priority and the maximum walk range. In this way our System will be able to schedule the user's calendar according to both the user preferences and the task preferences, giving more priority to the latter. If an overlap occurs or if a certain task is unreachable the System will notify the user asking respectively to choose from one of the overlapping tasks or to change some task setting. At this purpose, see the Sequence Diagram below that models this situation.

Notice that, there are some preferences which are mandatory (the ones listed below under the Events Flow voice) while other are not, thus, default values will be automat-

ically set by the System. In addition, we must highlight that the Events Flow voices from point 2 to 5 might be executed by the user as he prefers.

Actors	• User
Goals	[G 1]; [G 1.1]; [G 1.3]; [G 1.6]; [G 4]; [G 9]
Enter Condition	The user should log in the *Travlendar+* System and he is on the calendar page
Events Flow	1. The user press on the *"Add a new Task"* button 2. The user select at least one *time* preference 3. The user select at least one *day* preference 4. The user insert a location, where the task will take place 5. The user select if the task has to be a periodic task or not. If the task has a periodicity, the user has to select when the task should be repeated 6. The user can select other elective options, for instance the priority of the task, the maximum walk range or a preferred vehicle to reach the task. If the user skips this phase, then automatically default preferences will be applied 7. The user confirm the task just created
Exit Condition	The task just created by the user is successfully added to the user's schedule
Exception	The task just created by the user is not added to the calendar. It may be happen either for an overlapping (e.g this task's time slot is overlapped with another task that is already in the calendar) or for some wrong information inserted in the creation phase. For the former problem, the user has to select which task he prefers, for the latter the user has to create again the task inserting the correct information

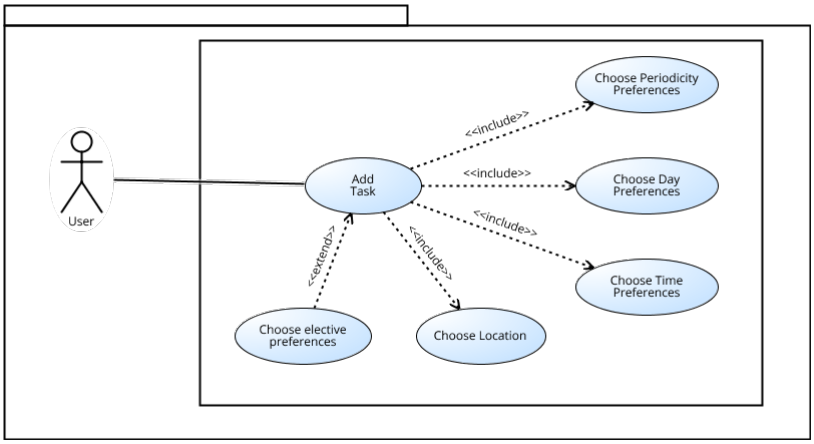

Figure 3.12: UML Use Case Diagram for the addition of a new task

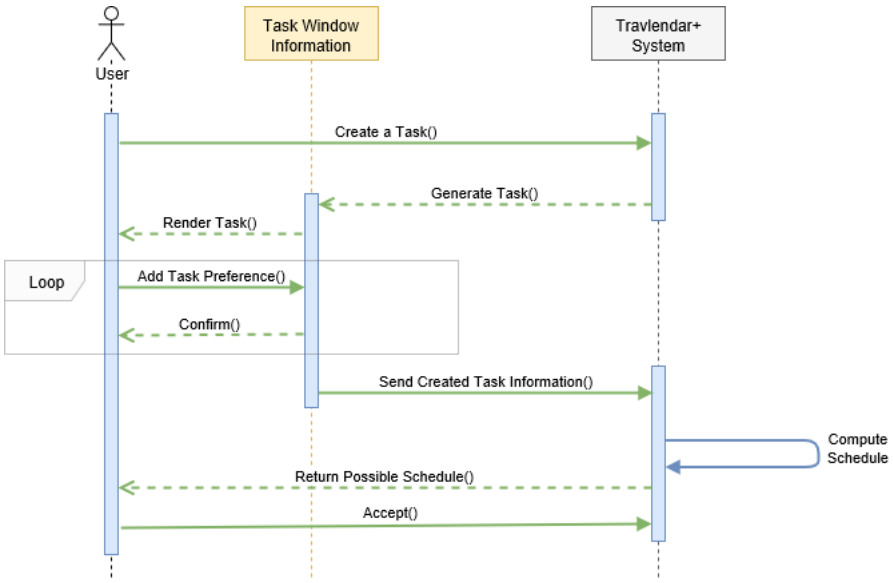

Figure 3.13: UML Sequence Diagram for the addition of a new task, *without overlapping*

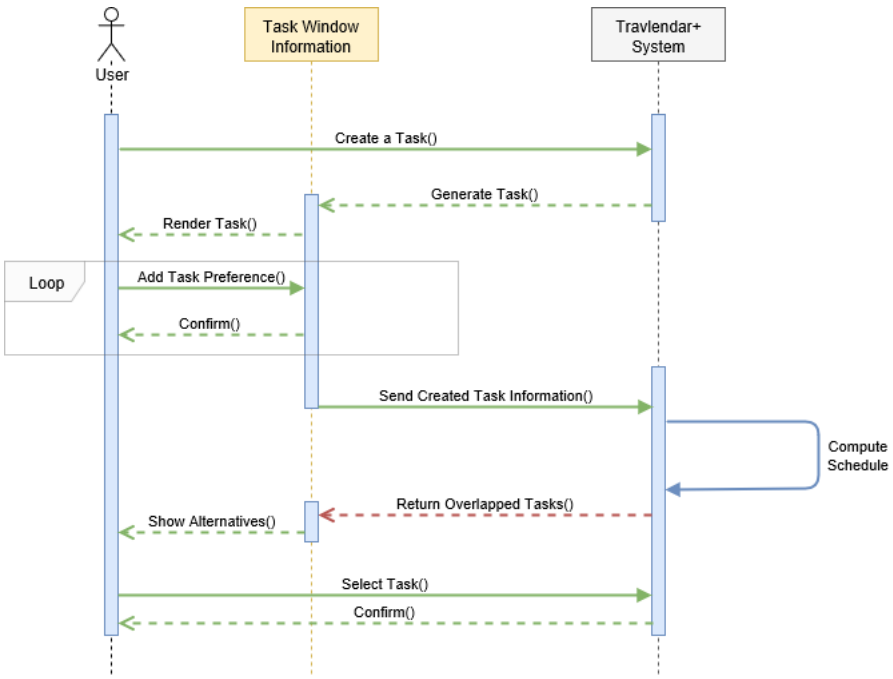

Figure 3.14: UML Sequence Diagram for the addition of a new task, *with overlapping*

3.3.3 Modify an existing task

It may happen that the user wants to modify an existing task since his travel mean preferences are changed or the appointment time or location are changed. It can also be the case that the user wants to delete a scheduled appointment. These features are described in this section. Notice that, an overlap may occur caused by the task modification, in this case the System will handle the situation as described in the Add a Task Use Case.

Actors	• User
Goals	[G 1]; [G 1.1]; [G 1.6]; [G 2]; [G 9]
Enter Condition	The user should be logged in the *Travlendar+* System
Events Flow	1. The user select the task he wants to change from his calendar (doing this he can also change the time scope of the calendar: daily, weekly, monthly time scope) 2. Now he can decide to modify some preferences, such as the time, the date, the location or the priority of the selected task. Moreover, the user can choose to cancel the entire task 3. The user confirm to apply the changes or discard them 4. The *Travlendar+* System computes a new schedule according to the new user preferences 5. Finally the user can choose from the new calendar or the older one
Exit Condition	The user has chosen which calendar he prefers: the newer or the older; So, from now on, the System will behave accordingly to the updated calendar
Exception	The modification of the task is not valid, for instance the user has inserted some wrong information about the task, then the user has to modify the task again. The modification of the task has created an overlap with another task, so the user has to choose which task he prefers to have in his calendar.

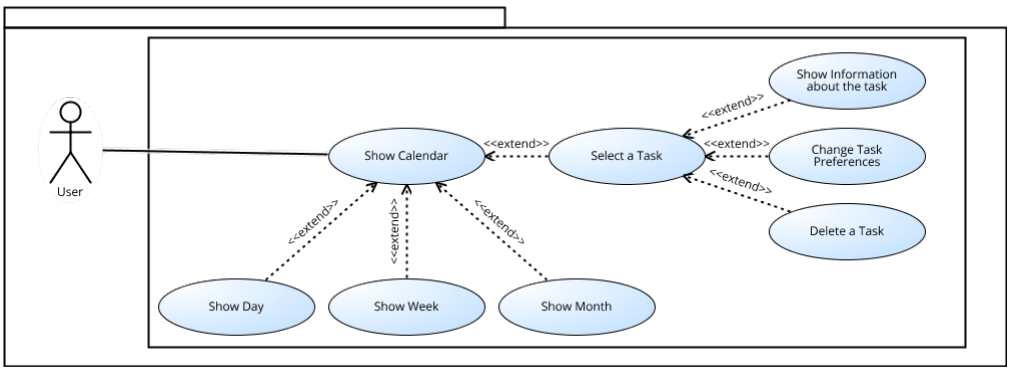

Figure 3.15: UML Use Case Diagram for the modification of a new task and the rescheduling of the calendar

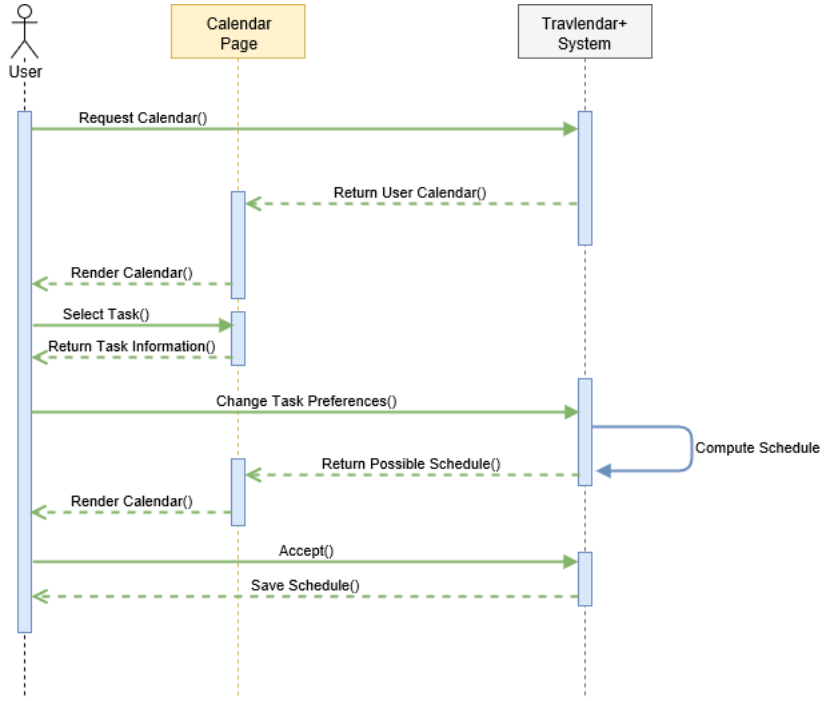

Figure 3.16: UML Sequence for the modification of a new task and the rescheduling of the calendar

3.3.4 Adding a break time

In the *Travlendar+* application if the user wants to have some free time he can create a break time. For instance, if the user wants to have lunch at 1 p.m. he can add to his calendar a break time task. This particular kind of task is scheduled according to the user's preferences as a normal task, with flexible time and periodicity but, without a specific location: we suppose that the user will have a break close to where he is. In addition, if the break time is scheduled close to lunch time, then the *Travlendar+* System will suggest the user a place where to eat.

Actors	• User
Goals	[G 1]; [G 1.2]; [G 1.1]; [G 2]; [G 9]
Enter Condition	The user should be logged in the *Travlendar+* system
Events Flow	1. The user presses the "Add a Task" button 2. The user select "Break Time" as task preferences 3. The user insert some additional information about the task, such as the time when he wants to do that break 4. The user select the repetition of that break time task 5. The user should approve or discard the task 6. Finally the system compute a schedule according the user's preferences and the break time just inserted
Exit Condition	The user either accept the proposed schedule or discard it
Exception	The break time just inserted is overlapped with other tasks yet inserted in the user's calendar, so the user should select another break time or cancel the other task

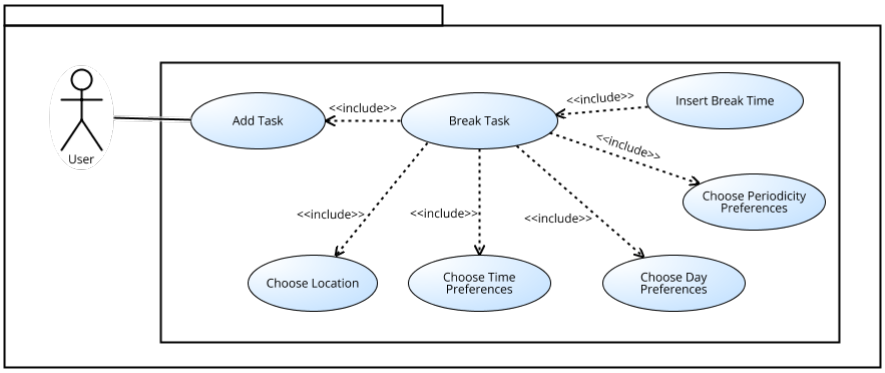

Figure 3.17: UML Use Case Diagram for the addition of a break time task

3.3.5 Show Transport Information about a Task

If the user wants more information about what kind of vehicles he needs to take to reach others tasks location, he can select a task from the calendar page and automatically more information about that task will be shown. For example, the trip indications and the travel means involved: if he needs to take a bus, a share-based vehicle or his own vehicle. Notice that, if the user take his own private vehicle, the application will schedule the day travels in such a way that allows the user to retake his own vehicle.

Actors	• User
Goals	[G 3]; [G 9]
Enter Condition	The user should be logged in the *Travlendar+* system and there is a calendar that is just scheduled
Events Flow	1. The user select a task from his calendar 2. The user presses the "Show Transport Information" button on the task menu 3. Now the user can choose from different kind of information: the vehicle he should take, the road trip he should do or the instructions to reach that location 4. Finally the user can look at the information he needs about transport systems
Exit Condition	The user closes the *Transport Information* screen and return to the *Calendar* page
Exception	

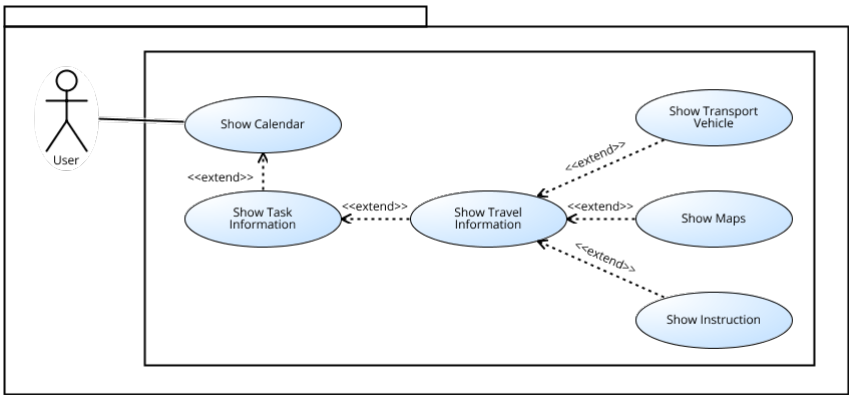

Figure 3.18: UML Use Case Diagram for the transport information about a task

3.3.6 Buying a vehicle-sharing or a public transport service

The user, for some tasks, maybe wants to take a bus or another public transport service. Furthermore, in Milan there are lots of alternatives to public transport, such as the car-sharing or the bike-sharing services, so the user might want to use one of these to reach a task location. For this reason, the *Travlendar+* System, not only support the user in the process of choosing and buying a possible rides from the public transport site or application, but also redirects the user to the right place where to book and rent a shared based travel mean service.

Actors	• User • Car-Sharing Service • Bike-Sharing Service • Public Transport Service
Goals	[G 3] [G 5]
Enter Condition	The user should be logged in the *Travlendar+* system and the calendar should be already scheduled
Events Flow	1. The user select a task 2. The user press the "Show Transport Information" button 3. The user select the type of transport service he wants to use 4. The user then selects if he wants to buy the ride
Exit Condition	The user should confirm the operation, then the system redirects he to the correct site of the selected service: Car-Sharing, Bike-Sharing or Public Transport
Exception	

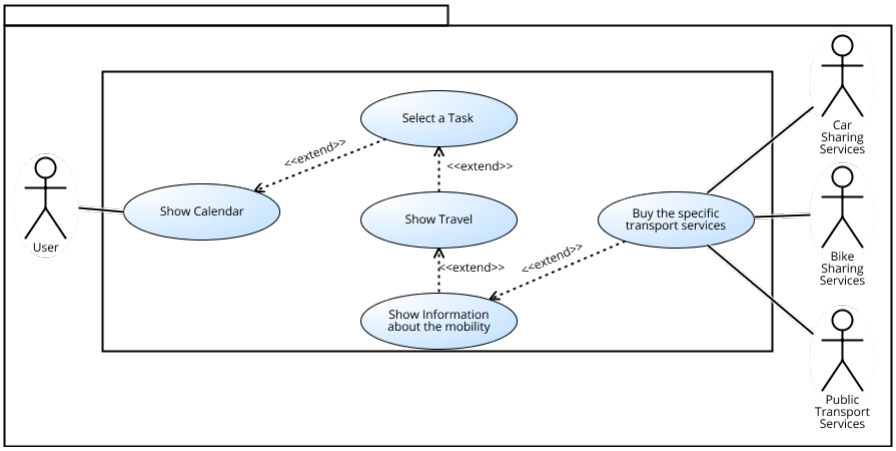

Figure 3.19: UML Use Case Diagram for buying services operation

3.3.7 Reminder System

A nice feature provided by the *Travlendar+* System is that it is possible to access the System either on traditional computer (through a web browser) or on a mobile phone (through the mobile application).

In both cases, the user may want to receive a notification that suggest him to start his travel, in order to reach the location of the next task. Moreover, it may happen that the user wants to take a vehicle-shared based service, but in the very moment he starts his trip, there isn't a vehicle available. To overcome these issues, the *Travlendar+* System sends a reminder to the user, and if the user wants to take a vehicle-shared based

service, our System, 30 minutes before the trip starts, will check if there are vehicles available, and suggest to the user to book that vehicle.

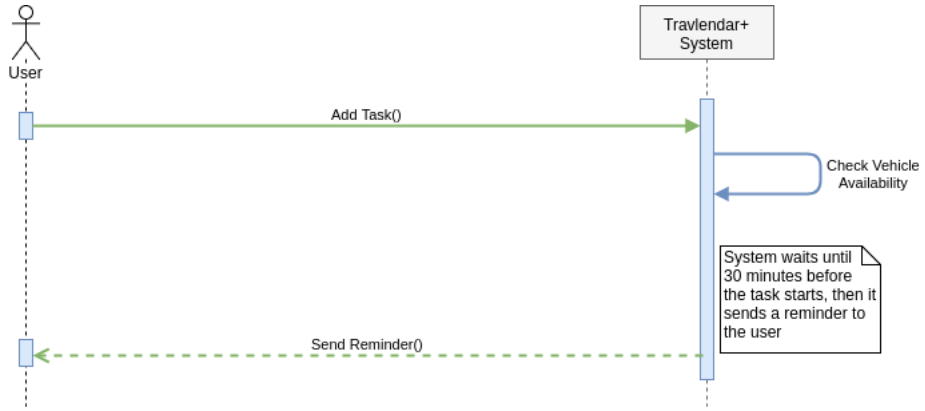

Figure 3.20: UML Sequence Diagram for the Reminder Notification System

3.3.8 Asking information about Public Transport Service

The *Travelndar+* System will interact with the Public Transport system, to suggest the best ride option in terms of money, time and considering the user agenda.

Several times public transport service is unavailable for many reasons, for instance an incoming strike or a maybe some metro line are stopped because of maintenance. Also in such situations the user wants to reach his appointments in time, and so the *Travlendar+* System should retrieve this information from the Public Transport system, to properly schedule the user's calendar.

Actors	• Public Transport System
Goals	[G 1]; [G 1.4]; [G 3]; [G 5.1]
Enter Condition	The *Travlendar+* system starts to schedule the user's calendar
Events Flow	1. The *Travlendar+* System asks mobility information or travel cards information to the *Public Transport* system through existing APIs 2. The *Public Transport* system sends back all the required information that the *Travlendar+* service has requested
Exit Condition	The *Travlendar+* system has requested all the mobility information it needed from the *Public Transport* service
Exception	It can be happened that either the *Public Transport* service is unavailable or an information request is rejected. In the former case it is impossible knowing useful information in order to schedule the user's calendar, so the *Travlendar+* system will compute a calendar without those information; otherwise in the latter case our system tries to ask again the information it needs

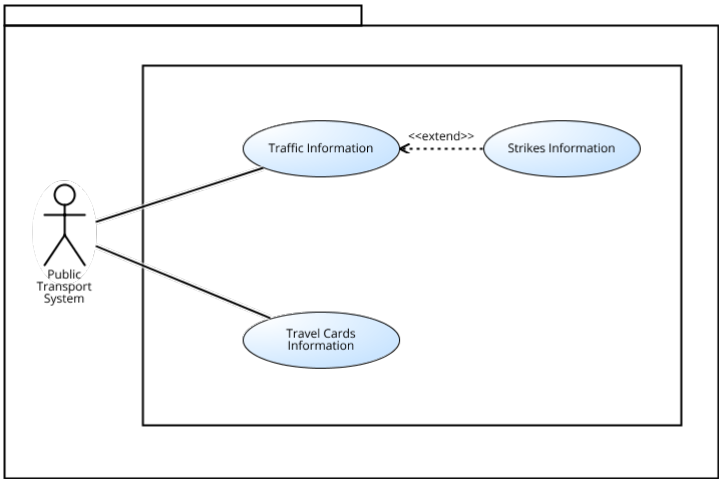

Figure 3.21: UML Use Case Diagram for the information requests asked to the *Public Transport* service

3.3.9 Using external Maps Services

In order to compute, as well as possible, the user's calendar, the *Travlendar+* service needs to know more information about the status of interested roads, the traffic information about the city and other useful information concerning the user's trip. To reach this goal the *Travlendar+* System needs to interact with an external maps service, retrieving these kind of information.

Actors	• External Maps Service
Goals	[G 1]; [G 1.4]; [G 1.5]; [G 3]; [G 7]
Enter Condition	The *Travlendar+* service is computing the user's calendar, so it needs to know more information about the user's trip
Events Flow	1. The *Travlendar+* system asks the maps information to the *External Maps* service through APIs 2. The *External Maps* service sends back the maps information 3. The *Travlendar+* now asks to the *External Maps* service to compute a trip 4. The *External Maps* service finally sends back the road trip just computed and more useful information about that journey
Exit Condition	The *Travlendar+* service knows all the information it needs in order to schedule as well as possible the user's calendar
Exception	It may be happened that the *External Maps* is unavailable for some reasons, in this case the *Travlendar+* system cannot compute the user's calendar, and our system tries to schedule it later with another request to the maps service

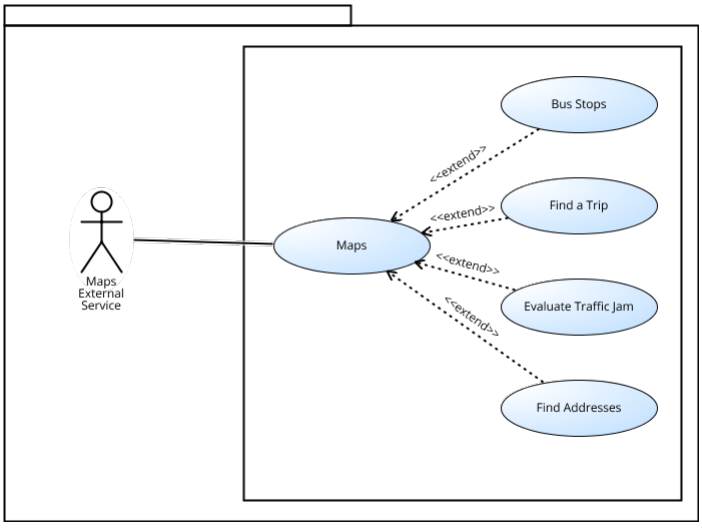

Figure 3.22: UML Use Case Diagram for the information requests about the maps

3.4 Performance Requirements

In this section we detail the Performance Requirements that we think are necessary for a better experience of our platform.

- From the very moment the System receives a re-scheduling request it should complete it within 5 seconds, at most.

3.5 Design Constraints

The Android version of *Travlendar+* will be designed to run under the Android operating system, version 4.0 or later. This because the 4.0 version of Android has already a great coverage through the users. The IOS version of the application will be designed to run under Apple IOS operating system, version 9.3.5 or later. Moreover, both the Android and the IOS version of *Travlendar+* requires an available Internet connection.

The *Travlendar+* application can be executed also on a web browser through a web interface of the software. In order to run *Travlendar+* on a browser, it has to support *HTML5*, *CSS3* and *Javascript*.

Finally the *Backend* system will have to run under modern UNIX systems, for example GNU/Linux systems or GNU/BSD systems. Indeed its easier to find people who can maintain a GNU based system.

3.6 Software System Attributes

In this Section we detail the attributes that our System must guarantee to be robust, scalable and secure.

3.6.1 Reliability

The system will have to be designed in order to be robust and fault tolerant. In particular, the system will have to manage either software or hardware modules failures without losing information.

3.6.2 Availability

The System must be available 24/7, exception made for maintenance interventions (both exceptional and routine ones). Among routine maintenance interventions we consider interventions due to: System updates or integration of new sharing based travel mean services, for instance.

3.6.3 Security

Since it will handle several private information, the System must guarantee a secure connection between Client and Server through the HTTPS protocol. Secondly, the System must properly handle and store the users data through a recognized standard, either for databases encryption and plain data encryption.

3.6.4 Maintainability

The System must provide a detailed documentation for each software module to guarantee an easy adaptability of the software to a new area and to easily integrate new travel mean services.

3.6.5 Portability

The software has to be able to work under different versions of the chosen work environment, either at back end and front end level.

Chapter 4

Formal Analysis

Here we present a possible formal model regarding the *Travlendar+* functionalities and context. This model has been written using Alloy, which is a bounded first logic prover that allows us to formalize a specific description of each entity used by the Software.

4.1 Signatures and Facts

Here we present all signatures composing our alloy model. We chose to include facts between signatures because it results much more simple to understand about what signature each FOL formula refers to, and so about what constraints each signature have. For this reason, we decided to comment each separate constraint, in order to gain readability compared to either uncommented code or separate fact modeling.

```
sig timeslot{
  start   : one Int,
  end : one Int
}{
    //no timeslots with negative starting or ending time, and
    // a timeslot must have end after his start.
start<=end and start > 0 and end > 0
}

//This signature represents a possible calendar task
sig task{
  time_slot : one timeslot,

//Task Preferences
    car : one Bool,
    bike : one Bool,
    atm : one Bool
}{
    //At least one preference from the task' preferences is true
    (car = true or bike = true or atm = true)
}
```

```
// This signature represents a possible travel between tasks
sig travel {

//it is mandatory to specify what are the two tasks being linked by the travel
  task_from : one task,
  task_to : one task,
  time_slot : one timeslot,

//Travel mean
//this must br unique, so only one of these can be true.
    car : one Bool,
    bike : one Bool,
    atm : one Bool

}{
    //Start and End time constraints on task_from and task_to
    task_from.time_slot.end  < task_to.time_slot.start

    and time_slot.start >= task_from.time_slot.end

    and time_slot.end <= task_to.time_slot.start

    //One and only one travel mean selected for a travel
    and XOR3[car, bike, atm]

}
```

This signature represent a possible day schedule produced by Travlendar+ note that this is not a good schedule, but only a possible one, so here are described only the most general constraints about how a schedule has to be

```
sig schedule{
    // this is a dummy task representing the schedule's starting point
    dummy_start_task : one task,

    // this is a dummy task representing the schedule's ending point
    dummy_end_task : one task,

    // the set of tasks complained by the schedule
    tasks : some task,

    // this task serves as a reminder to the user to retake his vehicle if
    // he leave it anywhere during the day's schedule
    retake_car_task : lone task,

    travels: some travel  }
```

```
{
    #travels = plus[#tasks, 1]

    (no t : tasks | t = dummy_start_task)

    (no t : tasks | t = dummy_end_task)

    (dummy_start_task = dummy_end_task implies #tasks = 0)

    ((some tr : travels |
                tr.task_from = dummy_start_task
                and tr.task_to = dummy_end_task ) implies
                                        #tasks = 0 )

    (all t : tasks | t.time_slot.start != t.time_slot.end)

    (all t : travels | t.time_slot.start != t.time_slot.end)

    // every task is not overlapped with other tasks
    (all u :tasks | no v :tasks |
            v != u and overlapped[v.time_slot, u.time_slot] )

    (all trav : travels | no task : tasks |
        overlapped[trav.time_slot, task.time_slot])


    (all trav : travels, u : travels |
    overlapped[trav.time_slot, u.time_slot] implies u = trav)

    (all t : tasks | some trav : travels |
        trav.task_from = t and not trav.task_to = t
        or  (
            not trav.task_from = t
            and trav.task_to = t ))

    (no t : tasks |
        t.time_slot.start <dummy_start_task.time_slot.start)

    (no u : tasks |
        u.time_slot.end > dummy_end_task.time_slot.start)

    (all tas : tasks + dummy_end_task |
        some trav : travels |
            trav.task_to = tas)
```

```
    (all tas : tasks + dummy_start_task |
        some trav : travels |
            trav.task_from = tas)

    ((no t : travels |
        (t.car = true and t.task_from = dummy_start_task) ) implies (
            (no t : travels |
                t.car = true
            )
            and (no t : task |
                t = retake_car_task
            )
        )
    )

    ((one t : travels |
        (t.car = true and t.task_from = dummy_start_task) ) implies
            (one tas : tasks |
                retake_car_task = tas and (one trav : travels |
                trav.task_from = tas and retaking_car[trav, this] )
            or (all t :travels | t.car = true)
        )
    )
```

Here's the formal model about the world where *Travlendar+* operates. This signature main goal is to constructively define the best schedule within a given set of tasks (all tasks have to be scheduled) and a given set of travels, among which the schedule has to select the better ones. Moreover, this signature contains information about weather through a set of weather slots, which represents weather conditions during a certain time slot. Eventually, this signature contains a preferences entity which is used to model user profiles, such as the "eco friendly profile".

```
sig World {
    // the best schedule given task preferences and weather conditions
    bestSchedule : one schedule,

    // tasks for this world. bestSchedule must contain every of these tasks
    tasks : some task,

    // possible travels connecting each task. bestSchedule chooses one travel
    // from each task.
    travs: some travel,

    wtrs: some weather_slot,

    // general preferences. there are only 2 profiles avaiable: "eco friendly"
    // and "only car"
    prefs: one preferences
```

```
} {
    // bestSchedule must contain every of these tasks
    tasks = bestSchedule.tasks

    (all tr1: bestSchedule.travels |
        one tr2 : travs |
            tr1 = tr2 )

    // Domain assumption : all tasks are don't overlap each other
    (all t1 : tasks, t2 : tasks |
        (t2 != t1) implies
            ( t1.time_slot != t2.time_slot and
            not overlapped[t1.time_slot, t2.time_slot])
    )

    // we use two predicates to check preferences and weather conditions.
    // see predicates section for further informations. here we check these
    // predicates for each travel in bestSchedule.
    (all tr1: bestSchedule.travels |
        (weatherCompliant[tr1 , this] and prefCompliant[tr1, prefs]))


    // here we assume that we want to chose the solution with as much as
    // short travels, given the task preferences. this behaviour is just for
    // demonstration purposes, as it can be modified by other preferences that
    // are hereby not modeled.
    (all tr : bestSchedule.travels |
        no tr2 : travs |
            ( tr != tr2 ) and (( tr.task_from = tr2.task_from )
            and (tr.task_to = tr2.task_to )
            and weatherCompliant[tr2 , this]
            and prefCompliant[tr2, prefs]
            and ( minus[tr.time_slot.end, tr.time_slot.start] >
                minus[tr2.time_slot.end, tr2.time_slot.start] ))
    )
}
```

Eventually, here we have some other support signatures; in particular, we defined weather slots, temperatures (which are contained only in weather slots) and eventually some signatures representing useful boolean types that we use to check the truth value of some flags contained in the signatures discussed above.

```
sig weather_slot{
    time_slot : one timeslot,
    temperature : one temp,
rain : one Bool
}{
}
```

```
sig temp {
    hot : lone Bool,
    cold : lone Bool,
    good :lone Bool
}{
    (not XOR[hot, cold]) implies good = true
}

abstract sig Bool {}

one sig true extends Bool {}
one sig false extends Bool {}
```

4.2 Support predicates

in this section we discuss about the predicates that had been used in order to model the facts presented in the previous section. We decided to use predicates whenever there was a too complicated axiom, in order to improve readability. Moreover, we decided to use Predicates to describe widely used conditions, thus improving code reuse and reducing the model's complexity.

```
//XOR predicate
pred XOR[ x1 : Bool, x2 : Bool] {
    (x1 = false and x2 = true) or (x1 = false and x2 = true)
}

//XOR predicate on three variables, true iif
//only one of the passed var is true
pred XOR3[ x1 : Bool, x2 : Bool, x3 : Bool] {

    (x1 = true and x2 = false and x3 = false)

    or (x1 = false and x2 = true and x3 = false)

    or (x1 = false and x2 = false and x3 = true)
}

Pred prefCompliant[trav: travel, p : preferences]{

    ( trav.bike = false and trav.atm = false) implies p.ecoFriendly = false

    trav.car = false implies p.oilFriendly = false

    (trav.task_to.car = false implies trav.bike = true)
```

```
    //use bike when user prefers only car
    and ( trav.task_to.atm = false  implies trav.atm = false)

    //use bike when user prefers only atm
    and ( trav.task_to.bike = false implies trav.bike = false)
}

//not use bike when is cold, hot or raining
pred weatherCompliant[trav : travel, w : World]{

    ((some w : w.wtrs |
        (w.temperature.good = false or w.rain = true)
        and overlapped[w.time_slot, trav.time_slot] ))
    implies (
        trav.bike = false)

    ((some w : w.wtrs |
        w.temperature.hot = true
        and overlapped[w.time_slot, trav.time_slot] )) implies
    (trav.atm = false)
}

pred overlapped[t,u : timeslot]{

    u.start = t.start or

    (u.start > t.start implies u.start < t.end) and

    (t.start > u.start implies t.start < u.end) or eq[t,u]
}

pred eq[t, u: timeslot]{

    (t.start = u.start and t.end = u.end) or t=u
}

// support predicate used to check if the user's car is bringed back home when
// used.
pred retaking_car[trav : travel, s : schedule] {

    trav.car = true and (
    (some tr : s.travels |
        (tr.task_from = trav.task_to and retaking_car[tr, s] ))
        // there's only one travel starting from each task
or trav.task_to = s.dummy_end_task )

}
```

4.3 Runnable predicates and assertions

In this section we include all predicates that can be run in order to check whether signatures and axioms are consistent. using the world visualizer included in Alloy IDE, it is possible to verify that the model fits the Application requirements, constraints and Domain assumptions. In particular, the two "show" predicates are useful to check that world and schedule signatures are realistic. Eventually, a small collection of assertions is used to check the deductibility of some formulas from the axioms presented in the above sections.

```
pred showSchedule[s: schedule]{

#s.tasks > 1
}
pred addTask[wr0 : World,  task0 : task,  wr1 : World]{

    //preconditions
    task0 != wr0.bestSchedule.dummy_start_task

    task0 != wr0.bestSchedule.dummy_end_task

    no t : wr0.tasks | overlapped[task0.time_slot,t.time_slot]

    //postconditions
    wr1.tasks = wr0.tasks + task0

    some tr, tr2 : wr1.travs |
        (tr.task_from = task0 and  tr2.task_to = task0)

    some tr, tr2 : wr1.bestSchedule.travels |
        (tr.task_from = task0 and  tr2.task_to = task0)
}

pred removeTask[ wr0 : World,  task0 : task,  wr1 : World]{

    //preconditions
    task0 != wr0.bestSchedule.dummy_start_task

    task0 != wr0.bestSchedule.dummy_end_task

    one t : wr0.tasks |
   task0 = t

    //postconditions
    wr1.tasks = wr0.tasks - task0

}
```

```
pred changeTaskPrefs[wr0 : World, task0 : task, task1 : task, wr1 : World]{

    //preconditions
    task0.time_slot = task1.time_slot and task1 != task0

    //postconditions
    (one wrTemp : World |
        (removeTask[wr0, task0, wrTemp] and
        addTask[wrTemp, task1, wr1]))

}

assert contained{
    no s : schedule | (
        bigSchedule[s] and some x : s.tasks, y : s.tasks | (
            x.time_slot.start > y.time_slot.start
            and x.time_slot.end < y.time_slot.end
            )
        )
}

assert continuous_sched {
    all s : schedule | (
            bigSchedule[s] implies
            one t : s.travels |
            t.task_to = s.dummy_end_task and continuous_pt2[t, s]
)
}

//NOTE: these two predicates are needed to overcome the limitate recursive power
//of Alloy prover. in particular, the maximum recuursion depth is 4,
//but there can be schedules with more than 4 travels

pred continuous_pt2[t : travel, s :schedule]{
    t.task_from = s.dummy_start_task
    or (some t2 : s.travels |
        t.task_from = t2.task_to and continuous_pt3[t2, s])
}

pred continuous_pt3[t : travel, s :schedule]{
    t.task_from = s.dummy_start_task
    or (some t2 : s.travels |
        t.task_from = t2.task_to and continuous_pt2[t2, s] )

}
```

4.4 Proof of Consistency and Assertion checking

here we present the result given by Alloy prover after running the above cited predicates and assertion. in this section we mainly omit the graphs given by the prover as result, because of their excessive reading complexity, given a reasonable number of instances for each signature.

4.4.1 Changing task preferences

```
run changeTaskPrefs for
    3 World,
    3 schedule,
    3 weather_slot,
    3 temp,
    exactly 15 timeslot,
    exactly 5 task,
    exactly 8 travel,
    exactly 1 preferences
```

```
Executing "Run changeTaskPrefs for 3 World, 3 schedule, 3 weather_slot, 3 temp, exactly 15 timeslot, exactly 5 task, exactly 8 travel, exactly 1 preferences"
  Sig this/World scope <= 3
  Sig this/schedule scope <= 3
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/timeslot scope <= 15
  Sig this/task scope <= 5
  Sig this/travel scope <= 8
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot == [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6], [timeslot$7], [timeslot$8], [timeslot$9], [timeslot$10
  Sig this/task == [[task$0], [task$1], [task$2], [task$3], [task$4]]
  Sig this/travel == [[travel$0], [travel$1], [travel$2], [travel$3], [travel$4], [travel$5], [travel$6], [travel$7]]
  Sig this/schedule in [[schedule$0], [schedule$1], [schedule$2]]
  Sig this/World in [[World$0], [World$1], [World$2]]
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences == [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  67261 vars. 1087 primary vars. 189919 clauses. 6104ms.
  Instance found. Predicate is consistent. 8615ms.
```

Figure 4.1: Alloy Result for changeTaskPrefs. Instance is omitted, but consistent with the non formal requirements

4.4.2 Showing world

```
run showWorld for
    1 World,
    1 schedule,
    3 weather_slot,
    3 temp,
    exactly 15 timeslot,
    exactly 4 task,
```

```
    exactly 8 travel,
    exactly 1 preferences
```

```
Executing "Run showWorld for 1 World, 1 schedule, 3 weather_slot, 3 temp, exactly 15 timeslot, exactly 4 task, exactly 8 travel, exactly 1 preferences"
  Sig this/World scope <= 1
  Sig this/schedule scope <= 1
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/timeslot scope <= 15
  Sig this/task scope <= 4
  Sig this/travel scope <= 8
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot == [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6], [timeslot$7], [timeslot$8], [timeslot$9], [timeslot$10],
  Sig this/task == [[task$0], [task$1], [task$2], [task$3]]
  Sig this/travel == [[travel$0], [travel$1], [travel$2], [travel$3], [travel$4], [travel$5], [travel$6], [travel$7]]
  Sig this/schedule in [[schedule$0]]
  Sig this/World in [[World$0]]
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences == [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  53549 vars. 944 primary vars. 151006 clauses. 1585ms.
  Instance found. Predicate is consistent. 1499ms.
```

Figure 4.2: Alloy Result for showWorld. Instance is omitted, but consistent with the non formal requirements

4.4.3 Remove a task

```
run removeTask for
    2 World,
    2 schedule,
    3 weather_slot,
    3 temp,
    exactly 15 timeslot,
    exactly 10 task,
    exactly 12 travel,
    exactly 1 preferences
```

```
Executing "Run removeTask for 2 World, 2 schedule, 3 weather_slot, 3 temp, exactly 15 timeslot, exactly 10 task, exactly 12 travel, exactly 1 preferences"
  Sig this/World scope <= 2
  Sig this/schedule scope <= 2
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/timeslot scope <= 15
  Sig this/task scope <= 10
  Sig this/travel scope <= 12
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot == [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6], [timeslot$7], [timeslot$8], [timeslot$9], [timeslot$10],
  Sig this/task == [[task$0], [task$1], [task$2], [task$3], [task$4], [task$5], [task$6], [task$7], [task$8], [task$9]]
  Sig this/travel == [[travel$0], [travel$1], [travel$2], [travel$3], [travel$4], [travel$5], [travel$6], [travel$7], [travel$8], [travel$9], [travel$10], [travel$11]]
  Sig this/schedule in [[schedule$0], [schedule$1]]
  Sig this/World in [[World$0], [World$1]]
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences == [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  122086 vars. 1448 primary vars. 325405 clauses. 67855ms.
  Instance found. Predicate is consistent. 17824ms.
```

Figure 4.3: Alloy Result for removeTask. Instance is omitted, but consistent with the non formal requirements

4.4.4 Adding a task

```
run addTask for
    2 World,
    2 schedule,
    3 weather_slot,
    3 temp,
    exactly 15 timeslot,
    exactly 10 task,
    exactly 12 travel,
    exactly 1 preferences
```

```
Executing "Run addTask for 2 World, 2 schedule, 3 weather_slot, 3 temp, exactly 15 timeslot, exactly 10 task, exactly 12 travel, exactly 1 preferences"
  Sig this/World scope <= 2
  Sig this/schedule scope <= 2
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/timeslot scope <= 15
  Sig this/task scope <= 10
  Sig this/travel scope <= 12
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot == [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6], [timeslot$7], [timeslot$8], [timeslot$9], [timeslot$10],
  Sig this/task == [[task$0], [task$1], [task$2], [task$3], [task$4], [task$5], [task$6], [task$7], [task$8], [task$9]]
  Sig this/travel == [[travel$0], [travel$1], [travel$2], [travel$3], [travel$4], [travel$5], [travel$6], [travel$7], [travel$8], [travel$9], [travel$10], [travel$11]]
  Sig this/schedule in [[schedule$0], [schedule$1]]
  Sig this/World in [[World$0], [World$1]]
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences == [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  125995 vars. 1496 primary vars. 335286 clauses. 64899ms.
  Instance found. Predicate is consistent. 15750ms.
```

Figure 4.4: Alloy Result for addTask. Instance is omitted, but consistent with the non formal requirements

```
run showSchedule for
    1 schedule,
    4 task,
    3 travel,
    7 timeslot,
    0 World
```

```
Executing "Run showSchedule for 1 schedule, 4 task, 3 travel, 7 timeslot, 0 World, 3 weather_slot, 3 temp, 1 preferences"
  Sig this/schedule scope <= 1
  Sig this/task scope <= 4
  Sig this/travel scope <= 3
  Sig this/timeslot scope <= 7
  Sig this/World scope <= 0
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot in [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6]]
  Sig this/task in [[task$0], [task$1], [task$2], [task$3]]
  Sig this/travel in [[travel$0], [travel$1], [travel$2]]
  Sig this/schedule in [[schedule$0]]
  Sig this/World in []
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences in [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  15901 vars. 439 primary vars. 41485 clauses. 309ms.
  Instance found. Predicate is consistent. 1241ms.
```

Figure 4.5: Alloy Result for showSchedule. Instance is omitted, but consistent with the non formal requirements

4.4.5 Checking if a schedule is connected

```
check continuous_sched for
    1 schedule,
    4 task,
    3 travel,
    7 timeslot
```

```
Executing "Check continuous_sched for 1 schedule, 4 task, 3 travel, 7 timeslot, 0 World, 3 weather_slot, 3 temp, 1 preferences"
  Sig this/schedule scope <= 1
  Sig this/task scope <= 4
  Sig this/travel scope <= 3
  Sig this/timeslot scope <= 7
  Sig this/World scope <= 0
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot in [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6]]
  Sig this/task in [[task$0], [task$1], [task$2], [task$3]]
  Sig this/travel in [[travel$0], [travel$1], [travel$2]]
  Sig this/schedule in [[schedule$0]]
  Sig this/World in []
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences in [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  16158 vars. 439 primary vars. 42295 clauses. 536ms.
  No counterexample found. Assertion may be valid. 33ms.
```

Figure 4.6: Alloy Result for continuous$_s ched. Instance is omitted, but consistent with the non formal requirem

4.4.6 Checking tasks not overlapped

```
check contained for
    1 schedule,
    4 task,
    3 travel,
    7 timeslot
```

```
Executing "Check contained for 1 schedule, 4 task, 3 travel, 7 timeslot, 0 World, 3 weather_slot, 3 temp, 1 preferences"
  Sig this/schedule scope <= 1
  Sig this/task scope <= 4
  Sig this/travel scope <= 3
  Sig this/timeslot scope <= 7
  Sig this/World scope <= 0
  Sig this/weather_slot scope <= 3
  Sig this/temp scope <= 3
  Sig this/preferences scope <= 1
  Sig logic/true scope <= 1
  Sig logic/false scope <= 1
  Sig logic/Bool scope <= 2
  Sig this/timeslot in [[timeslot$0], [timeslot$1], [timeslot$2], [timeslot$3], [timeslot$4], [timeslot$5], [timeslot$6]]
  Sig this/task in [[task$0], [task$1], [task$2], [task$3]]
  Sig this/travel in [[travel$0], [travel$1], [travel$2]]
  Sig this/schedule in [[schedule$0]]
  Sig this/World in []
  Sig this/weather_slot in [[weather_slot$0], [weather_slot$1], [weather_slot$2]]
  Sig this/temp in [[temp$0], [temp$1], [temp$2]]
  Sig this/preferences in [[preferences$0]]
  Sig logic/Bool in [[logic/true$0], [logic/false$0]]
  Sig logic/true == [[logic/true$0]]
  Sig logic/false == [[logic/false$0]]
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  17077 vars. 447 primary vars. 45310 clauses. 100ms.
  No counterexample found. Assertion may be valid. 907ms.
```

Figure 4.7: Alloy Result for contained. Instance is omitted, but consistent with the non formal requirements

Chapter 5

Effort Spent

This section of the document describes all hours of work for each team mate.

- Alberto Floris:
  - [6/10/2017] Team Reunion: Work Organization (1.5h)
  - [10/10/2017] Team Reunion: World Analysis Discussion (3.5h)
  - [13/10/2017] Team Reunion: Glossary and Domain Assumption sketch (1.5h)
  - [17/10/2017] Team Reunion: Requirement Analysis Discussion (3.5h)
  - [20/10/2017] Alloy Design (1.5h)
  - [21/10/2017] Team Reunion: Product Perspective, Functional and Non-Functional Requirements discussion (5h)
  - [22/10/2017] Team Reunion: Sequence Diagrams, E/R Diagram, Specific Requirements review (6h)
  - [23/10/2017] Alloy sketch (1.5h)
  - [24/10/2017] Alloy Implementation (3.5h)
  - [26/10/2017] Alloy Implementation (1h)
  - [27/10/2017] Alloy Testing and Debugging (5h)
  - [28/10/2017] Team Reunion: Alloy advanced model, Use Case Review (8h)
  - [29/10/2017] Team Reunion: Project Review, Consistency check (9h)
- Claudio Montanari:
  - [6/10/2017] Team Reunion: Work Organization (1.5h)
  - [10/10/2017] Team Reunion: World Analysis Discussion (3.5h)
  - [13/10/2017] Team Reunion: Glossary and Domain Assumption sketch (1.5h)
  - [17/10/2017] Team Reunion: Requirement Analysis Discussion (3.5h)
  - [20/10/2017] Alloy Design and first implementation (1.5h)
  - [21/10/2017] Team Reunion: Product Perspective, Functional and Non-Functional Requirements discussion (5h)
  - [22/10/2017] Team Reunion: Sequence Diagrams, E/R Diagram, Specific Requirements review (6h)

- [24/10/2017] Alloy Implementation, Use Case Review (3h)
- [26/10/2017] Alloy Implementation (1h)
- [27/10/2017] Use Case Review, Sequence Diagrams review, Goal and Requirement Review (4h)
- [28/10/2017] Team Reunion: Alloy advanced model, Use Case Review (8h)
- [29/10/2017] Team Reunion: Project Review, Consistency check (9h)

- Luca Napoletano:
  - [6/10/2017] Team Reunion: Work Organization (1.5h)
  - [10/10/2017] Team Reunion: World Analysis Discussion (3.5h)
  - [13/10/2017] Team Reunion: Glossary and Domain Assumption sketch (1.5h)
  - [17/10/2017] Team Reunion: Requirement Analysis Discussion (3.5h)
  - [20/10/2017] Alloy Design (1.5h)
  - [21/10/2017] Team Reunion: Product Perspective, Functional and Non-Functional Requirements discussion (5h)
  - [22/10/2017] Team Reunion: Sequence Diagrams, E/R Diagram, Specific Requirements review (6h)
  - [24/10/2017] UML Use Case Design, UML Sequence Diagram Design (3.5h)
  - [25/10/2017] UML Use Case Review, UML Sequence Diagram Review (2.5h)
  - [26/10/2017] Alloy Review (1.5h)
  - [27/10/2017] Mock ups Design (4h)
  - [28/10/2017] Team Reunion: Alloy advanced model, Use Case Review (8h)
  - [29/10/2017] Team Reunion: Project Review, Consistency check (9h)

5.1 GANTT Analysis

Here we attached to the document our *GANTT* Diagram with the corresponding table. This diagram was used to schedule as well as possible our work time, and we also used it to know at which point of the project we were.

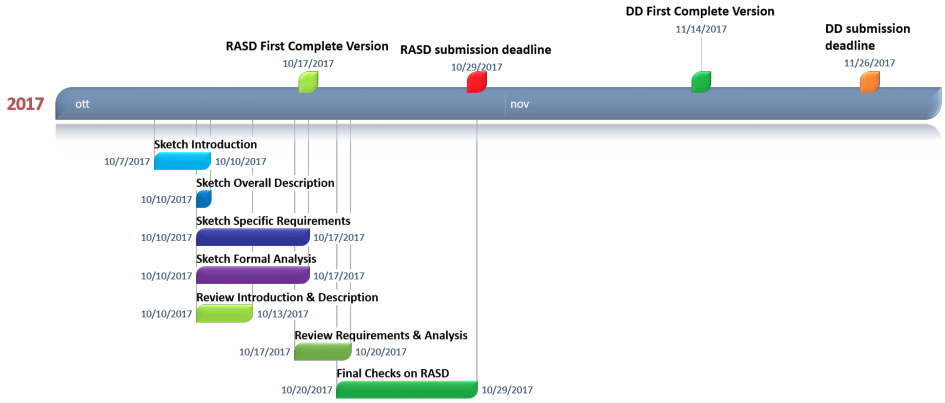

Figure 5.1: *GANTT* Diagram for *Travlendar+* Project

Timeline: 10/07/2017 - 11/26/2017

Milestone(s)	
Date	**Description**
10/17/2017	RASD First Complete Version
10/29/2017	RASD submission deadline
11/14/2017	DD First Complete Version
11/26/2017	DD submission deadline

Task(s)			
Duration (days)	**Start Date**	**End Date**	**Description**
4	10/07/2017	10/10/2017	Sketch Introduction
1	10/10/2017	10/10/2017	Sketch Overall Description
8	10/10/2017	10/17/2017	Sketch Specific Requirements
8	10/10/2017	10/17/2017	Sketch Formal Analysis
4	10/10/2017	10/13/2017	Review Introduction & Description
4	10/17/2017	10/20/2017	Review Requirements & Analysis
10	10/20/2017	10/29/2017	Final Checks on RASD

Figure 5.2: *GANTT* Table for *Travlendar+* Project