POLITECNICO
MILANO 1863

Travlendar+

Requirements Analysis & Specification Document

Alessandro Saverio Paticchio – 894092
Andrea Tricarico – 898406
Davide Santambrogio - 900204

Table of Contents

1. Introduction

a. Purpose

This document is the Requirement Analysis and Specification Document of the project *Travlendar+*, a calendar-based application that schedules users' appointments and support them in arranging travels.

The present is meant to be a guide-line for the implementation of the application, since it contains a deep analysis of the requirements, the goals and the domain of the environment surrounding the system and the users.

The analysis is carried out through the use of natural language, whose drawbacks and inconsistency will be amended by formalisms like UML and Alloy.

The application is meant to satisfy the following goals:

G1. Schedule and keep track of the user's appointments along the days.

G2. Notify the user of incoming appointments, with details about starting time and mean of transportation.

G3. Propose a suitable itinerary throughout the appointments' locations, according to user's preferences, appointment description and external information about public transportation, weather forecast, traffic conditions.

G4. Guide the user through all the appointments' locations as a navigator.

- G.4.1. React to real-time unexpected events, such as weather changes, car incidents and possible delays too.
- G.4.2. Edit the itinerary basing it on the real current position.
- G.4.3. Let the user be in time at all of its appointments.

G5. Give the possibility to buy a transportation ticket.

G6. Give the possibility to reserve a vehicle-sharing service.

b. Scope

The main goal of this application is to support people in arranging their days. It provides a calendar that can be filled by users with all their appointments, specifying time, estimated duration, location, type.

Once the user has inserted all this information, and according to its preferences, the system checks if it is compatible with others commitments and if a correct scheduling is feasible.

In addition to the data inserted by the user, the system will take into account many other variables (such as weather forecast, strikes, availability of private/public transportation) to compute the most efficient and suitable approach to every meeting, proposing a starting time, an estimated arrival time and the alternatives for travelling.

This application goes deep into the organization of the trip, indeed it also permits to buy a ticket for public transportation or to locate and reserve the nearest shared-vehicle.

By analyzing the problem the following phenomena are classified:

World phenomena	Shared phenomena	Machine phenomena
The user plans an appointment	The user creates an account	Most suitable path computation
The user starts its trip	The user edits its settings	Appointments consistency check
Meeting delay	The user creates a new appointment	Checking whether a ticket must be purchased
The user pays for vehicle sharing / ticket	The user edits an appointment	Appointment a = new Appointment (loc, start, end, type)
Weather conditions changing during the trip	The user cancels an existing appointment	The system detects the user's position through device's locating functions.
The user reaches its destination	The system proposes itineraries	Redirection to PayPal service
Unexpected event occurs during the trip	The user chooses an itinerary	Retrieve weather forecast information
The route is busy.	The system notifies the user if an appointment is going to be held	Retrieve the map of the area
The user reserves a vehicle-sharing service.	The user starts the navigation function of the application.	Retrieve public transportation schedule
A strike that involves public transportation services is announced	The user purchases public transportation ticket	
There is traffic jam along the itinerary	The system displays a warning for a meeting located in an unreachable place in the allotted time, or if overlaps with other appointments.	
	Traffic condition along the itinerary update through an external service (such as Google Maps)	
	The app displays all the vehicle-sharing services nearby	

c. Definitions, Acronyms, Abbreviations

Appointment: an entity that defines a period of time devoted to a user's activity.

Itinerary/Ride: Travel between two appointments, it is defined by starting time, ETA, vehicle, total cost.

Schedule: entity that represents the daily schedule of the user, each instance is composed by all the inserted appointments of that day.

Draft appointment: an appointment that is not yet inserted in the schedule, but whose details have already been defined by the user.

Inconsistency: A schedule is inconsistent if two appointments overlap or other variables make the appointment difficult or even impossible to reach (weather conditions, traffic, strikes, ...)

Priority: it is an integer number between 1 and 5. If the user does not insert a priority, the appointment gains the default value of its type.
The priority influences the scheduling in terms of assurance to be in time at the appointment's location: the higher is the priority more are the minutes in advance rather than the standard arrival time.

Priority	Minutes in advance
5	30
4	20
3	15
2	10
1	0

Here there are the vehicles that will be inserted in the application, with relatives avoided vehicles and default priorities.

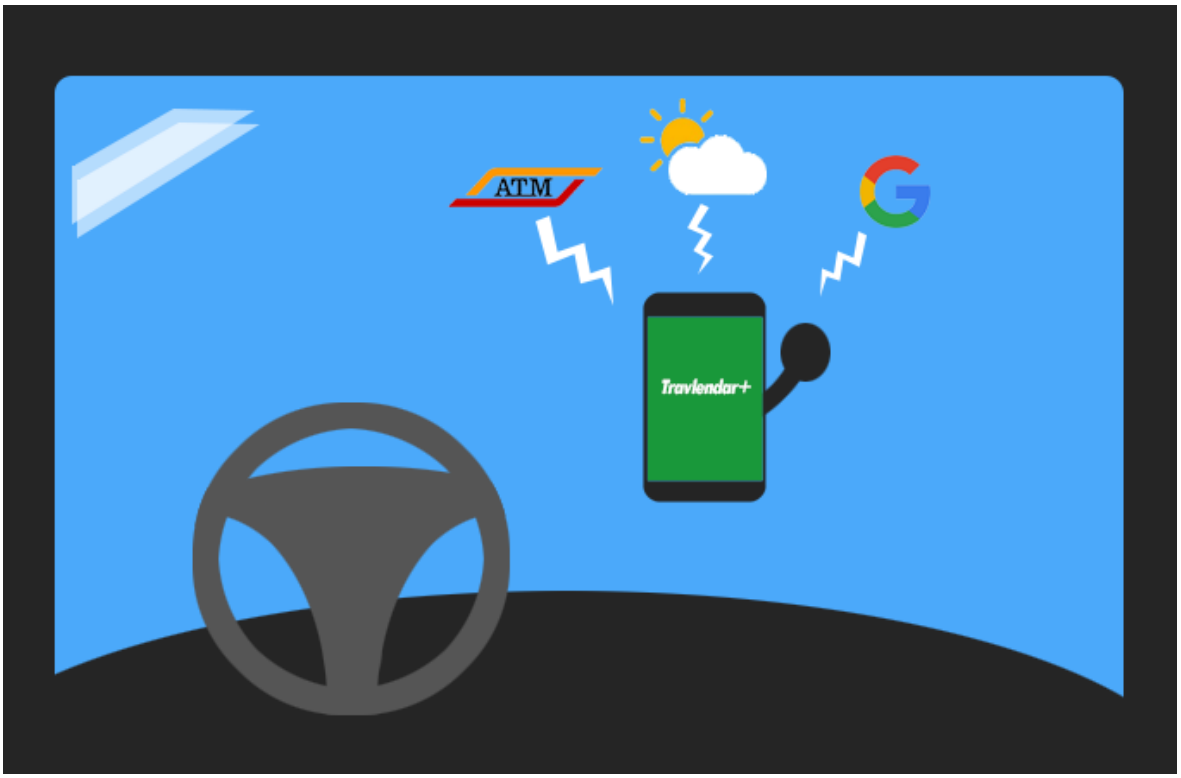
Type	Avoided vehicle	Default Priority
generic appointment	none	2
business meeting	bike, foot	5
party	car, motorbike	1
family appointment	public transportation, motorbike	2
date	bike, public transportation	4
office/school time	none	3
gym/workout	none	1

e. Reference Documents

- IEEE RASD standard document;
- Mandatory Project Assignment for Software Engineering 2

2. OVERALL DESCRIPTION

a. Product perspective



Example of usage

The following paragraph is a general and high-level description of the interaction between user and system.
In fact, its aim is to provide a mapping between the shared-phenomena (*italic*) highlighted in the previous paragraph and the application's functionalities, in order to let the reader understand how they are faced by the system to develop.

The user creates an account

The first time the user opens the app it must register, providing all the necessary details that will be processed by the system to accomplish his tasks.
The necessary information include username (unique for each user), password and email.
The next times the user will be able to access using its credentials or to recover its password using its email address.

The user edits its settings

In order to fit user's needs, the system allows a registered user to edit his account preferences in terms of: personal vehicles, public transportation passes, vehicle-sharing feasible to reserve, favourite type of transportation services, daily breaks and other useful information supporting the computation of the itinerary (such as maximum distance by foot, time preferences for the usage of particular transports, etc...).

The user creates a new appointment

Once the user has initialized the application, it is allowed to create appointments and to fill its daily schedule. For each new appointment the user is asked to fill a form in order to give a description of the event, that will also be processed by the application to efficiently plan the daily scheduling.

The app displays a warning for an appointment located in an unreachable place in the allotted time, or if overlaps with other appointments

At the end of the creation of each appointment the system checks the consistency of the daily schedule to verify whether it would be consistent after the insertion of the draft-appointment, if it was not then the application would show a warning to the user.

In order to perform this consistency checking, the system scans all the already inserted appointments and verifies whether they overlap and/or there is enough time between two adjacent events to move from one to another (also considering the lunch and further breaks).

The system proposes itineraries

Once the user fills all the information about a new appointment the system saves it as "draft". Then the system computes 5 itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance) to reach the appointment's location starting from the user's home (if it is the first appointment of the day) or from the previous appointment's location. The system must consider fundamentals factors such as the most suitable mean of transportation, weather forecast and all the user's preferences.

Furthermore, the application will also take into account the type of appointment.

The user chooses a system's itinerary.

The user can both accept one of the system's proposals or not, in the second case the appointment will be eliminated.

The user edits an appointment

In every moment, the user is able to edit an already filled appointment.

After the modification, the system will propose a new itinerary for the selected appointment.

The user cancels an existing appointment

In every moment, the user is able to delete an already filled appointment. After the modification, the system will propose a new itinerary for the next appointment.

The system notifies the user if an appointment is going to be held

In a given day, the system will notify the user with its appointments, suggesting to start its travel at a certain time with a specific mean of transportation.

The user starts the navigation function of the application

By starting your itinerary, the application will guide you as a navigator.
Actually, to better fit real-time conditions, when the application is eventually opened, a new and more accurate itinerary can be proposed, depending on the current position, possible shared-vehicles nearby, traffic conditions and weather information.

During the travel, if an unscheduled event affects the itinerary (for instance, unexpected traffic jam or weather change), the system will collect the foreseen delay and will propose a feasible solution to fix the track, if there is one.

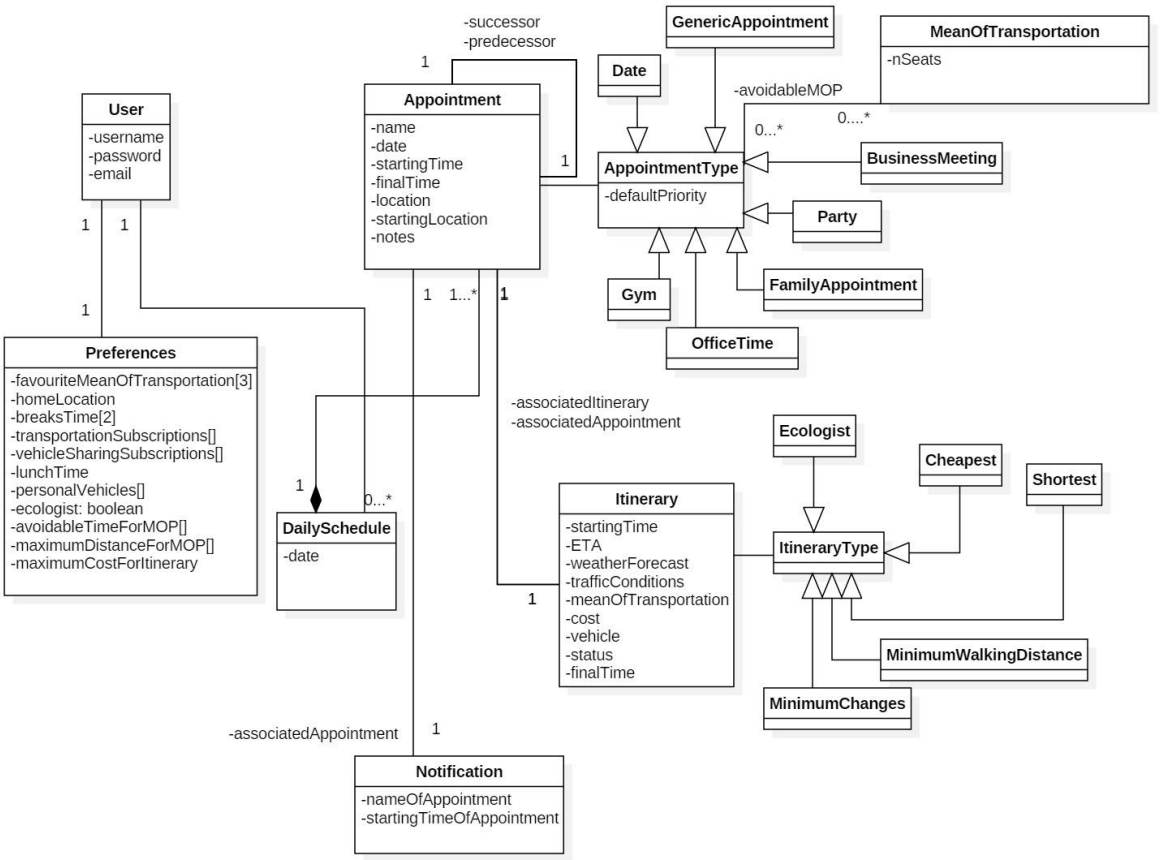
The user purchases public transportation ticket

Each time a public transportation is foreseen, the application notifies that a ticket is needed, and the user has the possibility to purchase one, unless it already has a pass.

The app detects and displays all the vehicle-sharing services nearby

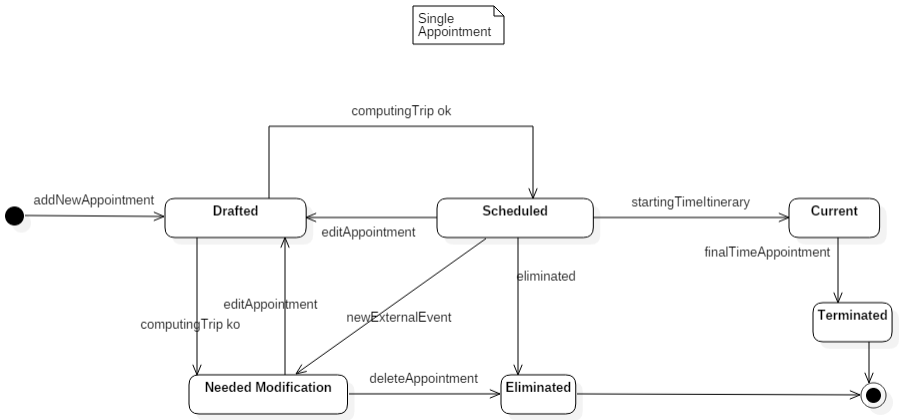
On the other hand, when the travel is going to start, the application will look for possible vehicle-sharing services in the neighbours of the user's location: if there is one (and it is convenient for the schedule), it is signaled on the map. The user, by clicking on it, can open the relative app to reserve a vehicle.

In order to let the developers better understand our view of the application, the following overall **UML** will be a guide-line (further details will be given in next pages).

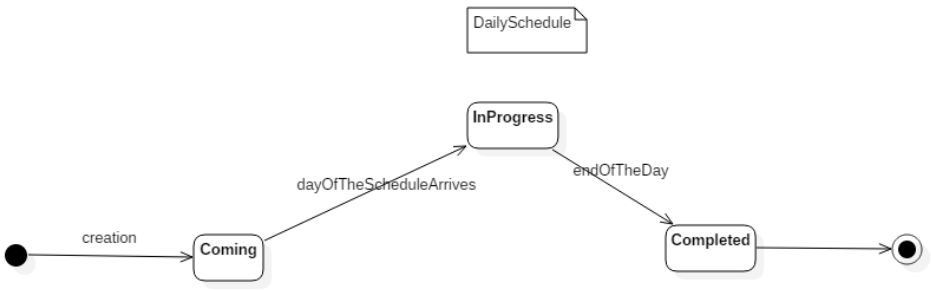


The system is expected to represent the following state-charts for the entities:

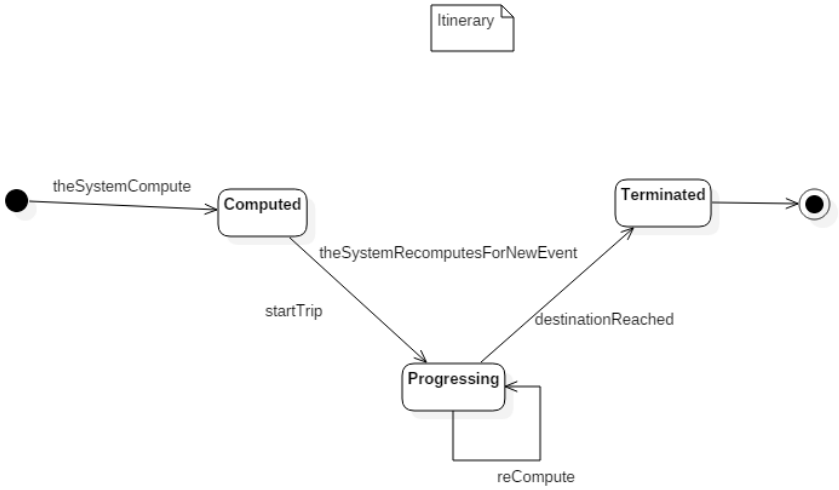
- Appointment
- DailySchedule
- Itinerary



Appointment state-chart



DailySchedule state-chart



Itinerary state-chart

b. Product functions

This section concerns about defining the correct and proper requirements to best fulfill the goals proposed for this application.
The requirements are grouped by the specific features they describe.

G1. Schedule user's appointments along the days.

Registration/Authentication

R.1. The user must register, if he is not yet.

R.1.1. Once the user fills the required information for the registration, the system must send him an auto-generated email with a link to verify its account.

R.1.2. Once the user clicks on the link sent by email, the system inserts the account into the database and the account is created.

R.2. If the user is already registered, the system must allow the user to authenticate himself through username and password.

Appointment Inserting

R.3. The system must allow the user to enter the entire description of its appointments, in terms of:

- Title
- Appointment Type (the type defines the default priority of the event and the default vehicle).
- Date
- Starting time
- Duration
- Location
- Starting location (by default, if it is the first appointment of the day it is home, otherwise it will be the location of the previous appointment)
- Notes

R.4. After the submit of a new appointment, the system must scan all the already present appointments, checking if there is inconsistency.

R.5. The system must not allow the user to insert an appointment with starting time previous than the system's time.

R.6. If an inconsistency is found, the system must warn the user, asking him to to edit/delete the new appointment.

R.7. The system must allow the user to view/edit/delete all its inserted appointments.

R.8. After the edit of an appointment, the system must re-scan all the already present appointments, checking if there is inconsistency.

Breaks Managing

R.9. The system must allow the user to enter two 30-minutes break during the day into its preferences.

R.10. The system must allow the user to specify a time of the day devoted to have lunch (minimum/default=15 min).

R.11. The system must not allow the user to insert an appointment during its breaks.

R.12. The system must allow the user to insert an appointment during its lunch-time, but only if:

- it only lasts the half of the lunch time;
- it starts before lunch time and it ends not more than (lunch-time in mins/4) after the start of lunch time;
- it starts after lunch time and it starts not more than (lunch-time in mins/4) before the end of lunch time;

G2. Notify the user of incoming appointments, with short details about starting time and mean of transportation.

R.1. The user must be registered and logged in.

R.2. The system must warn the user 30 minutes before it is proper to leave for an appointment.

R.3. The system must notify a possible change in the track before starting the itinerary, due to the presence of an available shared-vehicle in the neighborhood.

G3. Propose a suitable itinerary throughout the appointments' locations, according to user's preferences, appointment description and external information about public transportation, weather forecast.

Account Customization

R.1. The user must be registered and logged in.

R.2. The system must allow the user to insert all its personal vehicles.

R.3. The system must allow the user to choose an "eco-plan" for its trips.

R.4. The system must allow the user to enter a maximum-distance coverable for a certain mean of transportation.

R.5. The system must allow the user to enter a certain period of time in which a specific mean of transportation is avoidable

R.6. The system must allow the user to enter its 3 favourites mean of transportation: by foot, car, bike, motorbike, public transportation (no distinction between the various types), vehicle-sharing.

R.7. The system must allow the user to enter its transportation subscriptions with its type (daily ticket, monthly, season pass,...)

R.8. The system must allow the user to enter a maximum distance coverable for each mean of transportation.

R.9. The system must allow the user to enter its home location and to retrieve its position.

R.10. The system must allow the user to enter a maximum cost for a trip.

R.11. The system must allow the user to customize the priority of a certain appointment.

R.12. The system must allow the user to create new appointment type or to edit/delete the existing ones (inserting priority and avoided vehicle).

Getting Outsource Information

R.15. The system must retrieve all the necessary information about the maps from an external service (such as Google Maps).

R.14. The system must contact the public transportation service of the area to retrieve the necessary information of rides.

R.15. The system must contact a weather forecast service to retrieve information about the weather in the day of the appointment.

Itinerary Computation

R.16. The system must use an algorithm to produce suitable tracks, to reach the destination in time.

R.17. The system must consider the priority of the appointments during the scheduling.

R.18. The system must consider the type of the appointments during the scheduling:

R.19. The system must not propose an itinerary to an appointment with a mean of transportation avoidable for that appointment type

R.20. The system must not suggest “bike” as a mean of transportation if one of these conditions hold :

- rain/snow is foreseen
- temperature is lower than 18°
- it is set as an avoided vehicle for that appointment type

R.21. The system must not suggest “by foot” as a mean of transportation if one of these conditions hold:

- rain/snow is foreseen
- temperature is lower than 18°
- it is set as an avoided vehicle for that appointment type

R.22. The system must show the 5 best itineraries: the most ecologic, the cheapest, the shortest, the one with less walking distance and the one with minimum changes shown with an intuitive icon.

R.23. The system must show to the the user the type and the starting time of every single itinerary propose.

R.24. The system must allow the user to choose one of the itineraries, the default option is the shortest one (if he has not chosen the eco-plan in its preferences, otherwise the default option is the most ecologic).

R.25. Computing the most ecologic itinerary, the system must compute an itinerary that excludes any personal-vehicles or sharing-services different from bikes and/or electrical cars.

R.26. The system must not propose an itinerary that foresees a walking distance greater than the maximum walking distance inserted by the user.

R.27. The system must not propose a ride that foresees a cost greater than the maximum cost per itinerary inserted by the user.

R.28 . The system must not propose an itinerary that starts/ends more than 30 minutes after/before lunch time.

R.29. The system must not propose an itinerary that overlaps with the breaks inserted by the user.

R.30. The system must not propose an itinerary with car/bike if the user has not inserted it in its personal vehicles (except for vehicle-sharing services that are proposed at the moment of starting a trip).

R.31. The system, while computing the rides of every itinerary, must assign to them at least one of the user’s favourite vehicle, if it does not violate any constraint.

R.32. The system will retrieve the computation of the ETA and of the path (by foot, by car or by bike) from Google Maps.

R.33. If the user has indicated an avoidable period of time for a certain mean of transportation, the system must not propose itinerary in that period of time with that particular mean of transportation.

G4. Give the possibility to buy a transportation ticket.

R.1. The user must be registered and logged in.

R.2. If the user starts the system's itinerary and it foresees public transportations, the system must check the presence in the user's account of a pass for those transportation service.

R.3.In absence of the pass, the system must inform the user of the possibility to buy a ticket.

R.4. If the user chooses to buy the ticket, the system must perform a PayPal transaction between the user and the public transportation service.

G5. Give the possibility to reserve a vehicle-sharing service.

R.1. The user must be registered and logged in.

R.2. The system must allow the user to select which vehicle-sharing subscription he has in the account customization.

R.4. At the starting time the application, the system must contact the vehicle-sharing service (only the previously selected once) to retrieve the information about the location the vehicles nearby the user.

R.5. At the starting time of a trip, the system must look for shared-vehicle and, in case, verify if they are a suitable alternative to the previous track and show them on the map.

R.6. If the user selects a sharing-vehicle from the map, the system must redirect it to the application of the correspondent service.

G6. Guide the user through all the appointments' locations as a navigator.

Before Starting

R.1. The user must be registered and logged in.

R.2. At the starting time of a trip, the system must verify if the previously computed track is still consistent.

R.3. At the starting time of a trip (or if a new itinerary is going to be proposed), the system must look for sharing-vehicles in the neighbors.

R.4. (...continues) And in case, verify if they are a suitable alternative to the previous track and show them on the map, to let the user reserve them.

After Starting

R.5. When the user starts its travel, the system must run as a navigator-application.

R.6. If a weather change is detected and it affects the track conditions, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.7. If traffic is detected and it affects the track conditions, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.8. If a delay of the selected ride is detected and it affects the track conditions, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.9. If a delay by the user is detected and it affects the track conditions, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.10. If a public transportation strike is detected and it affects the track conditions, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.11. If the current position of the user is different from the one foreseen, the system will propose new itineraries (Shortest, Most Ecologic, Cheapest, MinimumChanges, MinimumWalkingDistance).

R.12. If it is impossible to compute an itinerary that lets the user get in time to its appointment, the system will compute an itinerary that minimizes the delay.

c. User characteristics

- Not Registered user: a person using the application without being registered. The only available functions are the possibility to proceed with the registration, to login in the system with a previously created account or to recover the password

- Registered User: a person passed through a successful registration/login process. This type of user has all the functions available so it can create and fill its own calendar, customize its account and schedule appointments.

d. Domain assumptions

The Requirements we have just listed are necessary but not sufficient to satisfy the goals this application wants to achieve.
In fact, this would be impossible if we did not take some facts for granted.
The following are the **Domain Assumptions that will support the Requirements in fulfilling the goals** mentioned above.

D.1. We assume that the registration's email always arrives to the destination address.

D.2. We assume that every username is unique.

D.3. We assume that the system retrieves the accurate position.

D.4. We assume that the weather forecast are always right.

D.5. We assume that, when an available shared-vehicle is selected, it remains available until the user completed the reservation process.

D.6. We assume that the application always has an available internet connection.

D.7. We assume that all the services that the application uses to retrieve all the necessary information are always available.

D.8. Once the user has reserved successfully a shared-vehicle, it is able to unlock it without any problem.

D.9. We assume that payment information correctness are verified using the PayPal service.

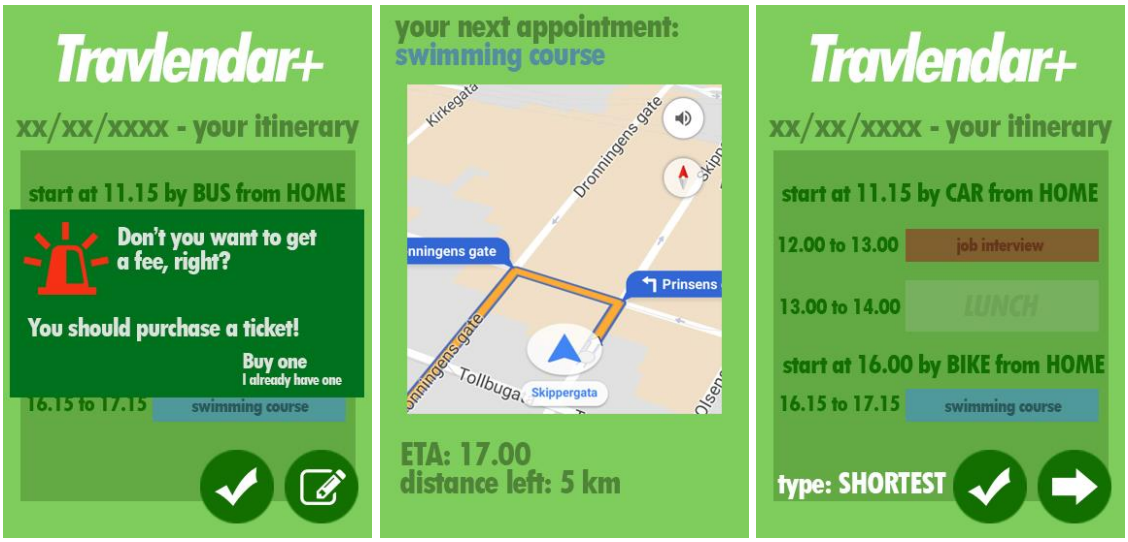
D.10. We assume that every appointment inserted by the user are all located in a ray of 400 km.

3. Specific Requirements

a. External Interface Requirements

a.1 User interfaces

Here there are some stylized mockups of the application to develop.



a.2 Software interfaces

This application is supposed to accomplish its task through the support of external services that collects data useful to efficiently arrange the trip of the users.

Google Maps

The system will connect to Google Maps, exploiting the relative APIs.
This is necessary to guide the user as a navigator, retrieving the maps of the service.
Furthermore, the Google service will provide all the data about the public transportation, the weather changes, the traffic conditions.

OpenWeatherMap

This service's APIs will provide all the weather forecast to optimize the travel planning along the day.
It will also be consulted every 5 minutes in real-time session to decide whether reschedule the travel or not.

PayPal

The PayPal API's will be exploited to start a transaction towards the right public transportation service to let the user buy a one-ride ticket.

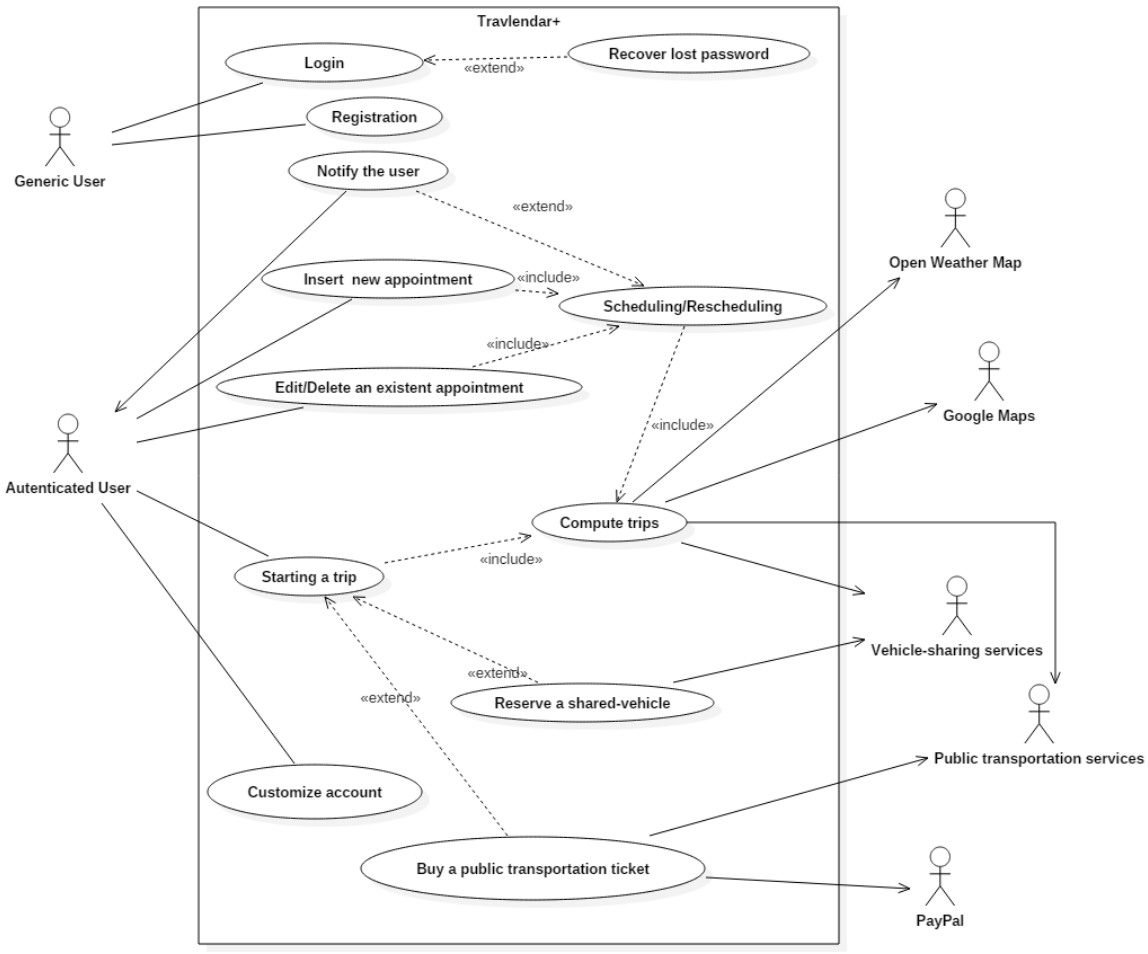
Enjoy, Car2Go, Share'n'go, Ofo, Mobike, BikeMi

The correspondent APIs of these sharing-services will be exploited to locate all the reservable vehicles.
If the user would like to reserve one of them, the system will redirect the user to the correspondent app or to the Play Store/Apple Store.

DBMS

The system is supposed to handle its databases through MySQL.

b . Functional Requirements



Use case diagram of the application

REGISTRATION

ACTORS	Generic user
GOALS	[G1]
INPUT CONDITIONS	The generic user opened the app
EVENT FLOW	1. The generic user clicks on the “Sign Up” button 2. The generic user fills every compulsory field displayed, such as a user ID, a password (following rules showed in the section 3.E.3) and his personal email 3. The generic user selects the “Confirm” button 4. The application system loads all inserted data in the database 5. It is send a confirmation email to the generic user 6. The generic user clicks on the link in order to definitely confirm the registration
OUTPUT CONDITIONS	The generic user now can Log In the app whenever it wants in order to reach the authenticated status.
EXCEPTIONS	1. The generic user is already registered 2. The user ID is already used by another user 3. The password does not follow the security rules (at least 8 chars, composed by all kind of them) 4. The generic user sets an already used email 5. The email inserted does not exist All exceptions are highlighted by a specific notification which describes the problem and the generic user has to retry the process from the point 2.

LOGIN

ACTORS	Generic user
GOALS	[G1],[G2],[G3],[G4], [G5], [G6]
INPUT CONDITIONS	The generic user opened the app
EVENT FLOW	1. The generic user fills every compulsory field displayed inserting its own username and password 2. The generic user clicks on the "Sign In" button 3. The application system checks all inserted data making a query to its database Password/username recovery
OUTPUT CONDITIONS	The generic user is now authenticated and it can customize its personal account and/or inserting its appointment in the Calendar directly from the main page now displayed.
EXCEPTIONS	1. The generic user inserts a wrong password 2. The user ID inserted does not exist All exceptions are highlighted by a specific notification which describes briefly the problem and the generic user has to retry the process from the point 2.

CREATION OF A NEW APPOINTMENT

ACTORS	User, Open Weather Map, Google Maps, Public Transportation Service
GOALS	[G1], [G3]
INPUT CONDITIONS	The user must be logged in.
EVENT FLOW	1. The user selects the option “new appointment”; 2. The user fills all the fields to describe the appointment (title, starting time, type, etc.); 3. The system queries the database in order to retrieve all the uploaded appointment with the same date of the one set by the user; 4. The system collects from these appointments the locations for the trips and the constraints given by their types and priorities; 5. The system queries its database in order to retrieve all the information about user’s preferences and its account customization; 6. The system retrieves from Open Weather Map the weather forecast of the appointment’s day; 7. The system retrieves the map of the area of the appointment's location from Google Maps; 8. The system retrieves all the information about public transports and their rides from external services (depending from the area of the trip);

	9. The system checks if there is at least one feasible way to reach the appointment without deleting the others; 10. If point 9 is passed successfully, the system applies an algorithm to compute the 5 best possible itineraries considering all variable (maximising user's preferences and minimizing cost and time); 11. The system proposes the computed itineraries to the user; 12. The user chooses one of the itineraries; 13. The system saves the changes in the database;
OUTPUT CONDITIONS	The new appointment and the correspondent trip is successfully saved into the database.
EXCEPTIONS	1. The new appointment is incompatible with the appointments already present in the calendar, because it generates an overlap or there is not enough time between two consecutive appointments for the trip; 2. One of the filled section in point 2 produces a syntax error 3. The user inserts an appointment with starting time previous than the system's time →The exceptions 1,2,3 are handled notifying the user of the inconsistency and taking back the flow of the events to the point 2 in order to change the appointment. 4. The system fails during the process; →The database rollbacks the operations and returns to the database's status that precedes the event of adding the new appointment.

BUY TICKET

ACTORS	User, Open Weather Map, Public Transportation Service,Vehicle-sharing services, PayPal
GOALS	[G2], [G3], [G5], [G6]
INPUT CONDITIONS	The user must be logged in. The user had to create an appointment in which the selected track uses public transportation. The appointment itinerary is starting in this moment.
EVENT FLOW	1. The system checks whether the user has the pass for the foreseen public transportation. 2. If the user has not the pass, the system notifies the user that a ticket is needed. 3. The user decides to buy a ticket. 4. The system starts a PayPal transaction between the user and the public transportation service, redirecting him to to the proper web-page. 5. The user performs the payment and get a ticket. 6. The system's database stores the operation and the ticket ID.
OUTPUT CONDITIONS	The user has its ticket and it is ready to continue its trip.
EXCEPTIONS	1. The itinerary is not consistent. → A new itinerary is computed and proposed by the system. 2. The payment is not completed. →When the app is re-opened, the system will again notify the user that a ticket is needed.

NAVIGATOR

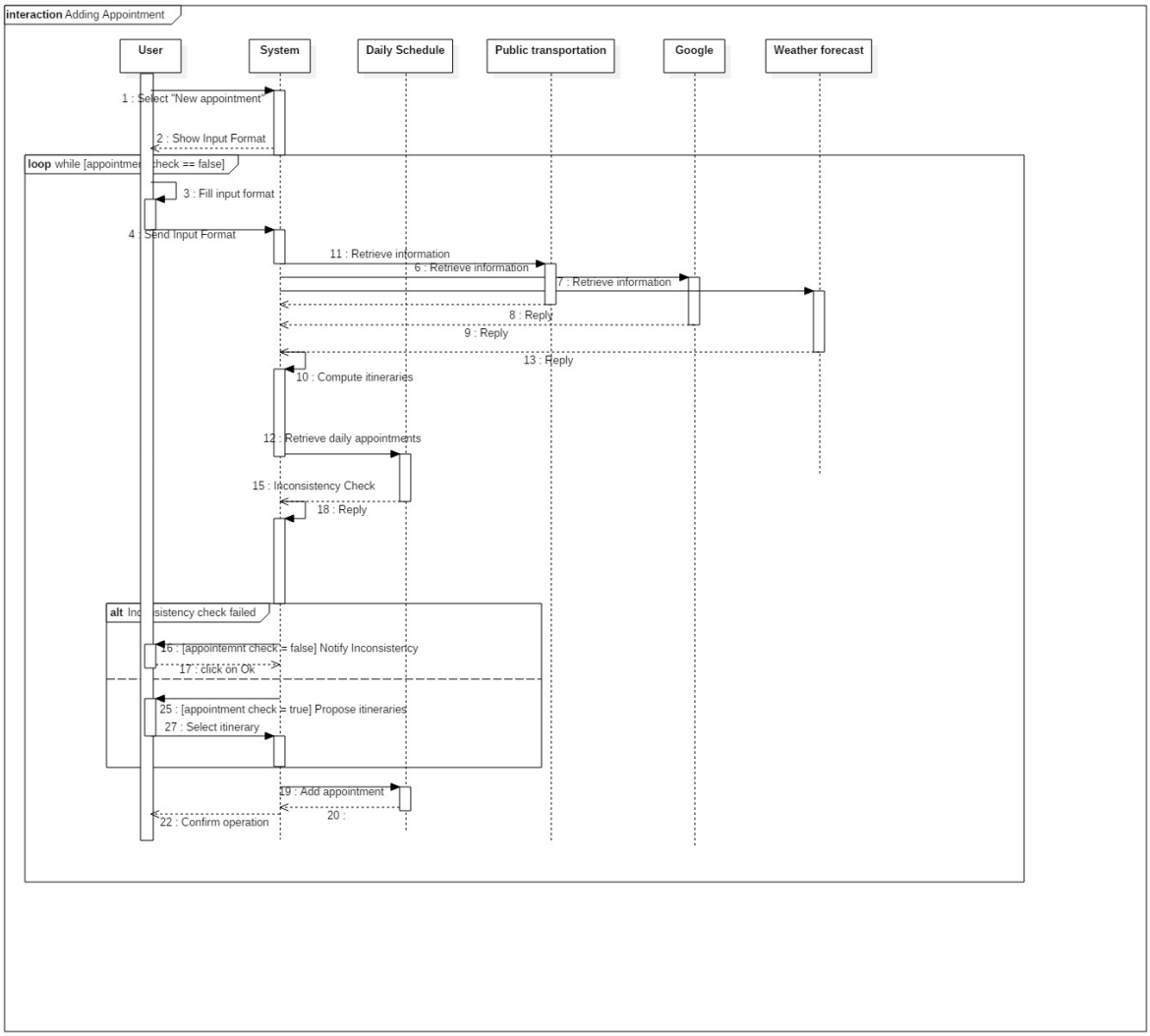
ACTORS	User, GoogleMaps, OpenWeatherMap
GOALS	[G4]
INPUT CONDITIONS	The user starts a trip to reach an appointment. The checking of the available vehicle-sharing services and of the necessity of buying a ticket has already been performed. Location service activated.
EVENT FLOW	1. The system notifies the user that it is proper for the user to leave for reaching an appointment. 2. The user opens the app. 3. The system decides whether the previously proposed itinerary is still consistent. 4. The system looks for vehicle-sharing services nearby the user's position and decides whether they are useful to get to the appointment in time. 5. If no shared-vehicle is detected the system will continue with the itinerary already computed. 6. The system checks whether public transportation is foreseen. 7. The system divides the trip in uniform segments for the type of navigation; 8. The system displays the indications to the next step to the user both listing instructions and showing them on the map, specifying the track that the user has to follow or the information of a particular public transportation's ride (e.g. the number and the direction of a bus or the platform of a train); 9. Each time an event that may affect the ETA occurs, the system proposes 5 new itineraries (as explained above). 10. If the user decides which he prefers, the navigation continues from point 7 .

	11. The navigation ends when the destination is reached and it is notified to the user.
OUTPUT CONDITIONS	The user has arrived to the location of the appointment. The appointment reached is saved in the database.
EXCEPTIONS	1. Location service unavailable during the navigation. →The system warns the user that it is unable to retrieve its position and the navigation will resume as soon as it is available. 2. It is impossible to compute an itinerary that lets the user get in time to the appointment. → The system proposes an itinerary that minimizes the delay.

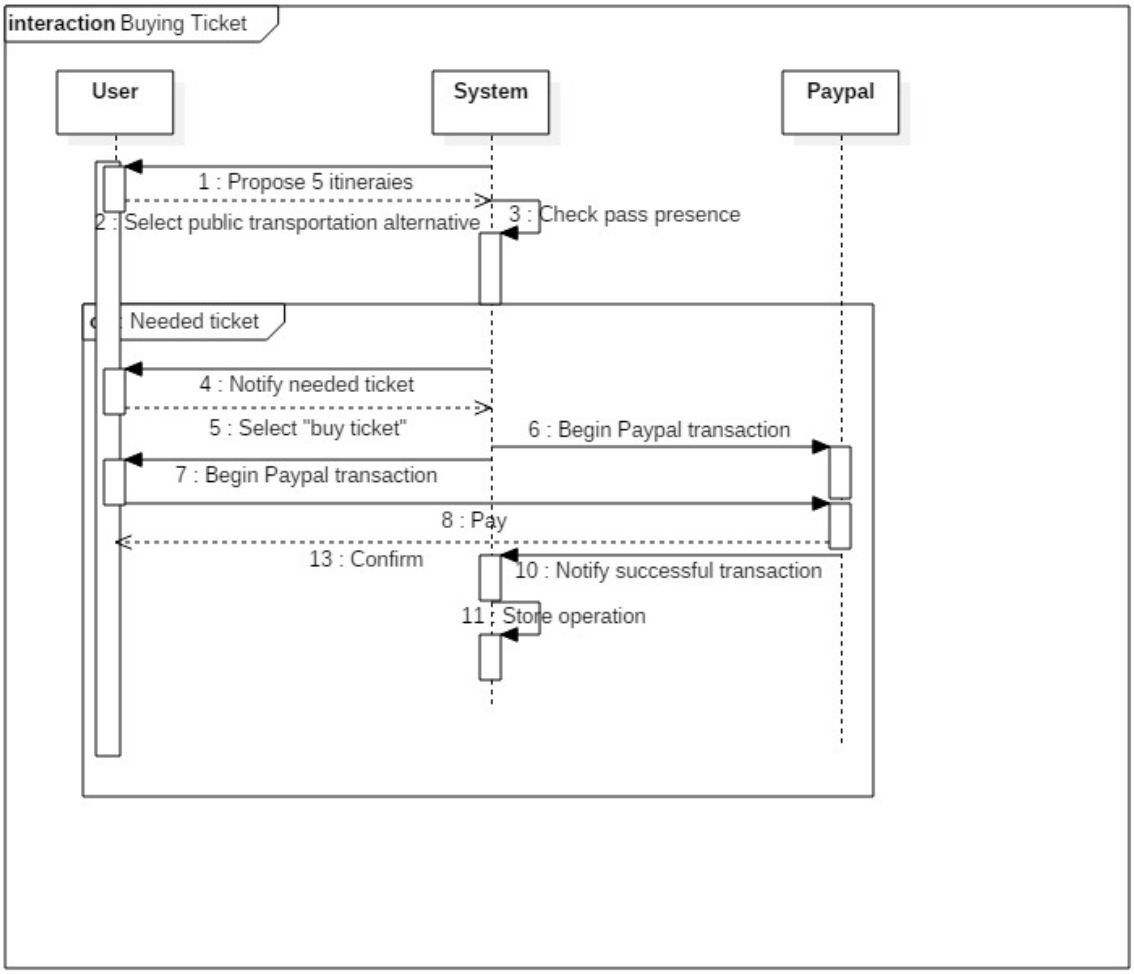
VEHICLE-SHARING RESERVATION

ACTORS	User, Vehicle-sharing services
GOALS	[G6], [G4]
INPUT CONDITIONS	The user starts a trip or, during a trip, a recomputing of the itinerary is done because of external event (such as the detection of a public transportation strike / an incident).
EVENT FLOW	1. The system looks for vehicle-sharing services nearby the user's position and decides whether they are useful to get to the appointment in time. 2. Some shared-vehicle are detected near the user's location. 3. The system checks if they respect the constraints of the appointment's type. 4. If they respect those constraints, the system recomputes the trip with the new information to check if the detected vehicle-sharing services are useful to reach the destination. 5. If they are, the system proposes the new itineraries to the user. 6. The user accepts one of the new itineraries. 7. The system redirects the user to the selected vehicle-sharing service's application in order to reserve the vehicle. 8. The system shows to the user the shared-vehicle's position on the map and the indications to reach it. 9. The user reaches the vehicle and unlocks it using the sharing service's application. 10. The system guides the user to the location like a common navigator.
OUTPUT CONDITIONS	The user has arrived to the location of the appointment.

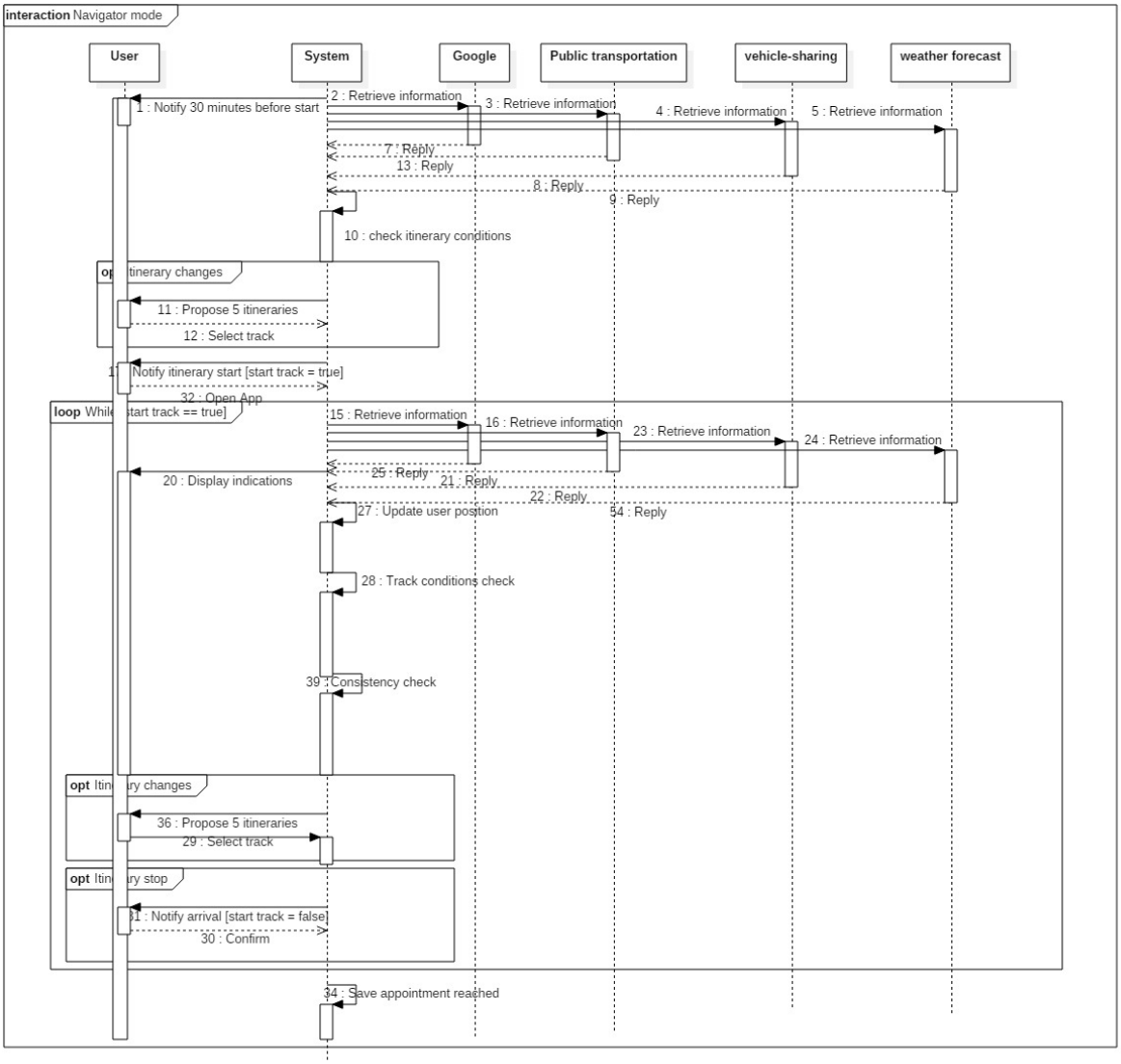
EXCEPTIONS	1. A shared-bike is detected but the use of the bike is prohibited by the appointment's type or by the weather forecast. →The system does not consider the shared-bike among the possible vehicles for the itinerary. 2. A problem occurs during the reservation of the shared-vehicle. →The system asks to the user if it wants to use another shared-vehicle (if there is one in the area) or if it prefers to continue without considering vehicle-sharing services. 3. Vehicle-sharing application is not installed in the user's device. →The system redirects the user to the application's page in order to install it. If some problems occur during the installation process and it is impossible to proceed with the reservation, the system recomputes the itineraries without considering that service.



Sequence Diagram of the insertion of a new appointment
Pay attention: the user is considered to be always online in this interaction

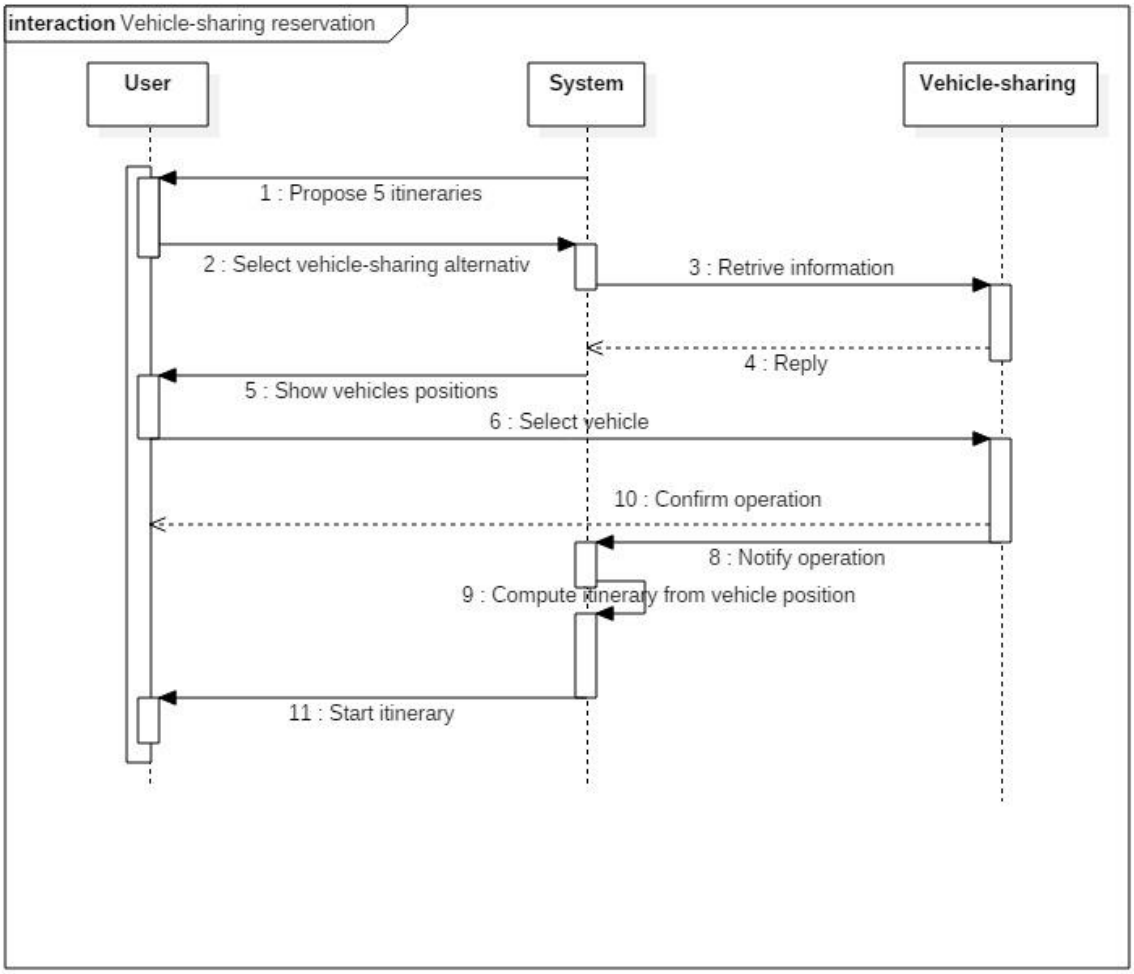


Sequence Diagram of the purchase of a ticket
Pay attention: the user is considered to be always online in this interaction



Sequence Diagram of Navigation mode

Pay attention: the user is considered to be always online in this interaction



Sequence Diagram of the reservation of a vehicle
Pay attention: the user is considered to be always online in this interaction

c. Performance Requirements

The system must guarantees a response time of, at most, 1.0 second in order to keep the user's flow of thought to stay uninterrupted and 95% of all response time should be less than 0.5 seconds for having the user feel that the system is reacting instantaneously.
The system must support only one concurrent user for each installation of the application.

d. Design Constraints

d.1. Standard compliance

The system must ask the user to retrieve its position.

The system will use user's email only to validate its registration, therefore no SPAM email will be sent.

The system will not collect any credit card number, since the transaction to buy public transportation tickets is entirely redirected to PayPal.

d.2. Hardware limitations

-iOS and Android
-Windows, MacOS, Linux
- 2G/3G/4G connection
- GPS/GLONASS/Galileo/WIFI for locating the user
-At most 500MB of space in the hardware storage is permitted for the application.

e. Software System Attributes

e.1. Reliability

The system must do the scheduling and the other computing operations producing the correct outputs within 30 seconds.

e.2. Availability

The application and the relatives online databases must be available 24/7, with only few monthly hours of downtime permitted in case of updates or other maintenance operations.

e.3. Security

Users credentials and preferences will be stored using proper cryptography. The security and privacy of the communications between the application and the external servers are a primary concern: all the communications must use cryptography and the request of information to external services must be anonymous in order to protect the privacy of the user.

The user has to choose a secure password with a minimum length of 8 characters and it must be composed of numbers, symbols and mixed-case letters.

e.4. Maintainability

The system must be developed in order to make it easy to add new functions.

e.5. Portability

The application must support a wide range of mobile operative systems (at least Android and IOS).

4. Scenarios

Scenario 1: Registration and usage

Alessandro is a very busy business man and wants to organize his several appointments in order to know how to reach them. His boss suggests him to download Travlendar+, so he signs up the app giving his username, password and email. Then he inserts in the calendar application all his weekly appointments and the system proposes an optimal solution to reach them in the minimum time last. He confirms all the application choices and finally he has a well organized calendar that permits him to not lose useful time.

Scenario 2: Overlap of two appointments

Andrea is a design student of Politecnico and has downloaded Travlendar+ in order to schedule his university calendar since he has to travel a lot to reach his classes. By the way, hir girlfriends wants to have a date on Monday afternoon, and Andrea add it in the app system.
Unfortunately, on Monday he has a lecture at 12 o'clock near Bovisa, while his partner will wait for him in Famagosta at 14 p.m. So the system highlights the impossibility to have both the appointments. For this reason Andrea decides not to go to the university, by deleting the lecture from its calendar. Now he can adds the date appointment reaching the date location perfectly in time.

Scenario 3: Appointment managing

Pit is a middle aged man who has a really busy life, full of appointments. In order to manage his calendar, he chooses to download the new Travlendar+ app. Now he has already inserted all his week appointment but the business meeting on Saturday is cancelled. For this reason he enters in the app and deletes the appointment from the calendar. More over, Pit selects the family gathering on the same date in the afternoon, changing its final time (indeed now he has more free time to spend). Finally, he exchanges the priority of the party on Thursday too, because he wants to arrive 30 minutes earlier (priority 5). The system reschedules all modifications and proposes the 5 alternatives for each itinerary. Pit chooses his favourite once and closes the app.

Scenario 4: Choosing preferences

Mario is a very green person, who prefers avoiding the usage of polluting transportation. After he signs up for the planning application Travlendar+, the system asks him to insert some information in order to customize his account. For this reason he selects the ecologist preference, in particular he chooses bike as the favourite vehicle adding his personal bike in the correspondent section and, in the maximum foot distance option, he puts 3 km. Actually few days later he has to go to visit an Art museum situated in the

other side of the city, and when he enters it as an appointment in the application, this one proposes him a bike track with his personal vehicle that lasts 2 hours and 17 minutes. Then, as a shorter chance, the system suggests an itinerary with public transports of the duration of 1 hour and 36 minutes and other three itineraries with an icon which shows them as the Cheapest, the MinimumChanges track, and the MinimumWalkingDistance one. The first is proposed by default, since Mario would rather face an ecologist travel. In fact Mario confirms the bike route proposed and starts it.

Scenario 4: Considering weather conditions

John has scheduled his several appointments in the new Travlendar+ app, selecting, inter alia, his preference in travelling by foot. However, for his business appointment on Monday there is a problem: rain is expected and the app is informed thanks to the support of OpenWeatherMap. Therefore, the system suggests him to use his own car, considering the type and the importance of the appointment too. John accepts the smart advice and confirms it.

Scenario 5: Considering appointments type and the priority

Steve is an important married businessman. Using his new Travlendar+ application he organised all his busy week with different type of appointment, such as a job meeting on Monday at 11 a.m., his yoga lesson on Wednesday at 7 p.m., his daughter's concert on Friday (9:45 p.m.) and a date with his wife on Saturday (8 p.m). in a restaurant). He also adds the priority of all these events giving a 5 five stars evaluation to the meeting and to his romantic evening, 4 stars to the concert and only 2 for yoga. The system considers all these details and proposes him to use the personal motorbike on Monday because of its celerity in order to arrive at 9 o'clock, while for Saturday advises the usage of a car taking in consideration in this case the comfortability. On the other hand, the app suggests a walk to reach the concert 30 minutes earlier, actually because it so so distant from Steve's home and finally proposes a bike track for Wednesday considering the fact that it is not necessary to be so tidy for this appointment.

Scenario 7: Navigator mode

Davide is a young student who uses the Travlendar+ application in order to have a support in organizing and reaching his several university appointment. It's Monday and he has to go to Sant'Ambrogio at Cattolica university for some lectures. The system at 7 o'clock notifies to him that in 30 minutes he has to start his itinerary in order to reach the location 15 minutes earlier. He opens the app and sees the updated 5 best itineraries (for example now the system consider a bike track because the weather conditions changed from respect to the first scheduling of 6 days ago). He chooses the public transportation one as usual clicking on "starts trip". Now the system begins to

give directions in order to reach the underground (5 minutes by foot from his house). Davide follows all of them and at the end arrives at university exactly 15 minutes earlier as prevented.

Scenario 8: Real -updating and minimizing eventual delay

Sarah, who lives in Milan, wants to reach a start up convention in the afternoon and inserted it in Travlendar+ app few days before, choosing a public transportation trip, suggested by the system in consideration of the fact that she has an ATM pass. However, an unexpected strike occurs and the application notifies the matter to Sarah. Sarah accepts the car-based alternative track and she begins the trip. Unfortunately, on the street she is going through, an accident occurs, blocking the traffic and the system updates another time the itinerary. The system finds a bike-sharing service nearby and, considering that it is prevented a possible 6 minutes delay (the minimum one), suggests it to the user. She takes rapidly the bike and arrives only 3 minutes after the beginning of the convention.

Scenario 9: Buy tickets

Martina is a young mother from Milan, who needs to organize her appointment very smart because she has also three child to whom takes care. She inserted all her week appointment in Travlendar+ and, when 30 minutes are left to the first itinerary, the app notifies that. One of the tracks suggested is by public transportation and Martina selects it. The system now checks if the user has any ATM pass, but she has not, so the app proposes her to buy a ticket directly through itself. She confirms this option and the system redirects her to PayPal. After Martina inserts all the information required, the systems saves the operation in the database and now she can use her personal ticket through her smartphone.

Scenario 10: Vehicle-sharing usage

Jennifer is a university student from Sicily but she studies in Milan. She has scheduled all her appointment in her personal account of Travlendar+. On Thursday she has to go to the gym with her friends after university and, when she is already in the classroom, the app notifies the starting itinerary 30 minutes before. Jennifer opens the app and sees that the application found a new possibility to reach the appointment's location, thanks to a vehicle-sharing service. So, she opens the app where the system shows all the reservable car of Enjoy, but not the Car2Go ones (indeed, she has not the subscription to that). She clicks on the nearest one and the app redirect her to the Enjoy system in order to take it. Then, when she arrives to the car, the itinerary starts and, by following all the directionsb she arrives to the gym.

5. Alloy Analysis

```
open util/integer
open util/boolean

sig System{
//it's the application and the date is the device's one
     time: one Time,
     users: some User
}

sig Time{
date: Int,
hour: Int}
{date>0
hour>=0}

sig User {
     calendar: set DailySchedule
}

abstract sig DailyScheduleStatus{}

one sig Coming extends DailyScheduleStatus{}
one sig InProgress extends DailyScheduleStatus{}
one sig Completed extends DailyScheduleStatus{}

sig DailySchedule{
     status: one DailyScheduleStatus,
     date: one Int,
     contains: some Appointment
}{
date>0
}

sig Appointment {
     predecessor: lone Appointment,
     successor: lone Appointment,
     startingTime: one Time,
     finalTime: one Time,
     associatedItinerary: one Itinerary,
}{
startingTime.date=finalTime.date
startingTime.hour<finalTime.hour }
```

```
abstract sig ItineraryStatus{}
one sig Computed extends ItineraryStatus{}
one sig Progressing extends ItineraryStatus{}
one sig Finished extends ItineraryStatus{}

sig Itinerary{
     associatedAppointment: one Appointment,
     startingTimeIt: one Time,
     finalTimeIt: one Time,
     itineraryStatus: one ItineraryStatus
}

fact AppointmentConstraints{
     predecessor=~successor
     all a: Appointment | a.predecessor!=a
     all a: Appointment | a.successor!=a
     //each appointment in a dailyshedule must have the same
date of it
     all d: DailySchedule, a: d.contains |
d.date=a.startingTime.date
     //If there is a predecessor, then it must end before its
successor
     all a1, a2: Appointment | (a2 in a1.predecessor) =>
(a1.startingTime.hour>a2.finalTime.hour)
     all d: DailySchedule,  a1, a2: Appointment | (a2 in
a1.predecessor) => (a1 in d.contains && a2 in d.contains)
     all d: DailySchedule,  a1, a2: Appointment | (a2 in
a1.successor) => (a1 in d.contains && a2 in d.contains)
     //If there is a successor, then it must start after its
predecessor
     all a1,a2 : Appointment | (a2 in a1.successor) =>
(a2.startingTime.hour>a1.finalTime.hour)
     //There is only one appointment in a daily schedule
without a predecessor/successor
     all d: DailySchedule | (#d.contains =
(add[#d.contains.predecessor,1] ))&& ( #d.contains =(
add[#d.contains.successor,1]))
}

fact AppointmentAndItineraryAssociated{
     associatedItinerary = ~associatedAppointment
     //each Itinerary of an Appointment must have the same
date of it
```

```
     all a: Appointment, i: a.associatedItinerary |
a.startingTime.date=i.startingTimeIt.date
     //each itinerary is between two consecutive appointments
     all i: Itinerary| i.finalTimeIt.hour =<
i.associatedAppointment.startingTime.hour &&
                    ( i.startingTimeIt.hour >=
i.associatedAppointment.predecessor.finalTime.hour)
}

fact UserSystemTree{
// each user must be in a system
     all u: User| u in System.users
// each user must belong with one and only one system
all u1,u2:User, s1,s2:System | (s1!=s2 && u1 in s1.users && u2
in s2.users)=>
(u1!=u2 && u1 not in s2.users && u2 not in s1.users)
}

fact DailyScheduleUserTree{
// each DailySchedule must be in a user's calendar
     all d: DailySchedule| d in User.calendar
// each DailySchedule must belong with one and only one user's
calendar
     all u1,u2:User, d1,d2:DailySchedule | (u1!=u2 && d1 in
u1.calendar && d2 in u2.calendar)=>
(d1!=d2 && d1 not in u2.calendar && d2 not in u1.calendar)
// each user must have at most one daily schedule per date
     all u : User, d1,d2: DailySchedule | (d1!=d2 && d1 in
u.calendar && d2 in u.calendar) => (d1.date != d2.date)
}

fact ItineraryAppointmentTree{
all i:Itinerary | i.startingTimeIt.date=i.finalTimeIt.date
all i:Itinerary | i.startingTimeIt.hour<i.finalTimeIt.hour
// each Itinerary must be in a appointment
     all i: Itinerary | i in Appointment.associatedItinerary
// each Itinerary must belong with one and only one
appointment
all i1,i2:Itinerary , a1,a2: Appointment | (a1!=a2 && i1 in
a1.associatedItinerary && i2 in a2.associatedItinerary)=>
(i1!=i2 && i1 not in a2.associatedItinerary && i2 not in
a1.associatedItinerary)
}
```

```
fact AppointmentDailyScheduleTree{
     // each appointment must be in a dailyschedule
     all a:Appointment | a in DailySchedule.contains
     //each appointment must be in one and only one
dailyschedule
     all a1,a2: Appointment, d1,d2: DailySchedule | (d1!=d2 &&
a1 in d1.contains && a2 in d2.contains)=>
     (a1!=a2 && a1 not in d2.contains && a2 not in
d1.contains)
}

fact DailyScheduleStateChart{
     all s: System, d: s.users.calendar | (d.date>s.time.date)
<=> d.status=Coming
     all s: System, d: s.users.calendar  |
(d.date=s.time.date) <=> d.status=InProgress
     all s: System, d: s.users.calendar  |
(d.date<s.time.date) <=> d.status=Completed
}

fact ItineraryStateChart{
     all s: System, i:
s.users.calendar.contains.associatedItinerary |
((i.startingTimeIt.date=s.time.date) && (i.startingTimeIt.hour
=< s.time.hour)
                    &&  (i.finalTimeIt.hour >= s.time.hour))
<=> i.itineraryStatus=Progressing
     all s: System, i:
s.users.calendar.contains.associatedItinerary |
(i.startingTimeIt.date>s.time.date or
                     (i.startingTimeIt.date=s.time.date  and
i.startingTimeIt.hour > s.time.hour))<=>
i.itineraryStatus=Computed
     all s: System, i:
s.users.calendar.contains.associatedItinerary |
(i.startingTimeIt.date<s.time.date or
                     (i.startingTimeIt.date=s.time.date  and
i.finalTimeIt.hour < s.time.hour))<=>
i.itineraryStatus=Finished
}
```

```
//There is only One DailyScheduleInProgress
assert OnlyOneDSInProgress{
     all u:User, d1,d2: DailySchedule | (d1.status=InProgress
&& d1 in u.calendar && d2 in u.calendar && d1!=d2)

          =>(d2.status!=InProgress)
}

assert AppointmentOrdering{
     all a1,  a2: Appointment | (a2 in a1.predecessor) =>
a1!=a2
     all a1,  a2: Appointment | (a2 in a1.successor) => a1!=a2
}

assert NoOverlappingAppointments{
//if two appointment overlap, they belong with different users
     all a1,a2: Appointment, u1,u2: User |
(a1.startingTime.date=a2.startingTime.date && a1!=a2 &&  (a1
in u1.calendar.contains && a2 in u2.calendar.contains)
                                                    &&
(a1.startingTime.hour>=a2.startingTime.hour &&
a1.startingTime.hour=<a2.finalTime.hour))
                                                    => (u1!=u2)
}

assert SamePredecessorSuccessorDate{
     //predecessor & successor have the same date
     all a1, a2: Appointment | (a2 in a1.predecessor) =>
(a1.startingTime.date=a2.startingTime.date)
     all a1, a2: Appointment | (a2 in a1.successor) =>
(a1.startingTime.date=a2.startingTime.date)
 }

assert OneFirstAndOneLastAppointment{
     all a1,a2: Appointment, d: DailySchedule |
(a1.predecessor = none && a2.predecessor = none && a1!=a2 &&
(a1 in d.contains))=> (a2 not in d.contains)
}
```

```
assert noOverlappingItineraries{
//if two itineraries overlap, they belong with different users
     all i1,i2: Itinerary, u1,u2: User | (i1!=i2 &&
i1.itineraryStatus=Progressing &&
i2.itineraryStatus=Progressing && i1 in
u1.calendar.contains.associatedItinerary && i2 in
u2.calendar.contains.associatedItinerary) =>
                                                    (u1!=u2)
}

assert ScheduleItineraryRelationFinished{
//Verify the time when itineraries are finished
     all s: System, d: s.users.calendar , i:
d.contains.associatedItinerary | ((d.status=Completed) or
(d.status=InProgress && i.finalTimeIt.hour<s.time.hour)) =>
i.itineraryStatus=Finished
}


assert ScheduleItineraryRelationProgressing{
//Verify that if the itinerary is progressing, then the daily
schedule is in progress
     all d: DailySchedule, i: d.contains.associatedItinerary |
i.itineraryStatus=Progressing => d.status=InProgress
}


pred isDrafted[a:Appointment]{
     all d: DailySchedule| not (a in d.contains)
}
pred timeOK[a:Appointment]{
     a.startingTime.date>=System.time.date
}

pred addAppointment[d:DailySchedule,
a:Appointment,d':DailySchedule]{
     (isDrafted[a] or a in d'.contains)
     timeOK[a]
     d.date=a.startingTime.date
     d'.date=d.date
     d'.status=d.status
     d'.contains=d.contains+a
}
```

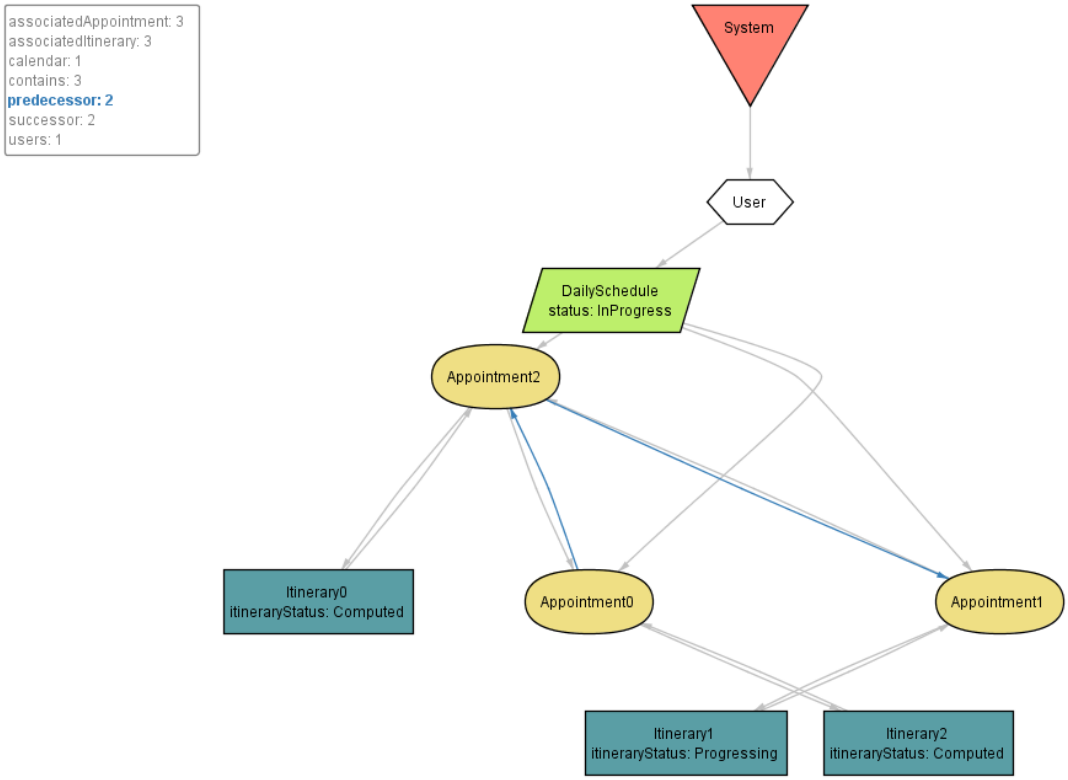
```
pred showAddAppointment[d:DailySchedule,
a:Appointment,d':DailySchedule]{
     addAppointment[d,a,d']
     a in d'.contains
}

assert checkAdd{
     all a:Appointment, d,d': DailySchedule | isDrafted[a]=> (
addAppointment[d,a,d'] => (not isDrafted[a]))
     all a:Appointment, d,d': DailySchedule | (a not in
d'.contains)=> addAppointment[d,a,d'] => (a in d'.contains)
}

pred show{
all a:Appointment | a in DailySchedule.contains
}


check checkAdd
check ScheduleItineraryRelationProgressing for 5
check ScheduleItineraryRelationFinished
check noOverlappingItineraries
check OneFirstAndOneLastAppointment
check NoOverlappingAppointments
check SamePredecessorSuccessorDate
check AppointmentOrdering
check OnlyOneDSInProgress

run show for 8 but exactly 1 System, exactly 1 User, exactly 1
DailySchedule, exactly 3 Appointment//, 10 Itinerary
run showAddAppointment
```

Proof of consistency

11 commands were executed. The results are:

#1: No counterexample found. checkAdd may be valid.
#2: No counterexample found. ScheduleItineraryRelationProgressing may be valid.
#3: No counterexample found. ScheduleItineraryRelationFinished may be valid.
#4: No counterexample found. noOverlappingItineraries may be valid.
#5: No counterexample found. OneFirstAndOneLastAppointment may be valid.
#6: No counterexample found. NoOverlappingAppointments may be valid.
#7: No counterexample found. SamePredecessorSuccessorDate may be valid.
#8: No counterexample found. AppointmentOrdering may be valid.
#9: No counterexample found. OnlyOneDSInProgress may be valid.
#10: **Instance found.** show is consistent.
#11: **Instance found.** showAddAppointment is consistent.

Generated world



6. Effort Spent

Alessandro Saverio Paticchio: ~33 hours
Andrea Tricarico: ~33 hours
Davide Santambrogio: ~33 hours

7. References

- Course slides about Requirement Engineering, Alloy Modelling
- Alloy tutorial on http://alloy.mit.edu/alloy/tutorials/online/
- RASD documents by previous Software Engineering 2 projects