POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

Travlendar+

Federico Betti - Tommaso Bianchi

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Federico Betti - 899914
	Tommaso Bianchi - 894183
Version:	1.0
Date:	October 29, 2017
Download page:	Github
Copyright:	Copyright © 2017, Federico Betti - Tommaso Bianchi
	All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document is a Requirement Analysis and Specification Document (RASD). The purpose of this document is to completely analyze and describe in a formal and non-ambiguous way the system in terms of its intended domain of use, of the goals it should aim at and of the requirements it should enforce. It also provides reasonable constraints within which the system has to behave in a correct way. This document is primarily addressed to the development team, who will be in charge of the implementations of the system in its entirety. Other stakeholders, such as investors, may read the first two chapters as they can be the basis of a contractual agreement.

1.2 Scope

1.2.1 Description of the project

Travlendar+ is a calendar-based service that helps users in managing the scheduling of their meetings, whether for work or personal reasons. The goal of this project is to create a system that:

- automatically computes and accounts for travel time between meetings to make sure that the user is never late
- supports the user in his/her travels, for example by identifying the best mobility option (e.g., use the train from A to B and then the metro to C) while taking into account his preferences (e.g., do not make me walk for more than 15 minutes)

Users can create meetings, and when meetings are created at locations that are unreachable in the allotted time, a warning is created. Travlendar+ should support a multitude of travel means, including walking, biking (own or shared), public transportation (including taxis), driving (own or shared), etc. A particular user may globally activate or deactivate each travel means. A user should also be able to provide reasonable constraints on different travel means (e.g., walking distances should be less than a given distance, or public transportation should not be used after a given time of day). Additional features could also be envisioned, for instance allowing a user to specify a flexible "lunch". For instance, a user could be able to specify that lunch must be possible every day between 11:30- 14:30, and it must be at least half an hour long, but the specific timing is flexible. The system would then be sure to reserve at least 30 minutes for lunch each day. Similarly, other types of breaks might be scheduled in a customizable way.

Some complementary services, such as planners or maps, already exists on the market; however they do not offer both the possibility to schedule events and to have meaningful information on how to travel between them.

1.2.2 Goals

G1 Allow someone to visit the homepage of the system and to register himself providing a valid email, a password and a unique nickname. As an alternative, an external login provider, such as Google+, can be used.

G2 Users can log into the system.

G3 Allow a user to visit its profile and to see a detailed schedule of any day containing all the meetings he is attending and all the travels the system has planned him.

G4 Allow a user to edit all information in its profile (e.g. displayed name, phone number, company, website, social accounts).

- **G4.1** Allow a user to add another one to its contacts.
- **G4.2** Allow a user to create a group and to invite other users into it.
- **G4.3** Allow a user to set his status to auto-decline meetings in a certain period.
- **G4.4** Allow a user to set default locations where he is supposed to be in certain time slots.
- **G4.5** Allow a user to set privacy and notification settings.

G5 Allow a user to create a meeting and to invite other users to attend it.

- **G5.1** Allow the administrator to categorize the meeting.
- **G5.2** Allow the administrator to change title, abstract and location of the meeting.
- **G5.3** Allow the administrator to nominate other administrators.
- **G5.4** Allow the administrator to send invitations and remove participants.
- **G5.5** Allow the team to communicate between them, to share files and to save personal notes about the meeting.
- **G5.6** Allow the invited users to accept or decline the meeting or to propose a rescheduling in a different time slot.
- **G5.7** Allow the administrator to change the date of the meeting after a rescheduling has been proposed.
- **G5.8** Allow the administrator to poll the team to reschedule the meeting; if everyone accepts the rescheduling, the date changes.
- **G5.9** Allow the administrator to create a copy of a meeting with the same team and settings on another date.
- **G5.10** Allow the administrator to see who's late at the meeting.

G6 Create a warning each time it is not possible to reach a meeting location from the previous one in time.

- **G6.1** Allow a user to decline or reschedule overlapping meetings in order to remove the warning.

G7 Allow a user to specify flexible breaks during the day.

G8 Manage users' travels between subsequent meetings, suggesting the best mobility option according to their preference list.

- **G8.1** Allow a user to create a preference list and constraints about the way he wants to travel.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

The following list contains all the main terms we will refer to in the rest of the document, together with a brief explanation of their meaning in this context.

- **User**: anyone that is registered into the system.
- **Nickname**: an alphanumerical string uniquely identifying a user.

- **User's Credentials**: either the pair <email, password>, the pair <nickname, password> or the token, returned by an external login provider; they do identify a precise user and have to be provided in order to log into the system.
- **Guest**: anyone that is not registered into the system.
- **Contact**: users will have the possibility to save other users as their contacts to find them more easily when needed.
- **Break**: a time slot in which a user does not want to have meetings; its exact starting and ending time may vary in a fixed interval to achieve more flexibility.
- **Group**: a collection of users; it may be used to simplify the creation of recurring meetings among the same people.
- **Status**: a flag indicating if a user is in normal or auto-decline status; in the latter case, all incoming request of meeting will automatically be declined.
- **Position**: the real-time latitude and longitude representing where a user is.
- **Location**: the latitude and longitude of a meeting or a user as inferred by the information known a priori by the system (e.g. the location inserted when a meeting is created).
- **Default Location**: users can provide default locations to indicate where they are when not at a meeting (e.g. the office); they are represented by a location and a start time and they are assumed to be valid until the start time of another default location.
- **Meeting**: an event involving at least two users taking place in a specific location and in a specific date.
- **Instant Meeting**: a special type of meeting that has only one participant and the same start and end time. It may be useful for organizing everything that has not a real duration (such as picking up someone or something) or that do not require interaction with other users. For all other aspects it is a meeting like normal ones, and in particular it can be incompatible and raise warnings.
- **Category**: a tag representing the type of a meeting (e.g. work, family).
- **Administrator**: a user that can manage most of the settings and data regarding a meeting.
- **Title**: a string to shortly describe the purpose of a meeting.
- **Abstract**: a text to give information about a meeting (e.g. agenda of the day).
- **Participant**: a user that has accepted an invitation to a meeting.
- **Team**: the collection of all participants of a given meeting.
- **Incompatible Meeting**: a meeting that overlaps with another one, that cannot be reached from the previous one in time or that prevents the correct placement of a break.
- **Warning**: a message displayed to a user indicating that he has been invited to an incompatible meeting.
- **Travel mean**: a way to travel between different locations.
- **Walking**: a travel mean in which you always go by foot.
- **Biking**: a travel mean in which you use your bike or a bike sharing service.

- **Driving**: a travel mean in which you use your car, a car sharing service or a taxi.
- **Public Transportation**: a travel mean in which you use metros, trains, buses and/or trams.
- **Constraint**: a restriction on when the system is allowed to suggest a specific travel mean; it will be composed as `subject operator value - target` (e.g. `'weather' 'is' 'rainy' - 'walking'`, `'time' '>' '21' - 'public transportation'`) and works as follows: if the operator applied to the observed state of the subject and the value yields true, then the target cannot be selected as suggested travel mean.
- **Constraint Subject**: a term representing an element of the world which the constraint refers to (e.g. the weather, the time of the day); it has to be fully observable by the system.
- **Constraint Operator**: a binary function from a pair of subject's states to a boolean value.
- **Constraint Value**: one of the states that characterize the subject (e.g. rainy, an integer).
- **Constraint Target**: the travel mean affected by the constraint.
- **Preference List**: an ordered list of the travel means where the highest ones are those preferred by a user; travel means that do not appear on this list are considered as non-usable by the user.
- **Weighted Preference List**: a sorted list of travel times calculated from the travel means of the preference list where (i * k) minutes have been added to the i^{th} element; it is a way not to suggest a user very long travels with their preferred travel means if a much shorter one is possible (e.g. if a user can walk for 4 hours or drive for half an hour suggest to drive even if walking is higher in the priority list than driving).

1.3.2 Acronyms

- **RASD**: Requirement Analysis and Specification Document (the present document).
- **UML**: Unified Modeling Language (a specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems).

1.3.3 Abbreviations

- **[Gn]**: n^{th} goal.
- **[Dn]**: n^{th} domain assumption.
- **[Rn]**: n^{th} requirement.

1.4 Reference Documents

The following list contains the main documents we have taken as reference. A more complete bibliography can be found at the end of this document.

- Assignment: "Assignments.pdf" (can be found in the root directory of the Github repository).
- 830-1993 - IEEE Recommended Practice for Software Requirements Specifications.
- Unified Modeling Language - Version 2.5

1.5 Document Structure

The RASD is organized into 5 main sections:

1. **Introduction**: this section contains a compact and high level view on the problem and on the goals of the system to be implemented in order to address it; it also contains a glossary with the definitions of the main terms we will refer to in the rest of the document.

2. **Overall Description**: this section describes the general factors that affect the product and its requirements, without stating specific requirements. Its contents are still at a high level of abstraction and should provide a background to understand more easily the following sections.

3. **Specific Requirements**: this section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Each requirement is stated in relation to goals and domain assumptions.

4. **UML Modeling**: this section contains the description of some typical scenarios of use of Travlendar+ , together with the relative use case diagrams and sequence diagrams. There are also present some other useful charts, in particular the class diagram modeling the domain of use of the service and some state chart diagrams.

5. **Formal Analysis Using Alloy**: this section contains all the code of a formal representation of the system using the alloy language and the results of the analysis of requirements in such a setting.

6. **Appendix**: this section contains some information about the work behind the drafting of this document, such as the amount of time spent by each of the authors, the tools used to produce all the material and a revision history that keeps track of all the modifications.

2 Overall Description

2.1 Product Perspective

The system will be implemented trying to combine together services already on the market as presented in the document's scope. Since there are no services like Travlendar+ , our system will be developed completely from scratch with the exception of geolocation and mapping services; for these ones, external providers can meet the needs of the system. This will guarantee that they are precise and reliable and will give system managers the opportunity to focus on developing innovative functionalities for the Travlendar+ project.

2.2 Product Functions

2.2.1 Meetings

A meeting is an event organized by a user with a title, a date and a location. The user who creates the event becomes the first administrator of the meeting and consequently can access all the functionalities described in the next sections. Those who are invited to a meeting can accept, decline or reschedule the invitation. The rescheduling request consists in asking the administrators if it is possible to change the date of the meeting; this proposal might have been generated because this meeting is incompatible with the user's schedule and signaled by a warning of the system at the moment of invitation.
A meeting has a chat that can be used for organizational purposes and users' interaction. It is possible to upload files and add personal or public notes.
Meetings are divided in categories and subcategories to organize them. There is a special type of meeting called *Instant Meeting* that is a meeting with only one participant and that does not last in time. Instant meetings are exactly like normal ones except for the fact that they are instantaneous (such as bring children at school). Even if they do not have a duration, it is essential to take them into account to organize schedule and travels during the day.

2.2.2 Travels

The system manages the user's travels through scheduled appointments during the day. The system will a priori calculate the duration of the trip from the default location to the appointment's one; from 15 minutes beforehand, it will begin to make use of the user's position to provide precise, real-time indications regarding the departure time and the path to follow. The available travel means are walking, driving (owned or shared), biking (owned or shared) and public transportation. The use of each of them may be restricted or completely inhibited by the user through customizable constraints. The system will propose to the user the best indications according to these constraints and its preference list. The system will then recommend and give real time indications to the user about the first path in the weighted preference list, according to constraints imposed. While computing the best path, system will take into account all of user's meetings during the day and will choose the more suitable travel mean.

2.3 User Characteristics

- **Guest**: a guest is an unregistered user who can visit the system's homepage to learn about the opportunities offered by our service. It may choose to register by providing valid credentials or using an external login provider. Before registering to the system, and becoming a user, the guest cannot access any of the features offered by the system such as meeting attendance and user profile management.

- **User**: the user is a guest who has completed the registration process. In order to have full access to the functionalities offered by the system, it must have provided at least one default location and submitted its preference list. The user has a profile that can be edited by adding a phone number or additional emails. Furthermore the system gives each user the opportunity to create a list of contacts with which they can organize meetings more easily. The user can be part of groups, such as colleagues working on the same project, to organize frequent meetings with the same people. In addition, the system provides real time information about the best path to reach the next meeting and organizes user's breaks according to the schedule of the day.

- **Administrator**: the administrator is a user who has created a meeting after defining a title, a date, a location and the participants. After the meeting has been created, the administrator can add or remove participants or designate other administrators. If a user responds to the participation request by proposing a rescheduling, the administrator can accept, decline or forward the reply to all members of the team; however the administrator has the authority to propose a change of date by itself.
  It can upload files both before the meeting, to provide a complete program on the topics to all participants, and after the meeting, to share results with the team. These files can also be maintained and updated for subsequent meetings. In fact, the administrator has the opportunity to re-propose a meeting with the same people, keeping the topic and files unchanged, on a new date.
  The system gives the administrator the power to monitor in real time whether a participant will arrive late - above a fixed threshold for privacy reasons - thanks to the system's ability to trace the position of the participants.

- **System Manager**: the system manager is an employee of Travlendar+ who has the ability to maintain and update the system. It does not have an account that it can use to interface with other users but acts on the system to add new features or improve existing ones.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain Assumptions

D1 All emails are valid according to RFC3696[1] standard.

D2 All emails belong to the user who enters them into the system.

D3 External login systems don't fail and always provide a valid email to identify the registered user.

D4 All users complete their profile right after the registration phase.

After registration, users must complete the profile to be able to use the system. They have to add a nickname, which must be unique, complete the preference list about travel means and add at least one default location. This is needed to prevent inconsistencies in the system (e.g. the system cannot infer the location of a user who has never participated to a meeting if it has not a default position).

D5 Each user has at least one default location.

The default location is where a user is considered to be located during the day. The system will compute the time when the user should start the travel to a meeting from the default location. Only after that time, the system will use the user's position to provide real-time information. This in needed to take into account the fact that, for example, someone may want to go back to his office between a meeting in the morning and one in the afternoon.

D6 Users can have lunch everywhere.

This assumption allows the system to consider that a user can always have a break without any necessary movement.

D7 Each user has a preference list.

D8 All users always have a position.

This is needed for the system to provide the user real-time information about the trip.

D9 External shortest path provider is always able to retrieve a path between any two locations.

D10 Each user is always able to communicate with our servers.

D11 The system does not differentiate between a travel mean that is shared and one that is owned.

A shared car can be considered exactly equal to a owned car because it takes the same amount of time to complete a certain path and it would be in the same position in the preference list. This assumption is taken to avoid the complexity to keep track of things like where an owned car is parked and how to get there. It also avoids the presence of mixed travel means in the system (i.e. take a bus up to your car and then drive it), that would be difficult to deal with, especially from the constraints point of view.

D12 The system treats taxis as a driving travel mean and not as public transportation.

This assumption is supported by the fact that taxis will take the same amount of time as a car to complete the path and, exactly like cars, can be considered to be always present in the same location of the user. This will also simplify the development process as most of external shortest path providers available on the market take the same assumption.

D13 If a user accepts the invitation to a meeting, then he really attends to it.

This assumption is needed to always correctly infer the location of a user. In fact, without this assumption a user could not attend a meeting and therefore the travel indications to go from that meeting to the following one could be completely wrong.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user will be able to interact with the system via a web interface and a mobile application, available for all major browsers and mobile operating systems. These two views will be optimized for different screen sizes and different scenarios of use (e.g. a 24 inch pc display in an office or a 5 inch display on a phone outside), but they will share a clean and modern look together with the same color scheme.
Below you can find some example mockups that show a part of the user interface; they are not intended to be a complete and exhaustive set, but rather to act as a guide for future implementation.

Figure 1: App Calendar

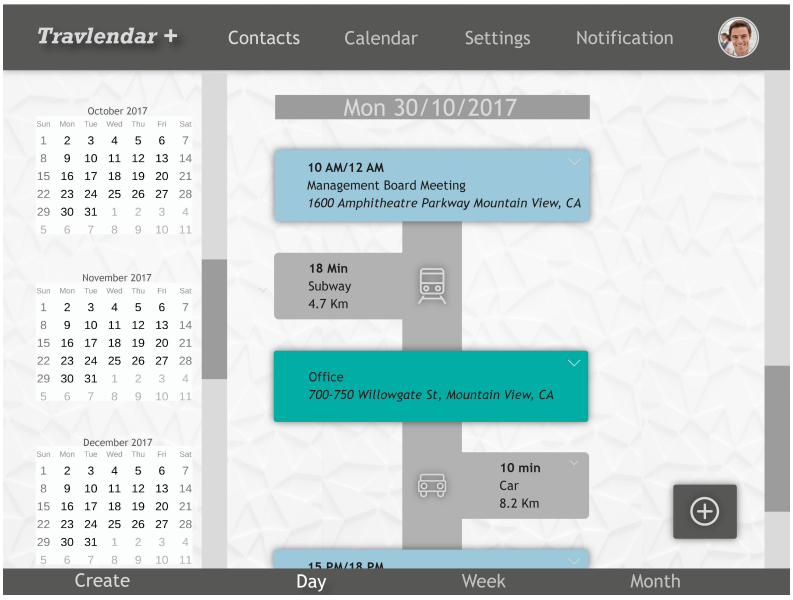


Figure 2: Web Calendar

Figure 3: App Warning

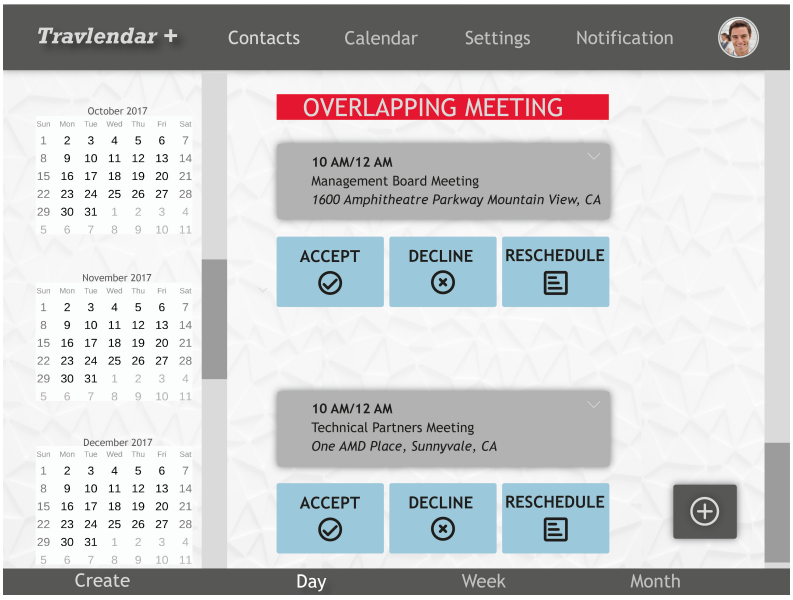


Figure 4: Web Warning

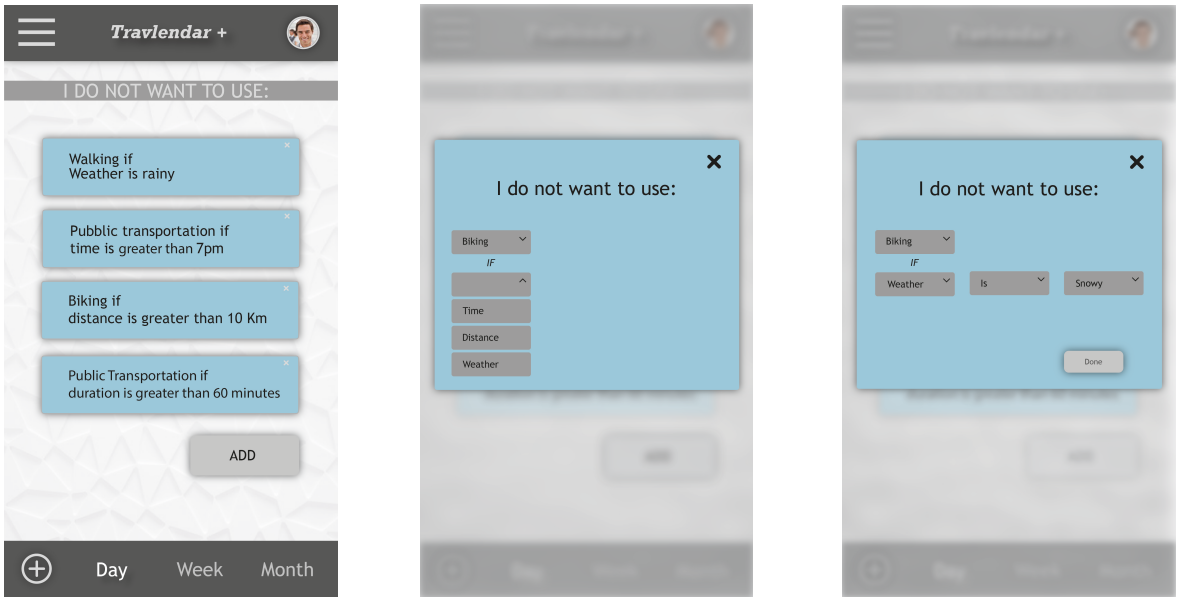


Figure 5: App Constraints

3.1.2 Hardware Interfaces

The system won't include any particular or dedicated hardware, and so no interface has to be devised.

3.1.3 Software Interfaces

The system will need to interact with a data management system, with external login providers and with an external shortest path provider.
For the first one, one of the many relational database on the market will be chosen as they all generally meet the needs of this project; a library or a framework will be used to interface with it, in accordance to the design and implementation choices that will be made.
As regards external login providers, the main ones will be supported in order to reach as many users as possible without too much development effort; the interface with them is usually dependent on the particular provider, so they have to be implemented on a case by case basis, perhaps with the help of a library.
For the external shortest path provider, a single one will be chosen to handle all different travel means in order to have to manage a single interface.

3.1.4 Communication Interfaces

The system won't include any particular communication protocol, and so no interface has to be devised.
For client-server communication we will use plain HTTP.

3.2 Functional Requirements

G1 Allow someone to visit the homepage of the system and to register himself providing a valid email, a password and a unique nickname. As an alternative, an external login provider, such as Google+, can be used.

- [R1] Each user must provide an email that is not already present in the system.
- [R2] Each user must provide a nickname that is not already present in the system.

- [D1] All emails are valid according to RFC3696[1] standard.
- [D2] All emails belong to the user who enters them into the system.
- [D3] External login systems don't fail and always provide a valid email to identify the registered user.
- [D4] All users complete their profile right after the registration phase.

G2 Users can log into the system.

- [R3] Users have to be registered into the system before logging in.
- [R4] Users have to provide an existing email or nickname and the associated password in order to log in; in alternative they can use a supported external login system.
- [D3] External login systems don't fail and always provide a valid email to identify the registered user.

G3 Allow a user to visit its profile and to see a detailed schedule of any day containing all the meetings he is attending and all the travels the system has planned him.

- [R5] A user must be logged into the system to perform any action except registering and logging in.

G4 Allow a user to edit all information in its profile (e.g. displayed name, phone number, company, website, social accounts).

- [R6] Each user must provide an email that is not already present in the system.
- [R7] Each user must provide a nickname that is not already present in the system.
- [R8] Users cannot have meetings while their status is set to auto-decline.
- [R9] A user cannot have different default locations sharing the start time.
- [R10] Time travel between subsequent default locations should be less than the difference between their start time.
- [R5] A user must be logged into the system to perform any action except registering and logging in.

G5 Allow a user to create a meeting and to invite other users to attend it.

- [R5] A user must be logged into the system to perform any action except registering and logging in.
- [R11] Each meeting has at least two participants.
- [R12] Each meeting has at least one administrator.
- [R13] Each meeting has a title, a date and a location.
- [R14] Each participant in a meeting can access shared files and the chat.
- [R15] Users participate in a meeting if and only if they accept the invitation.
- [R16] Users do not participate in a meeting if they decline the invitation.
- [R17] Users can write in the chat of a meeting if and only if they have received and accepted an invitation to it.

G6 Create a warning each time it is not possible to reach a meeting location from the previous one.

- [D13] If a user accepts the invitation to a meeting, then he really attends to it.

G7 Allow a user to specify flexible breaks during the day.

- [R5] A user must be logged into the system to perform any action except registering and logging in.
- [R18] The system suggests you a time, according to your settings, to have a break such that no meeting overlaps with it; if no time slot is valid, a warning is generated.
- [D6] Users can have lunch everywhere.

G8 Manage users' travels between subsequent meetings, suggesting the best mobility option according to their preference list.

- [R5] A user must be logged into the system to perform any action except registering and logging in.
- [R19] At least one travel mean is available in the preference list.
- [R20] The travel mean suggested by the system is always the first in the weighted preference list that satisfied all the constraints; if no travel mean satisfied all the constraints than the system suggests the fastest one.
- [D5] Each user has at least one default location.
- [D7] Each user has a preference list.
- [D8] All users always have a position.
- [D9] External shortest path provider is always able to retrieve a path between any two locations.
- [D10] Each user is always able to communicate with our servers.
- [D11] The system does not differentiate between a travel mean that is shared and one that is owned.
- [D12] The system treats the taxis as a driving travel mean and not as public transportation.
- [D13] If a user accepts the invitation to a meeting, then he really attends to it.

3.3 Performance Requirements

A system such as Travlendar+ may achieve exponential growth of users in a short time after appropriate marketing policies. For this reason, the system must be able to manage a large number of simultaneous requests and therefore have high-performance and scalable infrastructures. The systems used for saving data must be efficient, because with the growth of users, the increase in the amount of data about meetings, user profiles and transport means would be considerable.
It is expected that the number of users will grow up to 50000 in a short time because the system is an innovative service that could significantly simplify workers' lives. The main targets are big cities with a high number of people and companies. The system functionality that offers the possibility to add contacts and create groups could facilitate Travlendar+'s expansion.

3.4 Design Constraints

Since there is no pre-existing software on which to work, being Travlendar+ developed from scratch, there are no particular design or compatibility constraint to be considered. The hardware on which the system will operate will have the same needs as a common web server, thereby a standard external housing solution will be considered.

3.5 Software System Attributes

- **Reliability**: The reliability of the system is a crucial point in the development process since it is not possible to provide a professional organization service that contains failures. System managers should ensure that the system will be fully tested and error-free before entering the market. All of this, combined with a high maintenance speed in case of unexpected errors, is the starting point to provide high availability.
- **Availability**: Since the system must guarantee a complete service to the user, it must ensure 24/7 operation. Especially in view of a future global expansion, the service will have to offer all the functions 24 hours a day and for this reason it will be able to afford only a few exceptions from the 24/7 target. In the first implementation, however, as the system will operate only in a single country, availability requirements may be a little relaxed during the night time.
- **Security**: During registration phase or through subsequent changes, users will provide the system confidential and sensitive informations that must be properly protected to ensure their privacy. In addition, the system must also be designed in such a way that a user cannot access another user's schedule.
- **Maintainability**: Maintainability is essential because this system is created to allow future expansion and integration of new functionalities. This is possible only if the software has excellent characteristics in terms of testing, understandability and code modifiability (given by a great structural layout).
- **Portability**: The system is structured to be fully cross-platform. Its functionalities must be accessible by both web interfaces and smartphone applications.

4 UML Modeling

4.1 Scenarios

4.1.1 Scenario 1

Alice is a manager of a large company and mother of two children. She is always on the go, both to attend meetings all around the city and to bring her sons around. Up until now, she has tried her best to manage all this with her old style agenda, with the only result of being often late and under constant stress. Yesterday a colleague told her about a fantastic new service, Travlendar+ , that seems to be a perfect fit for her needs, so she immediately decides to try it out. Now she has a few minutes of break, so she decides to register to the system and customize her profile. She adds a profile picture, her phone number and her Facebook account.

4.1.2 Scenario 2

Bob is the CEO of a new, innovative startup, still substantially unheard of. His main concern now is to attract potential investors by describing his idea to as many of them as possible, so he is constantly around to attend meetings and to meet new people. To reach his goal, he has used Travlendar+ throughout last month and now he has a good number of contacts saved. Now he feels ready to try to arrange a meeting with the executives of a large company that he thinks should be interested in what he does; thereby he create a meeting on Travlendar+ inviting both them and a couple of his contacts, who he would like to get into his team if things were to get going.

4.1.3 Scenario 3

Charlie is a freelance web developer, so he has to frequently meet with all the different companies that hire him. To do so, he uses a popular service named Travlendar+ , which allows him to always avoid to arrange meetings in a way that will make impossible for him to attend them all. Today, he received an invitation from Dave, social media manager of a company for which Charlie maintains a website since years. All Dave's team has been invited, since they have to discuss some fairly substantial improvement on the graphics. The proposed date for the meeting is Thursday at 7 PM, but Charlie is a commuter so for him it is a problem as he has to take a train. Therefore he propose to reschedule the meeting on the subsequent day, at 4 PM, and Dave forwards this request to his team as he knows that it may be a problem for some of them to work till late also on Friday.

4.1.4 Scenario 4

Eve is a fashion store manager in Milan. During the Milan Fashion Week her agenda is full of meeting with brands, models and stylists. One of her colleague some weeks earlier suggested Eve to use Travlendar+ to manage her schedule and Eve, although initially a little skeptical, decides to try this new service and downloads the application. As hers nutritionist recommend, Eve sets a small break of 15 minutes in the afternoon between 4.00 PM and 6.00 PM to have a healthy snack and avoid stress caused by daily travels. In hers Travlendar+ 's agenda she has a meeting up to 5.20 PM and she has been invited to an appointment with a stylist at 6.00 PM on the other side of Milan. The system computes that, due to traffic jam, Eve should leave immediately after the previous meeting. This would prevent Eve doing her break so a warning is created to signal it. Eve, who had completely forgotten this chance, thanks Travlendar+ and decides to reschedule the second meeting half an hour later, with the button on the application. The stylist accepts the proposal and Eve manages to have her break and stay fit.

4.1.5 Scenario 5

Freddie is a pretty busy university professor. His Travlendar+ schedule is always full of meetings everywhere in London. He's deeply aware of today's pollution problems, so he always tries to use cars as less as possible. He's also somehow uncomfortable in getting by bike to formal meetings, so his preference list looks as follows:

1. Walking
2. Public Transportation
3. Driving

He has also setup some constraints, in particular because he's getting old and so he cannot walk for long distances or under a heavy rain. Moreover, he believes it is not wise to travel alone after the sunset so, in that case, he resorts to the car even if unwillingly. Here there are his constraints:

- Distance > 2 km - Walking
- Weather is rainy - Walking
- Time > 18 - Walking
- Time > 18 - Public Transportation

Now he has just accepted a meeting for Tuesday morning on 11 AM and the system has to calculate a few things in order to manage it correctly. Initially, all the travel times from the previous meeting and from his default location (that happens to be his office at 11 AM) are calculated, giving the following results:

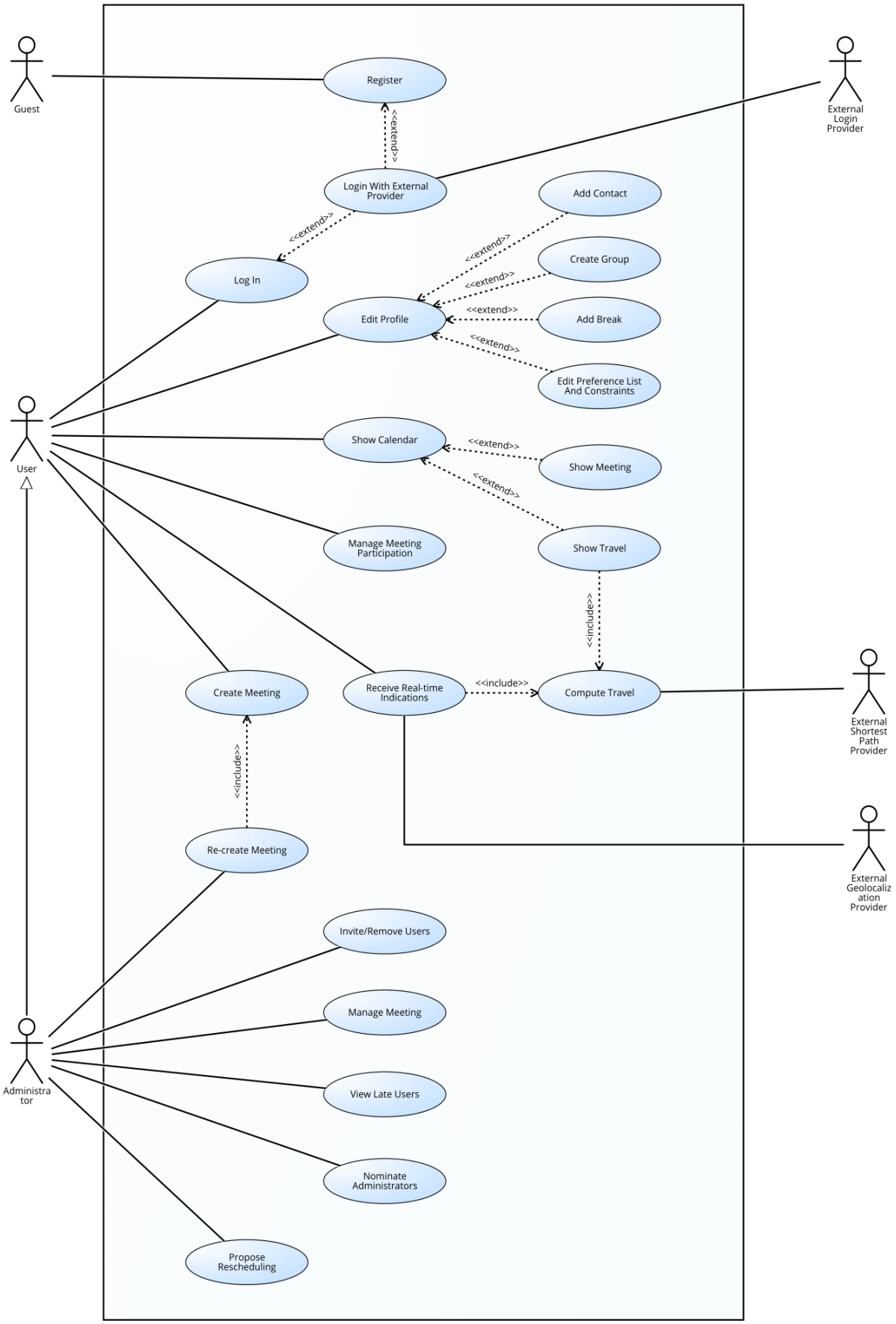
1. Walking from previous meeting 21 minutes, from office 53 minutes
2. Public Transportation from previous meeting 11 minutes, from office 23 minutes
3. Driving from previous meeting 8 minutes, from office 15 minutes

Then, all constraints are taken into account, eliminating the possibility to go by foot from the office as it is longer than 2 km. The previous meeting ends at 10 AM, so no warning has to be created. The selected travel means are walking from the previous meeting and public transportation from the office; as the slowest of the two takes 23 minutes, this value is saved for later use. On the day of the meeting, starting from a little before than 23 minutes before 11 AM, the real-time position of Freddie is used to calculate the distance to the meeting location and, if necessary, to notify him that he needs to leave in order not to be late.

4.1.6 Scenario 6

George is a manager of a construction company working in Modena and its province. After the 2012 earthquake lots of monuments and houses need to be reconstructed so his agenda is full of meeting with clients and architects. His company has decided to use Travlendar+ to organize all the meetings. George's secretary has create a meeting with some architects and engineers and has nominated George administrator.
After the meeting, George decides to make a summary of all the achievements reached so far and to upload that on the meeting's homepage. In this way the document can be seen by all the participants. The team has to organize another meeting to complete its goals and George, as meeting administrator, uses the Travlendar+ functionality that allows to create a new meeting with the same participants, keeping uploaded files. The following meeting will be in the same location, two weeks after. The system automatically invites all the participants to the second meeting. This is a meeting exactly like others, so each participant has to accept, decline or propose a rescheduling.

4.2 Use Case Diagrams



Actors	Guest, External Login Provider
Goals	G1
Input Conditions	A guest is on the homepage of the system.
Events Flow	1. The guest inserts its main emailaddress. 2. The guest enters its password. 3. The guest clicks on the "Sign Up" button.
Output Conditions	The guest is registered into the system with a pair <email, password>. The guest is redirected to a page where it can add to its profile the compulsory information needed to use the system, that are: the preference list, a unique nickname and at least one default location.
Exceptions	The guest inserts a wrong or already-used email. In this case the system shows the user a message reporting the errors and waits for another attempt.

Table 1: Registration (view Sequence Diagram)

Actors	User, External Login Provider
Goals	G2
Input Conditions	User is on the homepage of the system and it has not yet been authenticated.
Events Flow	1. The user clicks on the "Log in with `Service name`" button. 2. The user is taken to an external page where it has to follow the authentication procedure of the external login provider. 3. The user is redirected to our system with a valid token.
Output Conditions	User is logged into the system.
Exceptions	The external login provider procedure cannot be completed correctly. In this case the system shows the user a message reporting the error.

Table 2: Login with External Provider (view Sequence Diagram)

Actors	User
Goals	G2
Input Conditions	User is on the homepage of the system and it has not yet been authenticated.
Events Flow	1. The user inserts its main email address or its nickname. 2. The user inserts its password. 3. The user clicks on the "Log in" button.
Output Conditions	User is logged into the system.
Exceptions	The user credentials are wrong. The system shows the user a message reporting the errors and waits for another attempt.

Table 3: Login (view Sequence Diagram)

Actors	User
Goals	G4
Input Conditions	The user has already been authenticated by the system.
Events Flow	1. The user clicks on the "Edit" button on its profile page. 2. The user selects a field to edit. 3. The user inserts new data for the selected field. 4. The user can repeat from point 2 or click the "Save" button. 5. The system redirects the user to its profile page.
Output Conditions	The selected fields are updated with the new data inserted by the user.
Exceptions	The data inserted in a field are wrong (e.g. a non-unique nickname, a non-unique email, a syntactically wrong website, etc.). The system shows the user a message reporting the errors, clears the incorrect fields and restarts the event flow from point 2.

Table 4: Edit Profile (view Sequence Diagram)

Actors	User
Goals	G3
Input Conditions	The user has already been authenticated by the system.
Events Flow	1. The user clicks on the "Calendar" button in the menu. 2. The system generates a page containing all the meetings and the travels of the day. 3. The user may navigate to a week view and to a month view. 4. The user may click on a meeting or on a travel to see it in detail.
Output Conditions	This use case does not modify anything, so the state of the output is equal to the state of the input.
Exceptions	There are no exceptions.

Table 5: Show Calendar (view Sequence Diagram)

Actors	User
Goals	G5.6
Input Conditions	The user has already been authenticated by the system and has been invited to attend a meeting.
Events Flow	1. The user selects a meeting it wants to manage by clicking on it in the "Calendar" page. 2. The system displays all the information about the selected meeting, such as the title, the date, the location and the other invited users. 3. The user clicks on the "Accept", "Decline" or "Reschedule" button. 4. In the latter case, the user selects a new proposed start date.
Output Conditions	The acceptance, refusal or rescheduling proposal of the meeting is recorded by the system. In the latter case, it is forwarded to the meeting administrator.
Exceptions	The user selects an invalid start date (e.g. a date which is in the past). The system shows the user a message reporting the errors and wait for a new date to be inserted.

Table 6: Manage Meeting Participation (view Sequence Diagram)

Actors	User, External Geolocalization Provider, External Shortest Path Provider
Goals	G8
Input Conditions	The user has already been authenticated by the system and the system has detected that the start time of one of its meeting is approaching.
Events Flow	1. The system notifies the user that it's time for him to leave. 2. By using the real-time position of the user, the system keeps giving him indications on where to go and what travel means to use. 3. The user can click on the "Stop Notifications" button to stop the system giving him travel indications.
Output Conditions	The user has reached the meeting location or the user has clicked the "Stop Notifications" button.
Exceptions	There are no exceptions.

Table 7: Receive Real-Time Indications (view Receive Real-Sequence Diagram)

Actors	User
Goals	G5
Input Conditions	The user has already been authenticated by the system.
Events Flow	1. The use clicks on the "Create Meeting" button. 2. The user inserts a start date, an end date, a title and a location for the meeting. 3. The user may add other data, such as an abstract or a category. 4. The user chooses a non-empty list of invited users, either by inserting their email or by selecting them from its contacts; in alternative the user may specify that it is an instant meeting. 5. The user clicks on the "Create" button. 6. The system redirects the user to the new meeting page.
Output Conditions	A new meeting is created and an invitation is sent to all the selected users.
Exceptions	The location of the meeting may be incorrect or the start and end date may be inconsistent (i.e. the start is after the end). The system shows the user a message reporting the errors, clears the incorrect fields and restart the event flow from point 2.

Table 8: Create Meeting (view Sequence Diagram)

Actors	Administrator
Goals	G5.9
Input Conditions	A meeting has been created and completed normally. The team decides that another meeting is necessary to fulfill their goals.
Events Flow	1. One of the administrators clicks on the "Recreate Meeting" button. 2. The system shows a meeting creation form where title, abstract, uploaded file and participants fields are pre-filled with the previous meeting's values. 3. The administrator chooses the location and the date for the new meeting. 4. The administrator clicks on the "Create" button.
Output Conditions	A new meeting is created and an invitation is sent to all the selected users.
Exceptions	The location of the meeting may be incorrect or the start and end date may be inconsistent (i.e. the start is after the end). The system shows the user a message reporting the errors, clears the incorrect fields and restart the event flow from point 2.

Table 9: Recreate Meeting (view Sequence Diagram)

Actors	Administrator
Goals	G5.4
Input Conditions	A meeting has been created.
Events Flow	1. One of the administrators clicks on the "Add Participants" button on the meeting page. 2. The administrator selects some users either by choosing from its contacts or by inserting their nickname or email. 3. The Administrator clicks on the "Send" button to send invitations.
Output Conditions	The selected users receive meeting invitations.
Exceptions	The administrator may have write a wrong or non-existent email or nickname. The system shows the administrator a message reporting the errors, clears the incorrect fields and restarts the event flow from point 2.

Table 10: Invite Users (view Sequence Diagram)

Actors	Administrator
Goals	G5.4
Input Conditions	A meeting has been created.
Events Flow	1. One of the administrator clicks on the "Show Participants" button on the meeting page. 2. The system shows the list of all users that have been invited to the meeting. 3. The administrator can remove users from the participants list by clicking on the "Delete" button next to their name.
Output Conditions	The selected users are removed from the team.
Exceptions	There are no exceptions.

Table 11: Remove Users (view Sequence Diagram)

Actors	Administrator
Goals	G5.10
Input Conditions	A meeting has been created and its start time has passed.
Events Flow	1. One of the administrator clicks on the "Who's Late" button on the meeting page. 2. The administrator can contact one of the late participant clicking on his name.
Output Conditions	The administrator knows who is late and can take decisions based on that.
Exceptions	There are no exceptions.

Table 12: View Late Users (view Sequence Diagram)

Actors	Administrator
Goals	G5.3
Input Conditions	A meeting has been created.
Events Flow	1. One of the administrator clicks on the "Show Participants" button. 2. The system shows the list of all the users who have been invited to the meeting, with a "Nominate" button available only next to those who has accepted the invitation. 3. The administrator clicks on the "Nominate" button next to users who wants to nominate administrator.
Output Conditions	Selected users become administrators and gain all the functionalities of the role.
Exceptions	A participant may have decided to leave the meeting while the administrator's "Show Participant" page is open. In this case, if the participant is selected to be nominated as administrator, the system signals it with a message saying "this participant has left the meeting".

Table 13: Nominate Administrators (view Sequence Diagram)

Actors	Administrator
Goals	G5.2 G5.5
Input Conditions	A meeting has been created.
Events Flow	1. One of the administrators clicks on the "Meeting Settings" button. 2. The system shows all the functionalities available, such as edit title, modify abstract or upload files. 3. The administrator can update all the fields. 4. The administrator may chose to upload a file, to share it with the team, by clicking on the "Upload File" button. 5. The administrator selects the file he wants to upload. 6. The administrator clicks on the "Save" button.
Output Conditions	The fields are updated and the selected files are uploaded.
Exceptions	The administrator wants to upload a huge file or a file with an invalid extension. In this case, the system stops the upload of the file and shows the user a message saying "The file is not supported".

Table 14: Manage Meeting (view Sequence Diagram)

Actors	Administrator
Goals	G5.7 G5.8
Input Conditions	A meeting has been created.
Events Flow	1. One of the administrators clicks on the "Send Reschedule Proposal" button on the meeting page or on the reschedule proposal sent by an invited user. 2. The system shows the reschedule form, pre-filled with the reschedule proposal data if the request comes from a user. 3. The administrator completes the form. 4. The administrator clicks on the "Send" button.
Output Conditions	A rescheduling proposal is sent to all the participants. If everyone accepts than the meeting's date is changed.
Exceptions	The inserted date is incorrect (e.g. in the past) or it does not differ from the previous one. The system shows a message reporting the errors, clears the incorrect fields and restarts the event flow from point 3.

Table 15: Propose Rescheduling (view Sequence Diagram)

4.3 Sequence Diagrams

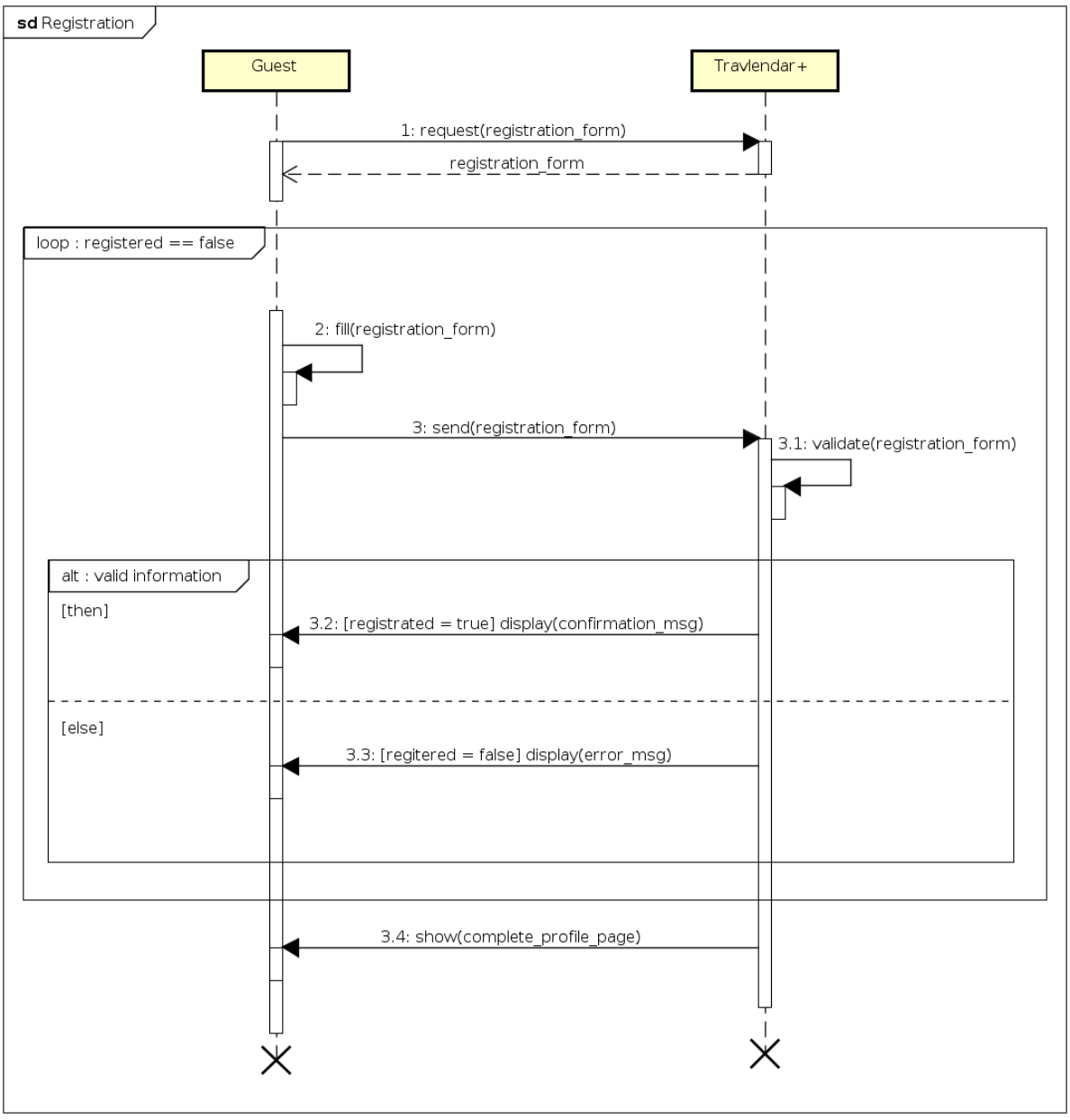


Figure 6: Registration (view Use Case Description)

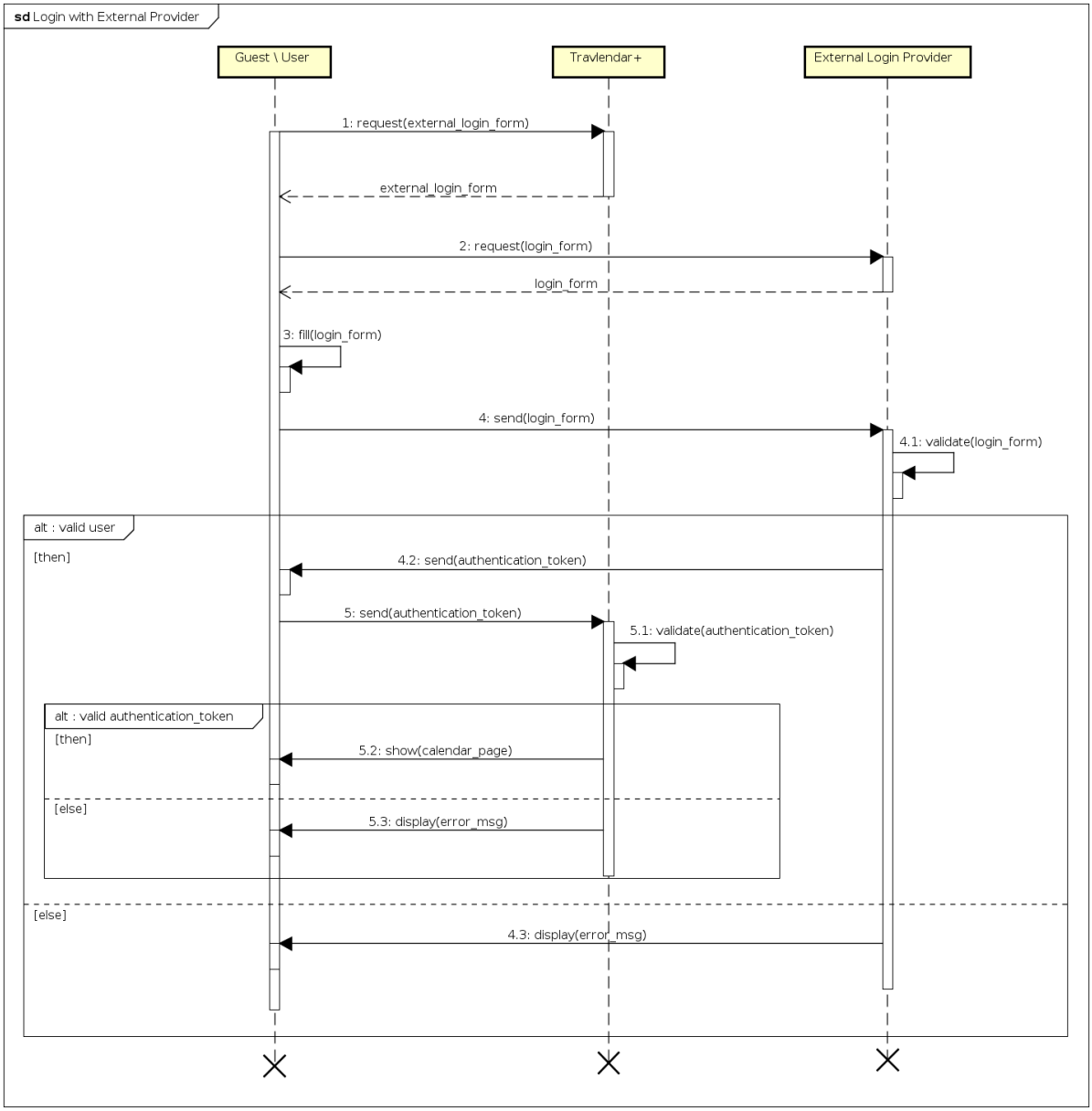


Figure 7: Login with External Provider (view Use Case Description)

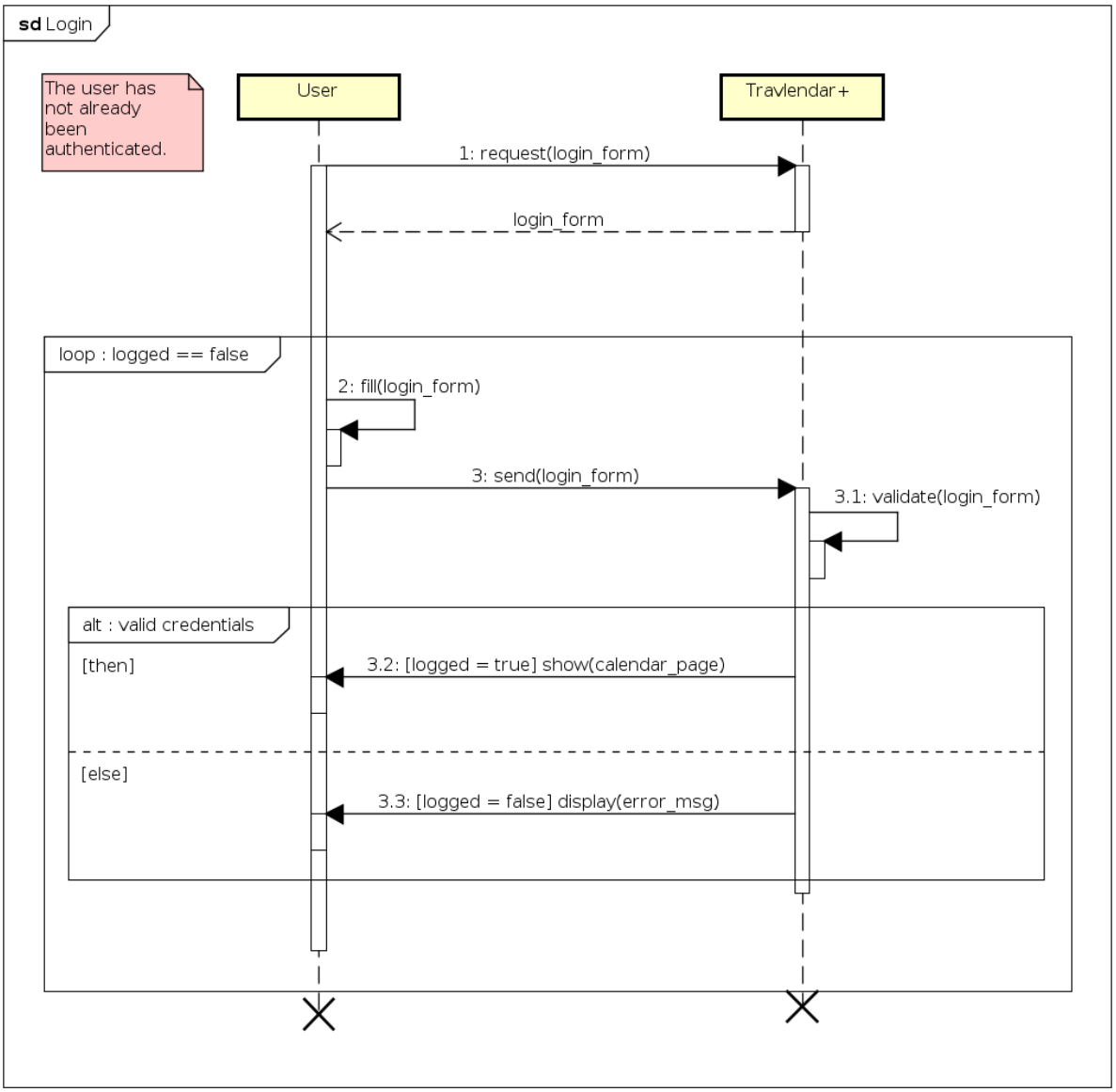


Figure 8: Login (view Use Case Description)

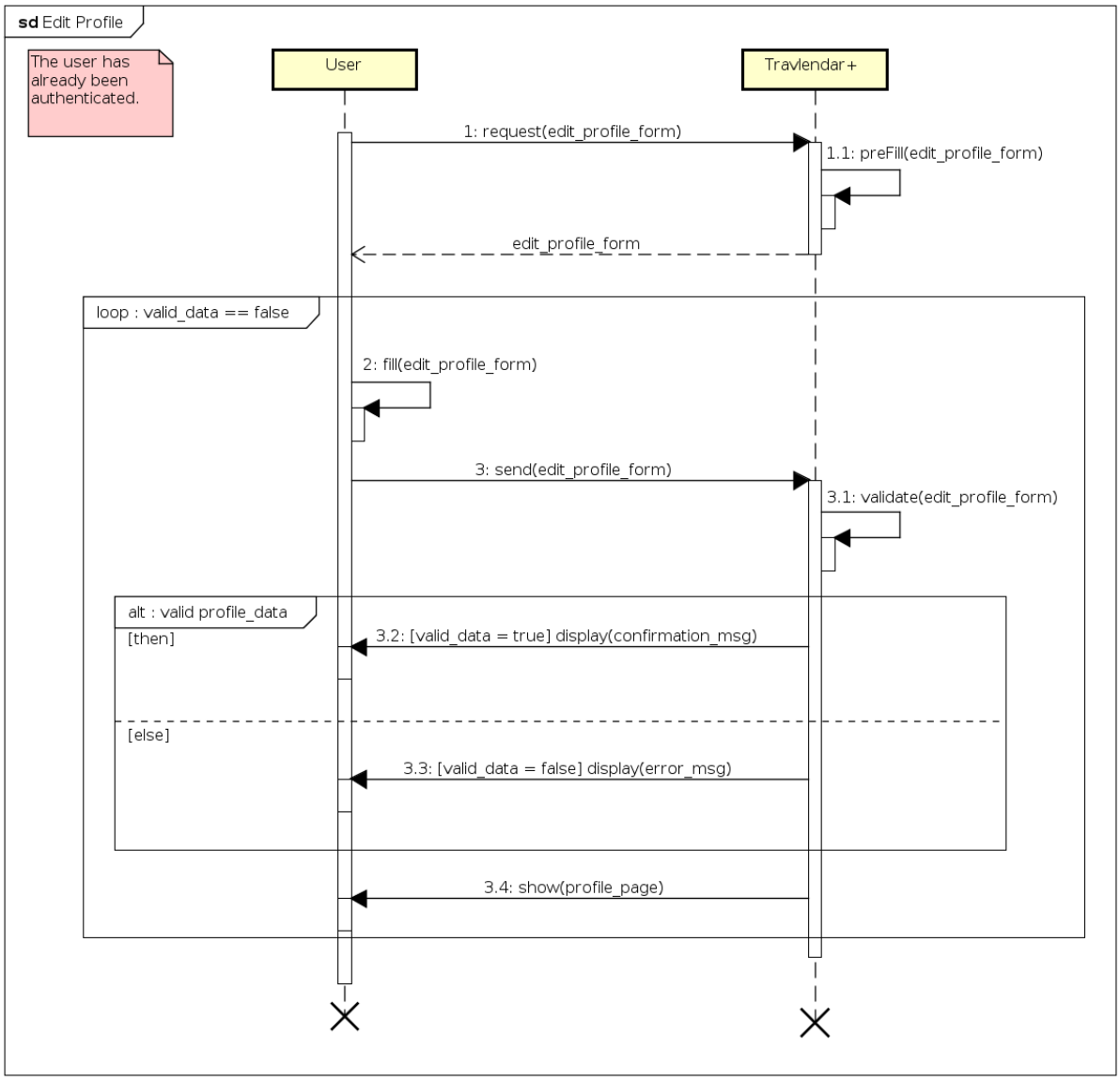


Figure 9: Edit Profile (view Use Case Description)

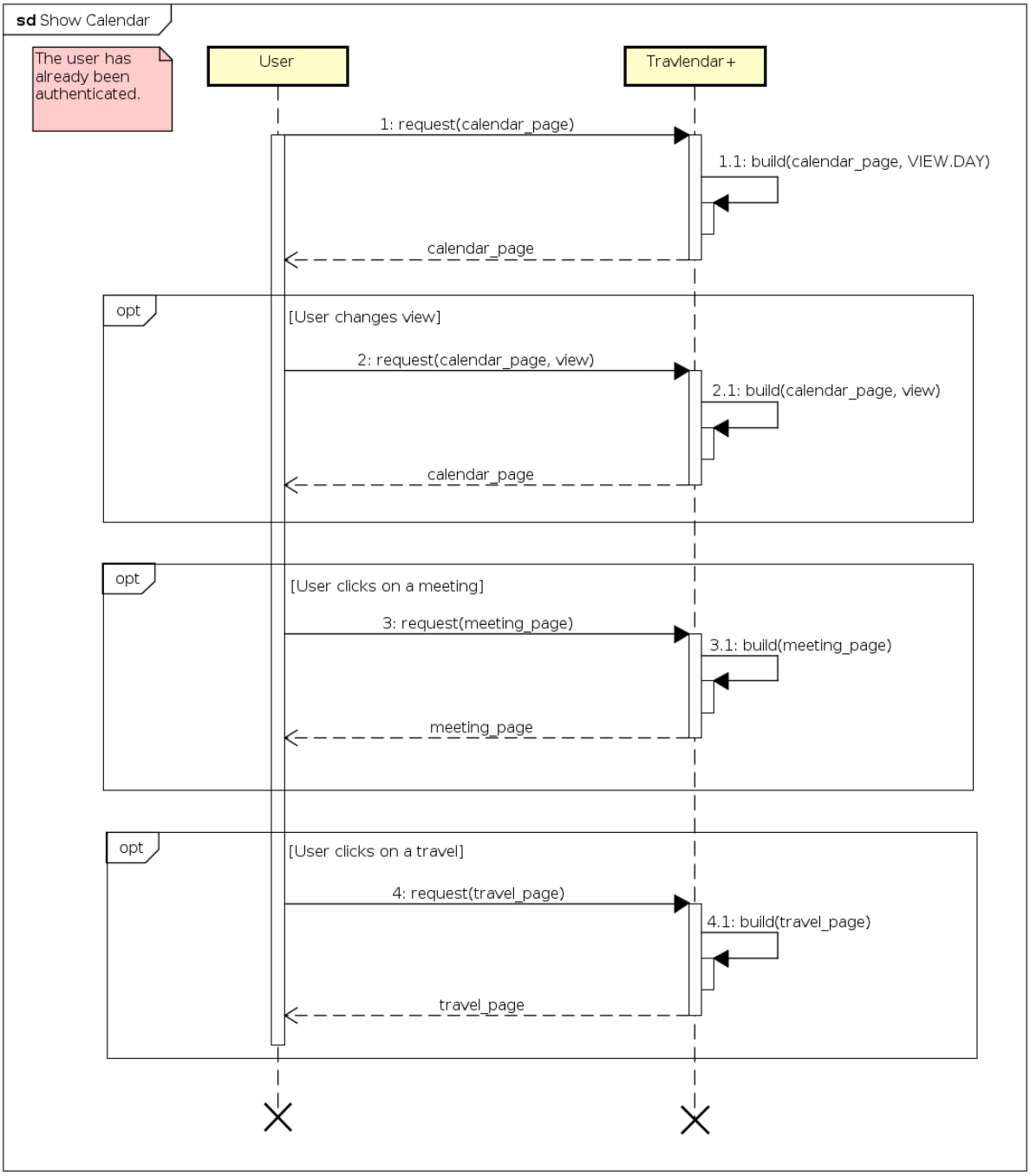


Figure 10: Show Calendar (view Use Case Description)

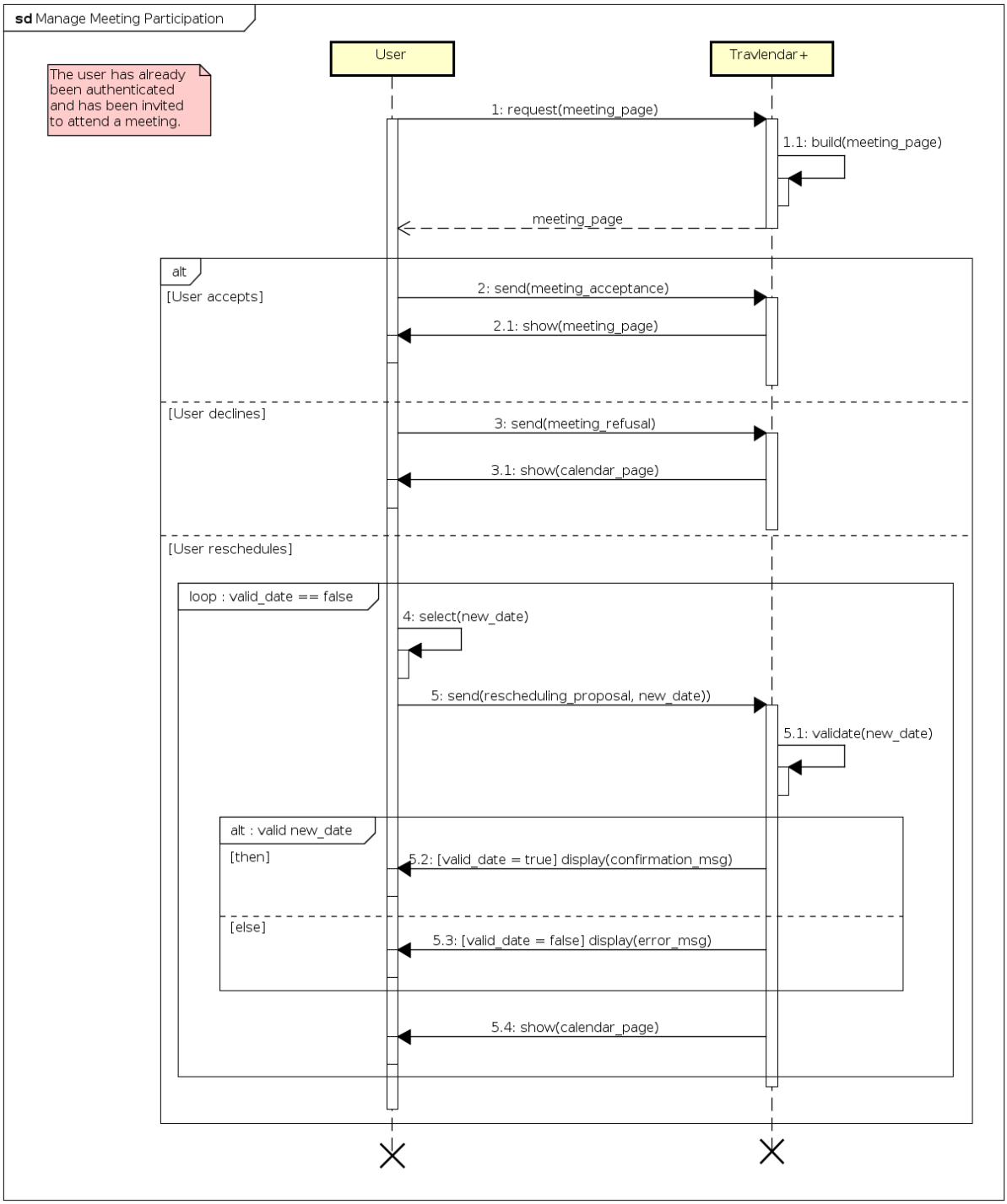


Figure 11: Manage Meeting Participation (view Use Case Description)

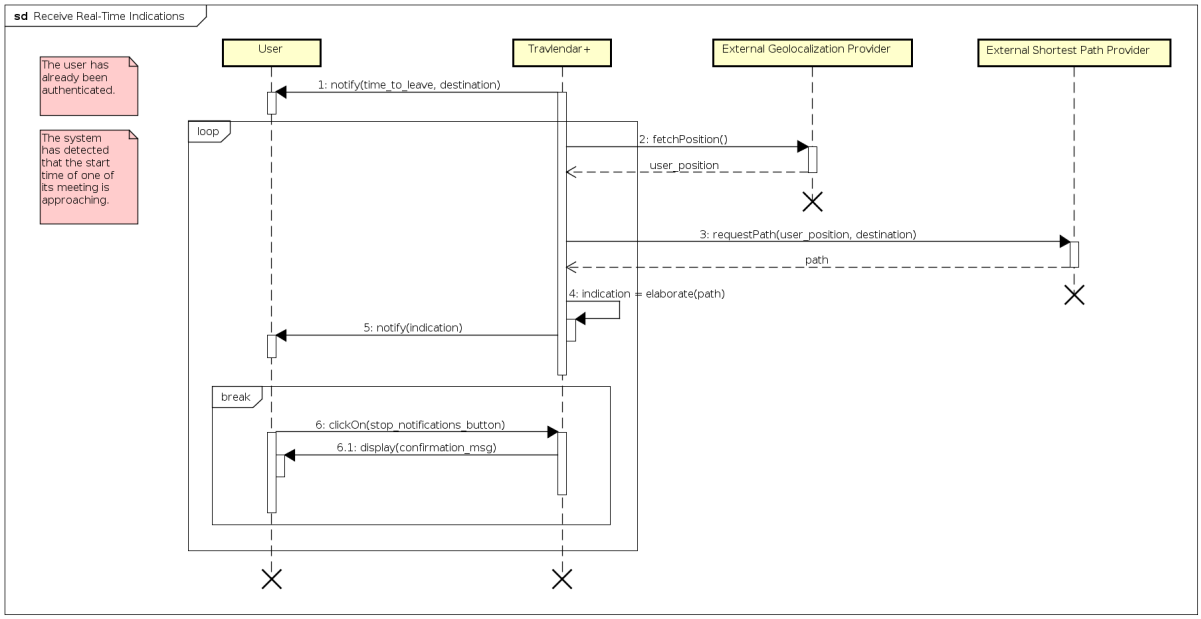


Figure 12: Receive Real-Time Indications (view Receive Real-Use Case Description)

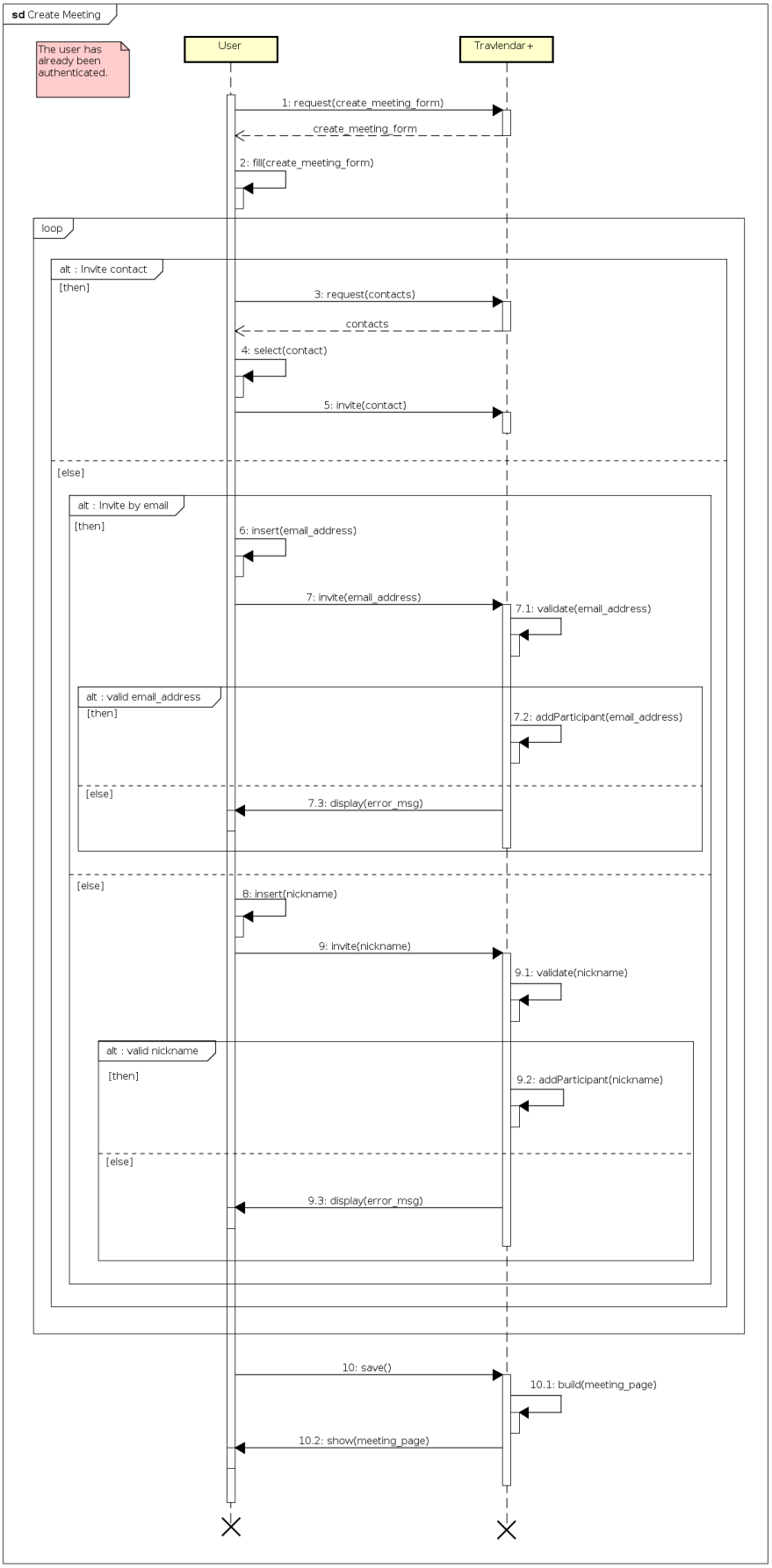


Figure 13: Create Meeting (view Use Case Description)

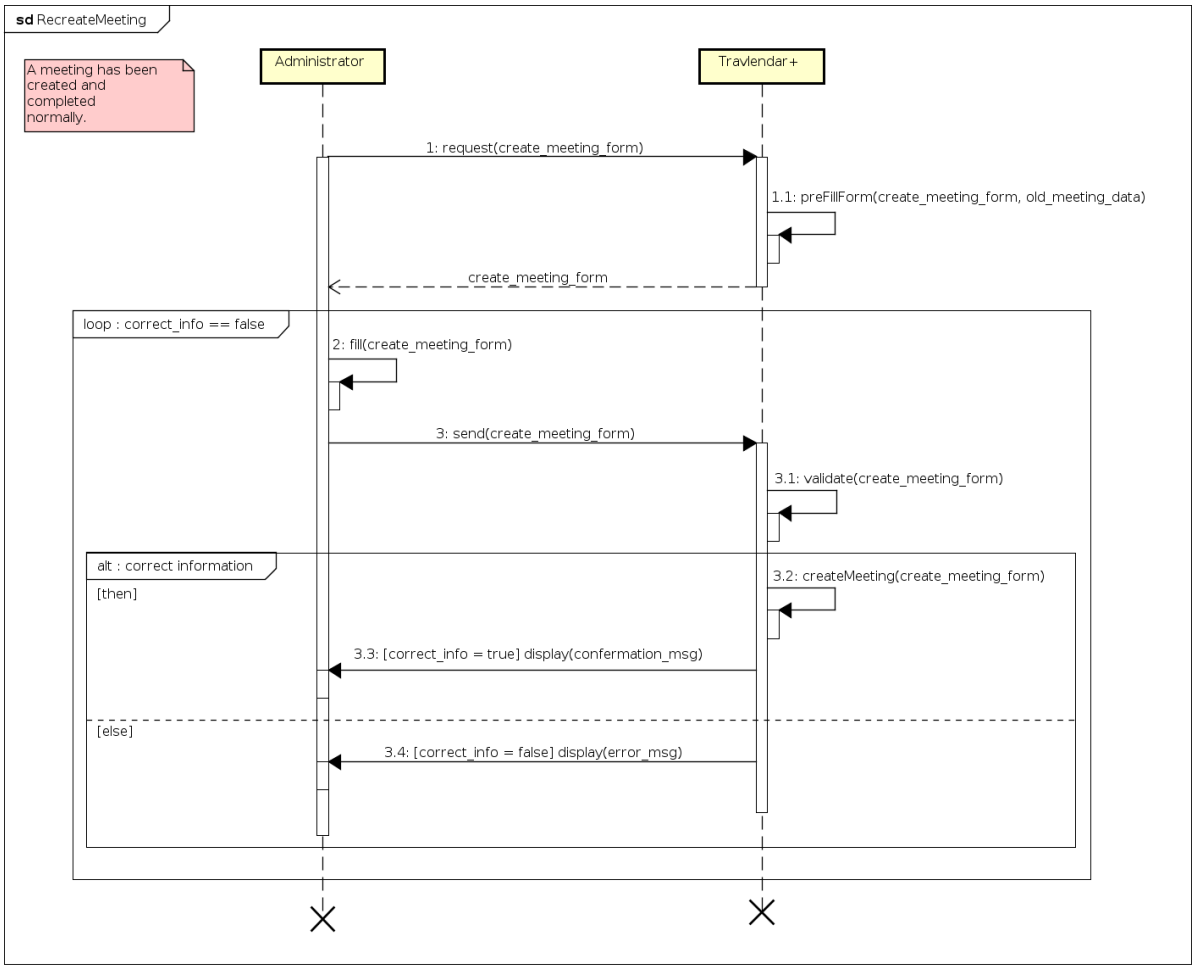


Figure 14: Recreate Meeting (view Use Case Description)

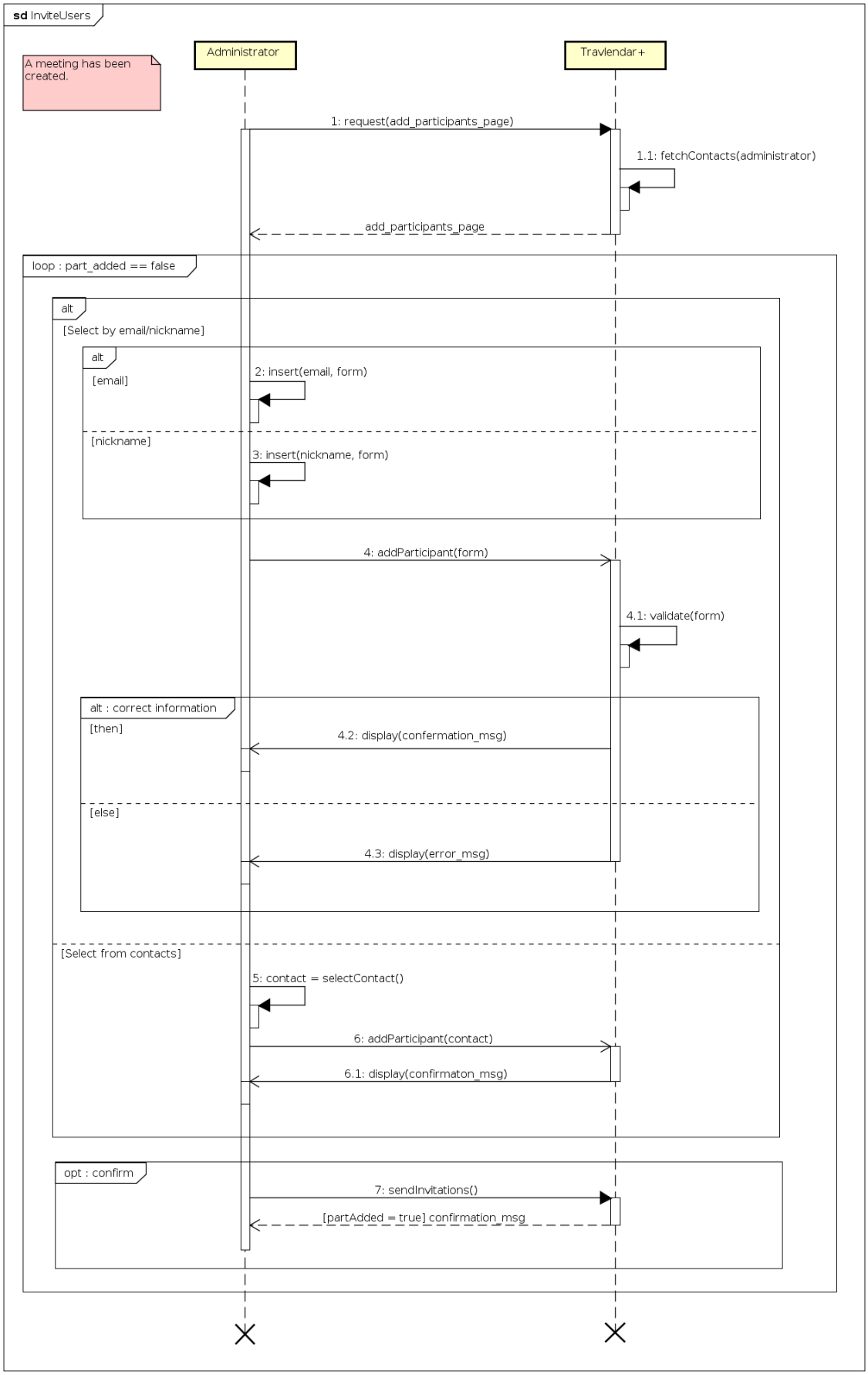


Figure 15: Invite Users (view Use Case Description)

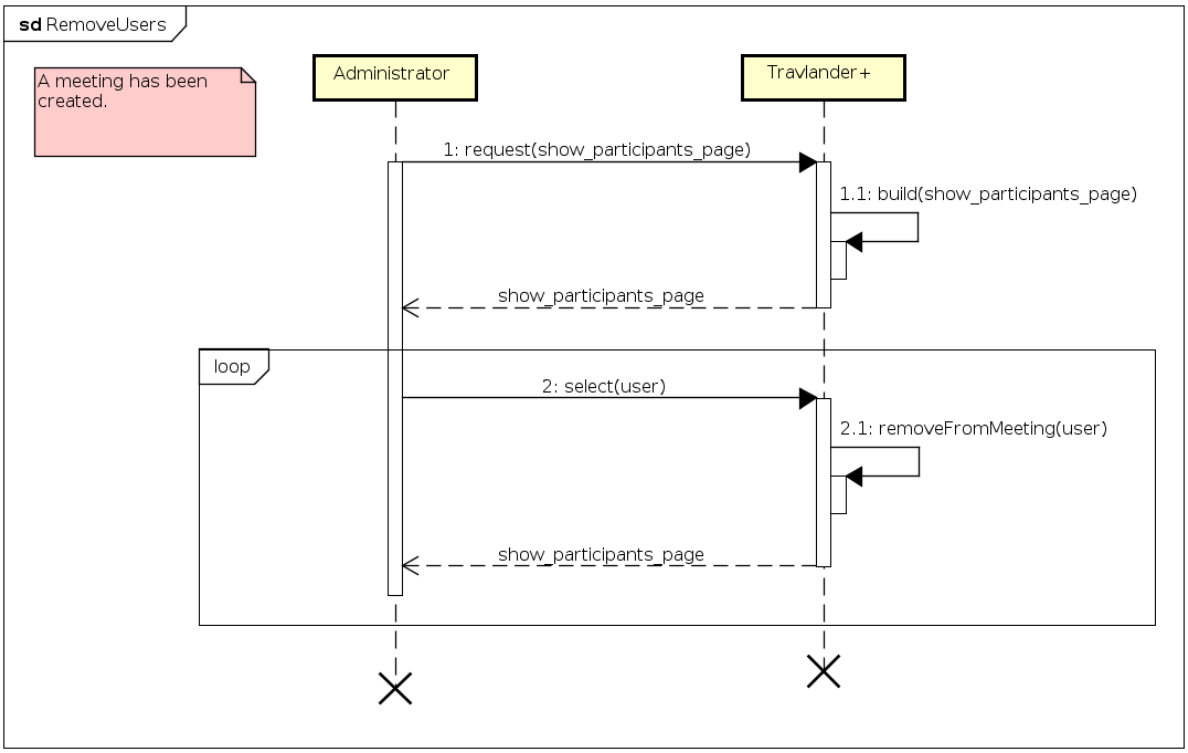


Figure 16: Remove Users (view Use Case Description)

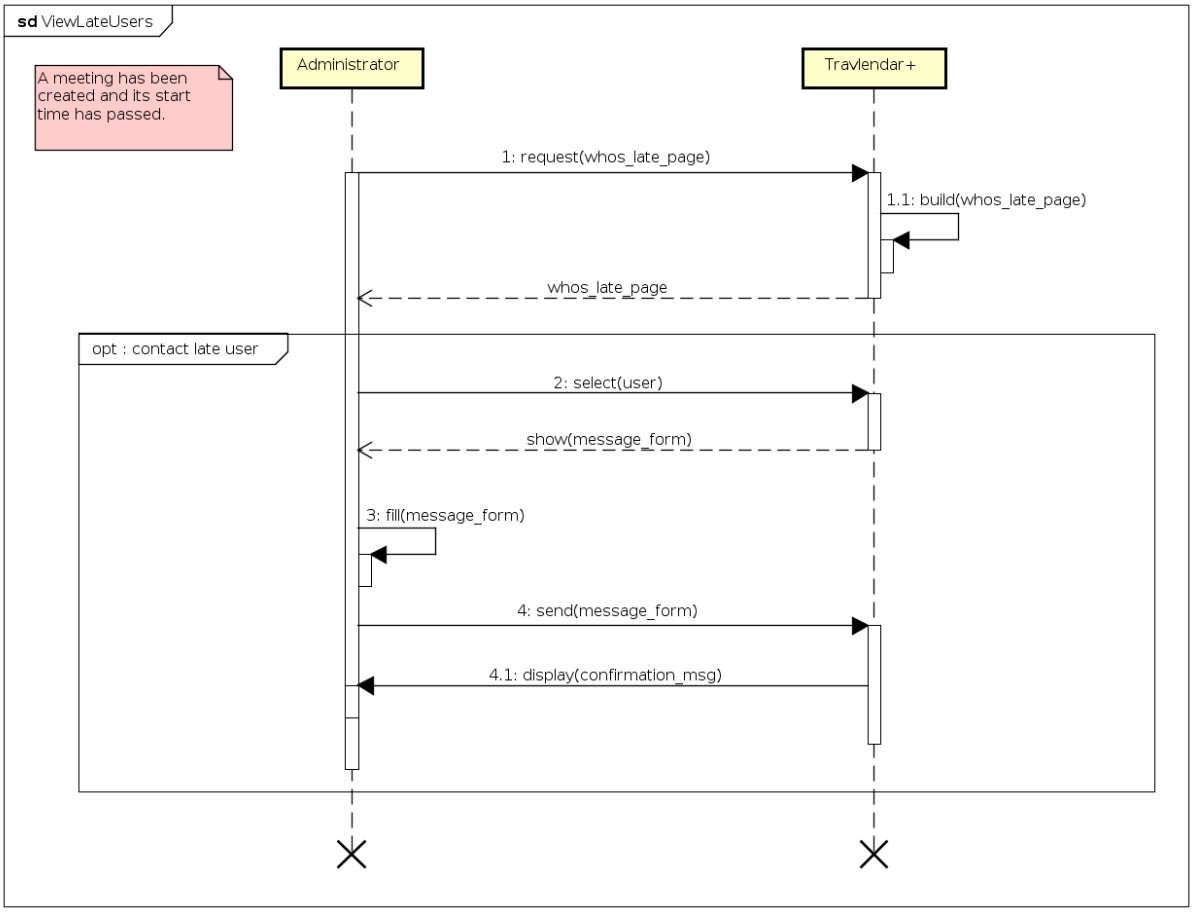


Figure 17: View Late Users (view Use Case Description)

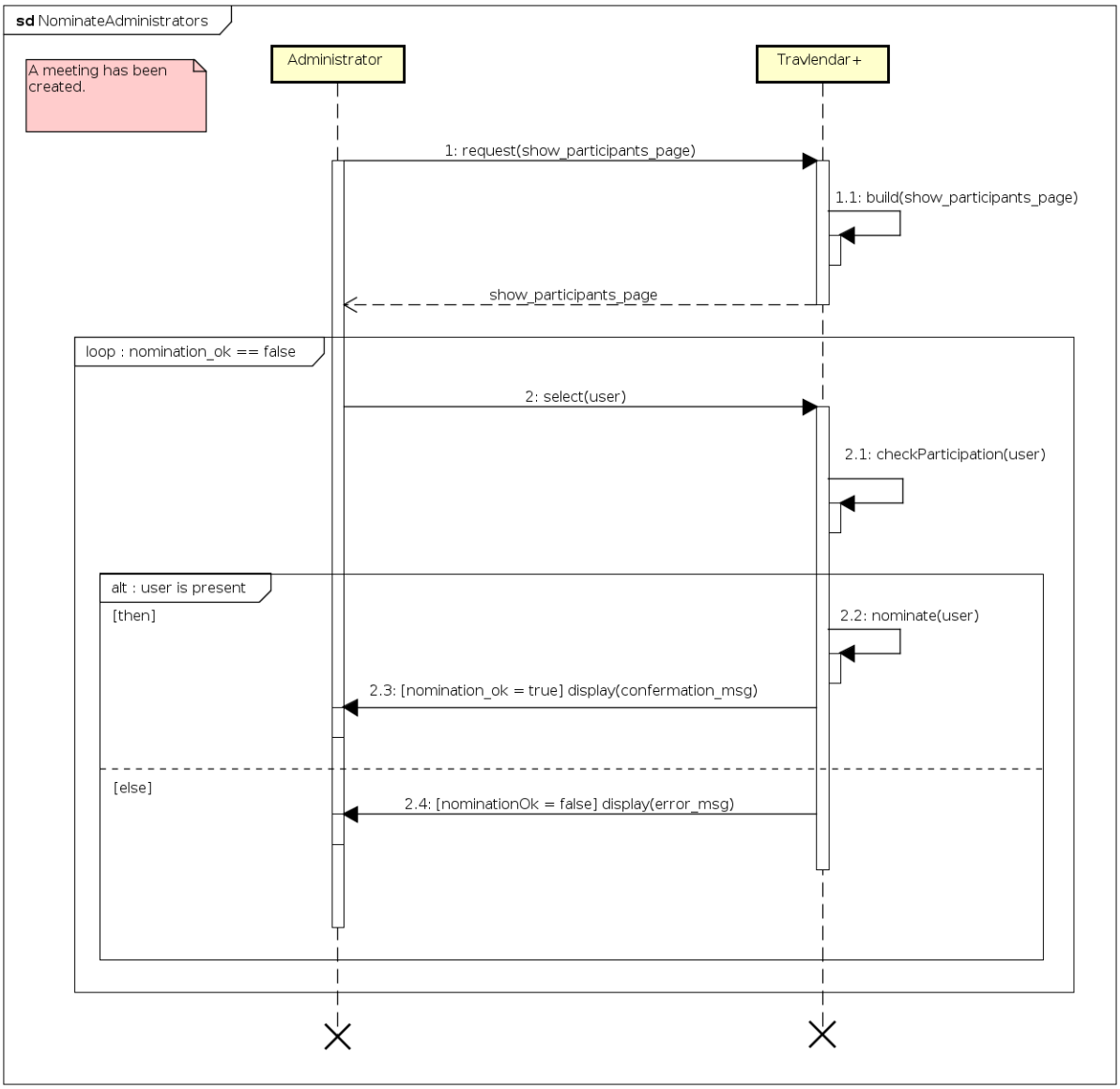


Figure 18: Nominate Administrators (view Use Case Description)

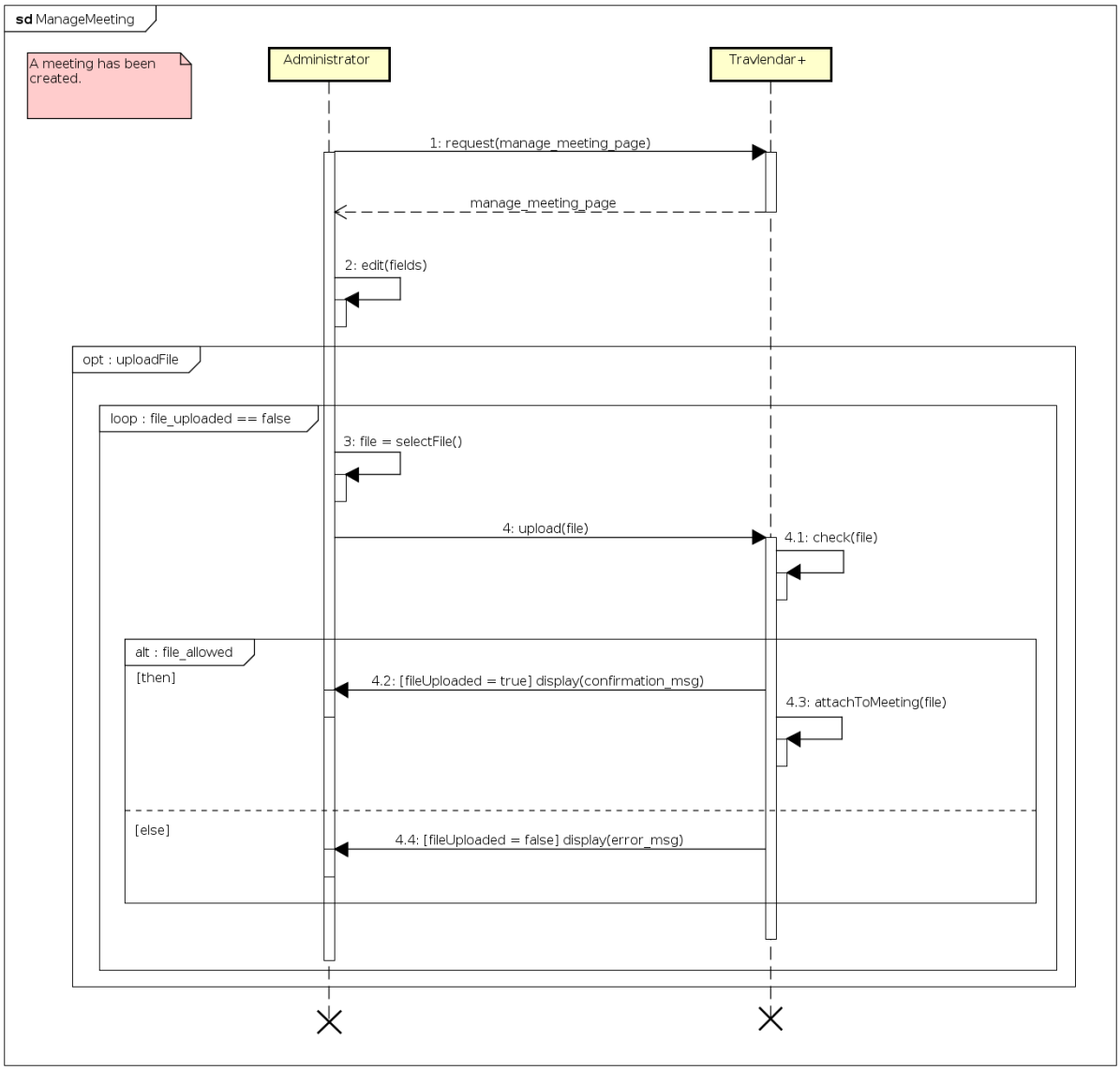


Figure 19: Manage Meeting (view Use Case Description)

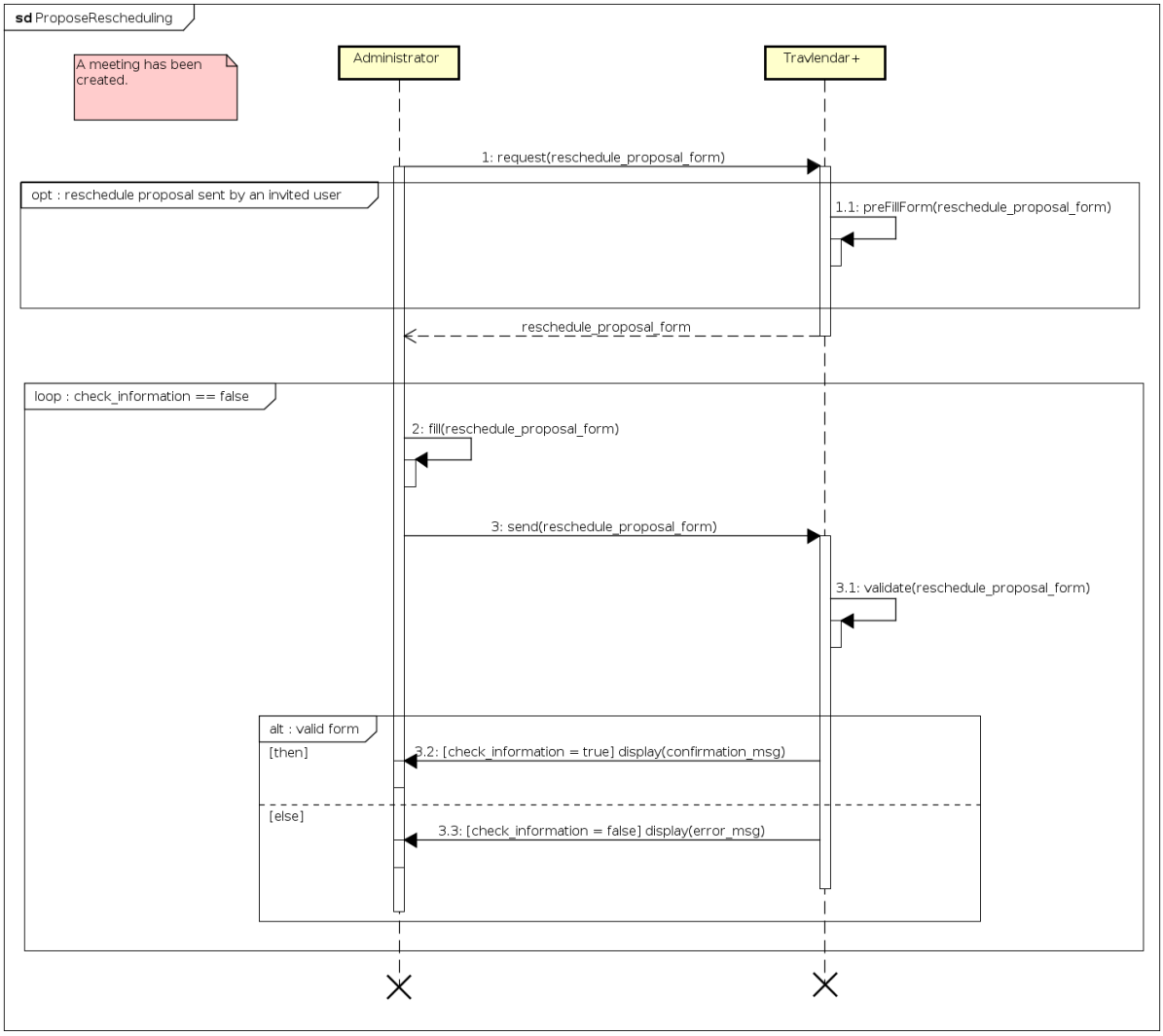


Figure 20: Propose Rescheduling (view Use Case Description)

4.4 State Chart Diagrams

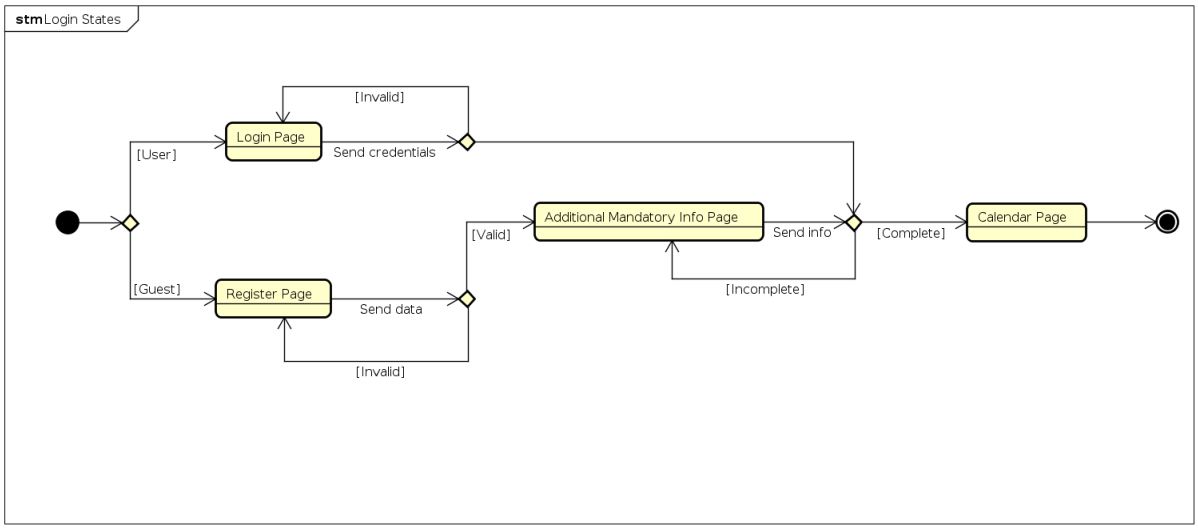


Figure 21: Login States

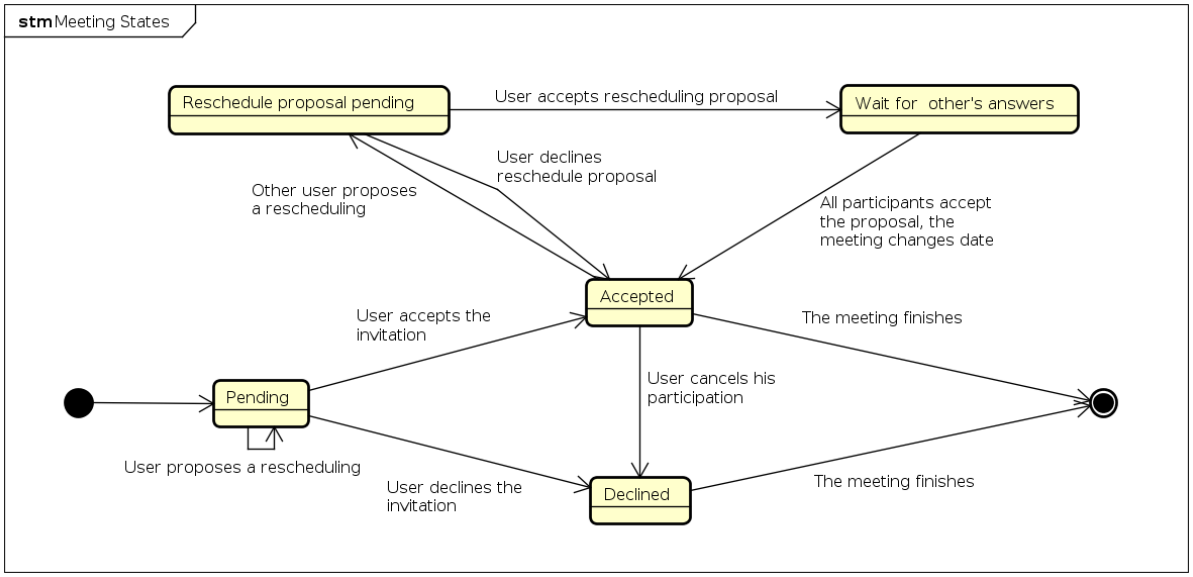
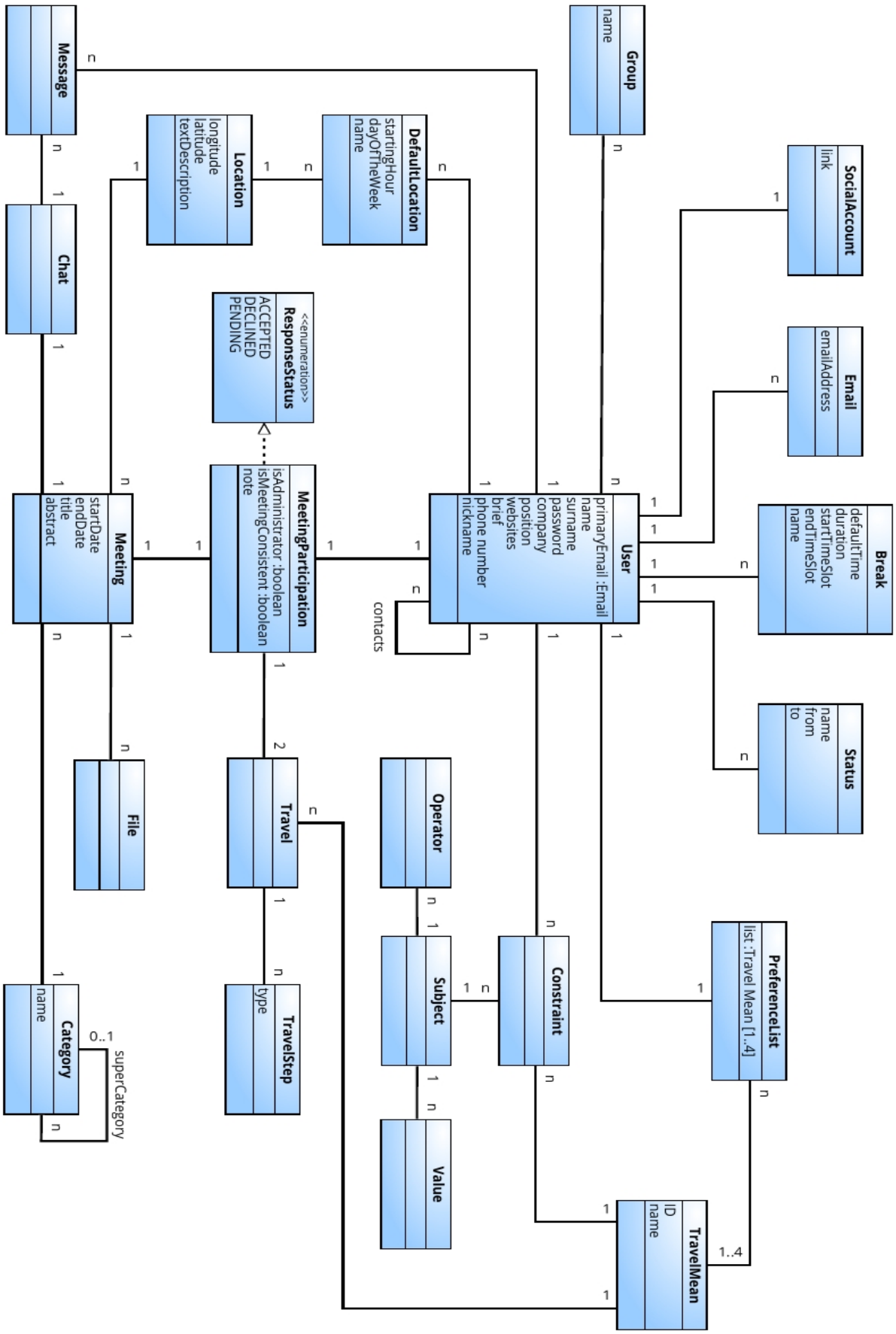


Figure 22: Meeting States

4.5 Class Diagram



5 Formal Analysis Using Alloy

5.1 Signatures

```
open util/ordering [Date]
open util/boolean

// User related signatures

sig User {
        primaryEmail: one Email,
        emails: some Email,
        breaks: set Break,
        socialAccounts: set SocialAccount,
        statuses: set Status,
        preferenceList: one PreferenceList,
        defaultLocations: some DefaultLocation,
        contacts: set User,
        constraints: set Constraint,
        messagesSent: set Message,
        meetingParticipations: set MeetingParticipation
}{
        primaryEmail in emails
        this not in contacts
}

sig Email {}{
        #(emails.this) = 1      // each email belongs to one and only
            one user
}

sig Break {
        defaultTime: one Date,
        duration: one Int,
        startTimeSlot: one Date,
        endTimeSlot: one Date,
        isDoable: one Bool
}{
        #(breaks.this) = 1      // each break belongs to one and only
            one user
        duration > 0
        DateInOrder[startTimeSlot, endTimeSlot]
        DateInOrder[defaultTime, endTimeSlot]
        DateInOrder[startTimeSlot, defaultTime]
```



```
        // A break is doable iff there are two consecutive "free"
           dates in between startTimeSlot and endTimeSlot
        isDoable = True iff (some d1: Date, d2: d1.next |
        (IncludingDates[d1, startTimeSlot, endTimeSlot] or d1 =
           startTimeSlot) and
        (no mp: breaks.this.meetingParticipations | OverlappingDates[
           d1, d2, mp.arrivingTravel.startTime, mp.leavingTravel.
           endTime]))
}

abstract sig Status {
        from: one Date,
        to: one Date
}{
        #(statuses.this) = 1    // each status belongs to one and
           only one user
        DateInOrder[from, to]
}

sig AutoDecline extends Status {}

sig SocialAccount {}{
        #(socialAccounts.this) = 1      // each socialAccount belongs
            to one and only one user
}

sig Group {
        members: some User
}

abstract sig ResponseStatus {}
one sig Accepted extends ResponseStatus {}
one sig Declined extends ResponseStatus {}
one sig Rescheduled extends ResponseStatus {}

// Locations related signatures

sig DefaultLocation {
        startingHour: one Int,
        dayOfTheWeek: one Day,
        defaultLocation: one Location,
        nextDefaultLocation: one DefaultLocation,
        travelToNext: lone Travel
}
```



```
{
        #(defaultLocations.this) = 1    // each defaultLocation
           belongs to one and only one user
        #(@nextDefaultLocation.this) = 1
        FirstLocationAfter[this, nextDefaultLocation] or (
           nextDefaultLocation = this)
        (defaultLocations.this.defaultLocations -> defaultLocations.
           this.defaultLocations) in (^(@nextDefaultLocation))
        defaultLocations.this = defaultLocations.nextDefaultLocation
           // the user owning this default location and the next one
           must be the same
        startingHour >= 0
        startingHour <= 24
        // travel from subsequent locations
        (one t: travelToNext | t.departure = defaultLocation and t.
           arrival = nextDefaultLocation.@defaultLocation) or
        (nextDefaultLocation = this and #(travelToNext) = 0)
}

sig Location {}{
        all l: Location | (some ts: TravelStep | l = ts.fromLocation
           or l = ts.toLocation) // there are no location that are
           not used by the system
}

// Meeting related signatures

abstract sig BaseMeeting {
        startDate: one Date,
        endDate: one Date,
        category: one Category,
        chat: one Chat,
        location: one Location,
        files: set File,
}

sig Meeting extends BaseMeeting {}{
        DateInOrder[startDate, endDate]
}

sig InstantMeeting extends BaseMeeting {}{
        startDate = endDate
}
```



```
sig MeetingParticipation {
        isAdministrator: one Bool,
        isMeetingConsistent: one Bool,
        meeting: one BaseMeeting,
        arrivingTravel: one Travel,
        leavingTravel: one Travel,
        responseStatus: one ResponseStatus
}{
        #(meetingParticipations.this) = 1       // each
           meetingParticipation belongs to one and only one user
        no mp: meetingParticipations.this.meetingParticipations | mp
           != this and mp.@meeting = meeting  // each user
           participates to a meeting at most once
        arrivingTravel.arrival = meeting.location
        leavingTravel.departure = meeting.location
        arrivingTravel.endTime = meeting.startDate
        leavingTravel.startTime = meeting.endDate
        // a leaving travel can arrive in a default location or in
           the location of the subsequent meeting
        leavingTravel.arrival in ((meetingParticipations.this).
           defaultLocations.defaultLocation +
           NextMeetingParticipation[this].@meeting.location)
        // if a leaving travel arrives in a meeting than the leaving
           travel is exactly the arriving travel of that meeting
        leavingTravel.arrival in NextMeetingParticipation[this].
           @meeting.location implies
        (leavingTravel = NextMeetingParticipation[this].
           @arrivingTravel)
        isMeetingConsistent = False iff (
                // There is another meeting that overlaps
                some mp: (meetingParticipations.this.
                   meetingParticipations - this) | (
                (OverlappingDates[arrivingTravel.startTime,
                   leavingTravel.endTime, mp.@arrivingTravel.
                   startTime, mp.@leavingTravel.endTime]
                and leavingTravel != mp.@arrivingTravel and
                   arrivingTravel != mp.@leavingTravel) or
                (OverlappingDates[arrivingTravel.startTime,
                   leavingTravel.endTime, mp.@meeting.startDate, mp.
                   @leavingTravel.endTime]
                and leavingTravel = mp.@arrivingTravel and
                   arrivingTravel != mp.@leavingTravel) or
                (OverlappingDates[arrivingTravel.startTime,
                   leavingTravel.endTime, mp.@arrivingTravel.
                   startTime, mp.@meeting.endDate]
                and leavingTravel != mp.@arrivingTravel and
                   arrivingTravel = mp.@leavingTravel)
        ) or
```



```
                // There is a break that cannot be done
                some b: meetingParticipations.this.breaks | (
                b.isDoable = False and OverlappingDates[b.
                    startTimeSlot, b.endTimeSlot, arrivingTravel.
                    startTime, leavingTravel.endTime]
                )
        )
}

sig Message {}{
        #(messagesSent.this) = 1        // each message belongs to
            one and only one user
        #(messages.this) = 1    // each message belongs to one and
            only one chat
}

sig Chat {
        messages: set Message
}{
        #(chat.this) = 1        // each chat belongs to one and only
            one meeting
}

sig Category {
        superCategory: lone Category
}{
        superCategory != this
        some m: BaseMeeting | this = m.category // categories are not
             empty
}

sig File {}{
        #(files.this) = 1       // each file belongs to one and only
            one meeting
}

// Travel related signatures

sig Travel {
        steps: some TravelStep,
        travelMean: one TravelMean,
        departure: one Location,
        arrival: one Location,
        startTime: one Date,
        endTime: one Date
}
```



```
{
        plus[plus[#(arrivingTravel.this), #(leavingTravel.this)], #(
           travelToNext.this)] = 1
        arrival != departure
        departure in steps.fromLocation
        departure not in steps.toLocation
        DateInOrder[startTime, endTime]
}

sig TravelStep {
        fromLocation: one Location,
        toLocation: one Location,
        nextStep: lone TravelStep
}{
        #(steps.this) = 1       // each  belongs to one and only one
           travel
        fromLocation != toLocation
        #(@nextStep.this) <= 1
        fromLocation not in (this.^@nextStep).@toLocation
        #(nextStep) = 1 implies toLocation = nextStep.@fromLocation
           and steps.this = steps.nextStep // if it is not the last
        #(nextStep) = 0 implies toLocation = steps.this.arrival
        and (no ts: TravelStep | ts != this and ts in steps.this.
           steps and #(ts.@nextStep) = 0) // if it is the last
}

abstract sig TravelMean {}
one sig Walking extends TravelMean {}
one sig Driving extends TravelMean {}
one sig Biking extends TravelMean {}
one sig PublicTransportation extends TravelMean {}

sig PreferenceList {
        travelMeans: seq TravelMean
}{
        #(preferenceList.this) = 1      // each preferenceList
           belongs to one and only one user
        #travelMeans > 0
        #travelMeans <= 4
}
```



```
sig Constraint {
        subject: one Subject,
        operator: one Operator,
        value: one Value,
        target: one TravelMean
}{
        #(constraints.this) = 1 // each constraint belongs to one and
            only one user
        operator in subject.operators
        value in subject.values
}

sig Subject {
        values: some Value,
        operators: some Operator
}

sig Operator {}{
        #(operators.this) = 1   // each operator belongs to one and
           only one subject
}

sig Value {}{
        #(values.this) = 1      // each value belongs to one and only
            one subject
}

// Helper signatures

sig Date {
        _next: lone Date
}{ _
        next = this.@next
}

abstract sig Day {
        nextDay: one Day,
        distances: Day -> one Int
}

one sig Monday extends Day {}
one sig Tuesday extends Day {}
one sig Wednesday extends Day {}
one sig Thursday extends Day {}
one sig Friday extends Day {}
one sig Saturday extends Day {}
one sig Sunday extends Day {}
```


5.2 Facts

```
// Each meeting as at least a participant and an administrator
fact oneParticipantAndAdministrator{
        no m: BaseMeeting | m not in MeetingParticipation.meeting
        all m: BaseMeeting | some mp: MeetingParticipation | m = mp.
           meeting and mp.isAdministrator = True
}

//Users cannot have meetings while their status is set to auto-
   decline. (Done)
fact autodecline{
        all u: User | all ad: statuses[u] | no m: u.
           meetingParticipations.meeting |
        ad in AutoDecline and (OverlappingDates[m.startDate, m.
           endDate, ad.from, ad.to])
}

//A user cannot have different default locations sharing the start
   time. (Done)
fact noSimultaneusDefaultLocations{
        all u: User | no d1,d2: u.defaultLocations |
        d1 != d2 and d1.dayOfTheWeek = d2.dayOfTheWeek and d1.
           startingHour = d2.startingHour
}

// each message of a chat belongs to a user who has accepted the
   invitation to the meeting of the chat
fact allMessagesFromUserInChat {
        all m: BaseMeeting | messagesSent.(m.chat.messages) in (
           UserParticipations[MeetingParticipationsByStatus[Accepted
           ]]).meeting.m
}

// each user uses only travel means in its preference list
fact travelMeansInPrefernceList{
        all u: User | u.meetingParticipations.(arrivingTravel +
           leavingTravel).travelMean in u.preferenceList.travelMeans[
           Int]
}

fact nextDay{
        Monday.nextDay in Tuesday
        Tuesday.nextDay in Wednesday
        Wednesday.nextDay in Thursday
        Thursday.nextDay in Friday
        Friday.nextDay in Saturday
        Saturday.nextDay in Sunday
        Sunday.nextDay in Monday
}
```



```
fact nextDayDistances {
        all d1: Day, d2: (Day - d1) | d1.distances[d2] = plus[1, d1.
           nextDay.distances[d2]]
}
```

5.3 Predicates and Functions

```
// Date a is before Date b
pred DateInOrder[a: Date, b: Date] {
        a not in b.^next and a != b
}

// two events (s1,e1) and (s2,e2) overlaps
pred OverlappingDates[s1: Date, e1: Date, s2: Date, e2: Date] {
        (DateInOrder[s1, s2] and DateInOrder[s2, e1]) or
        (DateInOrder[s2, s1] and DateInOrder[s1, e2]) or
        s1 = s2 or e1 = e2
}

// d1 is incuded between d2 and d3
pred IncludingDates[d1:Date, d2:Date, d3:Date]{
        DateInOrder[d2,d1] and DateInOrder[d1,d3]
}

// location after
pred FirstLocationAfter[d1: DefaultLocation, d2: DefaultLocation]{
        (no d3: defaultLocations.d1.defaultLocations |
        d1 != d3 and d2 != d3 and d1.dayOfTheWeek != d3.dayOfTheWeek
           and
        d1.dayOfTheWeek.distances[d3.dayOfTheWeek] < d1.dayOfTheWeek.
           distances[d2.dayOfTheWeek]) and
        (d1.dayOfTheWeek = d2.dayOfTheWeek implies d1.startingHour <
           d2.startingHour)
}

// returns a set of MeetingParticipations with a certain response
   status
fun MeetingParticipationsByStatus[rs: ResponseStatus]: set
   MeetingParticipation{
        {mp: MeetingParticipation | mp.responseStatus = rs}
}

// returns existing binary relationships between any User and the
   given MeetingParticipation
fun UserParticipations[mp: MeetingParticipation]: User ->
   MeetingParticipation {
        (meetingParticipations.mp -> mp) & meetingParticipations
}
```



```
// computes the next meeting a user is participating to (if any)
fun NextMeetingParticipation[mp: MeetingParticipation]: lone
   MeetingParticipation {
        {mp1: meetingParticipations.mp.meetingParticipations |
        DateInOrder[mp.meeting.endDate, mp1.meeting.startDate] or mp.
           meeting.endDate = mp1.meeting.startDate}
}
```

5.4 Results

```
// if there is a message in a meeting, then there is also a user with
    the accepted status
assert messageInMeetingEntailsAcceptedParticipation {
        all m: Meeting | #(m.chat.messages) > 0 implies (some mp:
           MeetingParticipation | mp.meeting = m and mp.
           responseStatus = Accepted)
}

// the arriving and leaving travel of a meetingParticipation are
   never equal
assert arrivingAndLeavingTravelAreDifferent {
        no mp: MeetingParticipation | mp.arrivingTravel = mp.
           leavingTravel
}

// the presence of a single non consistent meeting implies that is
   has been made inconstistent by the presence of a non doable break
assert singleInconsistentMeetingEntailsNotDoableBreak {
        #{mp: MeetingParticipation | mp.isMeetingConsistent = False}
           = 1 implies #{b: Break | b.isDoable = False} > 0
}

pred showUser {
        #User > 1
        #Break > 1
        #Status > 1
        #SocialAccount > 1
        #Group > 1
        #contacts > 1
        #Constraint > 1
        #MeetingParticipation = 0
}
```



```
pred showMeeting {
        #BaseMeeting = 1
        #Meeting = 1
        #MeetingParticipation > 2
        #Message > 4
        #File > 2
        some mp : MeetingParticipation | mp.responseStatus = Accepted
        some mp : MeetingParticipation | mp.responseStatus =
           Rescheduled
        some mp : MeetingParticipation | mp.responseStatus = Declined
}

pred showChat {
        some m: Meeting | #(messagesSent.(m.chat.messages)) > 1
}

pred showTravel {
        #{t: Travel | #(t.steps) > 1} = 1
        some t: Travel | #(t.steps) > 4
}

run showUser for 8 but 8 Int

run showMeeting for 8 but 8 Int

run showChat for 8 but 8 Int

run showTravel for 8 but 8 Int

check messageInMeetingEntailsAcceptedParticipation for 8 but 8 Int

check arrivingAndLeavingTravelAreDifferent for 8 but 8 Int

check singleInconsistentMeetingEntailsNotDoableBreak for 8 but 8 Int
```

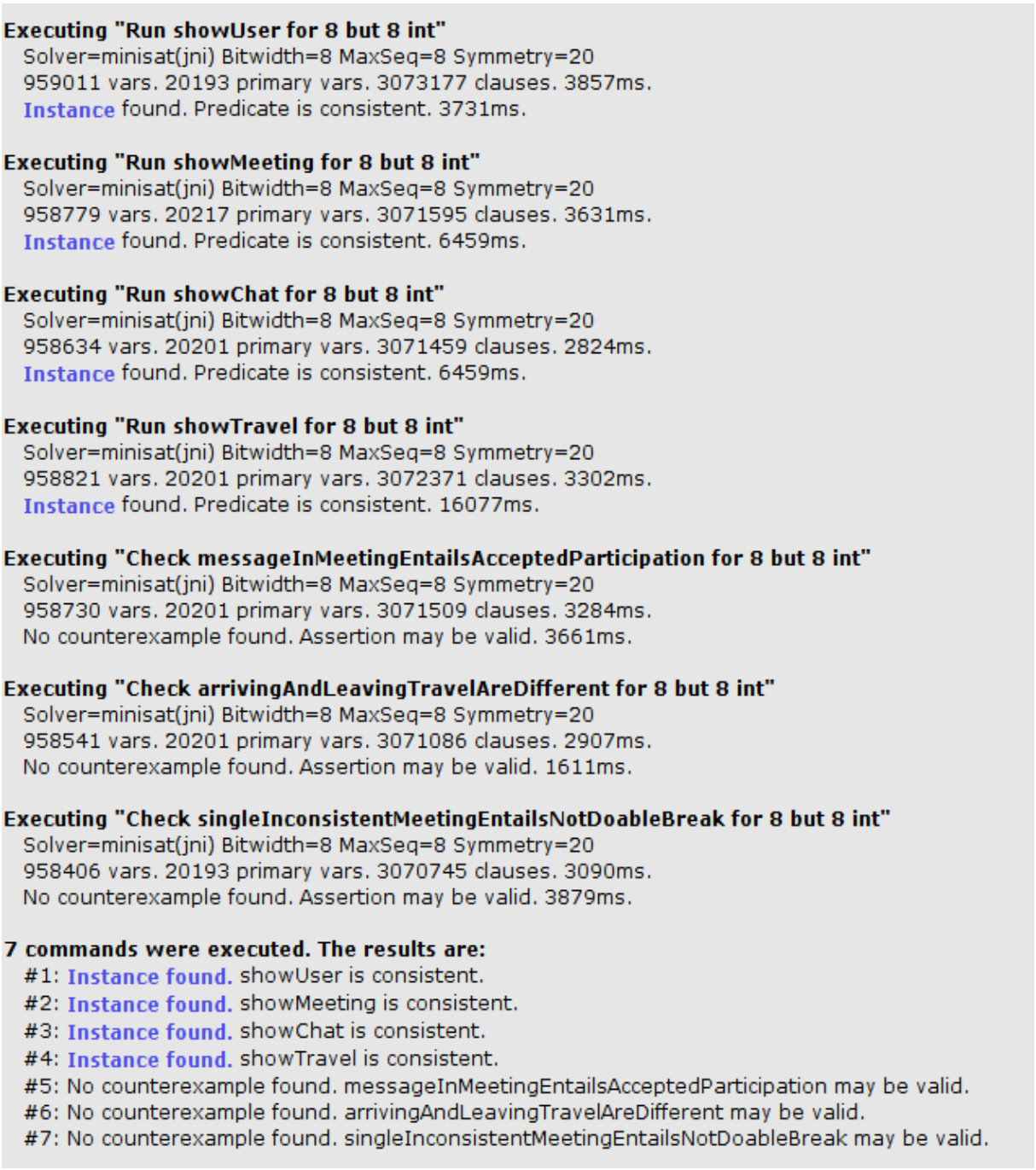



Figure 23: Execution of the Alloy model

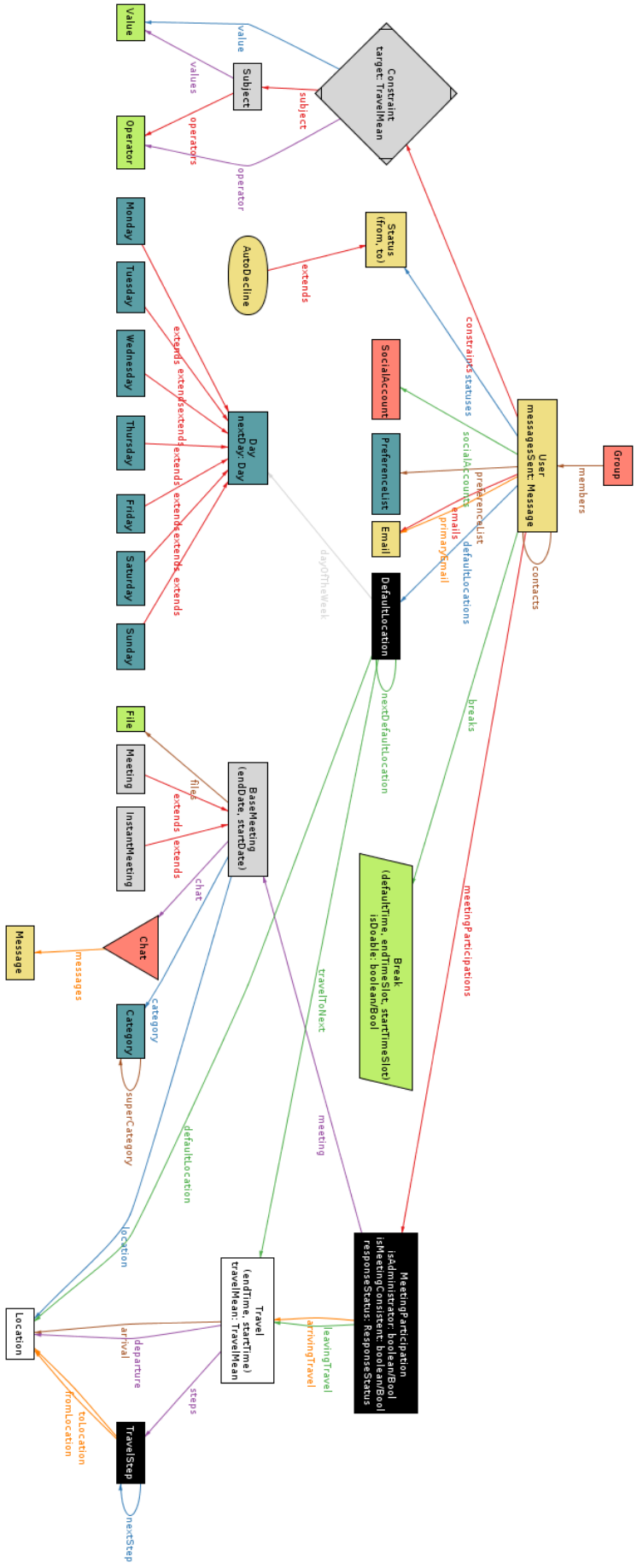


Figure 24: Graph for the Alloy Metamodel

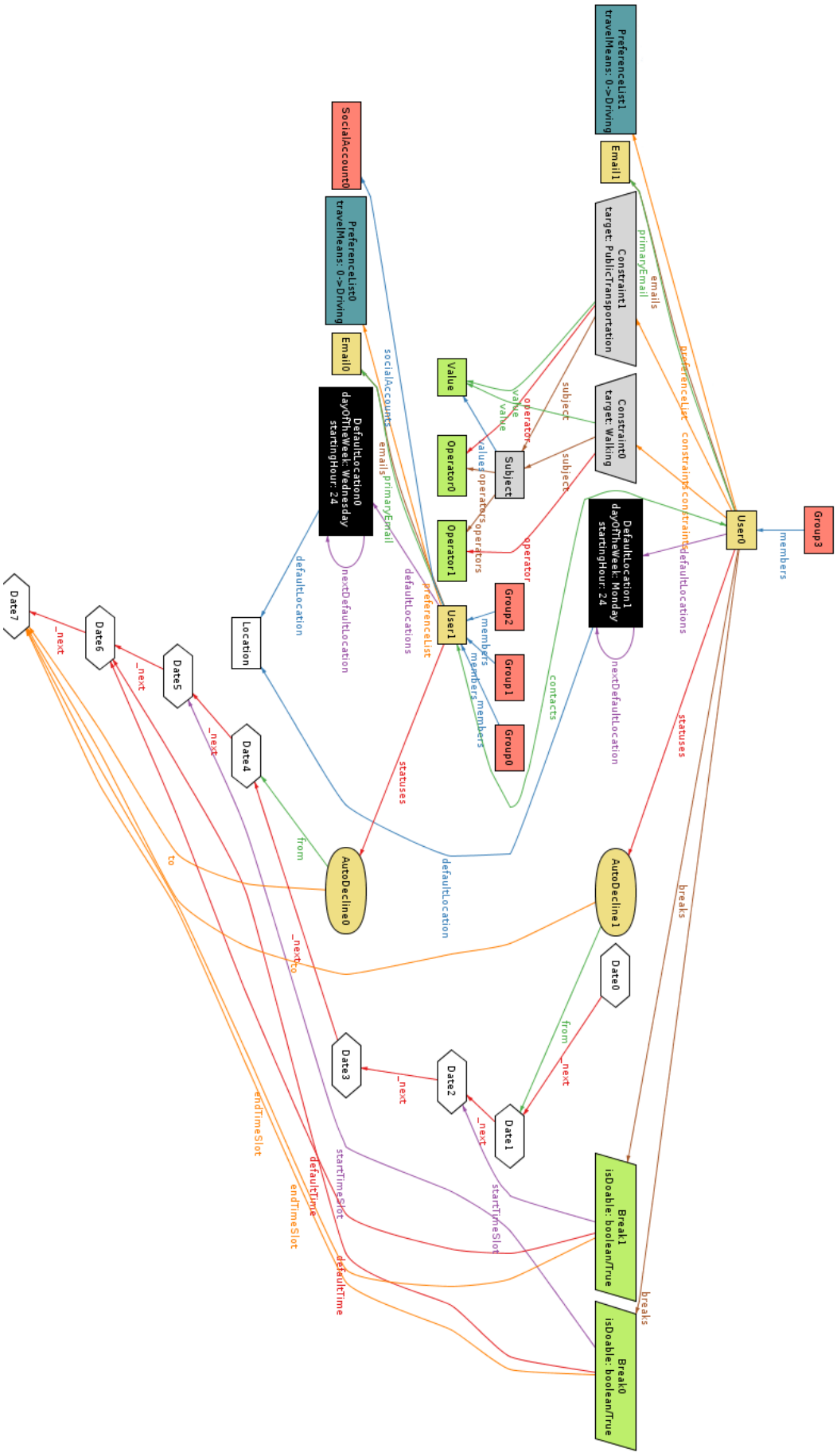


Figure 25: Graph for the Alloy Show User predicate

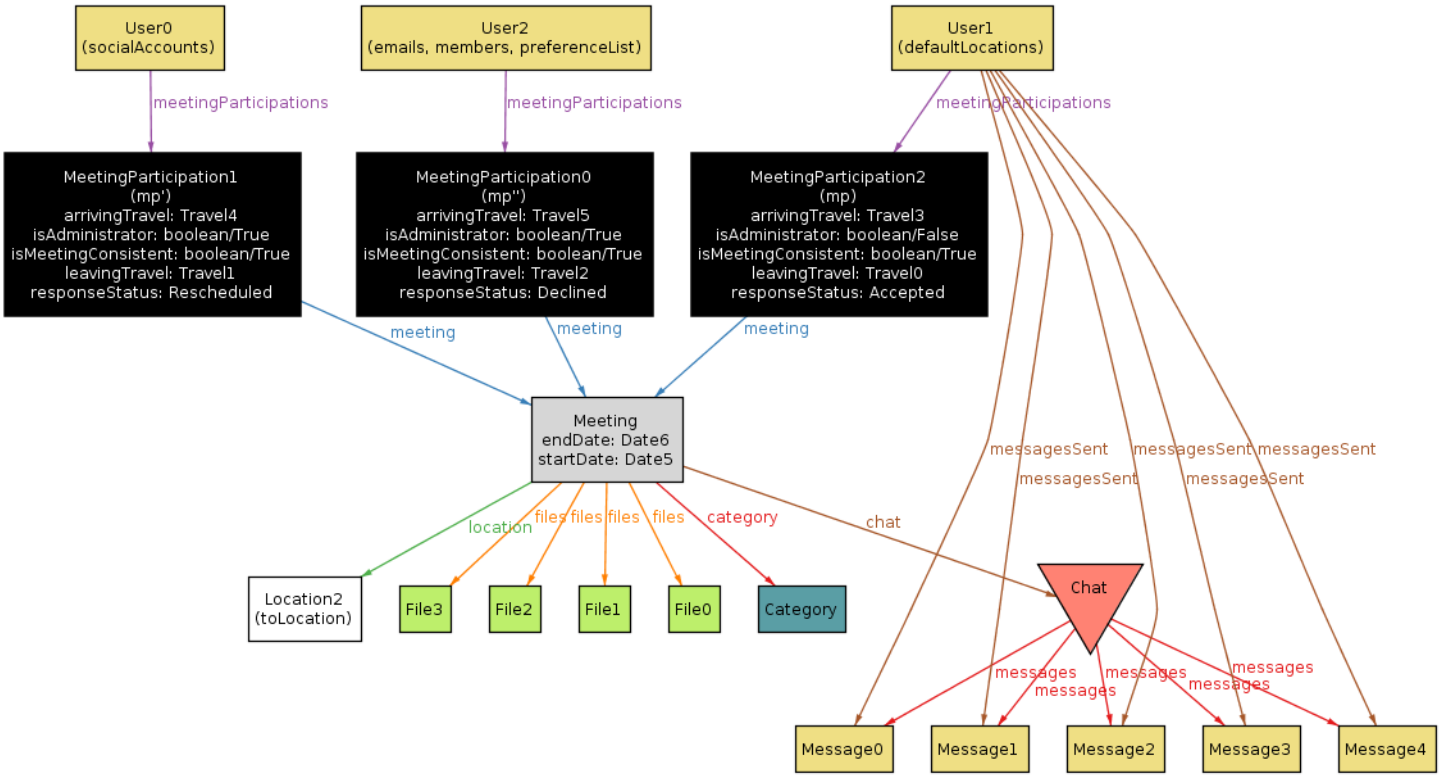


Figure 26: Graph for the Alloy Show Meeting predicate

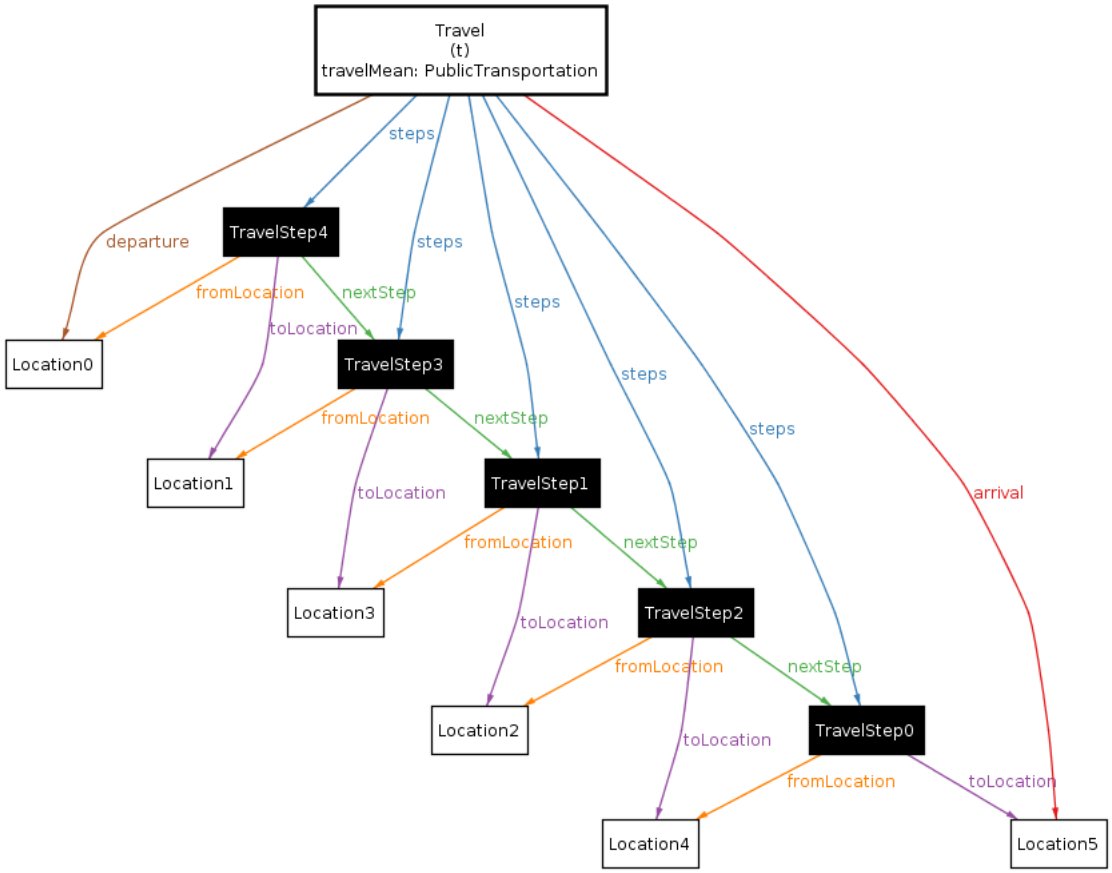


Figure 27: Graph for the Alloy Show Travel predicate

6 Appendix

6.1 Effort Spent

Date	Federico Betti	Tommaso Bianchi
8/10/17	3	3
9/10/17	2,5	2,5
12/10/17	2	2
13/10/17	4	4
14/10/17	2	1,5
16/10/17	3,25	3,5
17/10/17	2,5	3
18/10/17	2,5	3
19/10/17	5,5	3,5
20/10/17	2,5	2,5
21/10/17	1,5	3
22/10/17	1	1,5
23/10/17	2	2
24/10/17	3,75	3,75
25/10/17	5,75	4,25
26/10/17	6,25	7,25
27/10/17	6,5	7,25
28/10/17	0	2,5
29/10/17	5,25	2

Table 16: Effort Spent

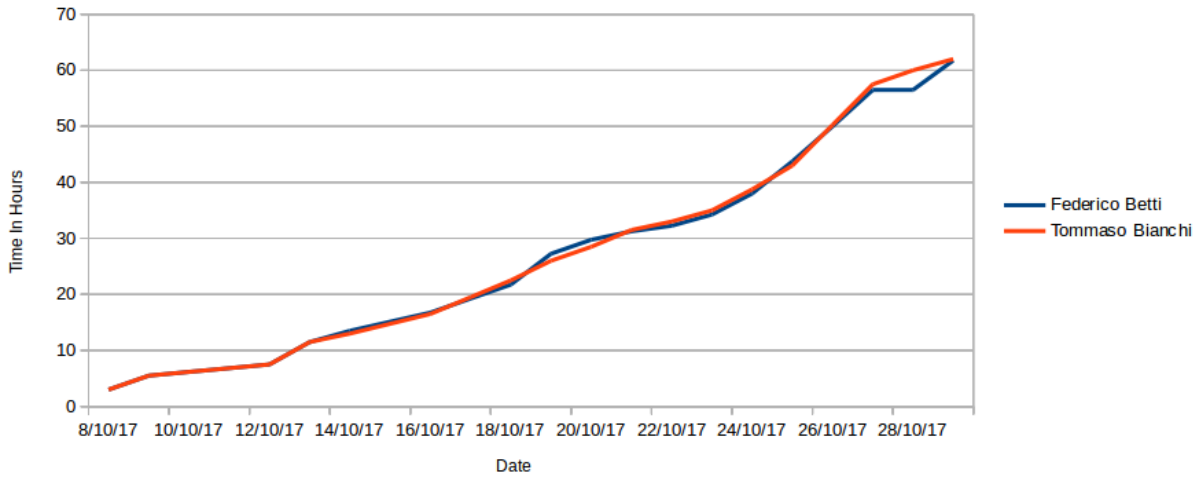


Figure 28: Effort Spent

6.2 Tools Used

- **TeXstudio**: a cross-platform open source LaTeX editor
- **Git**: a version control system for tracking changes in computer files and coordinating work on those files among multiple people
- **Signavio**: a tool for creating UML diagrams; we used it for the class diagram and the use case diagram
- **Astah**: a tool for creating UML diagrams; we used it for the sequence diagrams and the state chart diagrams
- **Alloy Analizer**: an IDE for the Alloy language, which includes the Kodkod model finder and a variety of SAT solvers, as well as the standard Alloy library
- **Adobe Illustrator**: a vector graphics editor; we used it for the mockups of the user interface
- **GIMP**: a free and open-source raster graphics editor; we used it for some small editing of the images exported from the other tools

6.3 Revision History

This is the first version of the document, released on 29/10/2017. All subsequent modification will be tracked in the github repository and briefly listed here.

References

[1] Network Working Group. Application techniques for checking and transformation of names. Technical report, , 2004. URL: https://tools.ietf.org/html/rfc3696.

