REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENTS

Project of Software Engineering 2

WEATHER-CAL

Authors:

PAOLO POLIDORI

MARCO EDEMANTI

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to present a description of WheaterCal system. It will explain the features of the system, the interfaces of the system, what the system will do and the constraints under which it must operate. This document is intended for both the stakeholders and the developers of the system.

1.2 Scope

We want to project and implement WheaterCal. The aim of this project is to develop a system that offers an online calendar in which user can schedule their events according to the weather conditions.
A registered user can create, delete and update an event and moreover he should provide information about where and when this event will take place and information about the invited user. Once the event is created the system should provide to its creator the weather forecast information regarding the scheduled day, and most of all

it should notify a bad weather condition one day in advance to all the participants. Also a user is able to make his/her calendar visible to all other registered user showing them only the time slots in which they are busy without letting know the event information unless either the event is public. In addition in case of bad weather, three day before the scheduled data of an event, the system will inform the event's owner and propose to him the closest sunny day.

1.3 Glossary

Anonymous User	People who access the system but didn't authenticate
Registered User	People who both registered to the platform and logged in. Sometimes referred to as "User"
Platform, System	The WeatherCal application
Bad Weather	Undesired conditions for an event to take place basing on the event owner's choices
Notification	A message that arrives to the user in dedicated area, which can be easily identified by the user at first sight
Event owner	The user who created the event and for which is responsibile of the organization
Event invited	A user invited to an event for which can decide if he wants to be a participant or not
Event participant	A user invited to an event for which decided to participate
Event	An occurrence, which was inserted in the system which should happen in a certain date and a certain time
Calendar	The set of all the events for a user, shown in a per-month view

1.4 References

IEEE,*IEEE Std 830-1998,IEEE Recommended Practice for Software Requirements Specifications*, IEEE Computer Society 1998

Chapter 2

Overall Description

2.1 System Environment

WeatherCal is a stand alone software made from scratch. It is a Web application that allows the registered users to manage their schedule basing on the weather conditions. The users are also able to invite people to their event and to publicize their event. The system is even capable of notifying the users, as they log on to the system, if the weather forecast for the event is not the desired one. The target of this software is, therefore, to give the users a tool for schedule their events smartly, giving the possibility to change preferences as the weather forecast changes.

2.1.1 Actors

Actors in the system are mainly the people registered into the web application. They will be referred as Registered Users. Whoever is not yet registered on the platform will be an actor for the system too and it will be identified as an Anonymous Users. No administration is required since the system is not intended for illicit purposes and this is a policy of the platform, plus no conflict among users can occur since the

only allowed interactions are the invite to an event and a participation to an event (in which other people partecipate).

2.1.2 Scenarios

- Maggie creates an event for her daughter's birthday, which will be in their garden. She invites all her friends telling them to join the event on WeatherCal to be sure that the forecast will be graceful for that day. Three days before the birthday, Maggie logs in two days before the event and she discovers it will be cloud but she still hopes it won't rain on that day. The day before the event Judith, Maggie's friend, is notified that it could be rainy for the birthday as she logs into the system. Anyway Maggie decides she won't posticipate the birthday. In the end that day was rainy, except for the afternoon, exactly for the birthday, and everything went fine.

- John wants to go cycling but right now the forecast is not so optimistic, so he decides to use WeatherCal to find the best day for doing so. The platform suggests him that he should go on Monday afternoon, so he plans to do that. Saturday he logs in and discovers that it will be windy on Monday, infact the system suggests him to go cycling on Wednesday and John chooses to do what the system suggests him. In the end he went cycling and the weather was perfect.

- Beth wants to have a picnic with her friends in two weeks, so she creates an event on WeatherCal when the temperature should be the hottest and there should be no clouds in the sky. Three days before she logs in discovering that for the next two weeks there will be bad weather conditions. She decides so to invite her friends at home for having a launch together.

2.2 Functional Requirements

The system allows the users to partecipate in various scenarios:

- Log on to and log out from the platform.
- Create, Modify, Delete an event.
- Invite other users.
- Manage invitation (accept, refuse).
- Manage event visibility (decide if an event's infos should be accessible only by its invited).
- View other users' profiles (see other users's calendar only if it's set public and see the event's calendar if they mark public too)

The system is the leading actor in:

- Warn all the participants to an event which will occur the following day that there will be bad weather conditions.
- Notify to an event's owner three day before the scheduled data that there will be bad conditions for it suggesting the closest available sunny day.

The Anonymous Users are only able to:

- Sign Up in to the Platform.

2.3 Goals

The WeatherCal system should offer and fulfill these main features:

- A simple management for the users' agendas
- A swift way to schedule an event according to the weather forecast
- Invite other registered users to an event
- Preserve the privacy of the users since only the user himself can make their info visibile to everyone.

2.4 Non Functional Requirements

2.4.1 User Interfaces

The WeatherCal system is a Web application intended to be used trough desktop device. Its interface is designed in a simple way so that the user can either easily understand what the system can offer to him and reaches all the main features of the system from a unique view.

Once that a user logs in to the system trough a defined web page he is redirected to the main page. This page is divided in two portion. The left one, which occupies most of the page, displays a monthly calendar in which the user manages its events while the right one shows the weather forecast for the following days and also shows the closest events.

In the figure 2.1 is shown a first sketch of the main user's page.

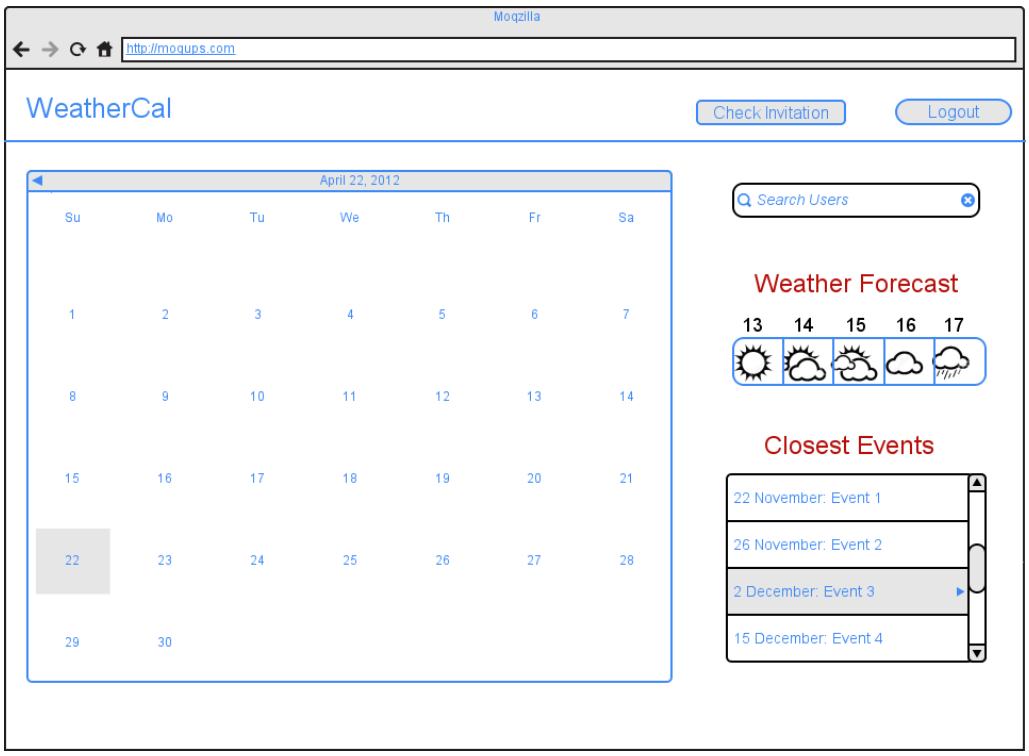

Figure 2.1: Main Page's MockUp

2.4.2 Performance

The system offers the users the possibility to use it without major slowdowns. Multiuser usage is allowed and it can be bore by the server application under reasonable conditions, basing on the server hardware.

2.4.3 Security

The system relies on HTTP protocol for transferring data over the Internet, so information sent both from the user to the server and vice-versa will be exposed to possible man-in-the-middle attacks. Unless this issue data stored on the server will be accessible only if valid credentials are provided and there is no possibility for a user to bypass privacy constraints, i.e. users will not be able to see non-public events of other users and modify events for which they are not owners.

Chapter 3

System Modeling

3.1 UML Diagrams

3.1.1 Use Case Diagrams

Event Management

The use case in figure 3.1 shows how a user can manage his agenda. After he logs into the platform he will see his calendar and being able to see schedule of his events.

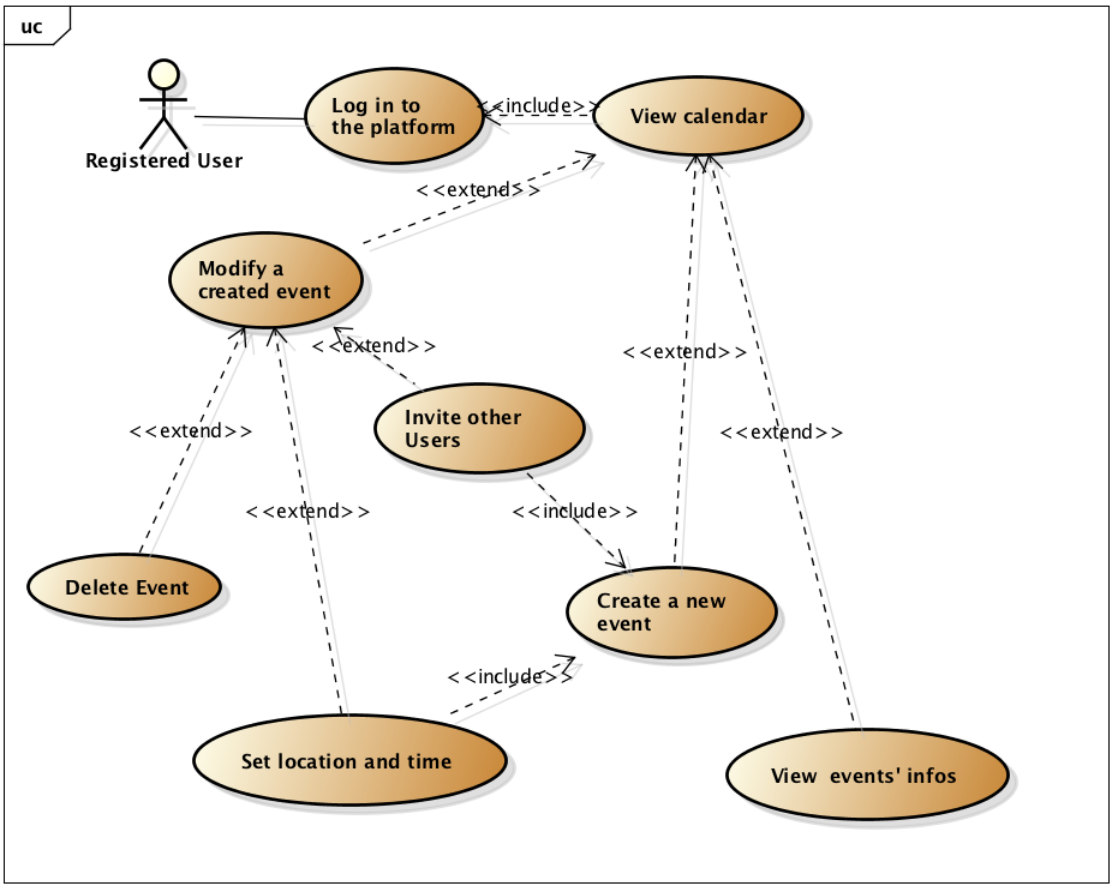

Figure 3.1: Event Management Use Case

As the user enter the platform he will be redirected to the main page of the system in which he will be able to manage in several different way its agenda. From that point he can either view his events and every events' details or create a new one. Whenever he creates a new event he will be able to set the place and the date and of course invite his friend. Naturally once a user created a event he will be capable of deleting the event whenever he wants.

Invitation

The use case in figure 3.2 explains what the user can do when he receive an invitation to an event from an other user.

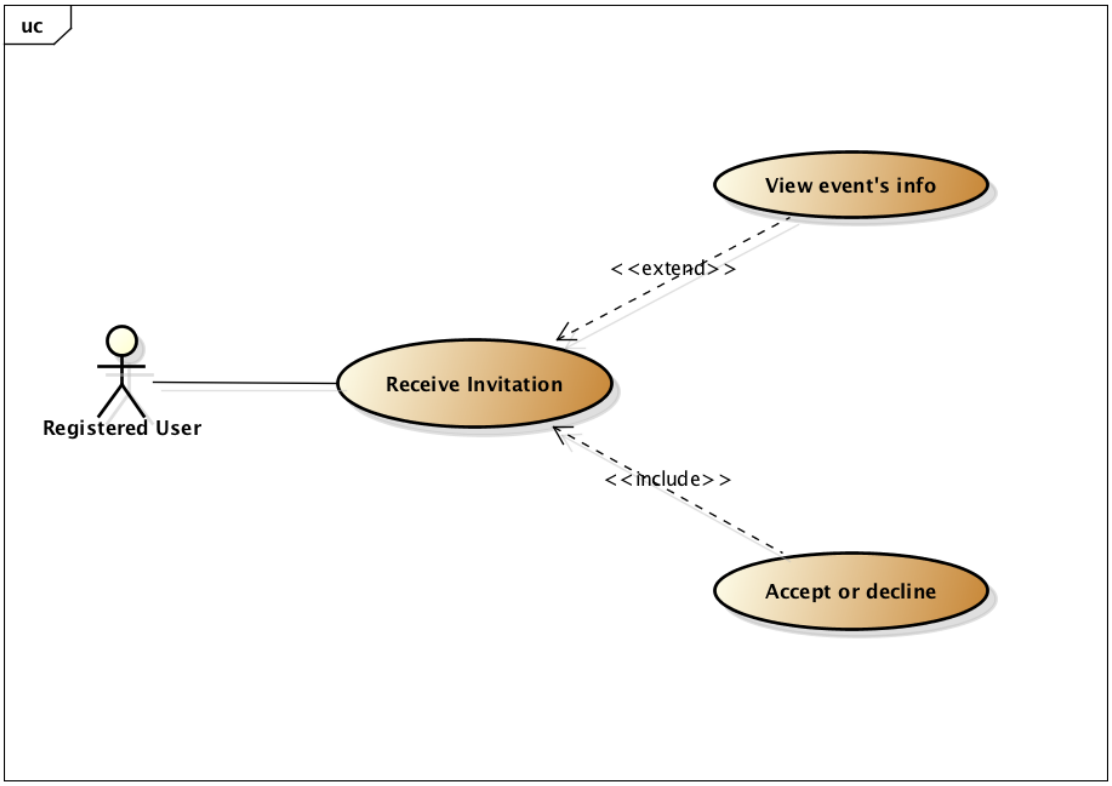

Figure 3.2: Invitation Use Case

Once a user invites an other user to his event, the guest receives a notification and from now on he can either see the event's info and accept or decline the invitation. Of course if he later changes its mind he is able to modify its participation at any time before the event's scheduled date.

View other users' profiles

The use case in figure 3.3 shows how an user can reach the profile of an other user and view his agenda.

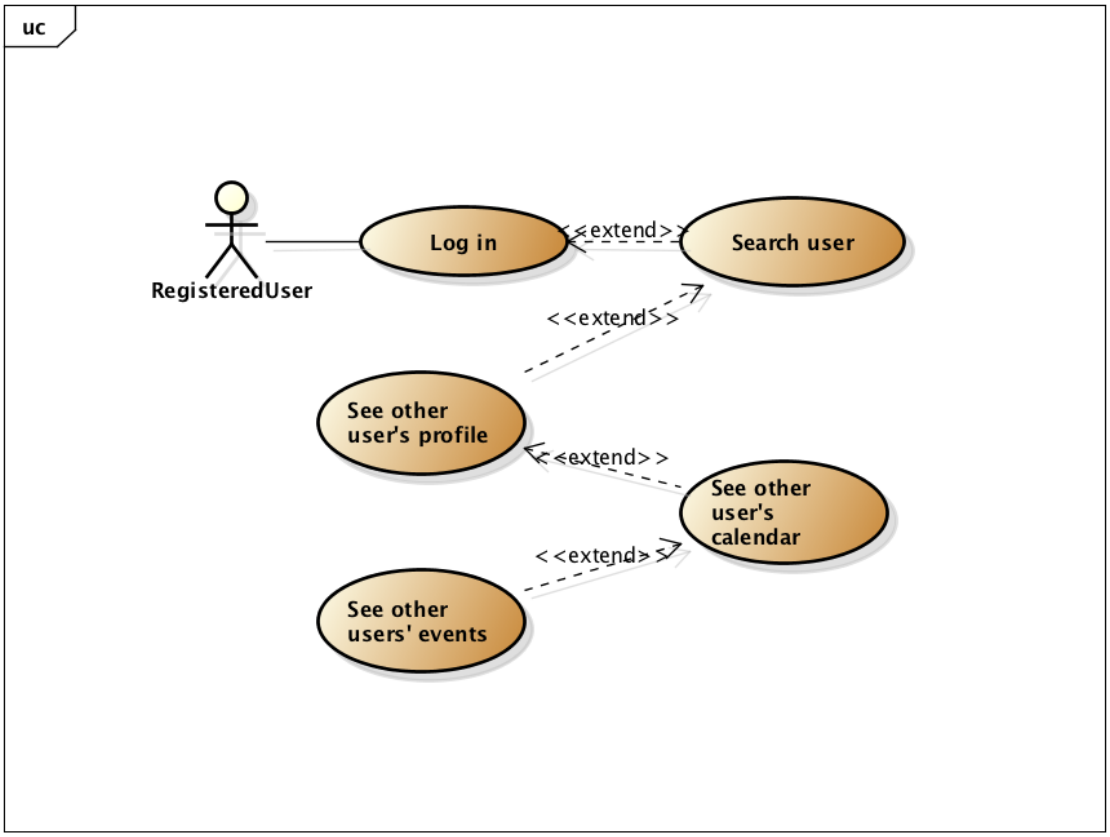

Figure 3.3: See Other User Profile Use Case

After the user logs in to the platform he will be able to search an user and see his profile, as soon as he reaches the user's profile he will be authorized to view other user's calendar only if it was set public by its owner, but even if a calendar was marked public the observing user is capable to see the calendar's events's details only if also the event was set as public.

Bad Weather

The use case in figure 3.3 explains how an user can behave when he get notified by the platform in case of bad weather condition.

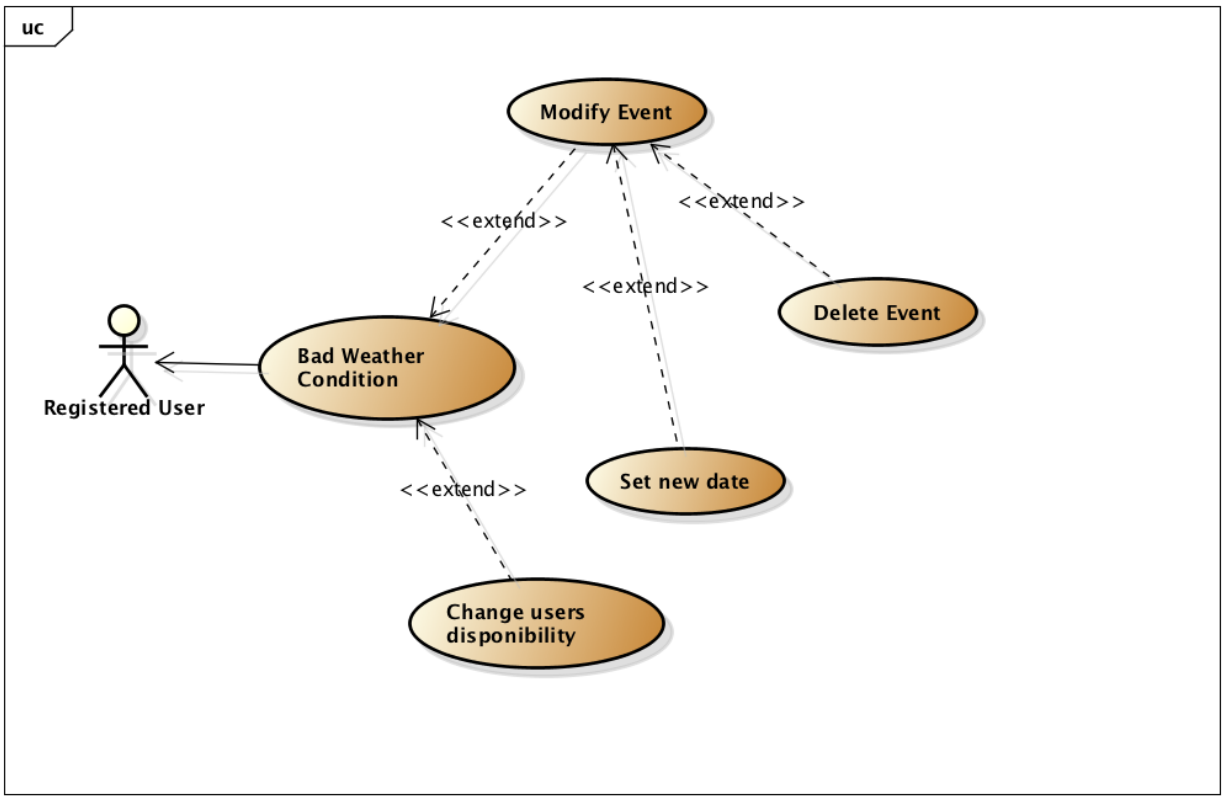

Figure 3.4: Bad Weather Use Case

Whenever an user receives a bad weather notifications he can perform two different activities depending how they relate to the event. If he's the event's owner he can delete the event or modify the event's location and date, while if he's just an invited to the event he can only modify his participation.

3.1.2 Class Diagrams

Once the requirements of our platform have been defined we are able to compose the class diagram of the system, as shown in figure 3.5.

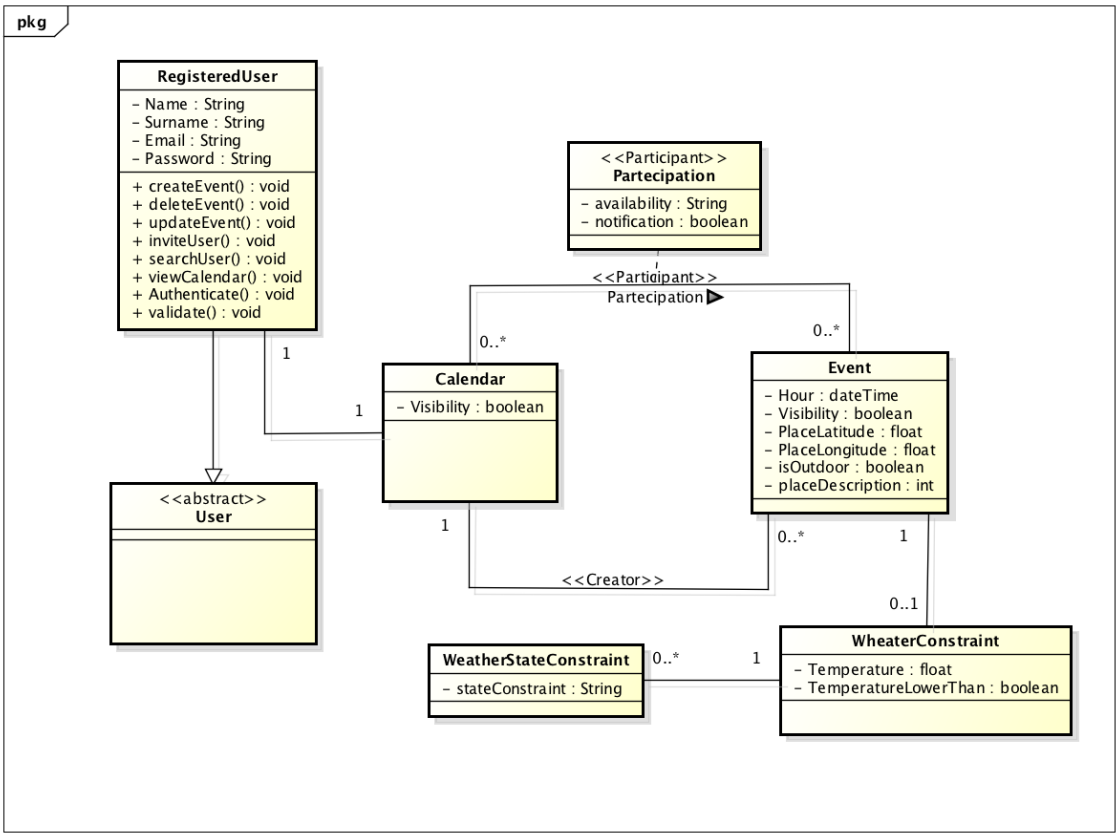

Figure 3.5: Class Diagram

3.1.3 State Diagrams

Invitation

In figure 3.6 the state diagram of an invitation to an event is shown.

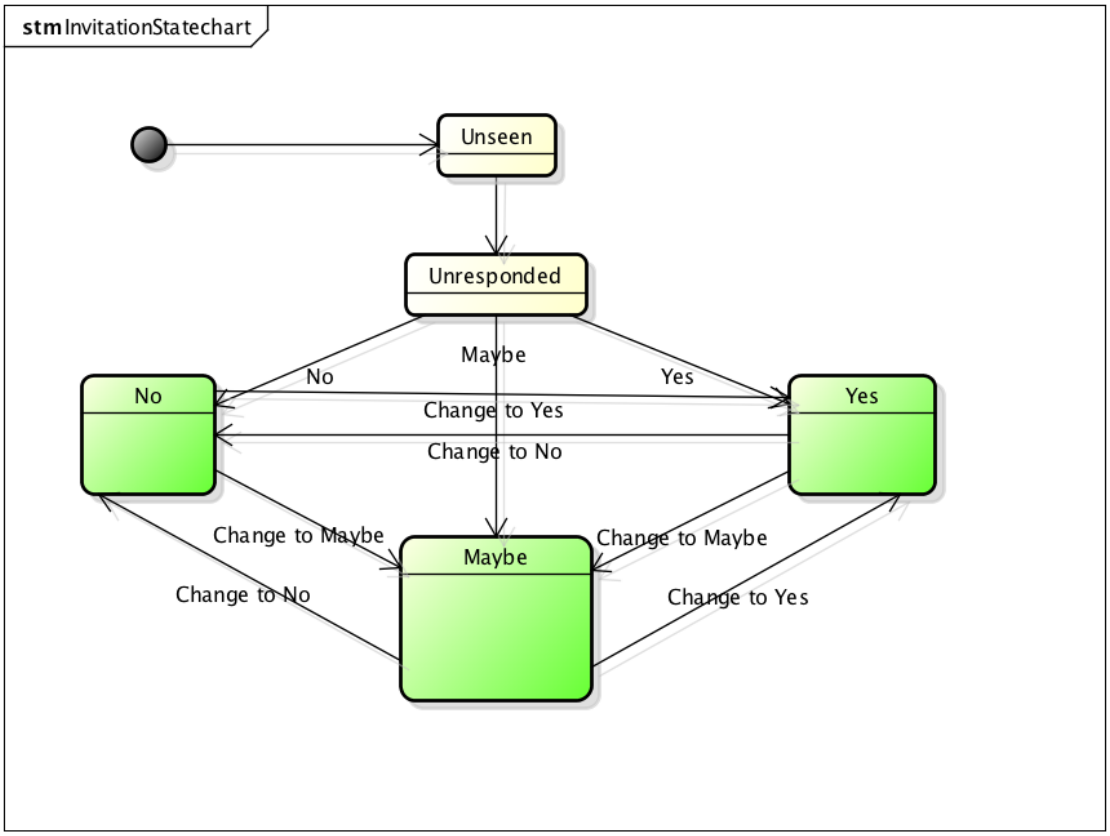

Figure 3.6: Invitation State Diagram

As the event owner invites a user, an invitation is created, having an unknown participation, which is defined by the first state. When an invitation is in this state means that an invitation for a user to an event has been made, but he didn't seen that yet. When he logs onto the platform the state of the invitation changes to unresponded, since it has been shown to the user, but he didn't choose what to do yet. As the user responds to the notification, the state of the invitation becomes definite, so in can be "Yes", "No" or "Maybe". This will not necessarily happen when the user is notified about an invitation. After the user chosen what to do, he can then change his participation among the defined states (marked in green),

and consequentially the state, until the event has ended. Infact, after the end of the event the modification of the participation will be disallowed.

3.1.4 Sequence Diagrams

User Registration

Figure 3.7 shows the process for the registration of a new user into the WeatherCal platform. As a user accesses the system, it will prompt it either to login or to register to the system. If the user chooses the latter this is what happens.

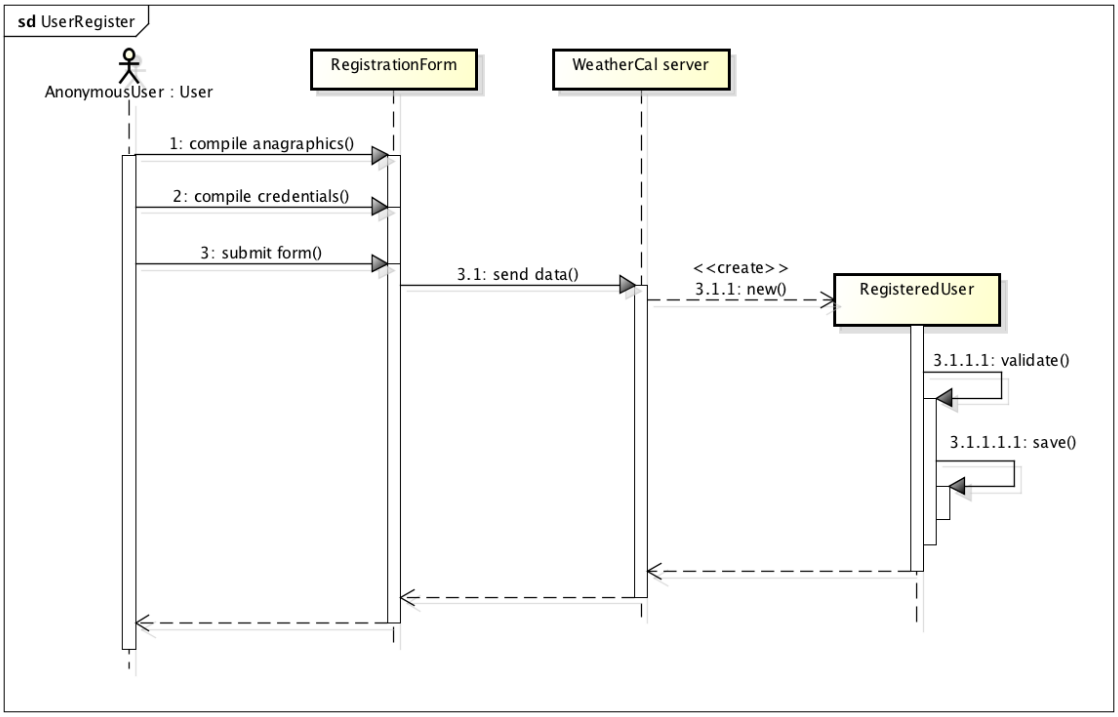

Figure 3.7: Sequence diagram of user registration

The user will see a registration form in which he will input his data and the credentials he wants to use for accessing the system. As the user submits the form and

data is sent to the server the validation occurs. It can happen that, for instance, a new user chooses an already taken nickname, or he registers with an email which is yet on the system. These cases will cause an exception to be thrown and an error to be displayed to the user. However, in this diagram, the case in which the validation succedes is represented. After that, the user data will be stored and the credentials can be used for authentication purposes.

Login

The diagram represented in figure 3.8 describes the login phase for a registered user in the system. As the user enters the platform the login form is showed him and this process begins.

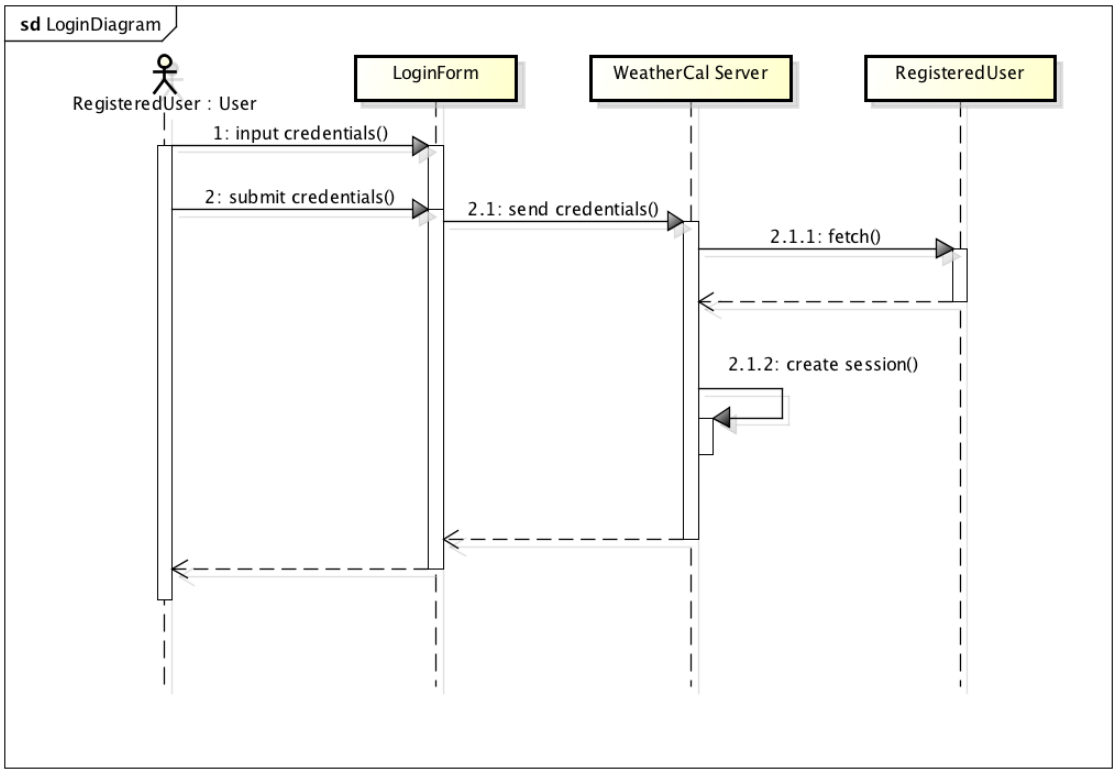

Figure 3.8: Login Sequence Diagram

First of all the user fills the form with his username and password, which he decided during the registration. As he submits his credentials they are sent to the server. The server then tries to fetch a User from the persistance, but if not found an exception will be thrown and the user will be notified about the wrong data inputed. In this sequence diagram, however, the credentials in the form are supposed to be valid, and

so the user will be authenticated to the system and identified by his own session, so he won't need to login for every action he wants to perform.

Event creation

In the diagram represented in figure 3.9 the user will create a new event and customize it as he wants. The user will be able to start this procedure by clicking on the calendar and choosing to create a new event, making a form for inputing event data appear.

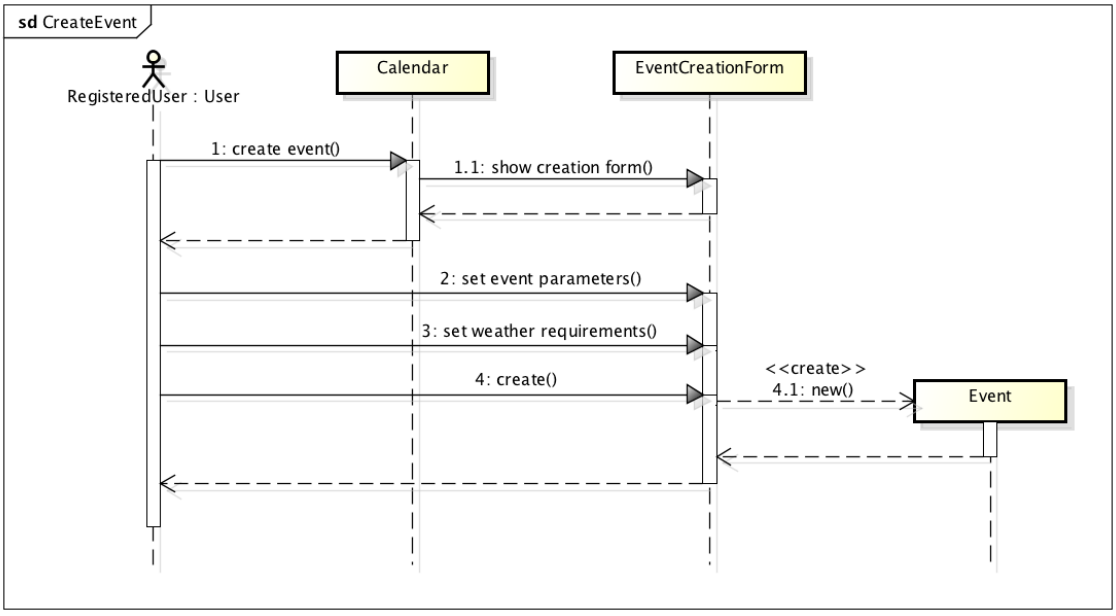

Figure 3.9: Create Event Sequence Diagram

As the form opens the user is able to set all the parameters he desires for the event, such name, place, time, visibility and so on. He will be also able to set the constraints he prefers for the event, i.e. a temperature higher than a certain value, or that the sky must be sunny. After that all the required fields are filled, inputed data is valid

and the user submits the form, the event is created and shown in his calendar. If data is not valid or required fields were not filled, an appropriate exception will be thrown and the user will be warned.

Event modification

When a user selects an event he will be able to enter a form in which he can modify certain parameters. This behaviour is described in figure 3.10.

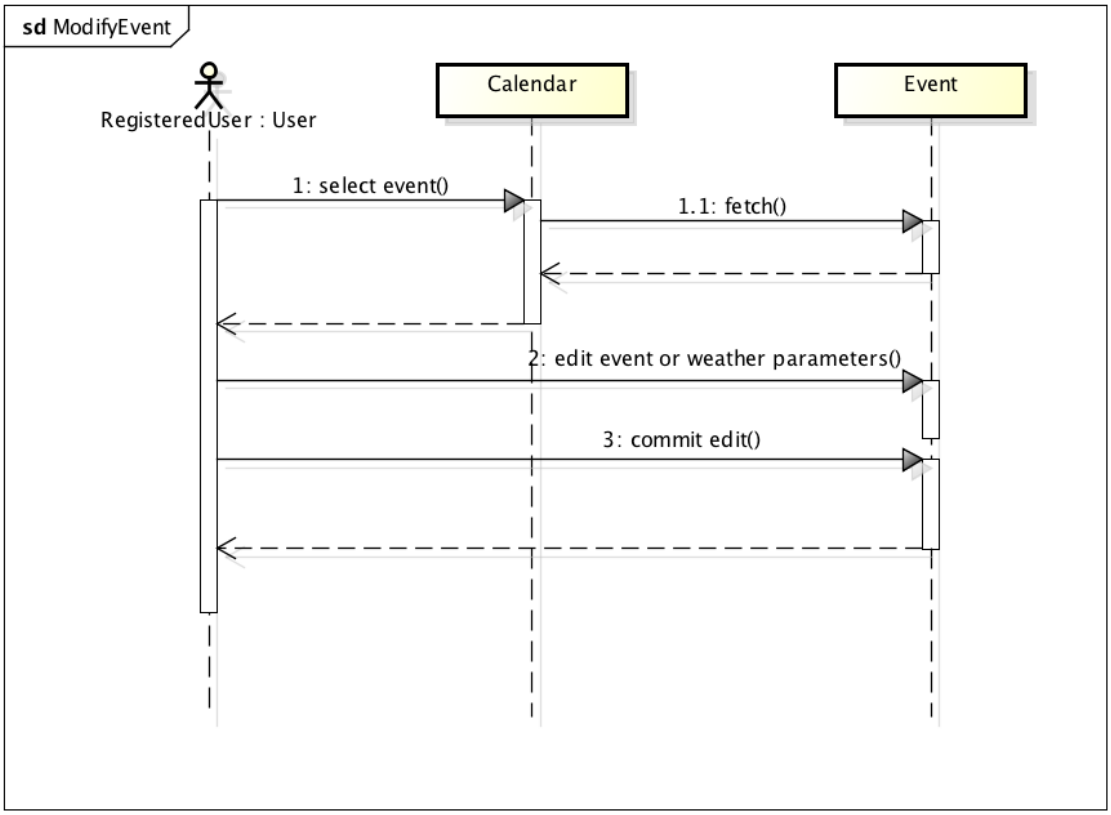

Figure 3.10: Sequence Diagram of event's modification

When a modification of an event is requested, it is fetched and displayed to the user, in a form similar to the creation form. He can so change whatever is shown

as editable and then, when he confirms the modification, they are saved serverside and showed both to his and to the invited users' calendars.

Event deletion

In this diagram, shown in figure 3.11, the process with which a user deletes an event is described.

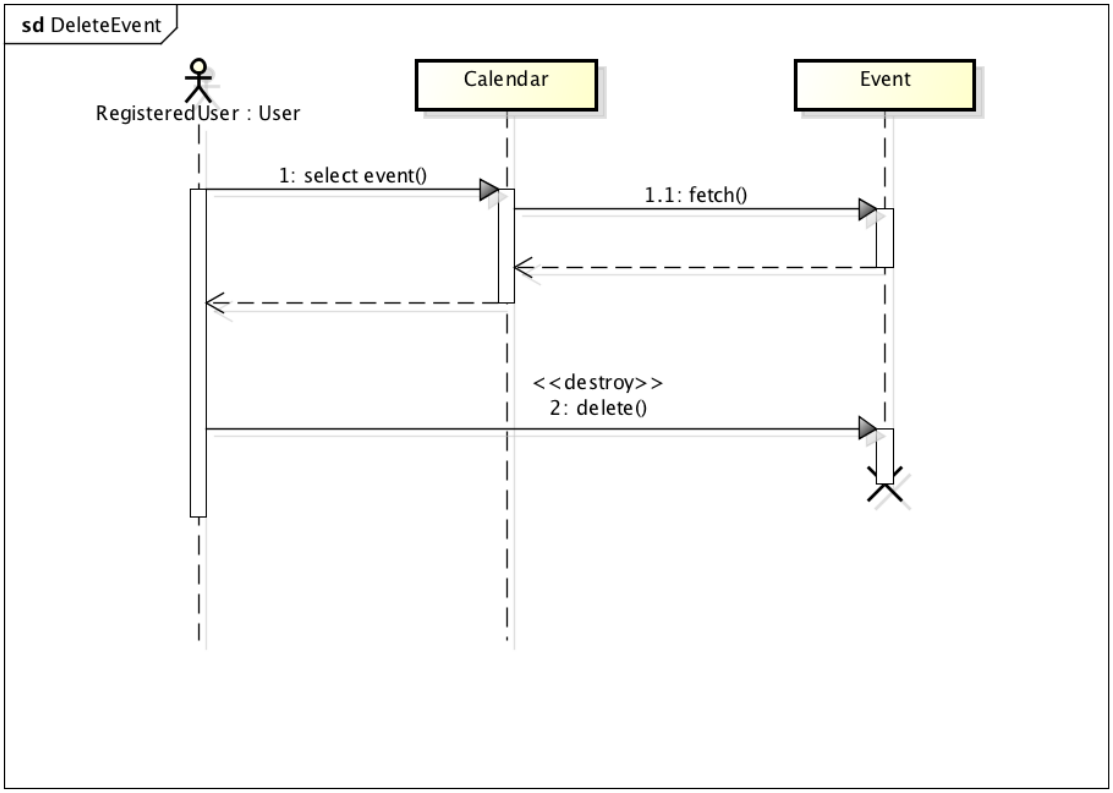

Figure 3.11: Sequence Diagram of event's deletion

Starting from the calendar, a user selects an event for which is owner, and clicks the deletion button. The first selection makes the event to be loaded and then the click on the button, makes it to be deleted. Thus, this last action must be performed by

the owner of the event, and an authorization check must be performed during the deletion, throwing an exception and so an error if a user tries to delete an event for which he is not the owner.

Invitation

Figure 3.12 shows the steps for inviting a user to an event. This process begins with a user selecting an event for which he is the owner.

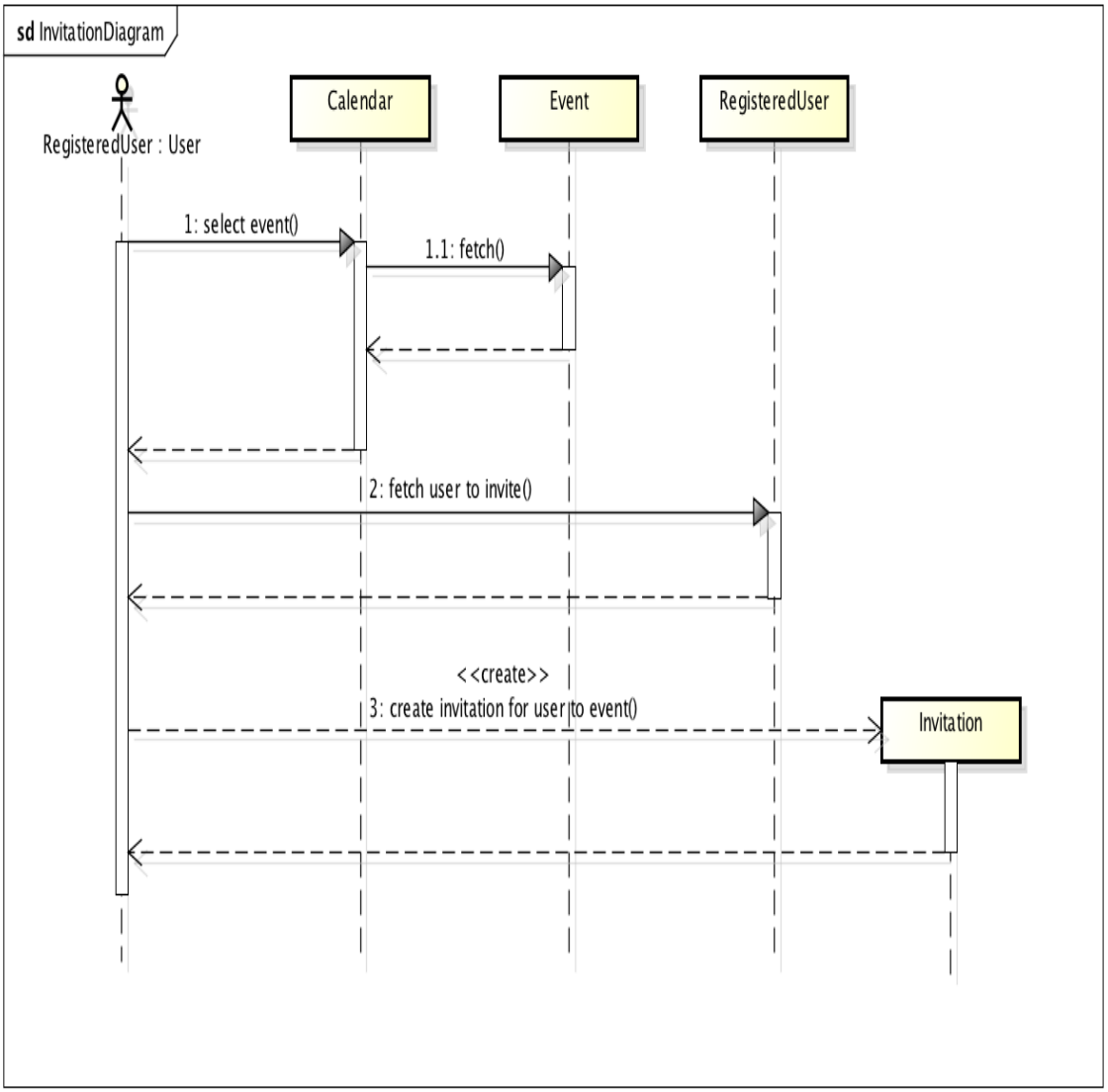

Figure 3.12: Sequence Diagram of an invitation

As he does so, the event is loaded and the user is able to look for people to invite to his event. As the users are selected, and the modification is confirmed, the invitation are created. This makes the invitation to be notified at the invited users' next login. Anyway this process can end with an error, and then an exception, when a user tries to invite people for an event for which he is not owner. This will be not allowed as an authorization politic and so any invitation will be created.

Invite notification

Here the invite notification is described, and shown in figure 3.13. The behaviour described in the former paragraph asynchronously triggers this procedure.

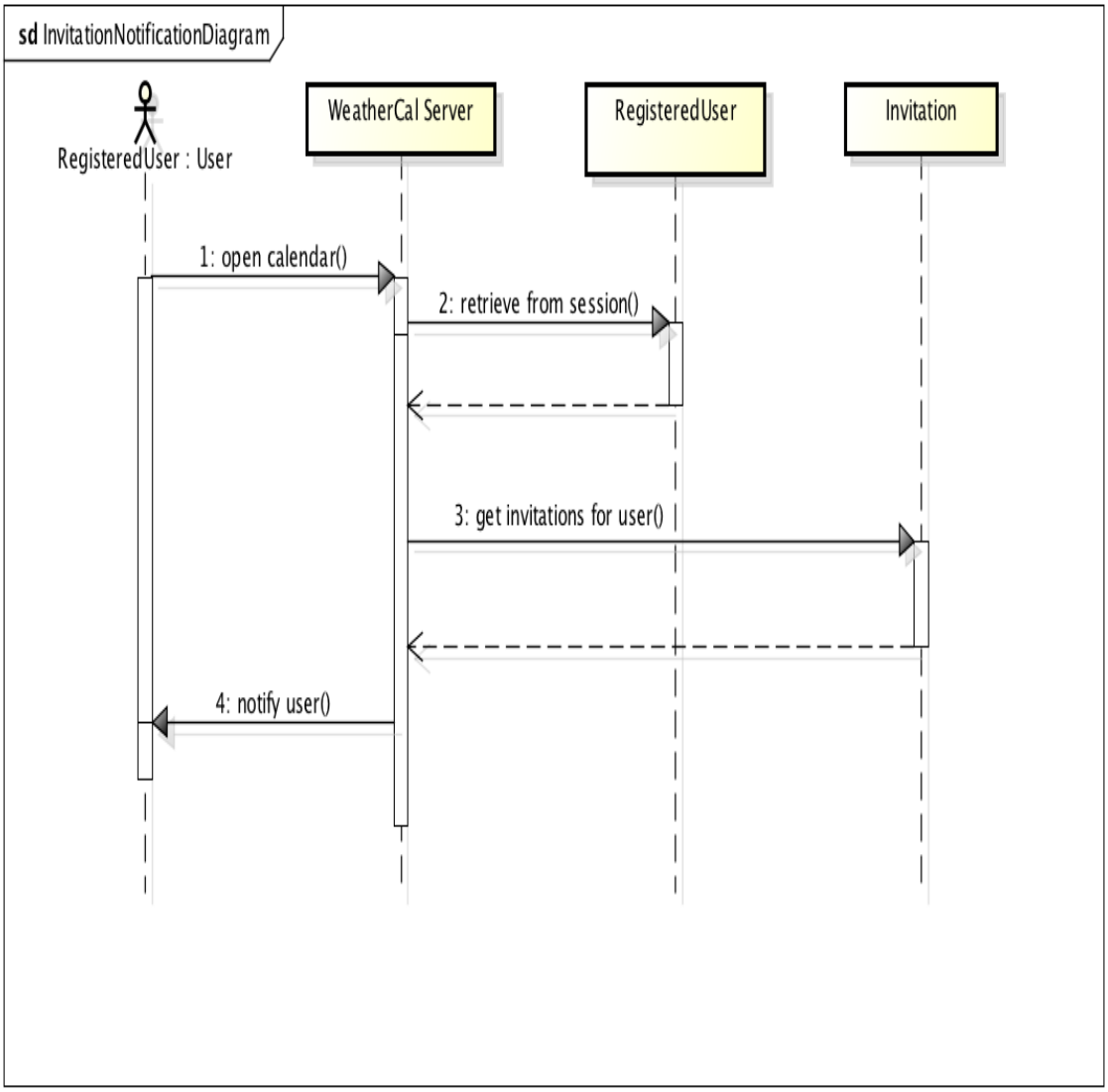

Figure 3.13: Sequence Diagram of an invitation's notification

As the user logs onto the platform and his calendar starts to be loaded he can notice that he was invited to an event (or more than just one). This happens because as the server notices that that user opens the calendar, it looks for unseen notifications. If it finds at least one notification it shows the user a message that informs him about that, giving him the possibility to choose what to do or even ignore that notification.

Modify participation

In the figure 3.14 a description of what happens when the user changes his mind about the participation to an event is illustrated.

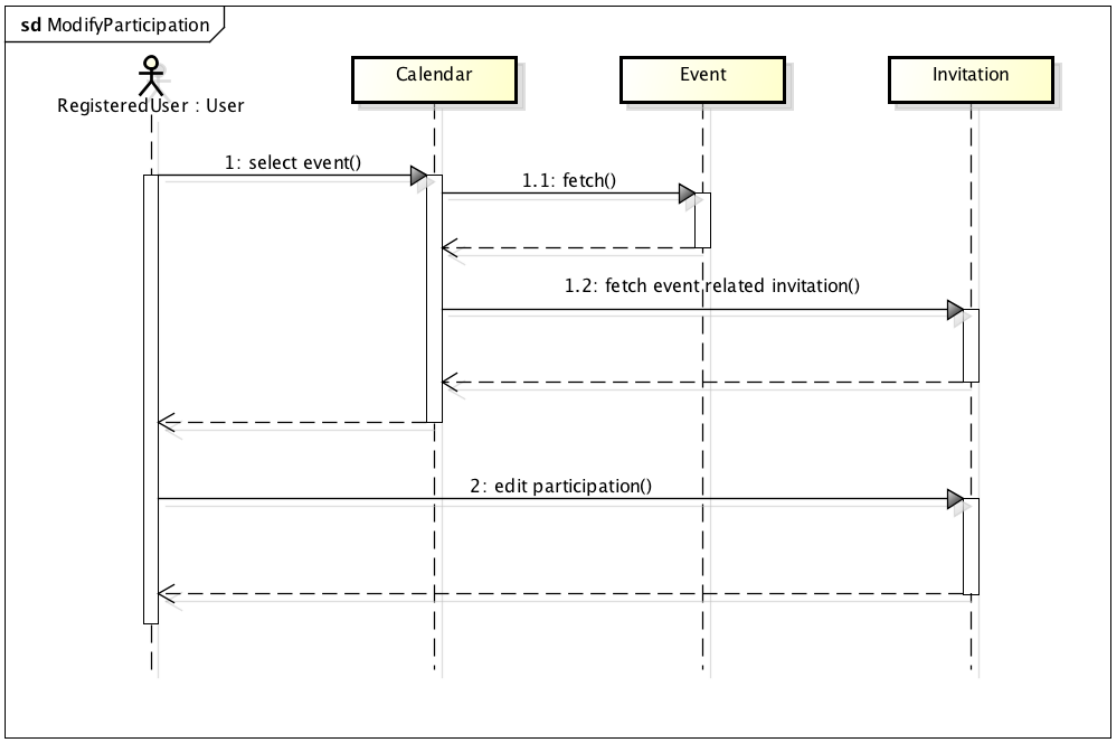

Figure 3.14: Modify participation Sequence Diagram

As he selects an event from his calendar, its data is fetched and, in case he is an invited to the event, the invitation is fetched too. In this case he will be able to

change the state of the participation, as described in the related diagram in section 3.1.3. As he modifies it the change is reported onto the server. If invalid values for the state are sent to the server, it will throw an exception and no modification will occur.

3.2 Alloy

We use Alloy Analyzer to figure out if our Class Diagram was designed in a consistent way so that even our system will be implemented in a consistent style.
Below are reported the code used to represent our system and some models generated by the Alloy Analyzer tool.

3.2.1 Code

```
//SIGNATURE
sig RegisteredUser{
calendar:one Calendar
}
sig Invitation{
Guest: some RegisteredUser,
event:one Event
}
sig WeatherConstrain{
}
sig Calendar{
}
sig Event{
 Owner:one RegisteredUser,
 constrain:one WeatherConstrain
}
//FACTS
fact calendarConstrain{
//the number of Calendar must be equals to the RegisteredUser
#Calendar=#RegisteredUser
//two or more RegisteredUser cannot refer to the same calendar
no disj r1,r2:RegisteredUser | r1.calendar=r2.calendar
}
fact eventConstrain{
```

```
25 //the number of Invitation must be equals to Event
   #Event=#Invitation
27 //each event refer to a different weather constrain
   #WeatherConstrain=#Event
29 no disj e1,e2:Event | e1.constrain=e2.constrain
   }
31 fact invitationConstrain{
   //two or more Invitation cannot refer to the same event
33 no disj r,p:Invitation | p.event=r.event
   //the owner of an event cannot receive the invitation of its event
35 all  disj e:Event , i1: Invitation | i1.event=e implies e.Owner not in
       i1.Guest
   }
37 //ASSERTION
   assert oneCalendarforUser{
39 no disj r1,r2:RegisteredUser | r1.calendar=r2.calendar
   }
41 //check oneCalendarforUser
   assert  ownerNotReceveir{
43 all  disj e:Event , i1: Invitation | i1.event=e implies e.Owner not in
       i1.Guest
   }
45 //check ownerNotReceveir
   assert oneInvitationforEvent{
47 no disj r,p:Invitation | p.event=r.event and p.Guest = r.Guest
   }
49 //check oneInvitationforEvent
   assert onecal{
51 #Calendar=#RegisteredUser
   }
53 //check cal
   assert oneConstrain{
55 no disj e1,e2:Event | e1.constrain=e2.constrain
   }
57 //check oneConstrain
   pred show(){}
59 run show
```

Listing 3.1: Alloy Analyzer code

As shown in figure 3.15 the *Assertion* hence even the *Facts* made in the listing 3.1 are verified because no counterexample was found therefore our system is designed in a coherent way.

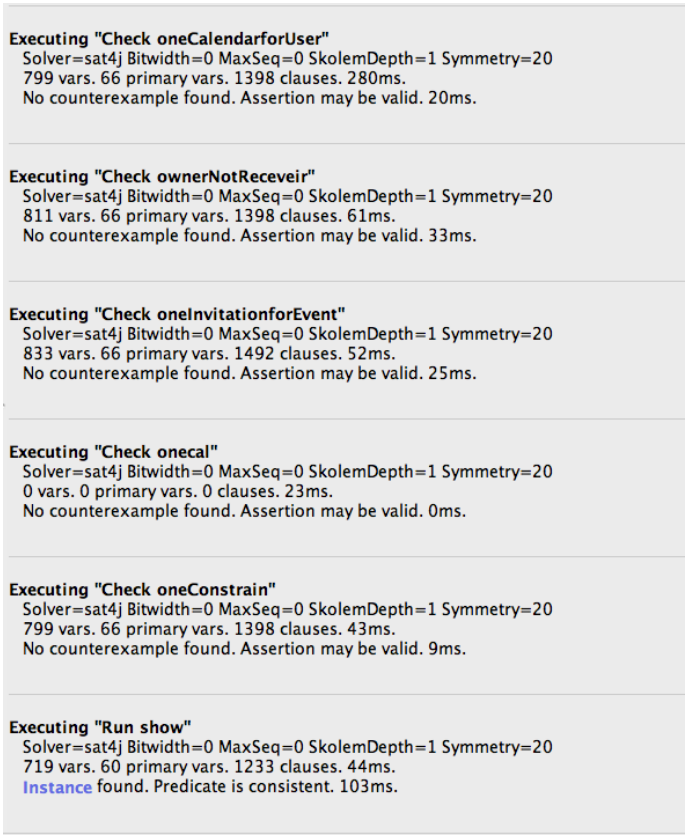

Figure 3.15: Assertion verified

3.2.2 Models generated

In this section are reported some of the various scenarios representing the model of our system.

Interaction between all users

The model in figure 3.16 represents a scenario in which all the users are both event's owner and event's invited. As we can observe from the model an event's owner cannot be a guest of the invitation and all the event should refer to one at only one owner.

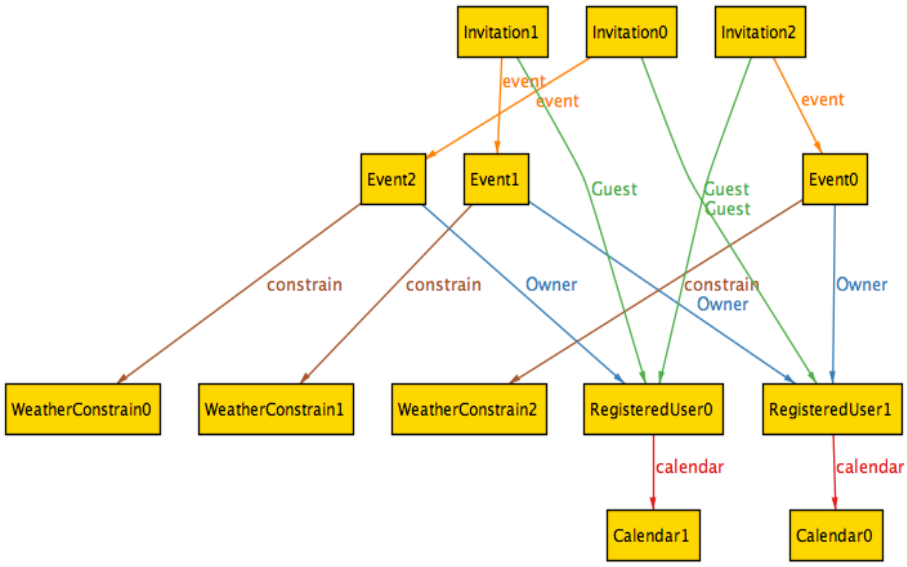

Figure 3.16: All user interaction model

Interaction only between a few users

The model in figure 3.17 represents a scenario in which only some users are interacting and other users are not invited at none of the scheduled event.

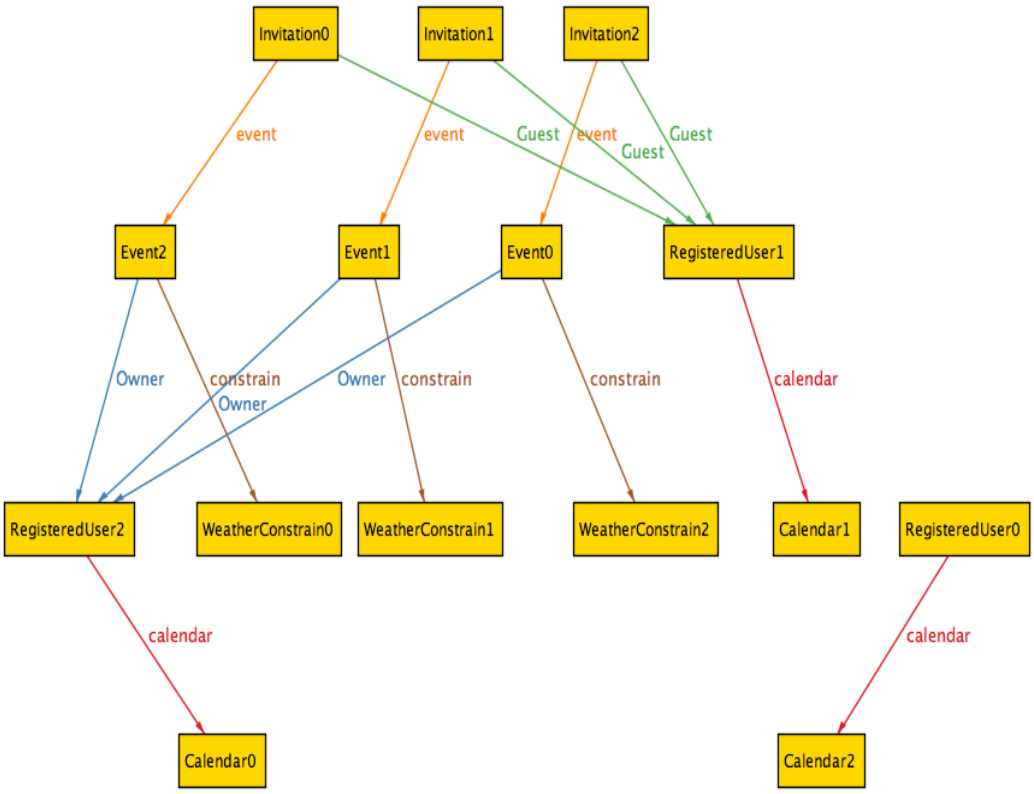

Figure 3.17: Mixed interaction model

No interaction between the users

The model in figure 3.18 represents a scenario in which the users are not interacting each other, it could represent some users that sign in to the platform but never use it.

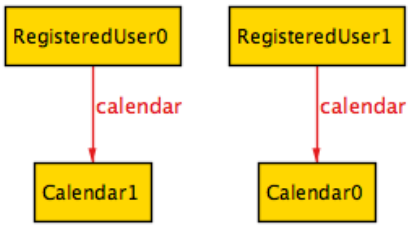

Figure 3.18: No interaction model

Time Reporting

	Paolo Polidori	**Marco Edemanti**
RASD writing	19 hours	19 hours

List of Figures

Listings