Politecnico Di Milano

Engineering of computing systems

Software Engineering II

A.A. 2014/2015

MeteoCal

Requirements Analysis and Specification Document

Professor:

Raffaela Mirandola

Authors:

Alessio Rossotti 823213
Matteo Pasina 837816
Simone Rubiu 835358

November 16, 2014

Contents

1 Introduction

1.1 Description of the given problem

MeteoCal is a new weather online based calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor activities.
The system allows users to registrate in order to create, delete and update events, which should contain information about when and where the event will take place, whether the event will be indoor or outdoor. Time consistency is managed when creating an event by avoiding conflicts with existing events.
Only the organizer is allowed to update or delete the event.
During the event creation, any number of registered users can be invited, who can only accept or decline the invitation.
Whenever an event has been saved, the system adds weather forecast information to it, enabling notification to all event participants one day before the event in case of bad weather conditions for outdoor events.
In that case, three days before the event, the system should propose to its creator the closest (in time) sunny day (if any). Notifications are received by the users by email and also when they log into the system.
Weather information associated to events are periodically updated every 12 hours and, of course, outdoor event participants are notified in case the forecast has changed.
Events can be defined as public or private by their owners, upon creation.
Public events are visible to other registered users, who can see its details, including the corresponding participants. Private calendars shows instead only when its owner is busy but without seeing the details of the event. Users are also allowed to export their calendars.

1.2 Goals

MeteoCal has to provide these main features to its users:

- Registration
- Creation of an event
- Updating or deleting the event (only for the organizer)
- Inviting other registered users (only for the organizer)
- Add other users to her/his personal address list
- Accept or decline an invite for an event
- Import and export calendars
- Set his/her own calendar private or public
- See details of public calendars' events
- See event's weather forecasts

1.3 Glossary

- **Calendar:** the calendar is the table where we will put user's events.
  It shows the current month and the user can select a single day and see programmed events for that day or create a new event (only for the calendar's owner)
- **Event:** for event we meant every possible activity that a user want to schedule in his calendar. It can be an appointment with the dentist, private and with only one participant, or a disco party, public and with hundreds of participants
- **User:** a user is every one with an account, it can be a person or a company
- **Invite:** is a request to participate to an event, it is sent through the system to his/her user profile and to his/her e-mail account
- **Notification:** is a message related to an event, it can be an invite or a weather forecast provided to event's organizer that alerts him/her of bad weather conditions
- **System:** is the overall of the software that we are going to develop

1.4 Assumptions

Some things in the assignment are not clear so we made this assumptions:

- Users have an address list that contains other users' contacts. It is used to invite people to events or to view their public calendars
- Contacts can be added in the list above through e-mail or username research
- When an organizer sends invites for his/her event to users not present in his/her address list and if he/she checks a check box ("automatically add new contacts"), new contacts will be added
- We think that the main feature of this application are the weather forecasts, so we don't want to confuse our users with friendships or other social features. We will focus on the main aspect: to do a good calendar software, clear and efficient
- We have not added the Administrator role, as we consider it unnecessary
- Every event can be organized by only one user, and the organizer can't be changed since event has been created
- When an event is created it can't overlap other user's events, but user can accept invites for events that are programmed on the same time slots

1.5 Proposed system

We propose a web application based mainly on Java EE, CSS 3 and HTML that achieves the goals above, helping the users to schedule their events.
With events are provided weather forecasts that allow users to be informed and organizers to schedule an outdoor event in a beautiful day. It will have all the features of a normal agenda but with the plus to manage the outdoor events checking the weather.

1.6 Identifying stakeholders

The financial stakeholder for our project is the professor who commissioned us the project. The professor wants to have a product that works and satisfies every specification gave in the assignment. A stakeholder for this software is every person that has to schedule his events and commitments that are for the most outdoor and so is interested of knowing the weather. So probably sportsmen and also organizers of big events that are outdoor, like concerts or marathons, can be interested in this product.

2 Actors Identifying

- **Guest:** a person that has not registered or that has not already authenticated.
- **User:** a person that has logged in the system.
- **Organizer:** a user that is owner of an event.

3 Requirements

The system has to provide the following:

1. **Registration of a person to the system:**
    - The system has to provide a sign up functionality: to the user are asked name, surname, a username, an e-mail and a password
2. **Create and manage the events:**
    - the system will let the user create an event and invite every person he wants using the mail or the username
    - the system will let the organizer define a place and a time for the event
    - the organizer has to specify if the event is indoor or outdoor
    - the system will let the organizer to update the event with new informations and invite new people
    - the system will let the organizer delete the event
    - the users invited can accept or decline the invitation
    - every time a user is invited an e-mail and a notification are sent
3. **Show and manage calendars:**
    - every user can see his own calendar with his events
    - every user can set to public or private the visibility of the calendar
    - every user can see the calendar of someone else if its public, also the events have to be public to see the details about them
    - users can import and export calendars
4. **Update weather conditions:**
    - the system will update the weather for every outdoor event using a weather forecast service
5. **Notify weather:**
    - in case of bad weather conditions for outdoor events, three days before the event, the system proposes to its creator the closest (in time) sunny day (if any).
    - the system will send a notification and an e-mail to every user that takes part to an event if the weather changes

3.1 Functional requirements

- **Guest** he can't see events, users and calendars, so he can only:
  - register to the application;
  - log in.
- **User** he can:
  - create events;
  - accept or decline invitation;
  - make own calendar public or private;
  - import and export own calendar;
  - see public calendars of users;
  - see public events details, including partecipants;
  - create and manage address list;
  - search for users of the system;
  - add users to the address list.
- **Organizer** he can only operate on events that he made, so he can:
  - set an event public or private;
  - delete and modify events;
  - invite users to events throught address list or email.

3.2 Non functional requirements

- *User interface:* the goal of our web application is to develop a easy to use way to schedule events, so it has to be nice to see and intuitive to use.
- *Compatibility:* the system has to be reachable from almost all the today's browsers.
- *Performance:* the response time must be acceptable from the users, so the web pages and the queries to the database have to be quick enough.
- *Security:* the system has to be secure, so the web application have to be seen through an up-to-date browser and through a *https* connection. We will focus on the login phase that is one of the most critical. The passwords will be saved in the DB hashed to preserve the secrecy. There will be some checking in the inputs forms (whitelisting,escaping,ecc.) to avoid SQLinjection.
- *Availability:* the service is granted to be up 24/7, also in case of maintenance.
- *Reliability:* the data given by the users have to be preserved in the DB without the possibility of losing them, so RAID technologies must be used. The informations also must be consistent.

3.2.1 Documentation

We will draft these documents to well-organize our work in the way to do in a fewer time the best work as possible:

- RASD: Requirement Analysis and Specification Document, to well-understand the given problem and to analyze in a detailed way which are our goals and how to reach them defining requirements and specification.
- DD: Design Document, to define the real structure of our web application and its tiers.
- JavaDoc comments in the source code: to make anyone that wants to develop the platform or do maintenance on it understand the code.
- Installation Manual: a guide to install MeteoCal.
- User Manual: a guide to use MeteoCal.
- Testing Document: a report of our testing experience of another MeteoCal project.

3.2.2 Architectural considerations

The platform used for develop the system is Java EE version 8 leaning on glassfish 4.0 and MySql.

4 Scenarios Identifying

4.1 Boris registers

Boris searches for a calendar merged with the weather forecasts because he does a lot of sports. He finds MeteoCal and decides to register because the main page is nice. He clicks on register.The system shows the register form, he compile it with first name, last name, username, e-mail and password. Boris clicks register and gets an e-mail to confirm the registration, so he goes in his e-mail account and clicks on the confirm link. The token sends him to the main page of his new account on MeteoCal.

4.2 Magnus creates an event for his birthday

Magnus is a user of MeteoCal. Next week is Magnus's birthday so he decides to have a party with his friends. He creates a public event using the MeteoCal service and he invites his friends through meteocal system. Some of his friends aren't users of MeteoCal so he puts the directly the e-mails in the invitation form. The system sends e-mails and notifications to the invited people.

4.3 Judit accepts Bobby's invitation

Judit receives the invitation to Bobby's event on Wednesday from 20:00 to 23:00 in her e-mail account. She decided to join the party, so she clicks on the link in the e-mail and accepts the invitation. The system updates Judit's calendar and sets as busy the slots from 20:00 to 23:00 on next wednesday. She has also another event in her schedule and MeteoCal notifies it, but she decides to let both the commitments and to choose later on which event go.

4.4 Akiba declines Bobby's invitation

Akiba receive Bobby's invitation, but first he goes to the event details and he checks the partecipants because he is mad at Judit. Akiba sees that Judit will join the event and he decides to decline the invitation.

4.5 Mikhail receives and e-mail from MeteoCal

Mikhail isn't a Metocal user, but Bobby invited him to his party with MeteoCal. The system generated an automatic e-mail with a link to the event invitation. If Mikhail has already an account on MeteoCal with a different e-mail, he can log-in with his credentials, also he can register and accept or decline the invitation of Bobby.
Mikhail decided to sign up with MeteoCal filling the registration form, after that Mikhail receive an e-mail with the activation link to his account. he activate his account and accept Bobby's invitation.

4.6 Garry searches his friends

Garry has to add several new friends in his address list so he starts to fill the address book with his contacts e-mails and by searching users in MeteoCal system. He writes in the search bar the names of the peopl that he wants to find. Garry finds his friend Judit and he can explore Judit's calendar because is public. Garry sees that Judit will partecipate to Bobby's party next week, he is also invited and he decides to accept the invitation.

4.7 Alexander wants to import his Calendar

Alexander has a habit to schedule all of his duties on an application on his desktop pc. He hears about MeteoCal from his friend Bobby and decides to begin to use it because he often uses his bike to move in the city. He doesn't have time to set all his appointments manually, so he clicks on import calendar and browses his pc till he finds the file of his old calendar. The file is imported and now Alexander has all of his events set.

4.8 Paul's futsal match

Paul wants to organize a futsal match with his friends but wants to know if is going to rain, because he doesn't want to play on a wet court. He uses MeteoCal and he schedules it for a thursday. Unluckily the weather forecast predict rain, so on monday the system sends an e-mail to Paul advising to change the day of the match on sunday that is the nearest sunny day. Paul changes the date of the event to sunday.

4.9 Fabiano goes to a concert with his friends

Fabiano wants to search for his friends for invite them to a concert, so he clicks on the search bar an writes the usernames that he wants to find. The system shows the results, he clicks on add and the user is put in the address list. Then he creates the event and invites them.

4.10 Emanuel changes place

Emanuel has organized a barbecue in the park for his promotion at work and for doing so he uses MeteoCal. On the day choosen the weather forecasts notified are not good, so he clicks on modify event and changes the place of the event from the park to his house(and from outdoor to indoor). The systems sends notifications to the participants. Some of his guests change their minds and delete themselves from the event.

5 UML Models

5.1 Use case diagram

This is the use case that we thought to model our application.

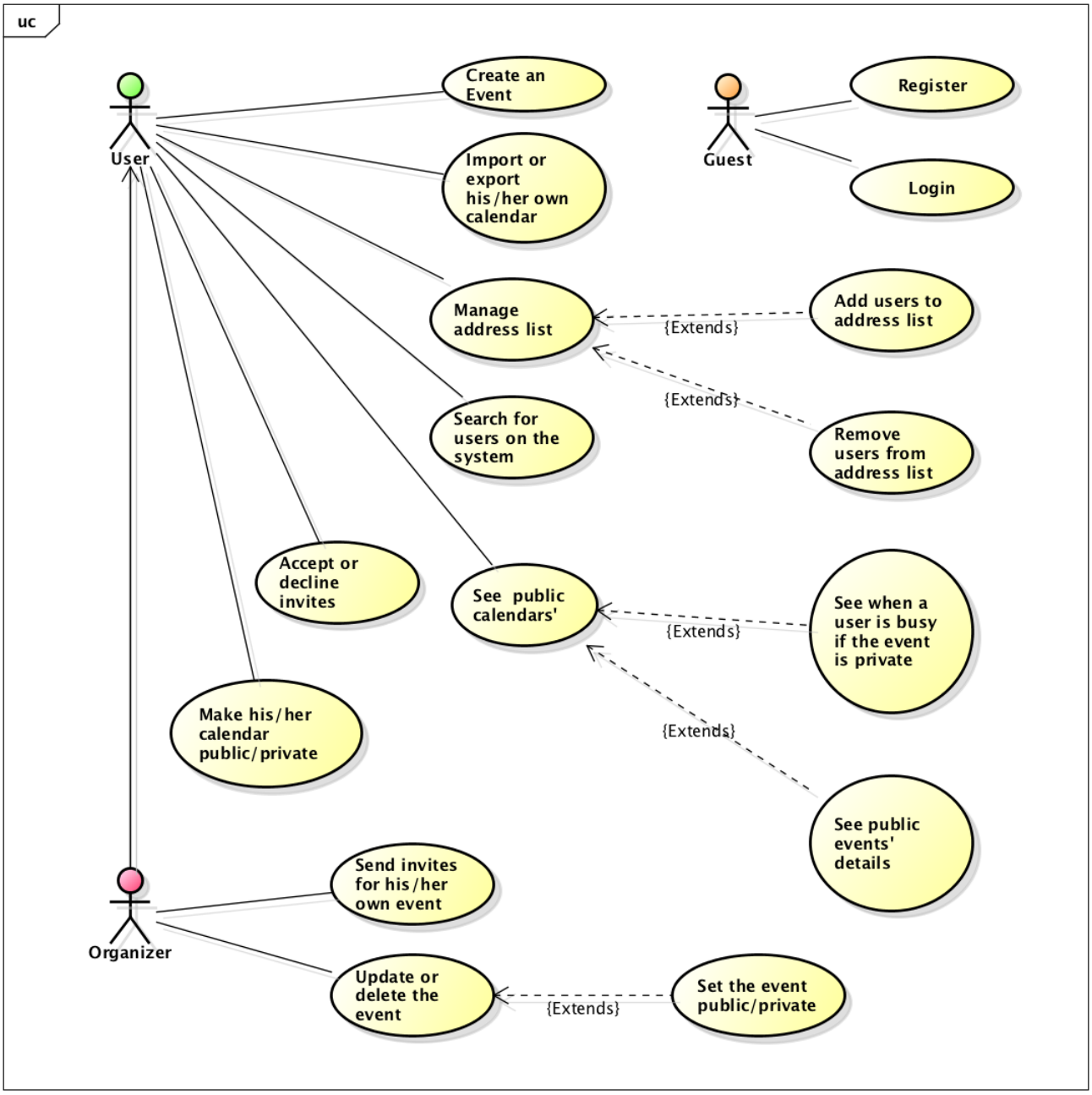

Figure 1: Use case diagram

5.2 Use case description

5.2.1 Registration

Registration

ACTORS: Guest

ENTRY CONDITIONS: Guest user is not registered yet

FLOW OF EVENTS:

- Guest access registration page
- Guest compiles the registration form
- The system verifies the uniqueness of the username and email
- The system sends a confirmation email to the address provided
- Guest confirms the email
- The system confirms the registration

EXIT CONDITIONS: Guest becomes a registered user

EXCEPTIONS:

- Email or username are not unique
- Some fields of the registration form are incorrect
- Guest doesn't confirm the email

5.2.2 Login

Login

ACTORS: Guest

ENTRY CONDITIONS: Guest is registered but hasn't logged yet

FLOW OF EVENTS:

- Guest access home page
- Gust inserts username and password
- The system authenticates the user and redirects him to profile page

EXIT CONDITIONS: User is logged in the system
EXCEPTIONS:

- Email or username are not correct

5.2.3 Create an event

Create an Event

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User clicks on create new event
- The system redirects him to event input form
- User fills in the form and selects users to invite
- The system verifies the correctness of the data
- The system shows the page of the created event
- The system sends an email to invited users

EXIT CONDITIONS: Event created

EXCEPTIONS:

- Form fields incorrect
- Some invited users not present in the system
- User is already occupied in that date

5.2.4 Import/Export calendar

Import or Export Calendar

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User goes on his/her calendar's page
- User clicks on import or export button
- If he/she clicked on import the system makes the user choose a file from the pc
- Old calendar is overwritten
- If he/she clicked on export the system sends the calendar file to the user

EXIT CONDITIONS: Calendar imported/exported

EXCEPTIONS:

- Form fields incorrect
- User is already occupied by that date

5.2.5 Manage address list

Manage Address List

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User goes on his/her address list's page
- The systems shows to the user the list of his/her contacts
- User clicks on add or remove button
- If he/she clicked on remove the system deletes that contact
- If he/she clicked on add the system shows a text field and requests a username
- The user fills in the text field
- The system adds the new contact into the address list

EXIT CONDITIONS: Contact added/removed

EXCEPTIONS:

- Username doesn't exist

5.2.6 Search for users

Search for Users

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User fills in the text field and clicks on search button
- The systems search for matches in the db
- The systems shows results to the user
- User can add him/her to address list or see his/her calendar

EXIT CONDITIONS: The user has found what he was looking for

EXCEPTIONS:

- Username doesn't exist
- User's calendar is private
- Contact already present in the address list

5.2.7 See public calendars

See public Calendars

ACTORS: User

ENTRY CONDITIONS: User is logged into the system, the calendar is public

FLOW OF EVENTS:

- User searches for a user in the system or selects a contact in his/her address list
- User selects view calendar button
- The systems shows the calendar
- User can see when the user is busy if the event is private
- User clicks on a public event
- The system shows event page with all the details including participants

EXIT CONDITIONS: The user has found what he was looking for

EXCEPTIONS:

- The event is private, user can't see the details

5.2.8 Accept or decline invites

Accept or Decline Invites

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User receives both an invitation email and a notification in her/his personal panel
- User clicks on the link in the email
- The systems shows page event
- User clicks on accept or decline button
- If he/she clicked decline nothing happens
- If he/she clicked accept the event is added in his/her own calendar

EXIT CONDITIONS: The user has declined (accepted) the invitation (the event has been added in the personal calendar)

EXCEPTIONS:

- The event date is incompatible (user is busy in that date)
- The event date has already expired

5.2.9 Make calendar public/private

Make his/her calendar public/private

ACTORS: User

ENTRY CONDITIONS: User is logged into the system

FLOW OF EVENTS:

- User clicks on his calendar button
- The system shows up user's calendar
- User clicks on change visibility option and selects private or public
- The system registers the change on the db

EXIT CONDITIONS: The user has changed calendar's visibility

EXCEPTIONS:

- None

5.2.10 Send invites for his/her own event

Send Invites for his/her own Event

ACTORS: Organizer

ENTRY CONDITIONS: Organizer has created an event

FLOW OF EVENTS:

- The organizer is on the event's page
- The organizer clicks on invite button
- The organizer adds usernames from address list or by manual typing
- The organizer checks/not checks the 'add contacts to address list' checkbox
- If the organizer has checked the checkbox, new contacts are automatically added to address list
- The system sends email invites to invited users

EXIT CONDITIONS: The organizer has invited other users

EXCEPTIONS:

- Invalid username(s)

5.2.11 Update or Delete the Event

Update or Delete the Event

ACTORS: Organizer

ENTRY CONDITIONS: Organizer has created an event

FLOW OF EVENTS:

- The organizer is on the event's page
- The organizer clicks on modify button
- The system shows event's edit page
- The organizer can change event's parameters like date,name.location..
- The organizer can click on delete button
- The system registers the change on the db and sends notifications to invited users

EXIT CONDITIONS: The user has updated or deleted his/her own event

EXCEPTIONS:

- Incorrect parameters

5.3 Class diagram

Below we present the class diagram that we propose to you, we have made some changes with respect to the use case described above. During the analysis we found that the shape of the organizer is not necessary since any user can become an organizer, so we modeled the system by holding only the shape of the user, which in any case must be able to change the events of which is the owner.

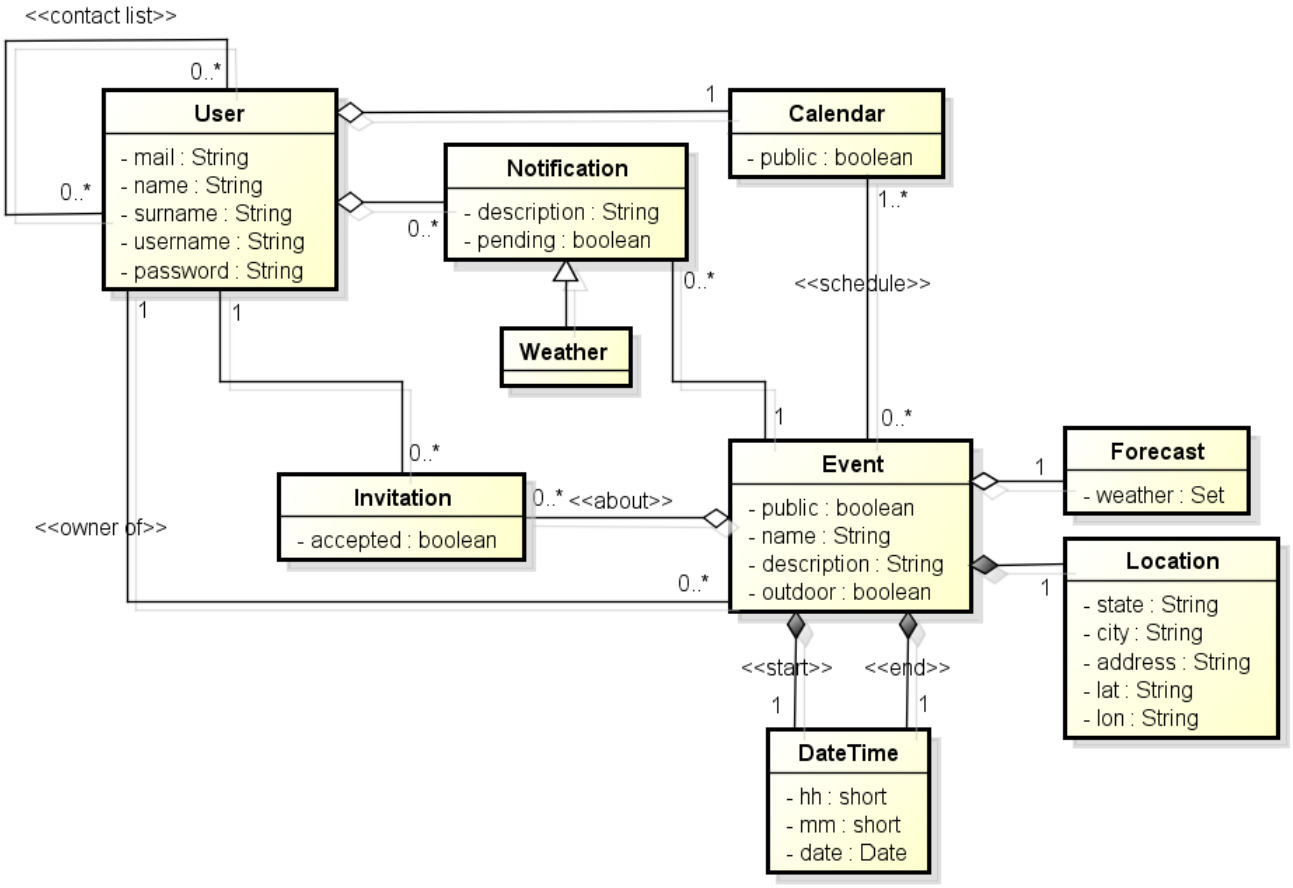

Figure 2: Class diagram

5.4 Sequence diagrams

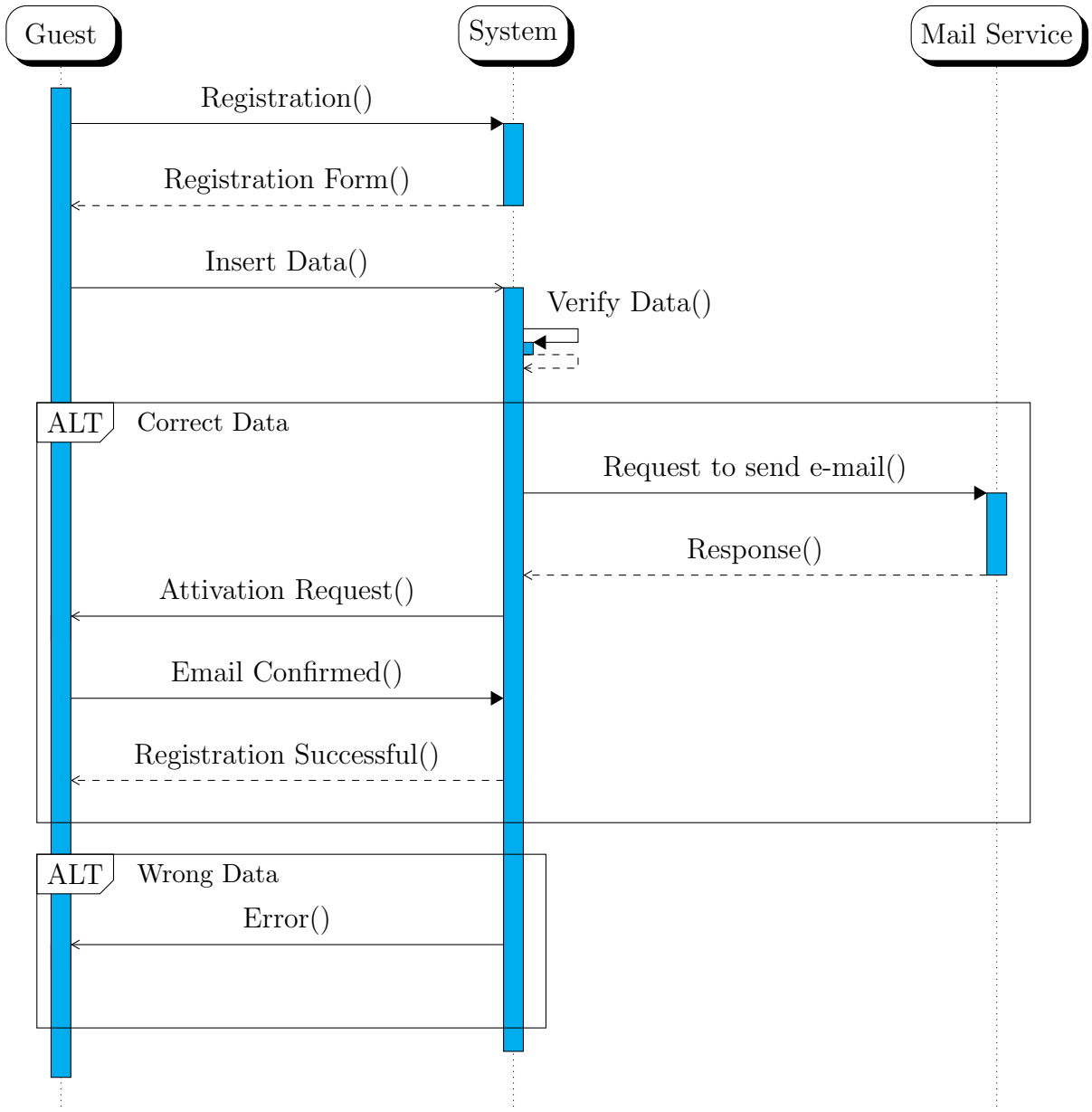

Figure 3.1: Registration

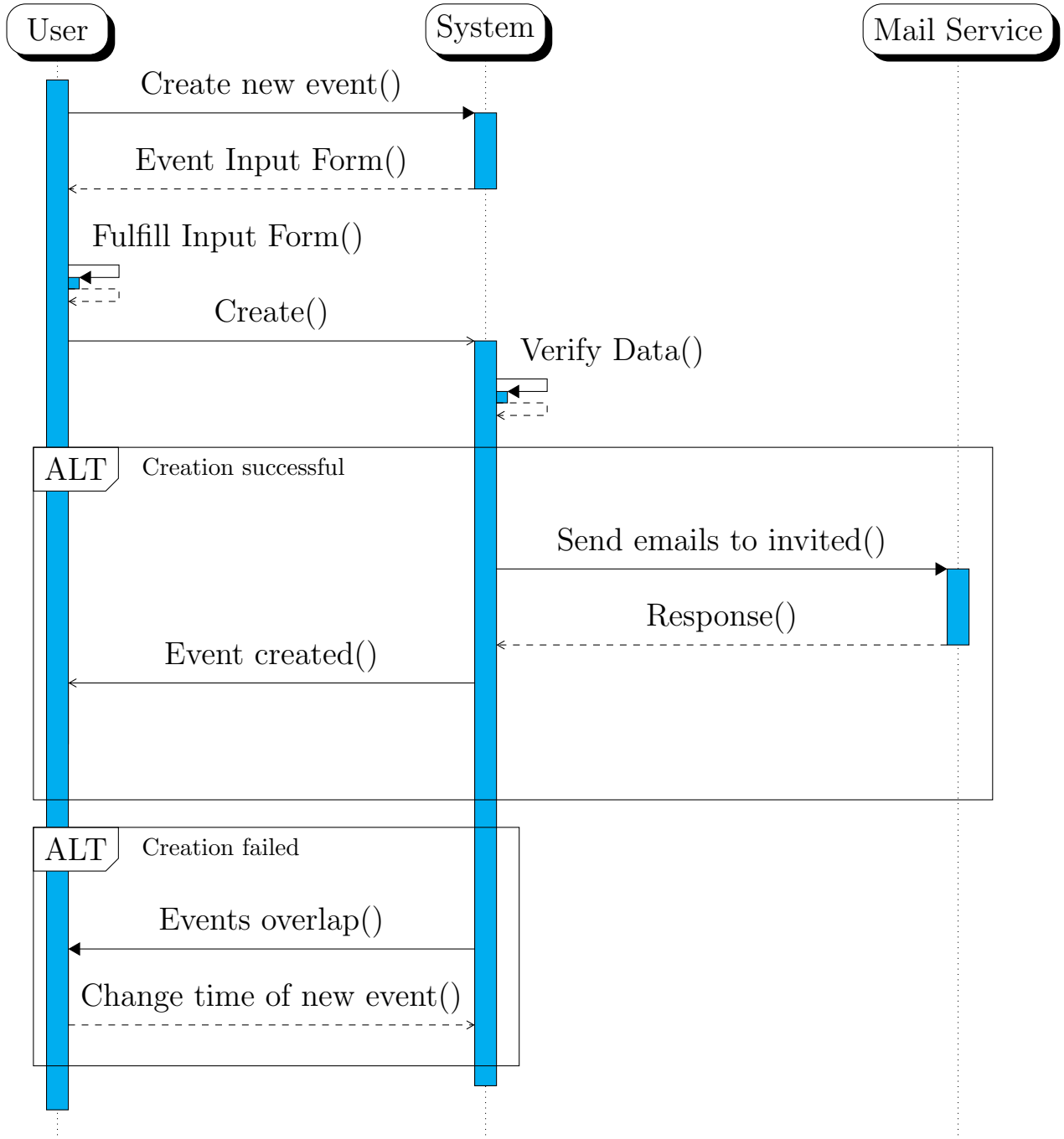

Figure 3.2: Create Event

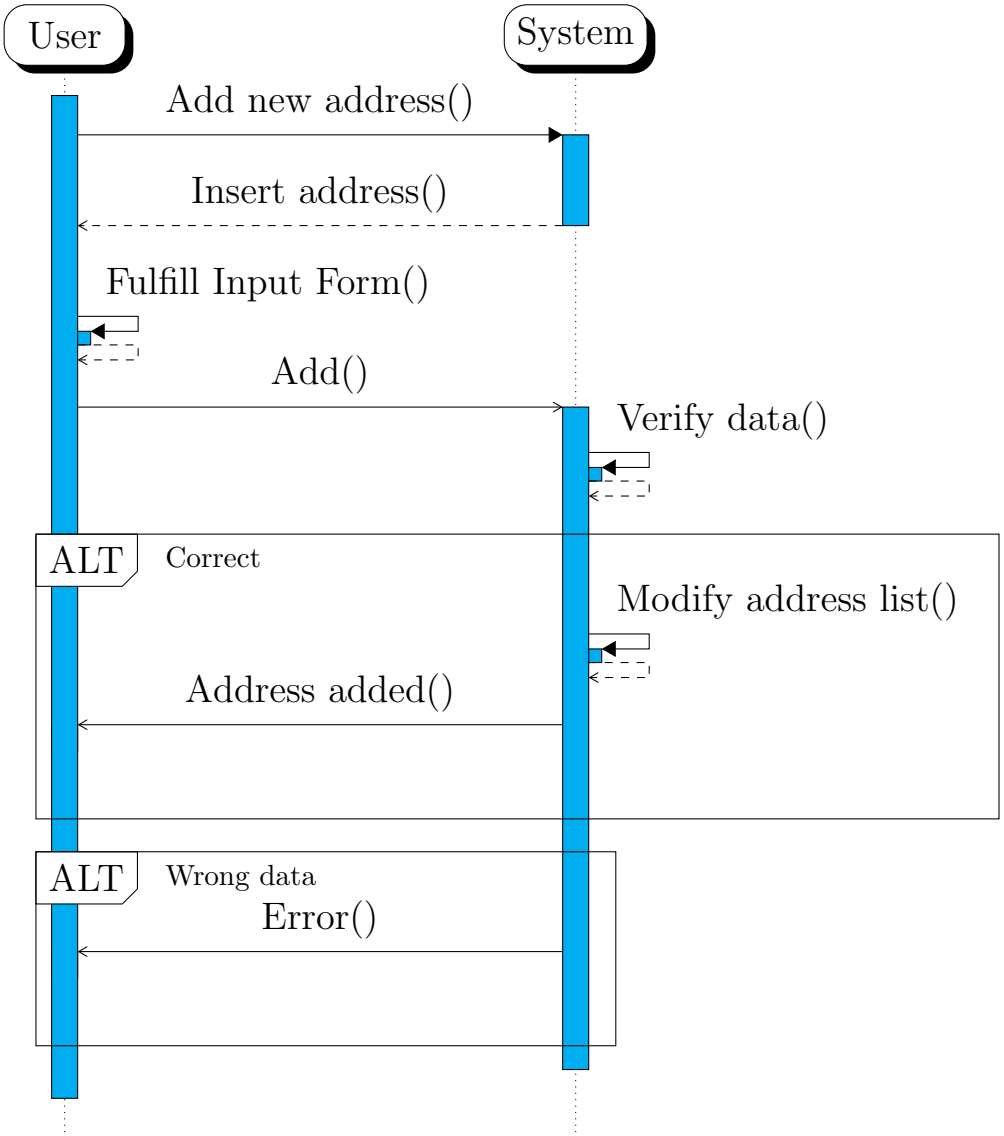

Figure 3.3: Add user in address list

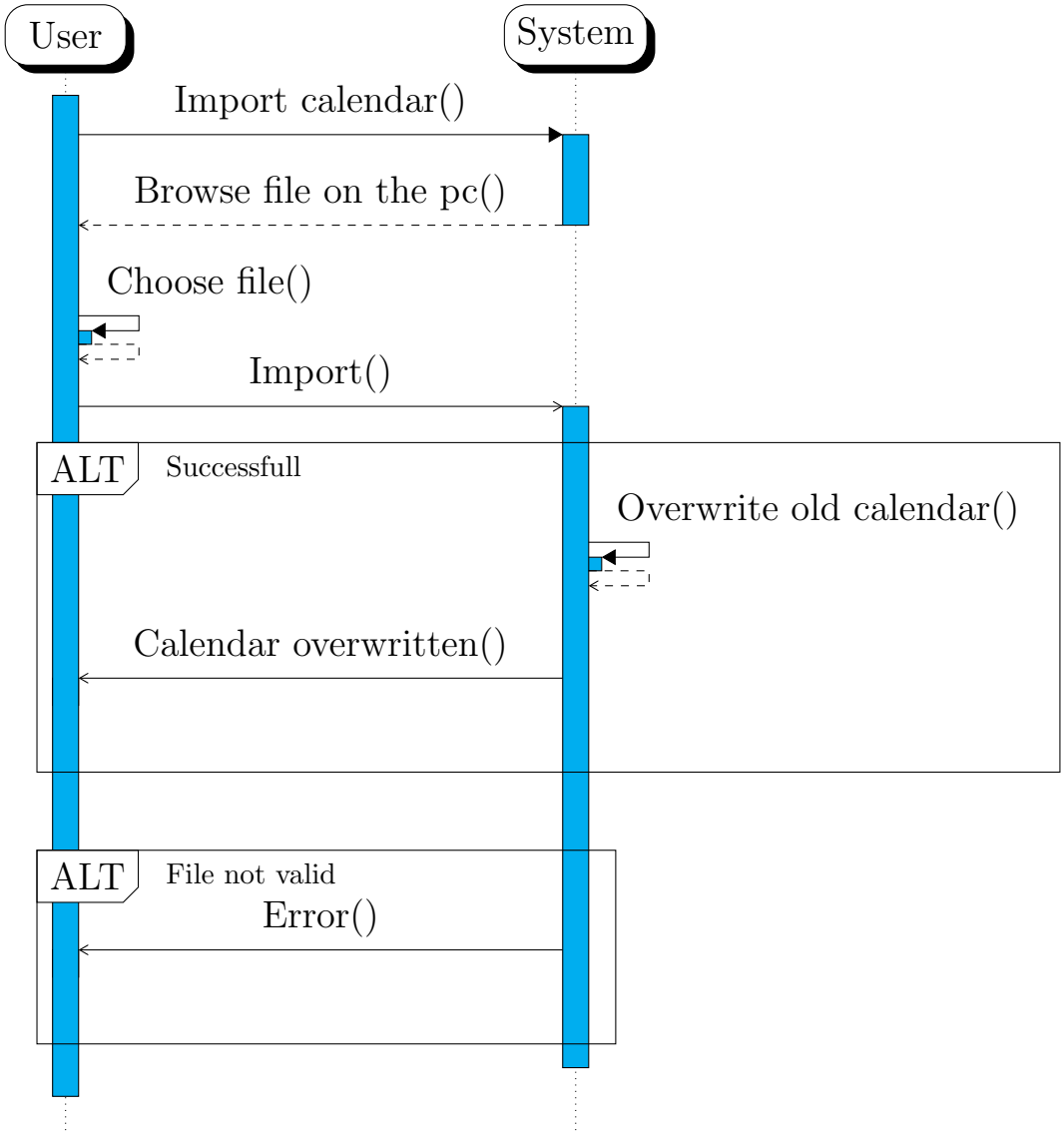

Figure 3.4: Import calendar

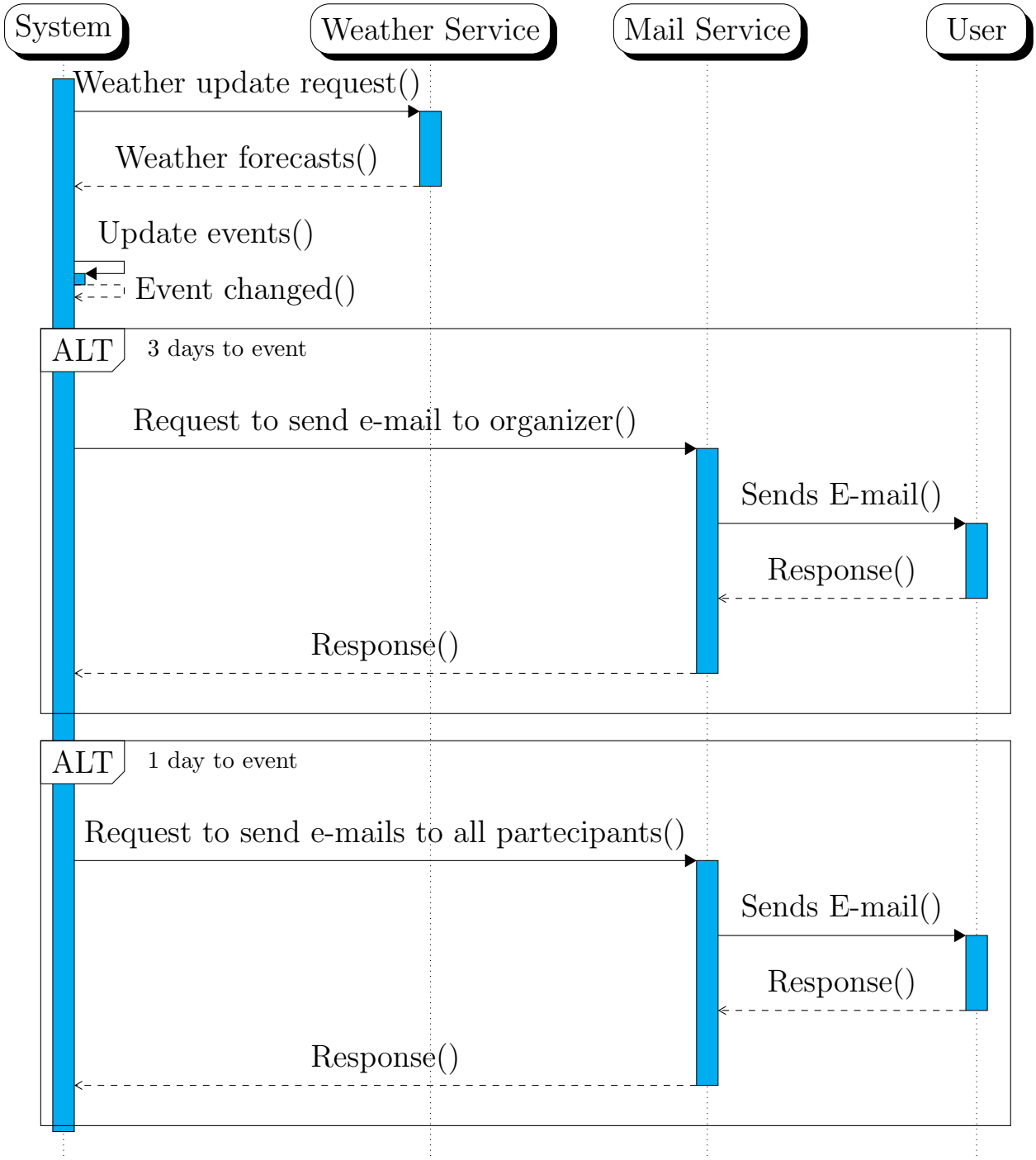

Figure 3.5: Weather changes

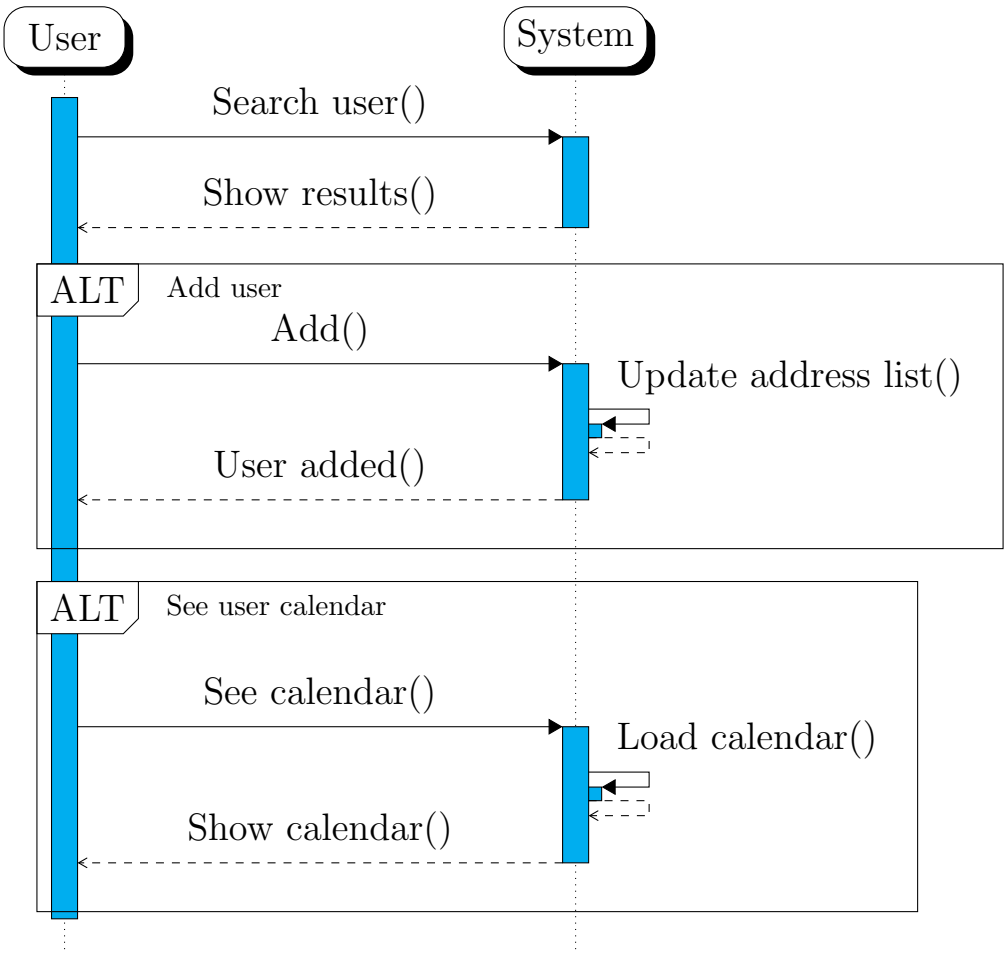

Figure 3.6: Search users

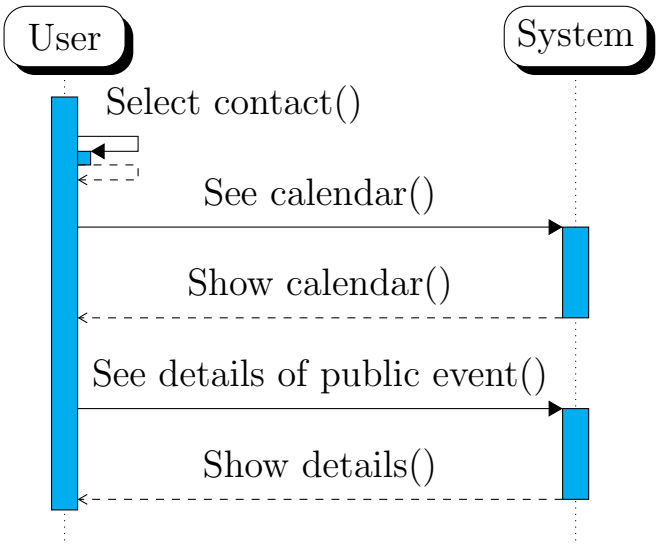

Figure 3.7: See public calendars

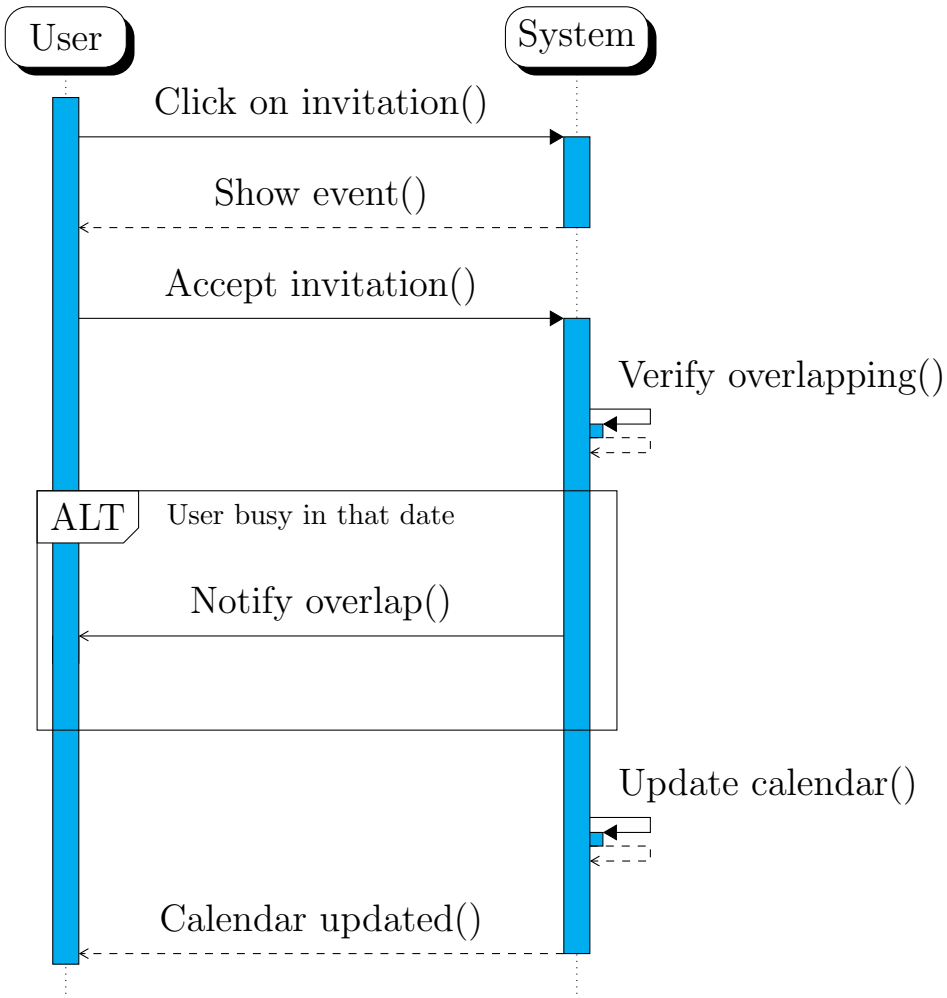

Figure 3.8: Accept invites

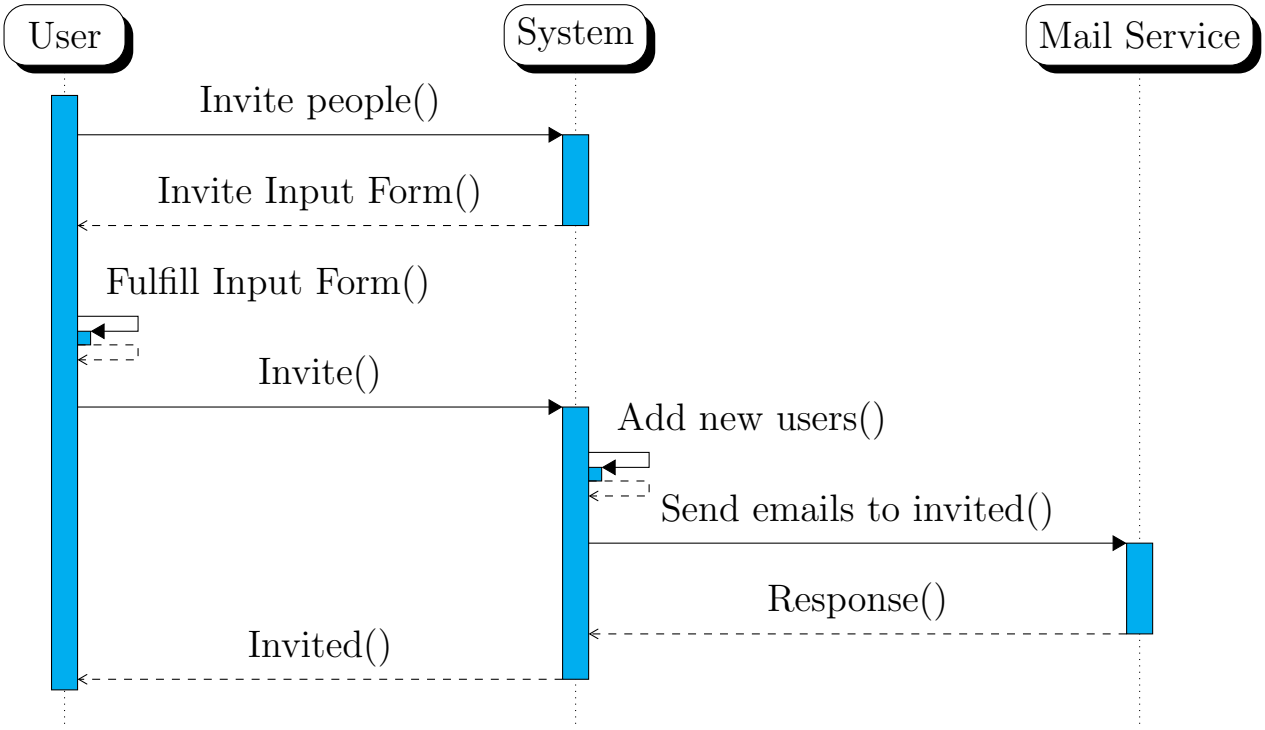

Figure 3.9: Send invites for his/her own Event

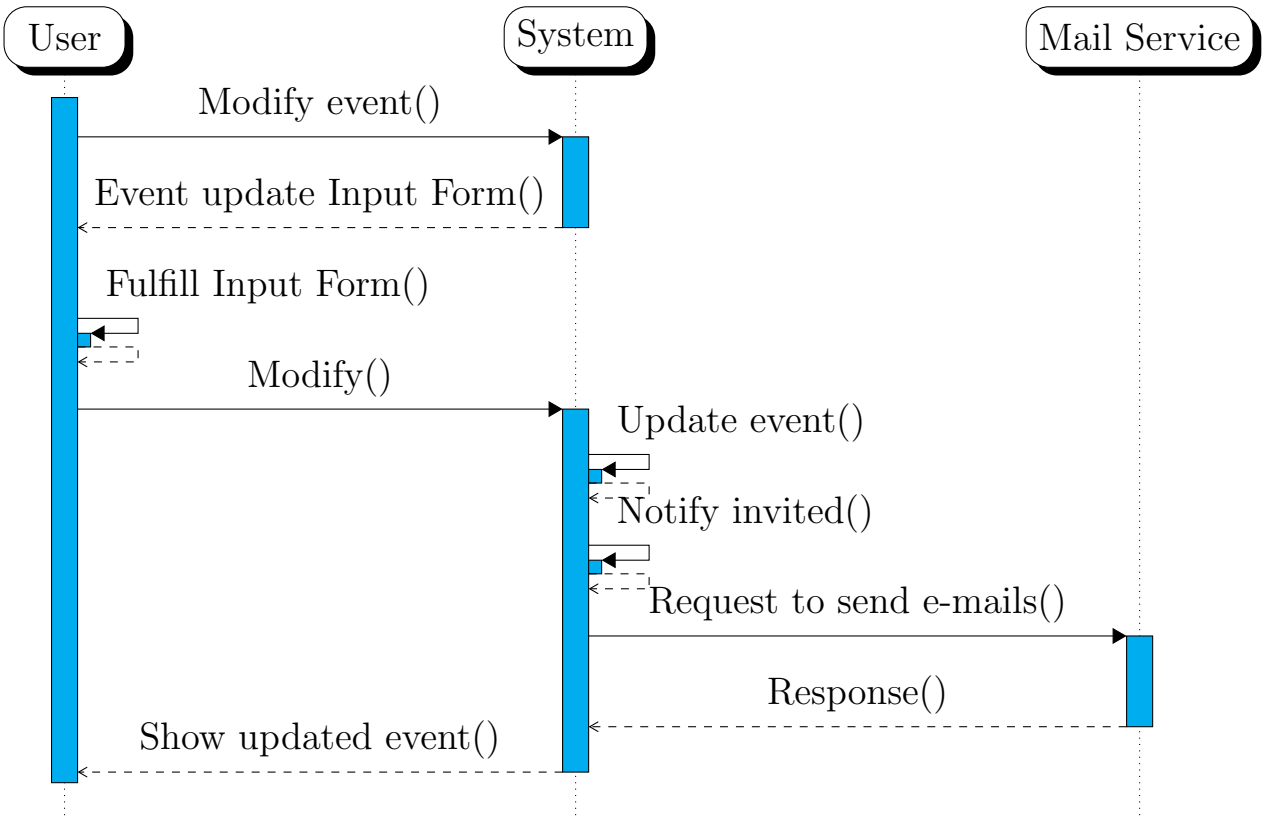

Figure 3.10: Update event

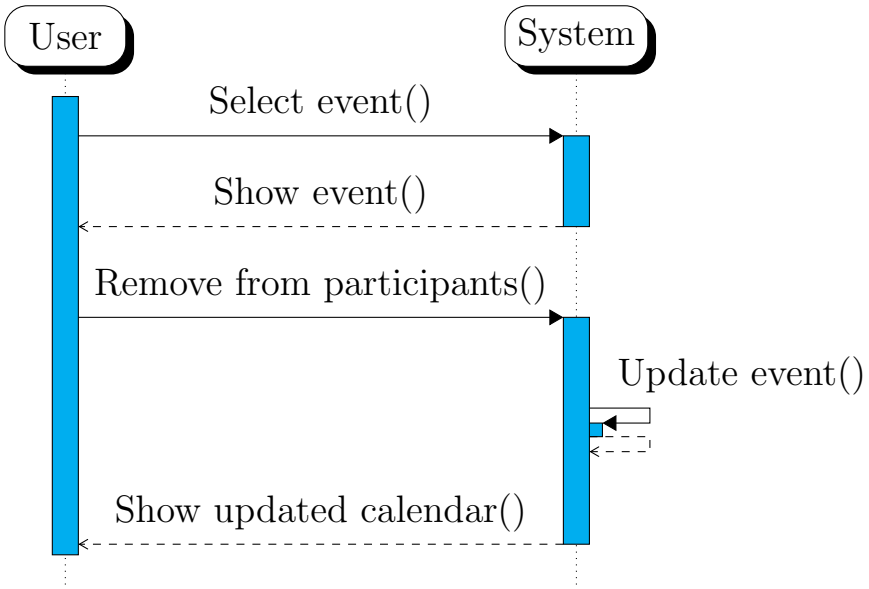

Figure 3.11: Remove from participants

6 Alloy Modeling

In this paragraph we try to understand if our Class Diagram can be consistent using Alloy Analyzer. We report below the code used and some World generated by our predicates just to let understand that our model is consistent.

Listing 1: DateTime definition

```
open util/integer
sig DateTime{
     mm: one Int,
    hour : one Int,
    day : one Int ,
    month : one Int ,
    year : one Int
}
//check that the data are consistent
//year is between 0-5 because in alloy you can't have large integer, so 0 is the current year, while 1 is the next year and so on.
{
    hour>=0 and hour<24
    mm >=0 and mm <60
    day >= 1 and day <= 31
    month >= 1 and month <= 12
    ( month = 4 or month = 6 or month = 9 or month = 1 ) => day <=30
    month = 2 implies day <= 28
    year >=0 and year <= 5
}
//check if d1<d2
 pred before[ d1 , d2 : DateTime ] {
    ( d1.year < d2.year ) or
    ( d1.year = d2.year and d1.month < d2.month ) or
    ( d1.year = d2.year and d1.month = d2.month and d1.day < d2.day ) or
    ( d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.hour < d2.hour) or
    ( d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.hour = d2.hour and d1.mm < d2.mm)
}

//check if d1=d2
pred equal[ d1 , d2 : DateTime ] {
    d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.hour = d2.hour and d1.mm = d2.mm
}

//check if d1<=d2
pred beforeOrEqual[ d1 , d2 : DateTime ] {
    before[ d1 , d2 ] or equal[ d1 , d2 ]
}

//check if d2<d1
 pred after[ d1 , d2 : DateTime ] {
    before[ d2 , d1 ]
}

//check if d2<=d1
pred afterOrEqual[ d1 , d2 : DateTime ] {
    not before[ d1 , d2 ]
}
```

Listing 2: MeteoCal alloy

```
module meteocal
open DateTime
/*******************************
**********SIGNATURES***********
*******************************/
sig Email{}
sig Username{}
sig Forecast{}
sig Location{}
sig User{
    contactList : set User,
    mail : one Email,
    username :one Username,
    calendar : one Calendar,
    notifications : set Notification
}
sig Event{
    place : one Location,
    start : one DateTime,
    end : one DateTime,
    forecast : one Forecast,
    invitationList : set Invitation,
    owner : one User
}
sig Calendar{
    events : set Event
}
sig Notification{
    event : one Event
}
sig Invitation{
    user : one User
}
/*******************************
************FACT***************
*******************************/
//Every mail has a user
fact{
    all m:Email | one u:User | m = u.mail
}

//Every Username has a user
fact{
    all un:Username | one u:User | un = u.username
}

//Every calendar has a User
fact{
    all c:Calendar | one u:User | c = u.calendar
}

//e-mails must be unique in the system
fact {
    no disj u1,u2: User | u1.mail = u2.mail
}

//usernames must be unique in the system
fact {
    no disj u1,u2: User | u1.username = u2.username
```

```
60 }
61
62 //I can't have myself in my contacts
63 fact {
64     all u1,u2:User | u1 in u2.contactList implies u2 !=u1
65 }
66
67 //I can't be among the participants of events that I haven't been invited
68 fact {
69     all u1:User, e:Event |e in  u1.calendar.events implies u1 in e.
       invitationList.user
70 }
71
72 //Every user has his own calendar
73 fact {
74     all u1,u2 :User | u1.calendar = u2.calendar implies u1=u2
75 }
76
77 //I receive notifications only from events in which I participate
78 fact {
79     all u1:User | u1.notifications.event in u1.calendar.events
80 }
81
82 //Every invite is only from one event
83 fact {
84     all i:Invitation| one e:Event | i in e.invitationList
85 }
86
87 //A user must receive only one invite from the same event
88 fact {
89  all e:Event,  i1,i2 :Invitation | (i1 in e.invitationList) and (i2 in e.
       invitationList) and i2.user = i1.user implies i1=i2
90 }
91
92 //a calendar can't contain a event in which the owner of that calendar has not
       been invited
93 fact {
94      all c:Calendar, e:Event, u:User |e in c.events  and c=u.calendar implies u
       in e.invitationList.user
95 }
96
97 //every forecast is relative to an event
98 fact {
99     all f:Forecast | one e:Event | f in e.forecast
100 }
101
102 //every location is relative to an event
103 fact {
104     all l:Location | one e:Event | l in e.place
105 }
106
107 //every datetime is relative to an event
108 fact {
109     all d:DateTime | one e:Event | d = e.start or d=e.end
110 }
111
112 //every notification is relative to an user
113 fact {
114     all n:Notification | some u:User | n in u.notifications
115 }
```

```
116
117 //the owner of an event is automatically a participant of that event
118 fact {
119 all e:Event| e.owner in e.invitationList.user
120 }
121
122 //owner's calendar must contain the created event
123 fact {
124 all e:Event| e in e.owner.calendar.events
125 }
126
127 //invited are in the contact list of the owner
128 fact{
129 all e:Event | all i:Invitation | all u:User | ( u!=e.owner and i in e.
      invitationList and u = i.user ) implies u in e.owner.contactList
130 }
131
132 //check data integrity
133 fact{
134 all e:Event, d1,d2:DateTime | d1=e.start and d2 = e.end implies before[d1,d2]
135 }
136
137 pred show (){
138  #Email >0
139  #Event>0
140 }
141 run show for 5 but 7 int
```

Listing 3: MeteoCal Assert

```
1 //a user can't receive notifications from events in which is not invited?
2
3 assert NoNotificationsIfNotInvited{
4     no u:User|some e:Event| all n:Notification|all i:Invitation |n in u.
      notifications and  e= n.event  and i in e.invitationList and u !=i.user
5 }check NoNotificationsIfNotInvited for 10
6
7 //can arrive a notification to a user from an event that is not in his calendar?
8 assert NoExternalNotifications{
9     all u:User |all n:Notification | n in u.notifications and n.event not in u.
      calendar.events
10 }check NoExternalNotifications for 10
11
12 //exists an event that is not in any calendar?
13 assert NoEventsWithoutCalendar{
14     no e:Event | all c:Calendar | e not in c.events
15 }check NoEventsWithoutCalendar for 10
```

And here are the results of Alloy Analyzer:

4 commands were executed. The results are:
#1: No counterexample found. NoNotificationsIfNotInvited may be valid.
#2: No counterexample found. NoExternalNotifications may be valid.
#3: No counterexample found. NoEventsWithoutCalendar may be valid.
#4: **Instance found.** show is consistent.

Figure 4: Alloy results

7 Worlds Generated

In this paragraph we report some worlds generated by Alloy Analyzer in order to make understand that our model is consistent. We report below four different worlds generated.
Further specific representations are reported below in order to better understand the characteristics of the model.

7.1 The complete world

This is the complete world, showing all possible links. We have two events, three users, some invitations. We note that User2 is the owner of Event0, but he is also involved in Event1, as you can see from the calendar to User2.

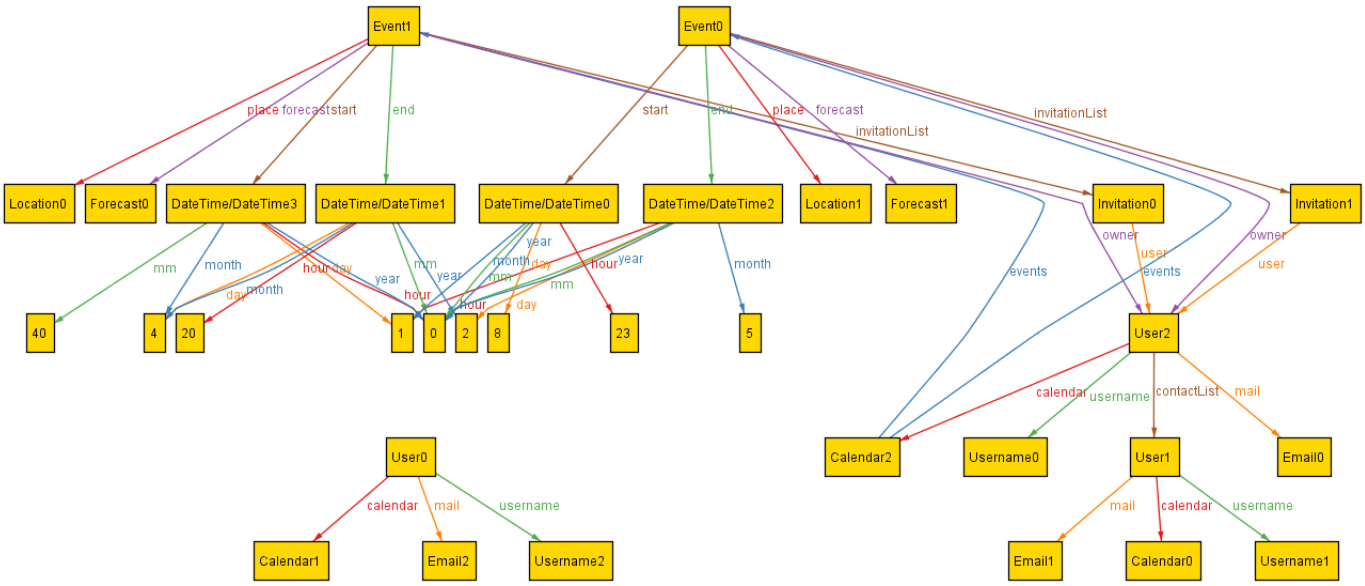

Figure 5.1: The complete world.

7.2 The event and its owner world

In this world we have tried to focus on a single user and a single event to highlight the relations established. Event has as owner the User that should be an invitee and must participate in the Event, as shown by the relationship between Calendar and Event.

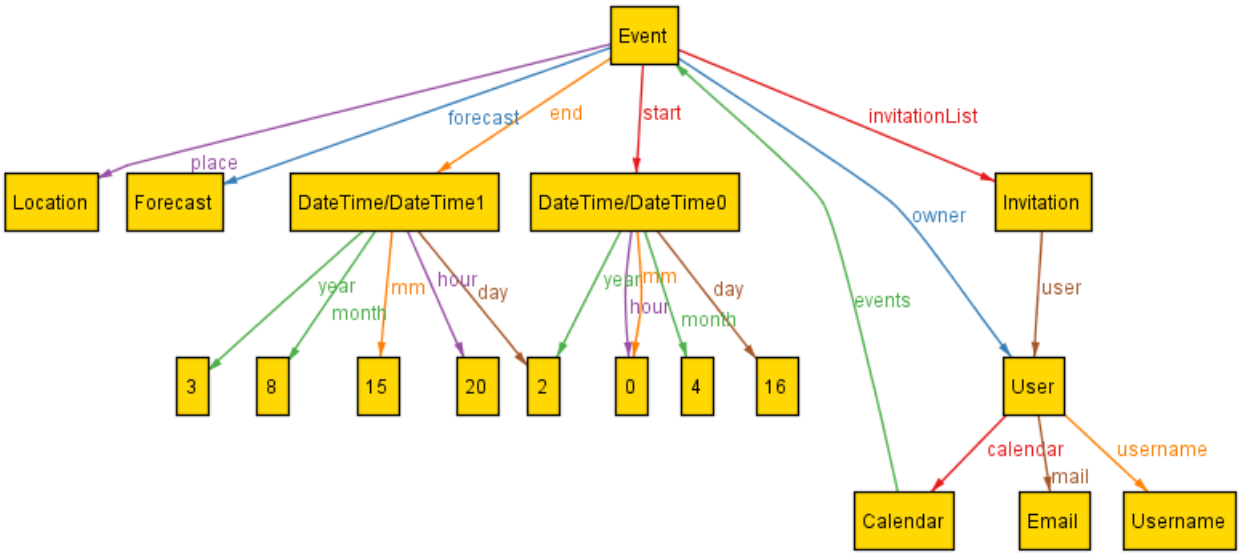

Figure 5.2: The event and its owner world.

7.3 The Invitation world

This site is derived from the previous one with the difference that there are more users. Here we have tried to show, in addition to the previous link, that a user may receive an invitation to the event, but it is possible that the event is not included in its schedule. This is shown between Event and User0, which receives an invitation from Event, but the event does not appear in his own Calendar

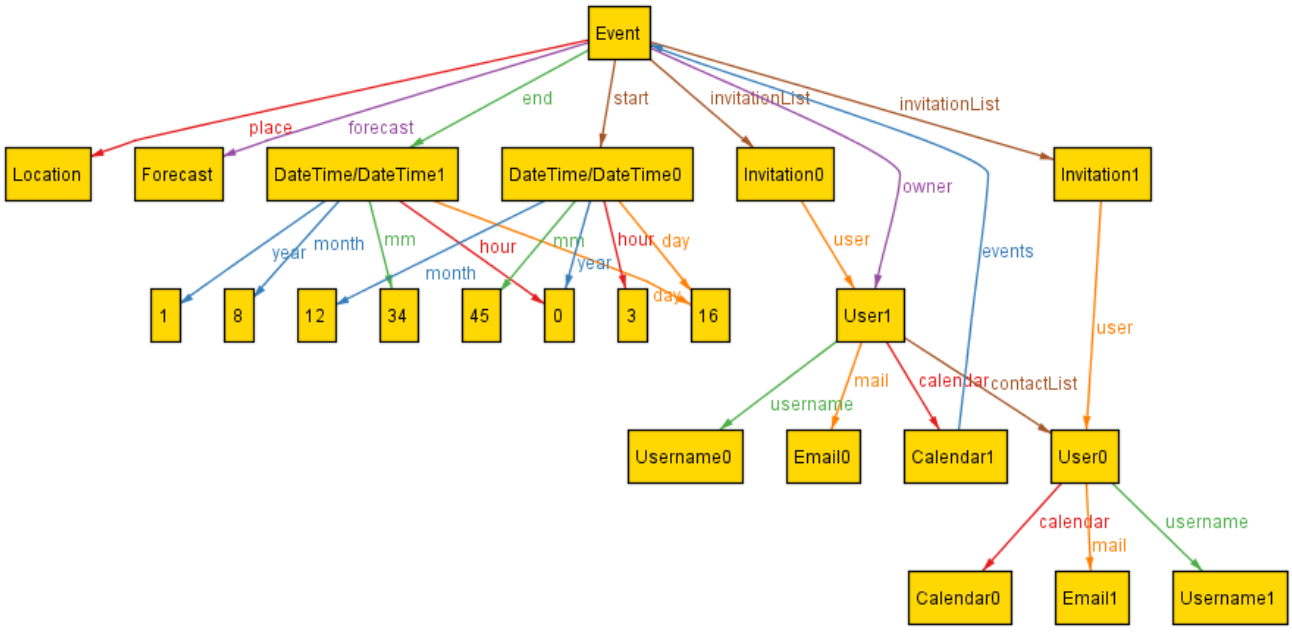

Figure 5.3: The Invitation world.

7.4 The Users world

In this world we have tried to show the relationships between users. Each user has a unique User Name, Calendar and Email.The relationship between users is called ContactList and each user cannot have self-join like this link. User4 is linked with User1, User2 and User3, User3 is connected only with User1, and finally User0 and User1 not have any contacts yet.

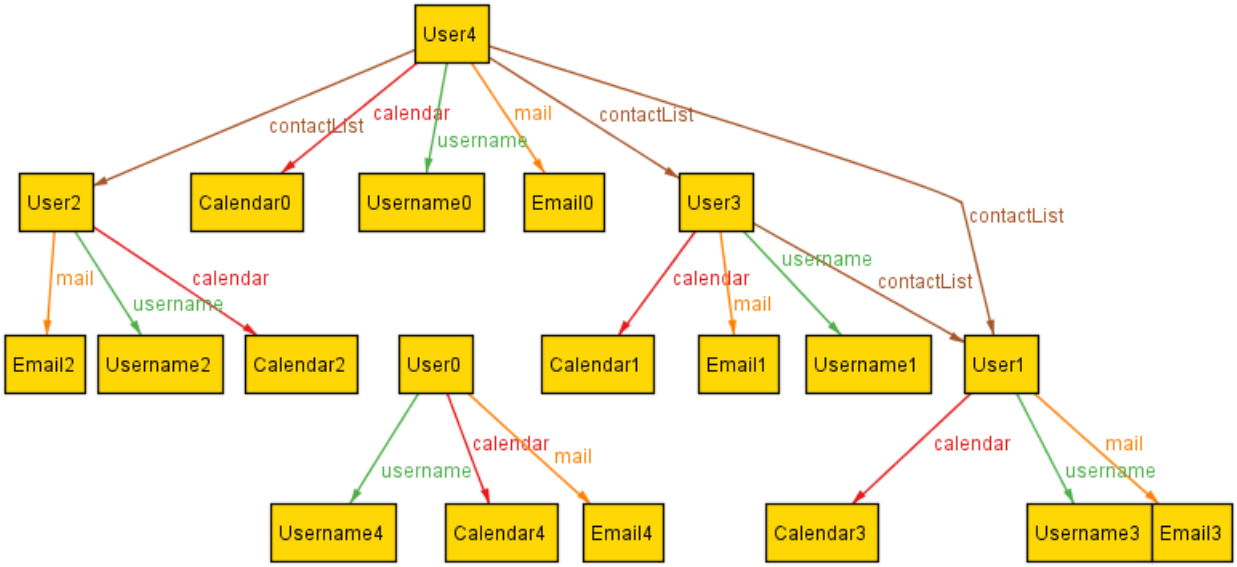

Figure 5.4: The Users world.

8 Division of the roles

	Rossotti	Pasina	Rubiu	Total hours
Problem analysis and roles division	2	2	2	6
Product general description and non functional requirements	8	6	5	19
Functional requirements: • Scenario • Use cases • Sequence diagrams	5	7	6	18
Formal specifications of the system: • Class diagram • Alloy	8	8	10	26
Total hours:	23	23	23	69

9 Used Tools

The tools we used to create this RASD document are:

- https://www.writelatex.com/ to redact and format this document with LaTex language
- Astah Professional to make the use case diagram and the class diagram
- Alloy Analyzer 4.2: to prove the consistency of our model