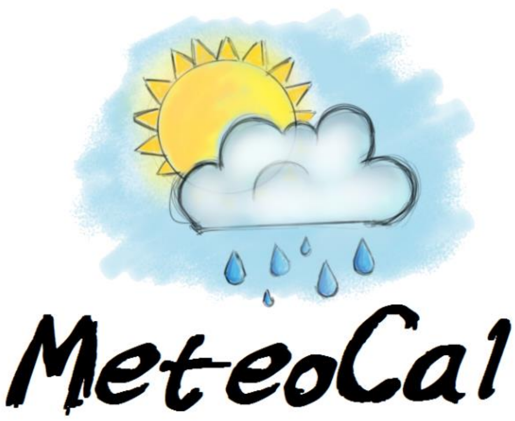


January 25th, 2014

Politecnico Di Milano

Computer Science

***Authors*:**

- ***Alessandro Cianferotti***
- ***Alessandro Cimbelli***
- ***Alessandro De Angelis***

SUMMARY

Requirements analysis and specification document (RASD)

1 Description of the given problem

The given problem consists in the implementation of a new weather based online calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor events.

1.1 Goals

The system will provide the following features:

- Registration
- Authentication
- Create, delete and update events
- Invite user
- Accept or decline invitation
- Enrich the event with weather forecast information (if available)
- Notification system
- Import and export the calendar
- Update weather information periodically (12h)
- Share own calendar
- View public calendars
- Advice for rescheduling events
- Select desired weather in event creation/update
- Forecast info for each event in the calendar

- Warning notification for bad weather (3 days before and every 12h for the creator and 1 day before for invited users)

1.2 Glossary

In this section we define the common words used in our documents.

- ***Guest:*** a person that hasn't already signed up.
- ***User:*** a person registered in the system.
- ***Notification system:*** software that provides a means of delivering a message to a set of recipients.
- ***Main weather conditions:*** the forecast system we will use offers different weather information. "Main weather conditions" means the main condition that occurs in the day.
- ***Bad weather:*** the meaning of these words can vary from user to user. The system will provide to the creator the possibility to specify what "bad weather" means to him.
- ***Event details:*** the event details are name of the event, date, hour, location, participants list and selected weather conditions.

1.3 Domain Properties

We suppose the conditions of the analysed world:

- We assume that the forecast given by the service used is exact.
- Users have the same time zone

1.4 Assumptions

In this section we want to specify a few points that are not very clear.

- For each event we set as weather condition the main condition that the forecast server gives us. We have to consider the impossibility to find a day, or more, that satisfies the constraint of “good weather”. In this case the system will ask user to change the parameters of bad weather.
- Accepting an invitation means that the user accept also the choice of the creator of bad weather.
- The system will notify the forecast change only if it cross the line of bad weather.
- If a user creates an event for the current day he will not receive the 3 days before notification. Depending on the start hour he could receive the 12h notify. Anyhow he will get the notification from the warning section.
- If a user creates an event that will be taken within 3 days he/she will not receive the 3-days notifications “bad weather”

- A user can accept or decline an invitation of passed events (the system will not take action)
- An invited user will receive event updates also if he has declined , accepted or he hasn’t taken a decision yet .
- The user cannot accept event invitation if he has already created or accepted a previous event that occurs in the same time.
- When creating an event, the user must insert all the data except for the invitation and the preference.
- The User can import only his own calendar
- The reference date for control has the server timezone
- The import file is assumed to be correct, so there are no empty fields except for preference and invitation.
- If the user doesn’t specify a (or more) weather preference the system set, as desired, all of them.
- An invited user will receive a 1 day before bad weather notification only if he/she has accepted the invitation.
- The system requests weather forecast for the next 10 days starting from the current date only for the registered city.

1.5 Other considerations

Nowadays there is a wide range of solutions, so, in order to implement a modern system that can compete with others, we have decided to focus our attention also to the graphic UI.

2 Actors Identifying

The actors of our informative system are:

- ***User:*** has access to all the functionalities of the system
- ***Guest:*** can only have access to the main page
- ***Forecast server:*** *provides weather forecast for outdoor event*

3 Requirements

3.1 Functional requirements

The following lines describe the system requirements according to the goals that have to be achieved during the realization of the project:

3.1.1 Registration

Users have to sign up to the website through the insertion of personal data to authenticate

.

3.1.2 Authentication

Users have to authenticate to access the functionalities of the system

3.1.3 Creation of an event

Users can create an event specifying where the event will take place (city name) and whether the event will be indoor or outdoor. Furthermore, the event must be classified as public or private event. In case of outdoor event the user may select which forecast conditions he/she desired.

3.1.4 Delete of an event

Users can delete own created events.

3.1.5 Update of an event

Users can update own events like date, place etc.

3.1.6 Send of invitation

User can invite any number of users.

3.1.7 Accepting or declining of an invitation

Users can accept or decline the invitation. If a user decline before accepting, the invitation will be deleted, while if he/she has accepted, the user can decline and eventually accept in the future.

3.1.8 Association between event and weather information

The system associates for each event the corresponding weather information (if available) based on the location given by the user.

3.1.9 Notification system

The system must be able to supply notifications to users through two main ways:

1) Showing them on the user profile in a dedicated part.
2) Sending email.

3.1.10 Make the calendar public

Users can set their calendar public.

3.1.11 View of all public calendars

Users can see details of public events from other users' calendars that have been set public.

3.1.12 Notification to the event participants

The system must be able to notify all the event participants (invited users that have accepted the invitation) one day before the event in case of bad weather conditions for outdoor events.

3.1.13 Notification to the creator

The system must be able to notify the creator of an outdoor event three days before the event and every 12h in case of bad weather conditions.

3.1.14 Proposal of a new schedule

After a notification in case of bad weather conditions, the system has to propose to the creator the closest (in time) schedule with admissible weather conditions (if any). The system searches only for the next 10 days from the current date.

3.1.15 Import and Export of calendars

The system enable importation and exportation of calendars. In case of import the system has to prevent from overlapping problem. While importing, in case of overlapping with an invitation the related event will not import. The import file must be in the document folder for both Windows and Mac OS.

3.1.16 Update weather information

The system must update the weather information every 12 hours and in case notify the creator the forecast changes.

3.1.17 Selection of weather preferences

The system must provide to users the possibility to specify, during creation or updating, the weather preferences.

3.2 Non Functional Requirements

3.2.1 User Interface

Although no specifications were provided about the GUI , we tried to identify an interface as intuitive and attractive as possible .
The use of shortcuts and icons gives users immediate access to the required functionality.
The next images will provide a basic idea of the principal sections:

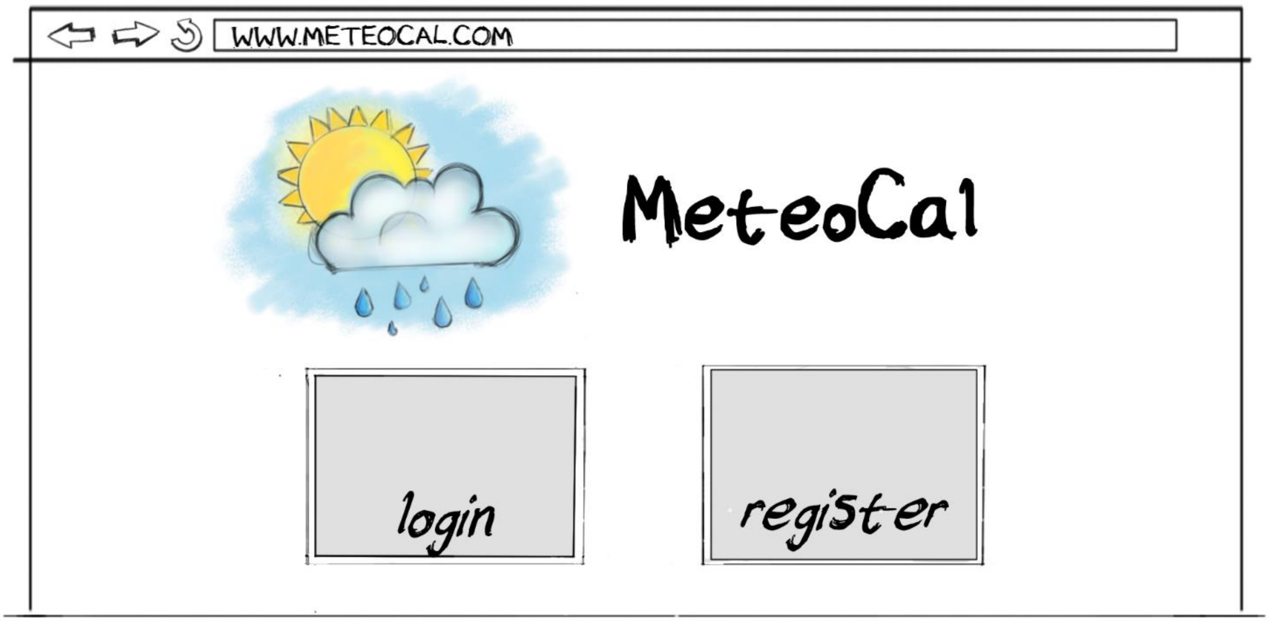
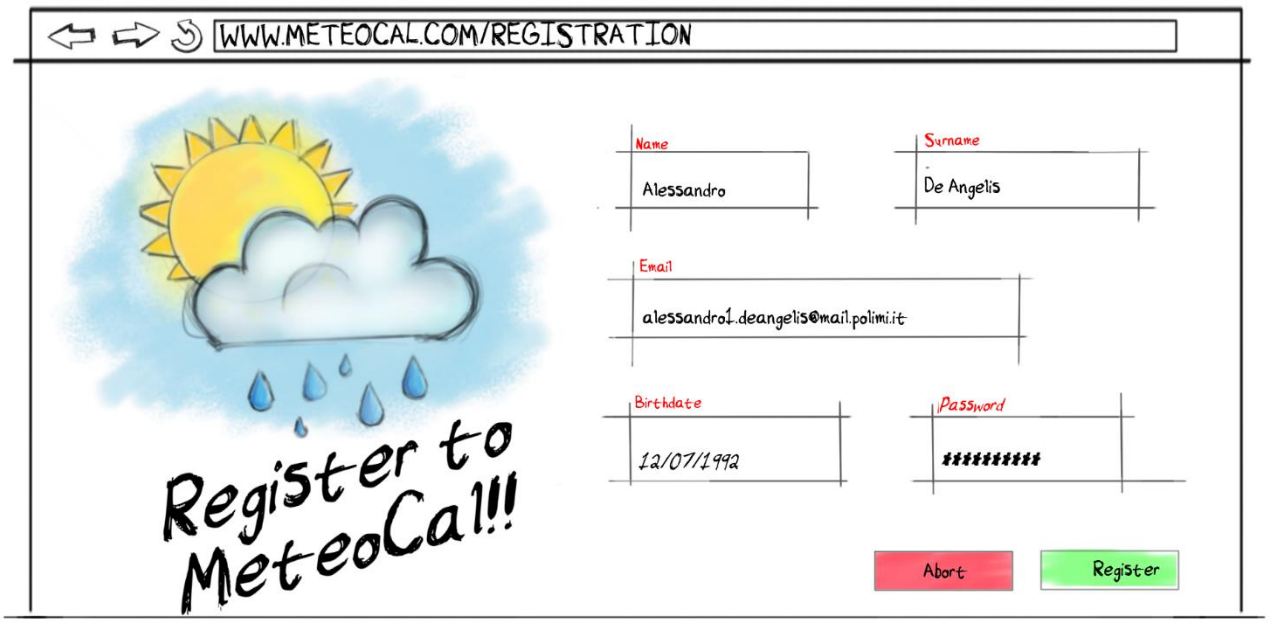
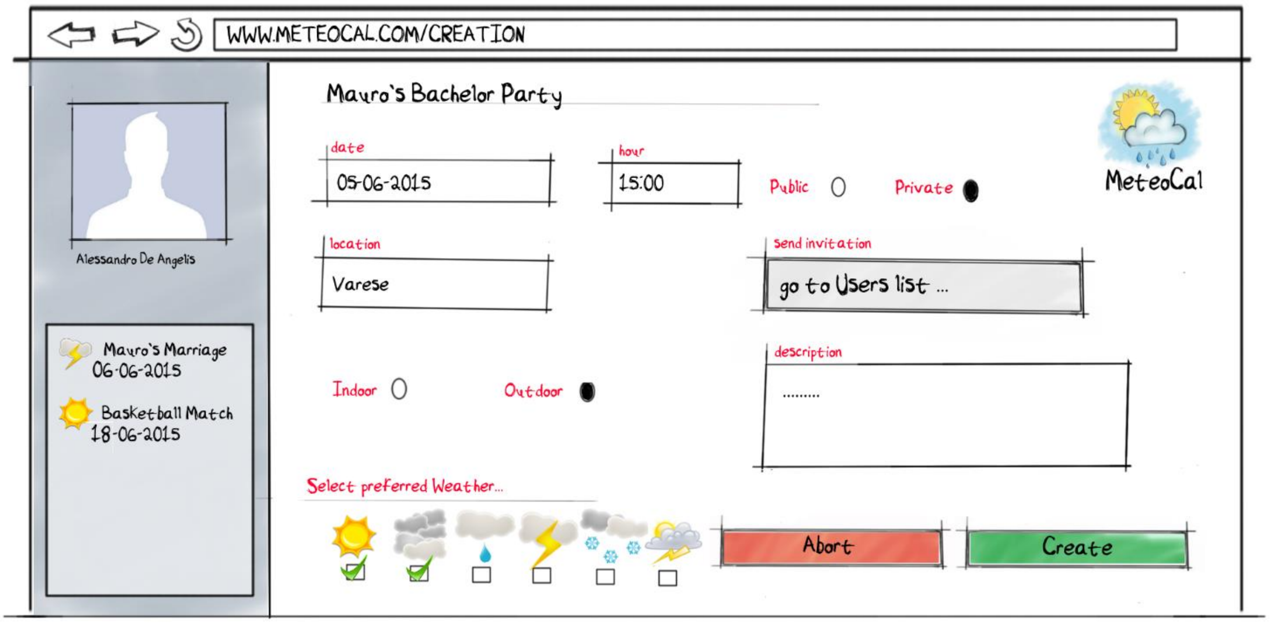


Figure 1 – Homepage



Name	Alessandro
Surname	De Angelis
Email	alessandro1.deangelis@mail.polimi.it
Birthdate	12/07/1992
Password	**********

Figure 2 – Registration



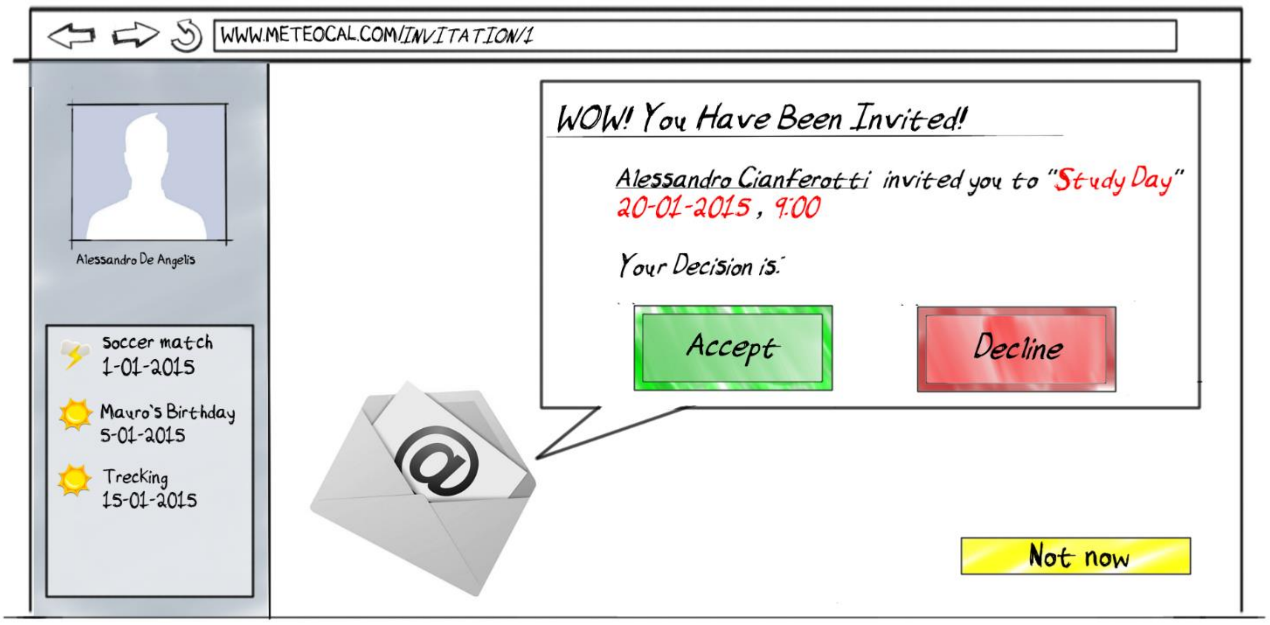
Mauro's Bachelor Party

date	05-06-2015
hour	15:00
location	Varese
send invitation	go to Users list ...
description	....

Public ○ Private ●

Indoor ○ Outdoor ●

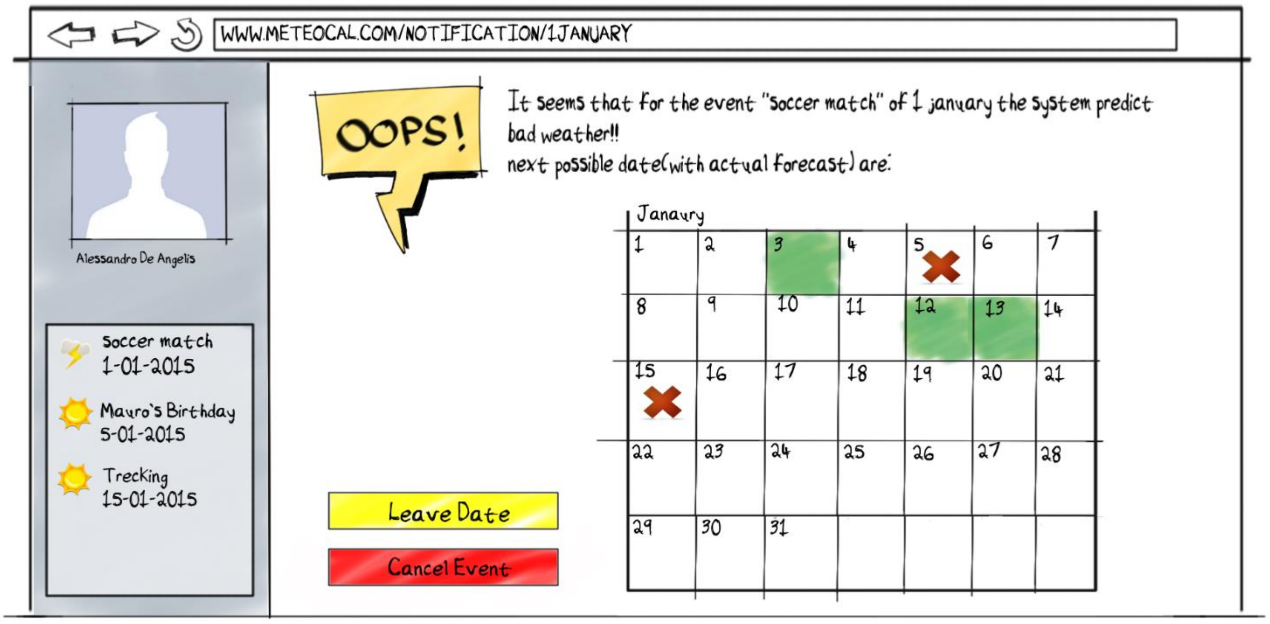
Figure 3 – Event Creation


WWW.METEOCAL.COM/INVITATION/1
Alessandro De Angelis
Soccer match 1-01-2015
Mauro's Birthday 5-01-2015
Trecking 15-01-2015
WOW! You Have Been Invited!
Accept
Decline
Not now

Alessandro Cianferotti invited you to "Study Day" 20-01-2015, 9:00

Your Decision is:

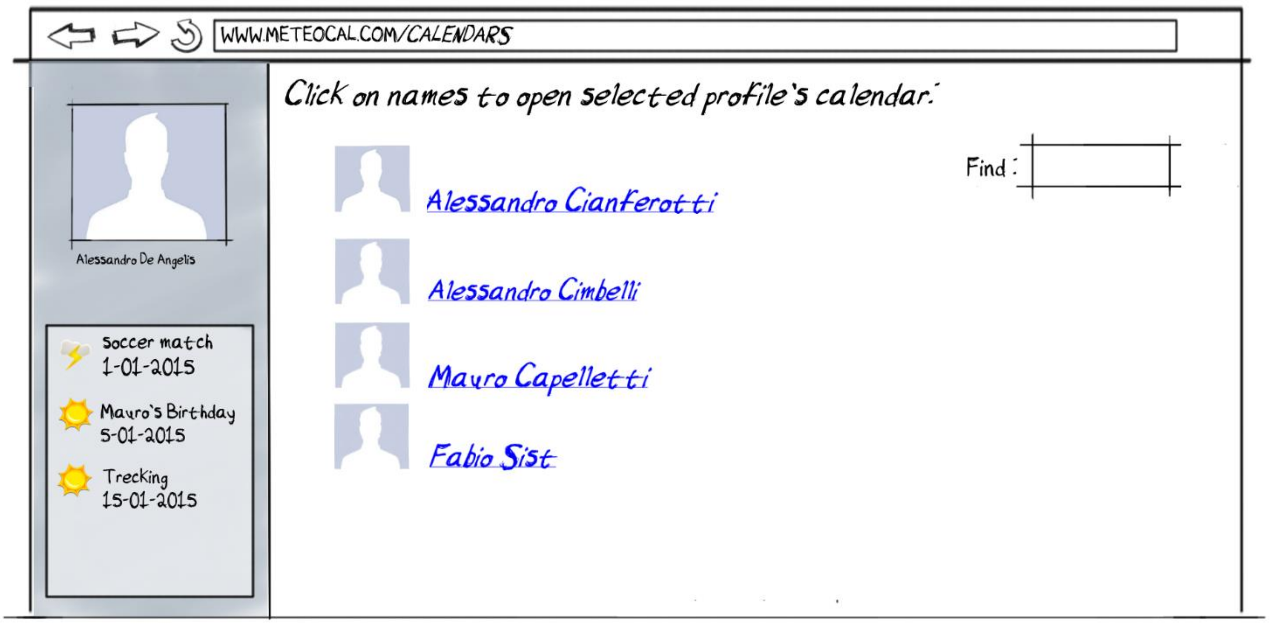
Figure 4 - Invitation



It seems that for the event "soccer match" of 1 january the system predict bad weather!!
next possible date(with actual forecast) are:

1	2	3 🟩	4	5 ✗	6	7
8	9	10	11	12 🟩	13 🟩	14
15 ✗	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figure 5 - Weather Change Notification



WWW.METEOCAL.COM/CALENDARS

Click on names to open selected profile's calendar:

Find :

Alessandro Cianferotti

Alessandro Cimbelli

Mauro Capelletti

Fabio Sist

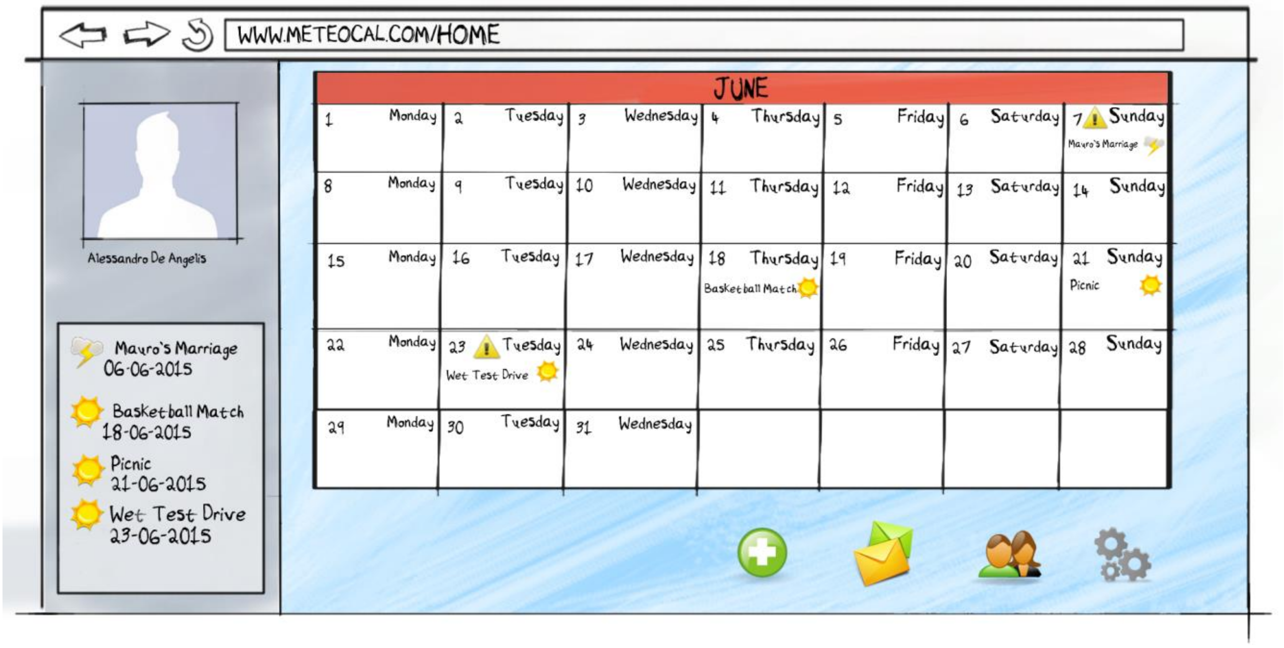
Alessandro De Angelis

Soccer match
1-01-2015

Mauro's Birthday
5-01-2015

Trecking
15-01-2015

Figure 6 – Users List



WWW.METEOCAL.COM/HOME

Alessandro De Angelis

Mauro's Marriage
06-06-2015

Basketball Match
18-06-2015

Picnic
21-06-2015

Wet Test Drive
23-06-2015

JUNE

1 Monday	2 Tuesday	3 Wednesday	4 Thursday	5 Friday	6 Saturday	7 Sunday Mauro's Marriage
8 Monday	9 Tuesday	10 Wednesday	11 Thursday	12 Friday	13 Saturday	14 Sunday
15 Monday	16 Tuesday	17 Wednesday	18 Thursday Basketball Match	19 Friday	20 Saturday	21 Sunday Picnic
22 Monday	23 Tuesday Wet Test Drive	24 Wednesday	25 Thursday	26 Friday	27 Saturday	28 Sunday
29 Monday	30 Tuesday	31 Wednesday				

Figure 7 - Homepage

3.2.2 Documentation

We will provide the following documents:

- ***RASD:*** Requirement Analysis and Specification Document, to well- understand the given problem and to analyse 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***
- ***User Manual***
- ***Testing Document*** :a report of our testing experience

3.2.3 Architectural Consideration

The project will be developed using the Java EE platform. In particular, we will use EJBs to develop the business logic. Data will be stored in a centralized database on the Server.

An Internet Connection and an up-to-date browser is needed to use the application.

3.2.4 Hardware Specifications

The centralized architecture of the system includes a server that can handle all requests without difficulty.
We decided to develop a thin client to not require high computing power users .

4 Software Characteristics

4.1 Reliability

To ensure reliability to users avoid losing access speed, we propose a storage system that implements a RAID 1 + 0

4.2 Security

To ensure the functionality of the system the user must log in with username and password provided at registration. User data will be protected from hacking attempts (sql injections) and the password will be encrypted by hashing code.

4.3 Availability

We expect the system is available 24/07 except for maintenance

4.4 Portability

We decided to develop a light application that can be run on every devices, taking advantage of java compatibility.

This option also avoid continuous download by the user at every update.

5 Scenarios

Alessandro needs to find a system for managing his calendar. He is a sportsman and he often organize outdoor activities, but he is getting tired of checking forecast every time to be sure to carry out his planned activities. He is looking for a web calendar that integrates a forecast system able to notify weather changes.

During a five-a-side football match, speaking with two teammates, he proposes to develop a web application satisfying his need.

Therefore, they decided to project and implement "MeteoCal", a system able to fully manage events including an intuitive graphic user interface.

5.1 "Event Creation"

Alessandro wants to test the system organizing a football match with his mates. After registration and login, he selects the creation button; consequently, the system opens a related interface in which Alessandro can insert basic information (date, hour, place etc.). Moreover, Alessandro can both specify the desire weather condition and invite other users.

5.2 "Accept/Decline Invitation"

Marco receives a notification of the invitation. He logins and selects the notification. The system opens a new page that contains all the information about the event and two buttons, one to accept and one to decline.

5.3 "Event Update after Notification"

Alessandro receives the "bad weather" notification. The system sends him to a new page suggesting the closest, in time, days that offer the right weather condition. Alessandro analyses them and can decide if change the date (suggested or not) avoiding overlapping, delete the event or take it anyway. Eventual changes will be notified to participants.

5.4 “Event Update”

After creating the event, Alessandro realizes he has a dental appointment in the same date and hour of the match, so he decides to postpone the match. He logins and selects the day of the match in the calendar getting a list of daily event. On the right, he finds the update button, one for each event. After clicking, the system opens a new interface in which Alessandro can update all the information. The systems notifies changes to participants

5.5 “Event Cancellation”

Some of the invited players tells Alessandro that they cannot participate to the match, so he decides to cancel it. He logins and selects the day of the match in the calendar getting a list of daily event. On the right, he finds the delete button, one for each event. The systems notifies invited users.

5.6 “See Others Calendar”

Marco tells Alessandro that Mauro has organized a match without inviting him. Driven by curiosity, Alessandro selects Mauro from the users list for watching his calendar. He discovers that Marco is right. He tries to see the details, but Mauro has set the event as private, so Alessandro can only see the title of the event.

5.7 “Export Calendar”

Fabio wants to have a local copy of his calendar in his laptop . From the home page, he selects the export calendar link getting a file that contains all the information about each event.

6 Use Case Description

6.1 Registration

Description	The system will provide to guests the possibility to register
Actor	Guest
Pre-condition	The guest goes to the registration page
Event flow	1. The user inserts his personal data, username and password 2. The user confirms the option
Post-condition	The system confirms the registration
Exception	1. User already registered. The system reports the error and cancels the registration 2. Data entered is not valid. The system requires the user to insert/correct the data

6.2 Authentication

Description	The system will provide to user the possibility to login
Actor	User
Pre-condition	The user goes to the registration page
Event flow	1. The user inserts username and password 2. The user confirms the data
Post-condition	The system confirms the login
Exception	1. Data entered is not valid. The system requires the user to insert/correct the data

6.3 Create Event

Description	The system will provide to user the possibility to create event
Actor	User
Pre-condition	The user logins
Event flow	1. The user selects the desired date or a random date 2. The user inserts date, hour and place 3. The user inserts a title and a description 4. The user specifies if the event is indoor or outdoor 5. If outdoor, he/she may select the "bad weather" condition 6. The user specifies if is a public or private event 7. If the user wants to, he invites some users
Post-condition	The system adds a new event in the user's calendar and eventually sends invitation
Exception	1. The user chooses a date and an hour that overlap with another event. The system asks for a change. 2. The user doesn't fill all fields

6.4 Update Event

Description	The system will provide to user the possibility to update events
Actor	User
Pre-condition	The user logins
Event flow	1. The user selects the event that he wants to modify 2. The user inserts new information filling the relative text box 3. The user confirms the update
Post-condition	The system update the event and the calendar. Moreover, the system notifies to invited user the changes.
Exception	1. The user sets a date and an hour that create a conflict. The system requests to the user a different choice. 2. The user leaves some blank fields.

6.5 Delete Event

Description	The system will provide to user the possibility to delete events
Actor	User
Pre-condition	The user logins
Event flow	1. The user selects the date of the event 2. The user selects the event that he wants to delete 3. The user clicks on delete button
Post-condition	The system delete the event and notifies to invited users.
Exception	

6.6 Set Calendar

Description	The system will provide to user the possibility to set the calendar public or private
Actor	User
Pre-condition	The user logins
Event flow	1. The user selects the “settings” button 2. The user selects the relative radio button 3. The user confirm the choice
Post-condition	The system sets the calendar public/private
Exception	

6.7 Import Calendar

Description	The system will provide to user the possibility to import a calendar
Actor	User
Pre-condition	The user logins
Event flow	1. The users selects the settings button 2. The users selects the import button
Post-condition	The system loads the new calendar
Exception	1. The file is damaged, incorrect or doesn’t exist in the document folder.

6.8 Export Calendar

Description	The system will provide to user the possibility to export his own calendar
Actor	User
Pre-condition	The user logins
Event flow	1. The users selects the settings button 2. The users selects the export button
Post-condition	The users download the calendar file
Exception	

6.9 View of public calendar

Description	The system will provide to user the possibility to see the public calendar
Actor	User
Pre-condition	The user logins
Event flow	1. The users selects the "user" button 2. The user selects a name in the list
Post-condition	The system shows the selected calendar
Exception	1. There are no public calendar

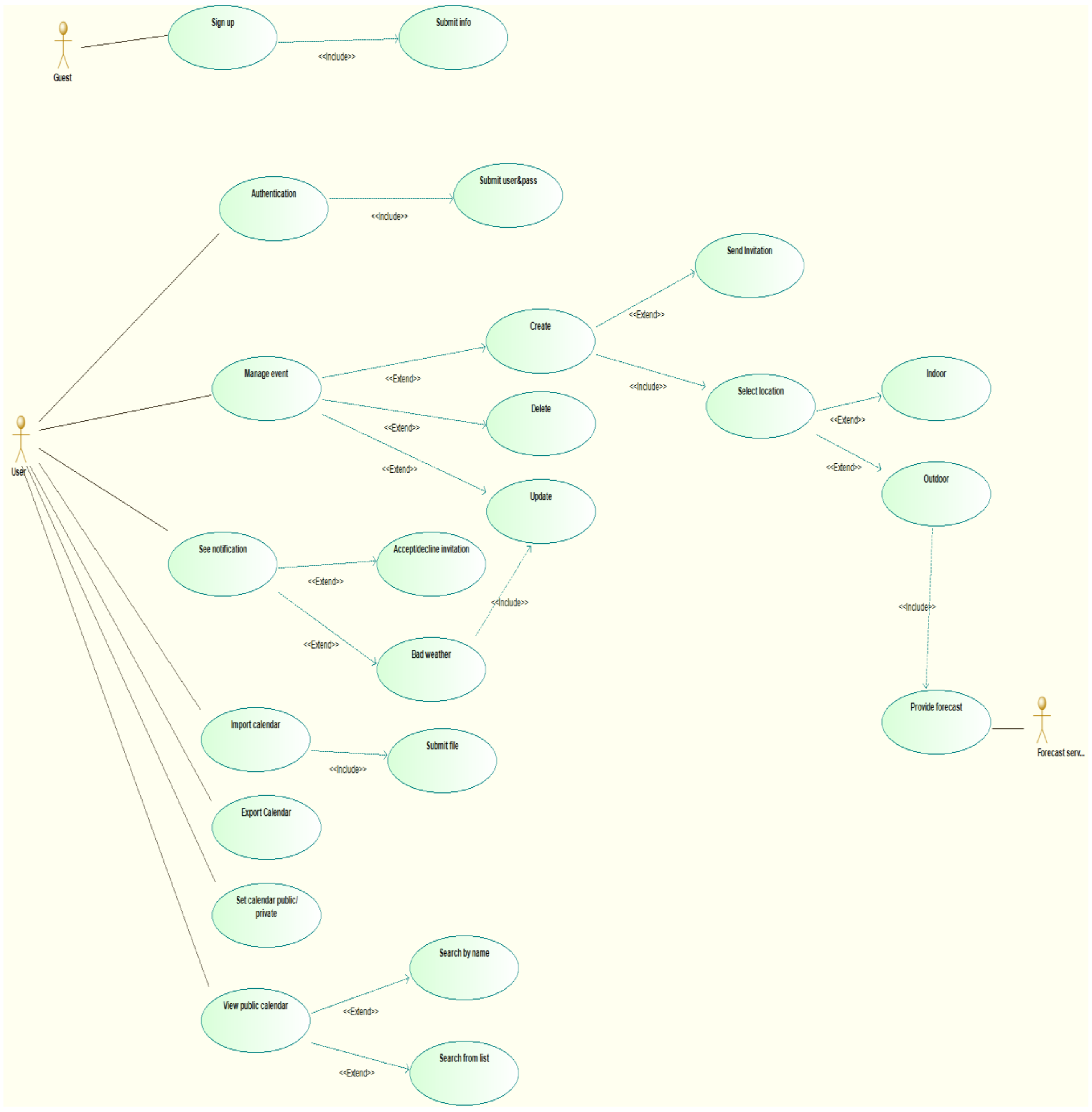
6.10 Accept/decline invitation

Description	The system will provide to user the possibility to accept/decline invitation
Actor	User
Pre-condition	The user logins
Event flow	1. The user goes to invitation section 2. The user selects the invitation 3. The user clicks on accept or decline button 4. After accepting the user can decline selecting the event from the calendar
Post-condition	The system notifies to the creator what the user has decided
Exception	1. Overlapping with existing events

7 UML Models

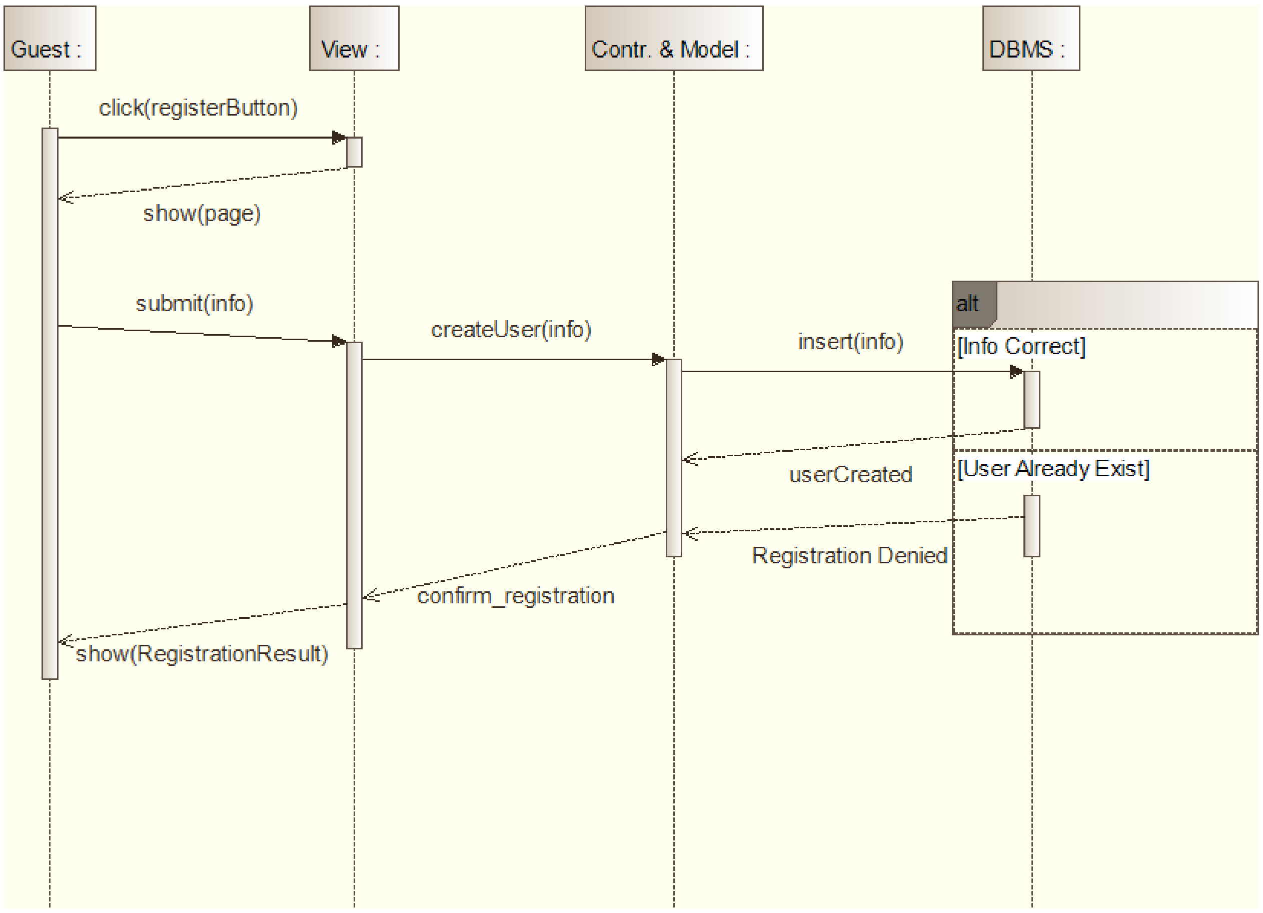
7.1 Use Case Diagram

We present here a general use case diagram, that will be analysed in the specific with BPMN and sequence diagram:

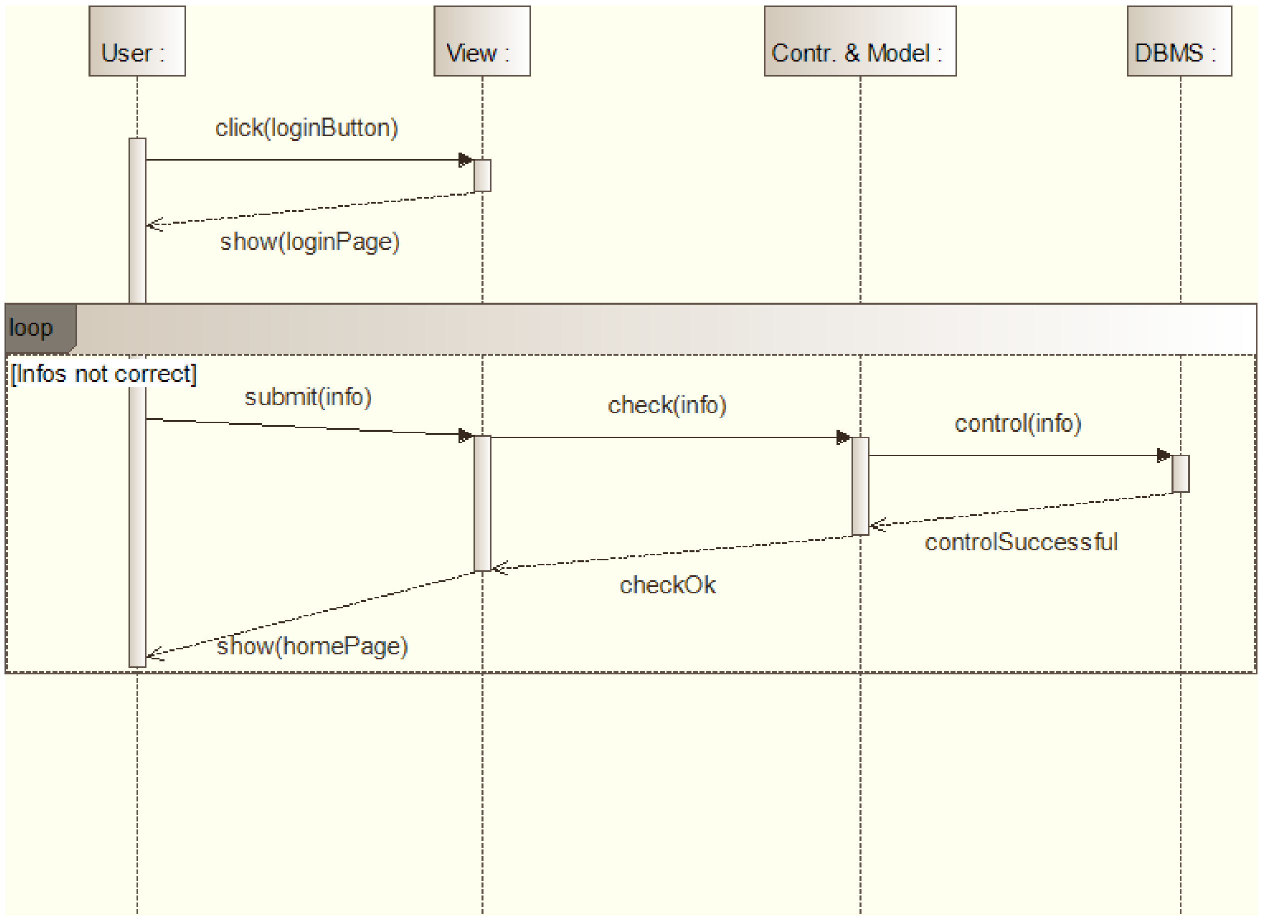

Guest
Sign up
<<Include>>
Submit info
Authentication
<<Include>>
Submit user&pass
Send Invitation
<<Extend>>
Create
<<Include>>
Select location
Indoor
<<Extend>>
<<Extend>>
Outdoor
Manage event
<<Extend>>
Delete
<<Extend>>
<<Extend>>
Update
User
See notification
<<Extend>>
Accept/decline invitation
<<Include>>
<<Extend>>
Bad weather
<<Include>>
Provide forecast
Forecast serv...
Import calendar
<<Include>>
Submit file
Export Calendar
Set calendar public/ private
Search by name
View public calendar
<<Extend>>
<<Extend>>
Search from list

7.2 Sequence Diagrams

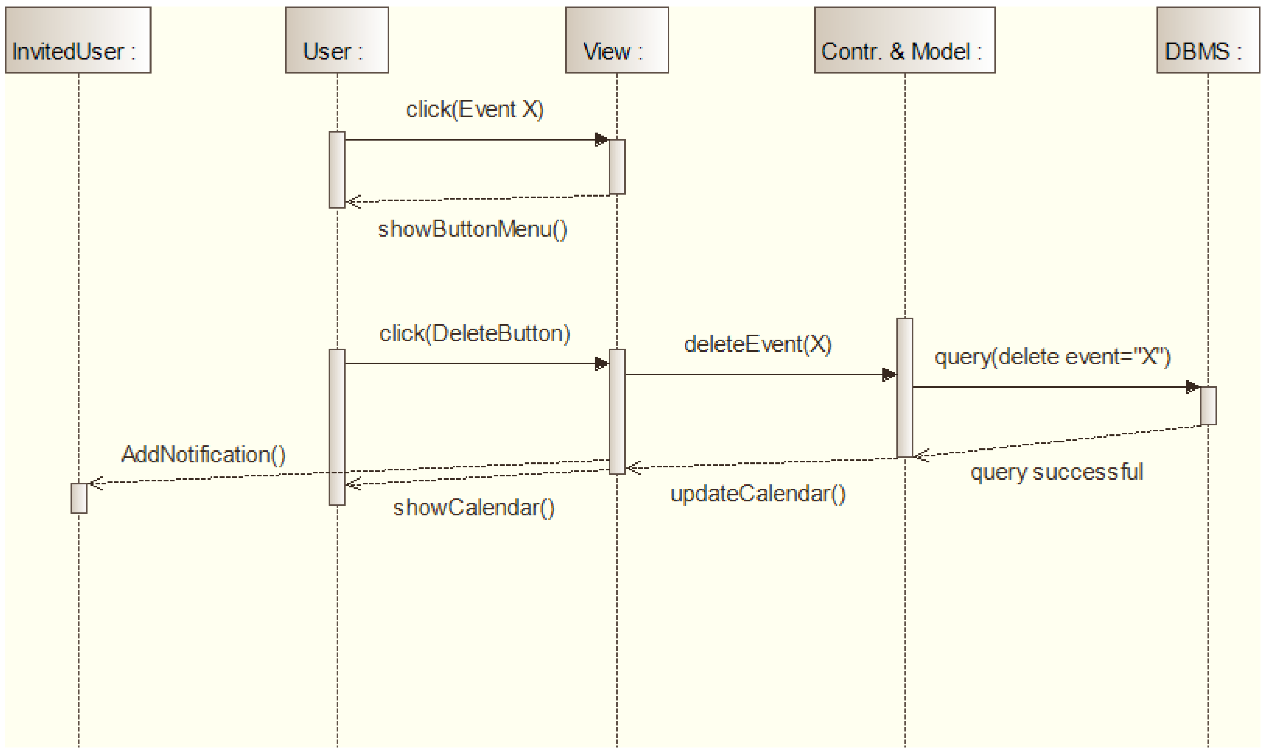
7.2.1 Registration



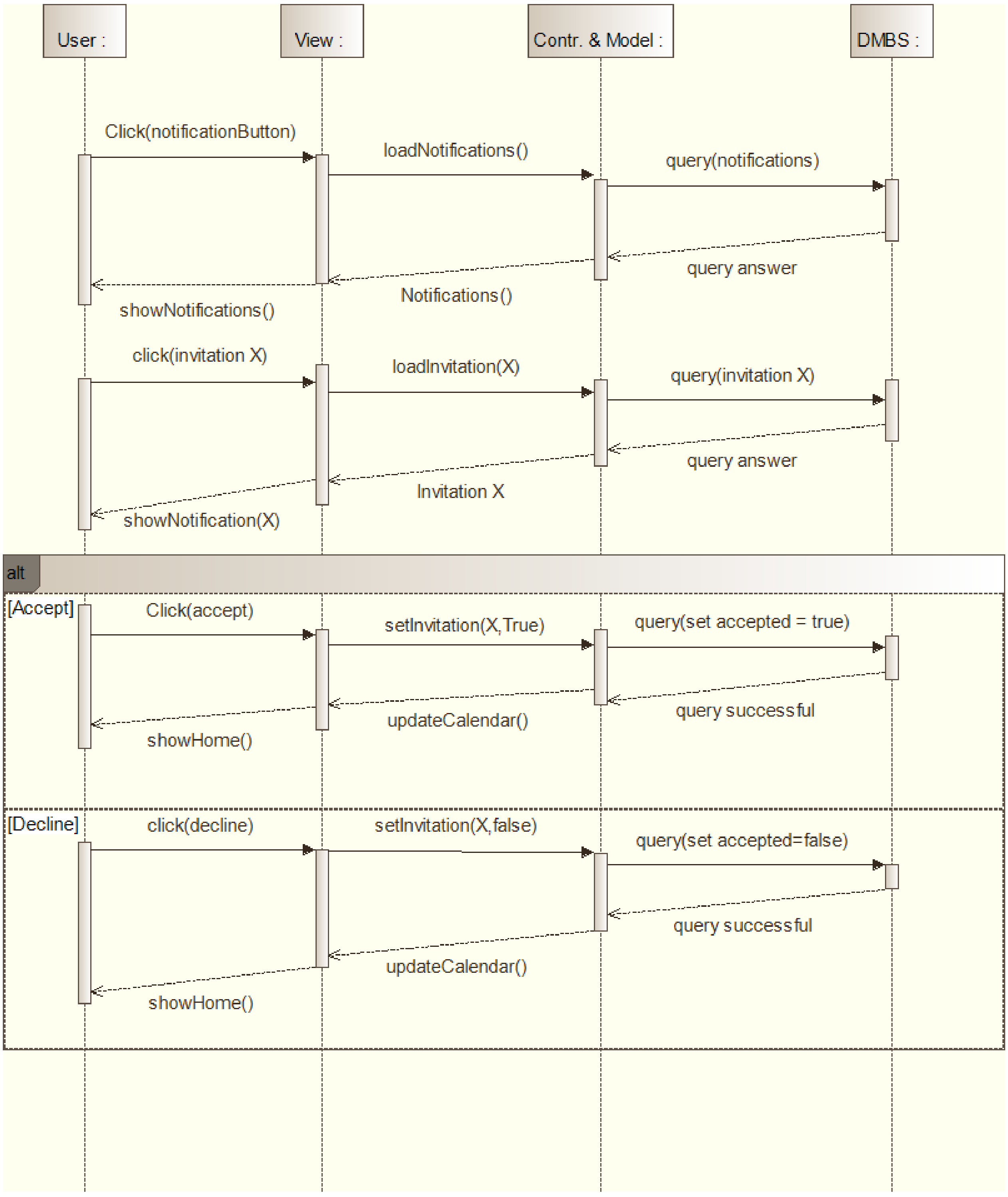
7.2.2 Authentication



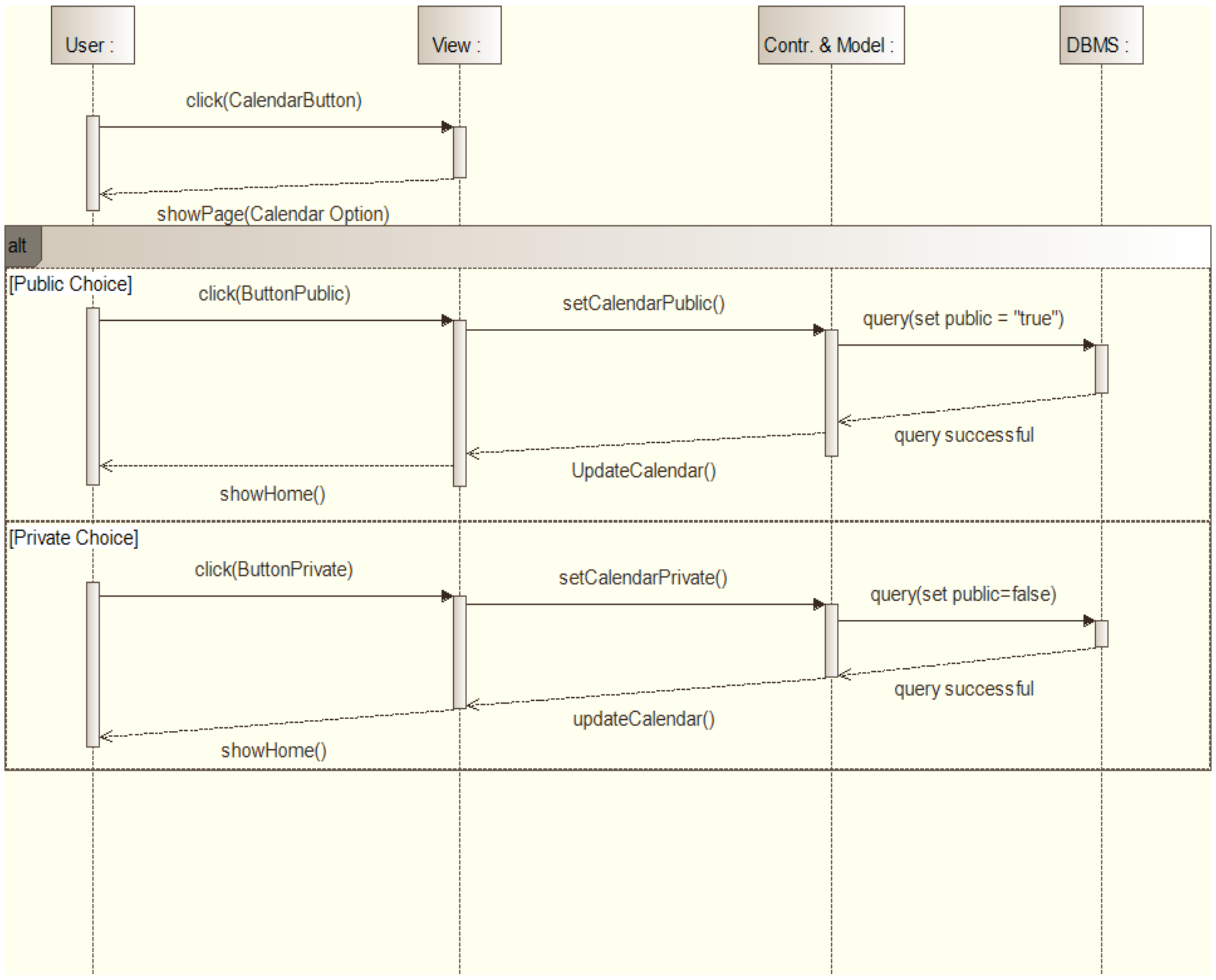
7.2.3 Delete Event



7.2.4 Invitation

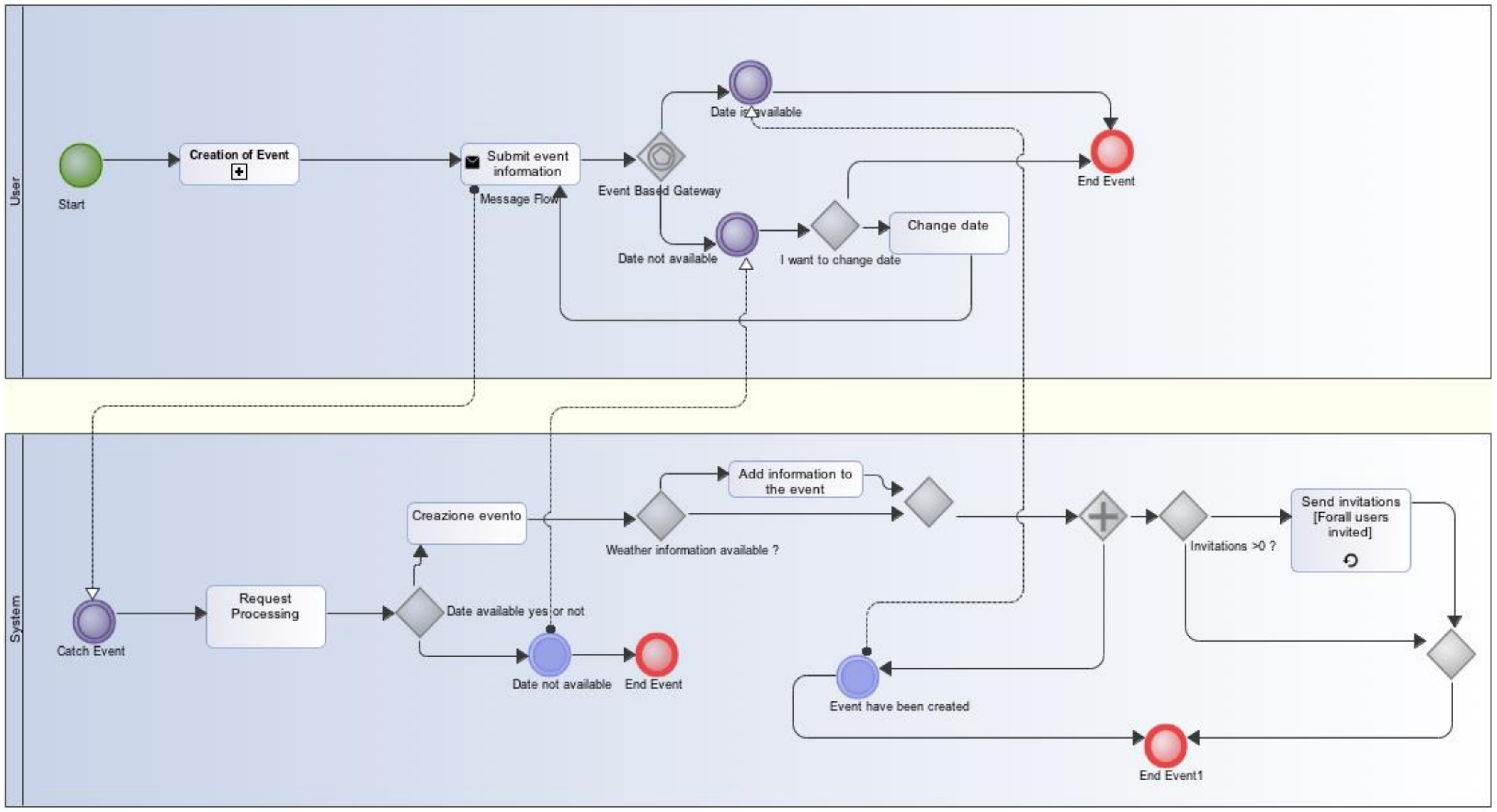


7.2.5 Set Calendar

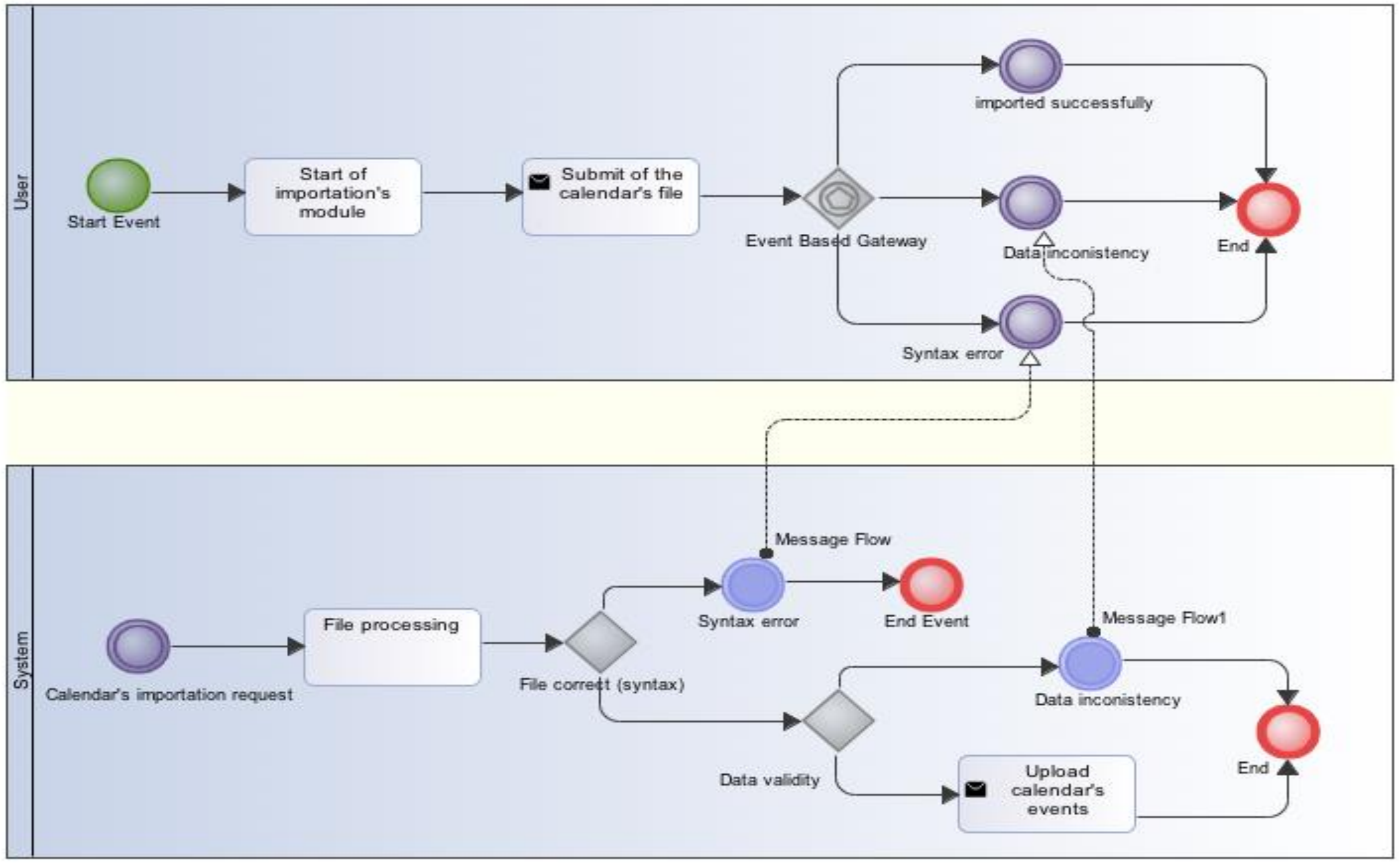


7.3 BPMN Diagrams

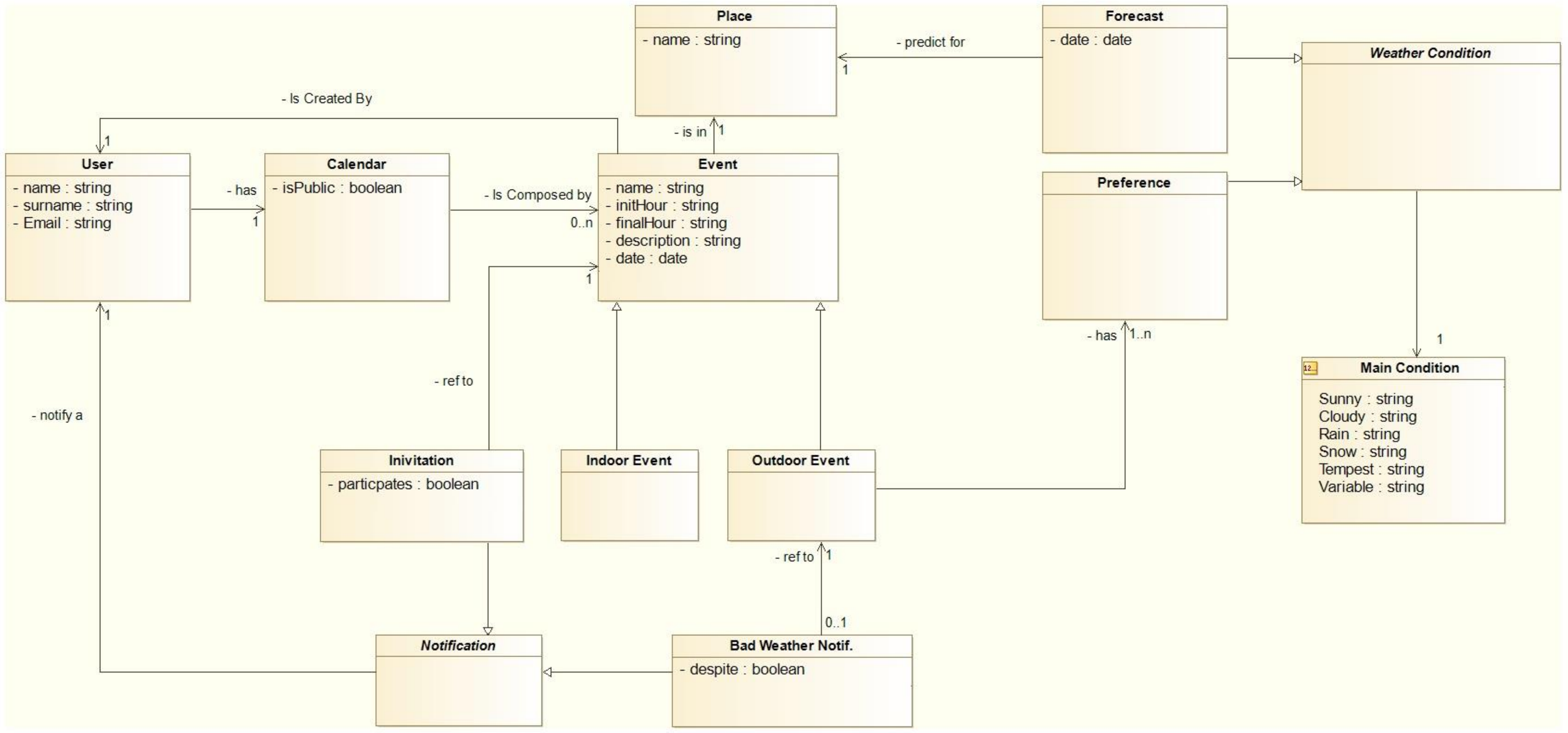
7.3.1 Create Event



7.3.2 Calendar Importation



7.4 Class Diagram



8 Alloy

In the following section we want to show the results we obtained testing our Class Diagram using Alloy. We want to demonstrate if our Class Diagram is consistent.

8.1 Code

We report below the code:

```
//SIGNATURES

sig Username{}

sig User{

        name: one Username,

         cal: one Calendar

}

sig Calendar{

        events: set Event

}

abstract sig Event{

        date: one Date,

        startingHour: one Hour,

        endingHour: one Hour,

        place: one Place,

        creator: one User,

        invitations: set User

}
```

```
sig OutdoorEvent extends Event{
        preference: some Preference
}
//Implemented for futher update
sig IndoorEvent extends Event{}
sig Preference extends WeatherCondition {
}
abstract sig WeatherCondition{
        mainCondition: one KindOfWeather,
}
sig Forecast extends WeatherCondition{
        place: one Place,
        date: one Date,
}
sig KindOfWeather{}
abstract sig Notification{
        owner: one User
}
sig Invitation extends Notification{
                ref: one Event
}
```

```
sig BadWeatherNotification extends Notification{
        ref: one OutdoorEvent
}
sig Place{}
sig Date{}
sig Hour{
 hour : one Int
}
//FACT
//No calendar without user
fact NoAloneCalendar{
        no c: Calendar | all u: User|
        c!=u.cal
}
//hour strictly positive
fact HourPositive{
all h1:Hour|
        h1.hour >0
}
```

```
//events created are in creator calendar

fact creatorCalendar{

all e1:Event | all user1 : User |

e1.creator = user1 implies e1 in user1.cal.events

}

//A calendar different for all the user

fact oneCalendarOwner{

             all u1:User | all u2:User |

            u1.cal=u2.cal iff u1=u2

}

//User can't invite himself

fact NoUserCanInviteHimSelf{

            no i :Invitation |

            i.owner=i.ref.creator

}

//Creator not invited

fact creatorNotInvited{

            all e:Event | e.creator not in e.invitations

}
```

```
//Informations associated at existing place
fact NoUselessForecastInformation {
        all f: Forecast |one p: Place |
        f.place=p
}
//Consistency of weather information according to the place and the time
fact NoForecastInfoForSamePlaceAndTimeDifferent{
        all f1: Forecast | no f2: Forecast |
        (f1.place=f2.place)and(f1.date=f2.date)and(f1.mainCondition!=f2.mainCondition)

}
//Preference associated at existing outdoor event
fact NoRandomPreference{
        all p : Preference | some eo: OutdoorEvent |
                eo.preference = p
}
//Forecast for date sets in at least an evet
fact ForecastOnlyForDayEvent{
        all f:Forecast | some e:Event|
        e.date=f.date and e.place=f.place
}
```

```
//Different starting hour from ending hour
fact DifferentHour{
        all e:Event|
        e.startingHour.hour < e.endingHour.hour
}
//Same forecast for the same place and date
fact ForecastOne{
        all f1:Forecast | all f2:Forecast |
                (f1.date=f2.date and f1.place=f2.place) implies f1=f2
}
//No User With The Same User Name
fact NoDuplicatedUser{
        all u1:User | all u2:User |
        u1.name=u2.name implies u1=u2
}
//Constraint for the association between BadweatherNotifications and Events
fact ConsistenceWeatherNotification{
        all b : BadWeatherNotification | all e: Event |
                b.ref=e  iff (some mainC : b.ref.preference.mainCondition | some f : Forecast
| f.date=e.date and f.place=e.place and f.mainCondition = mainC)
}
```

```
//Bad weather notification only if there is a forecast for the event associated to the
notification's event

fact BadWeatherNotificantionImpliesForecast{

         all e : OutdoorEvent |

                  (some bad:BadWeatherNotification |bad.ref=e) implies (some f : Forecast |
e.date=e.date and e.place=f.place)

}

//Only one bad Notification for outdoor event

fact OnlyOneBadNotificationForEventOutdoor{

                  all b1:BadWeatherNotification | all b2:BadWeatherNotification |

                  b1.ref=b2.ref implies b1=b2

}

//Username exists only if exists one user owner

fact UsernameOnlyWithUser{

         all u:Username | some user:User |

                  user.name=u

}

//Date with Event

fact UsefulDate{

         all d: Date | some e: Event |

                  e.date=d

}
```

```
//Useful place

fact UsefulPlace{

        all p: Place | some e: Event |

                e.place=p

}

//Useful hour

fact UsefulHour{

        all h: Hour | some e: Event |

                e.startingHour = h || e.endingHour = h

}


//Useful weather

fact UsefulKindWeather{

        all k: KindOfWeather | some f: Forecast|some p : Preference |

        f.mainCondition = k || p.mainCondition = k

}
```

```
//One event is in a Calendar iff the event's creator is the owner of the calendar or exists an
invitation for that event

fact EventCorrectlyInACalendar{

        all c : Calendar | all e : c.events | some u : User |

                                        (u.cal=c) and ( ( e.creator = u ) or ( some i : Invitation | i.owner
= u and i.ref = e ) )

}

//Avoiding that an event starts on a certain hour and finishes before this hour

fact HourCorrect{

        all e: Event | e.startingHour.hour<=e.endingHour.hour

}

//Avoiding overlap

fact NoOverlap{

        all c: Calendar | all e1: c.events | all e2: c.events |

                    (e1.date=e2.date && e1!=e2 )implies (e1.startingHour.hour >=
e2.endingHour.hour || e2.startingHour.hour >= e1.endingHour.hour)

}

fact AllPreference{

        all oe: OutdoorEvent | all p1: oe.preference | all p2: oe.preference |

        p1.mainCondition = p2.mainCondition implies p1=p2

}
```

8.2 Cases

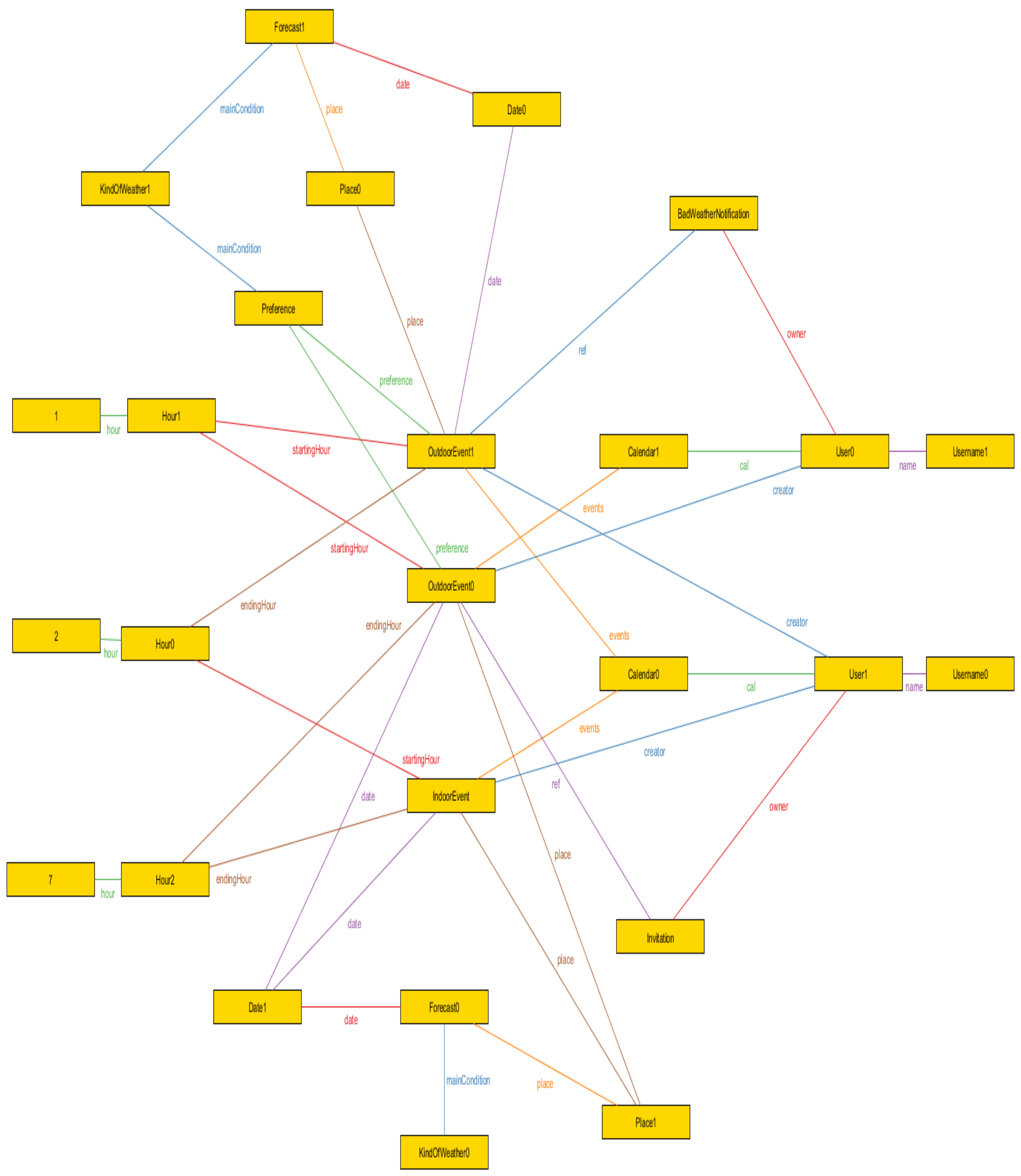
In this section we show different situation.

8.2.1 Global situation

Now, we specify a predicate to show the global situation:

```
pred showGlobal{

#OutdoorEvent=2

# Forecast= 2

#BadWeatherNotification =1

#IndoorEvent=1

#Invitation=1

}

run showGlobal
```

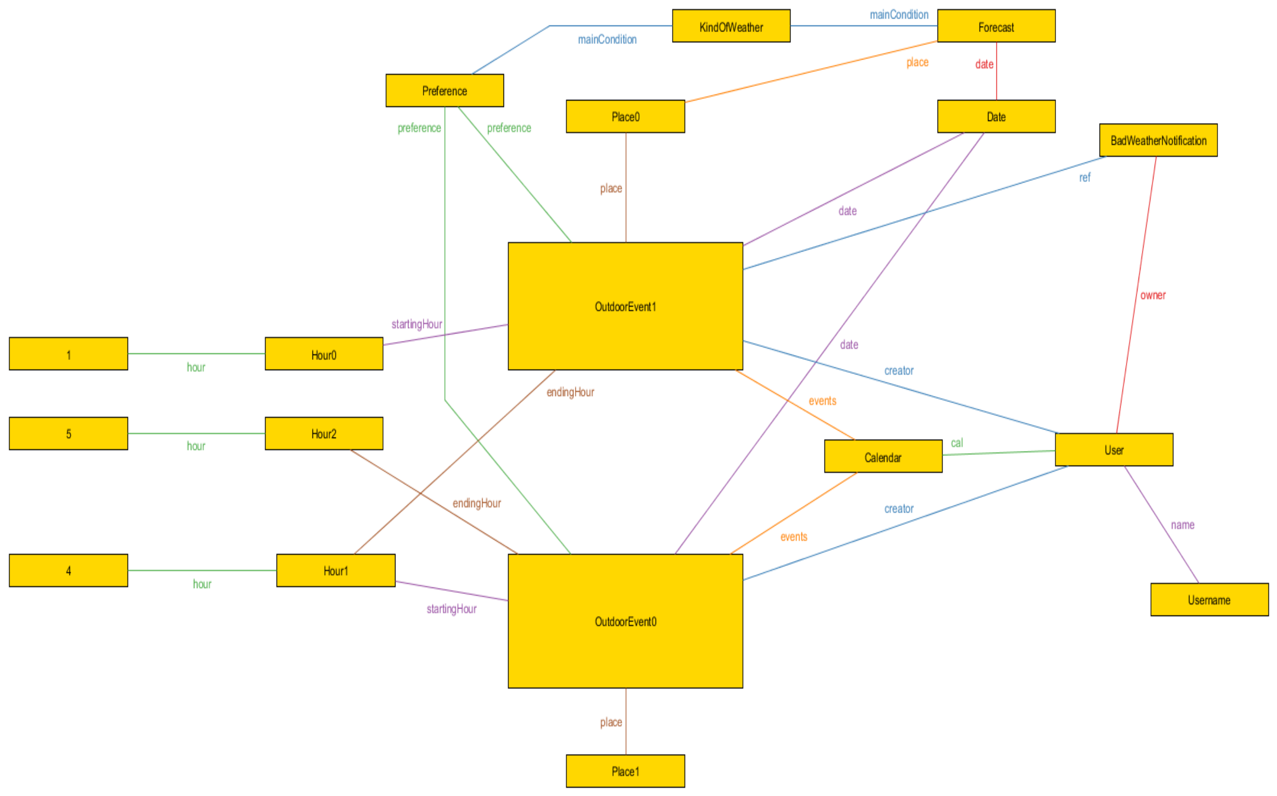
This is the result we obtained:



8.2.2 Bad weather notification

In this part, we want to check that a bad weather notification exists only if there is a forecast for the event associated to the notification's event.

```
pred showBadWeatherNotification{

#OutdoorEvent = 2

#BadWeatherNotification =1

#IndoorEvent=0

#Forecast=1

#Place=2

#Date=1

#User=1

}

run showBadWeatherNotification
```



8.2.3 No overlap

Here, we check if the system avoids overlap between events.

pred *showNoOverlap{*

#User=1

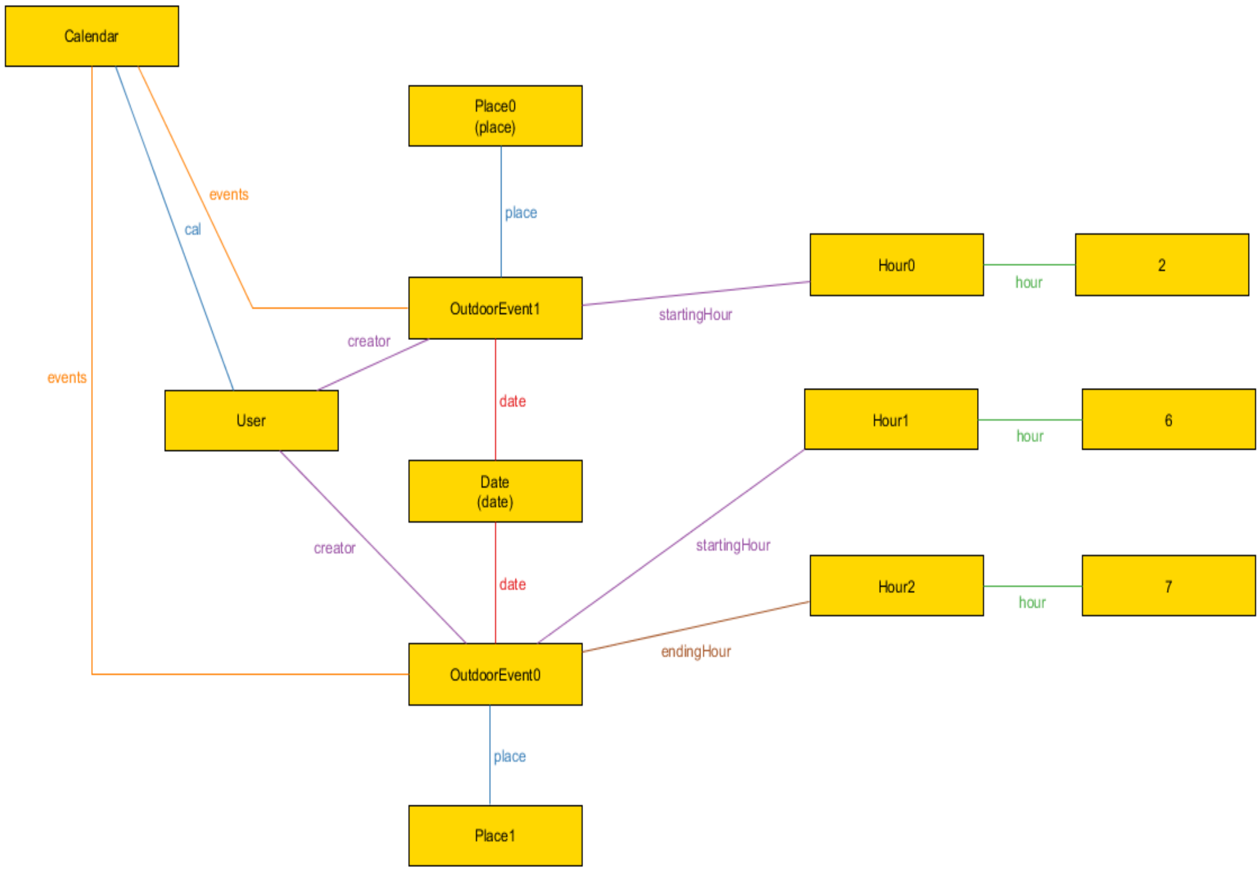
#Date=1

#OutdoorEvent=2

}

run *showNoOverlap*

This is the result:

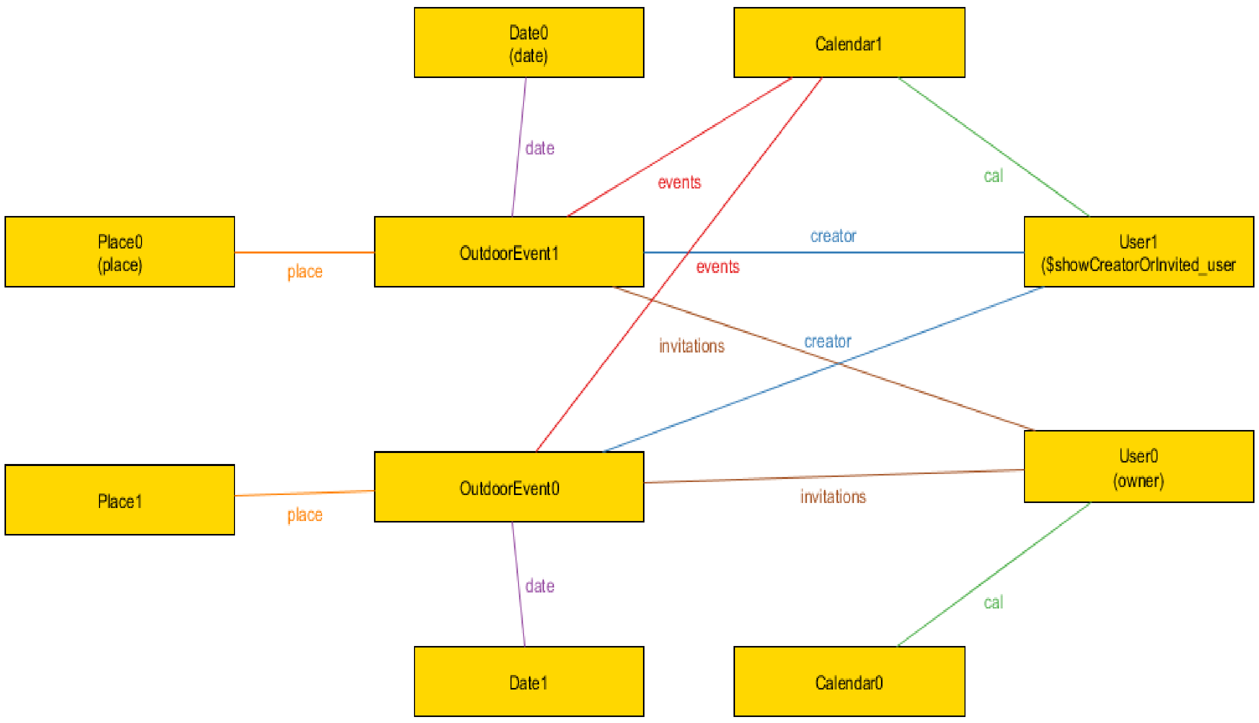


8.2.4 Creator or Invited

If a user creates an event, his username should not appear in the invitation list.

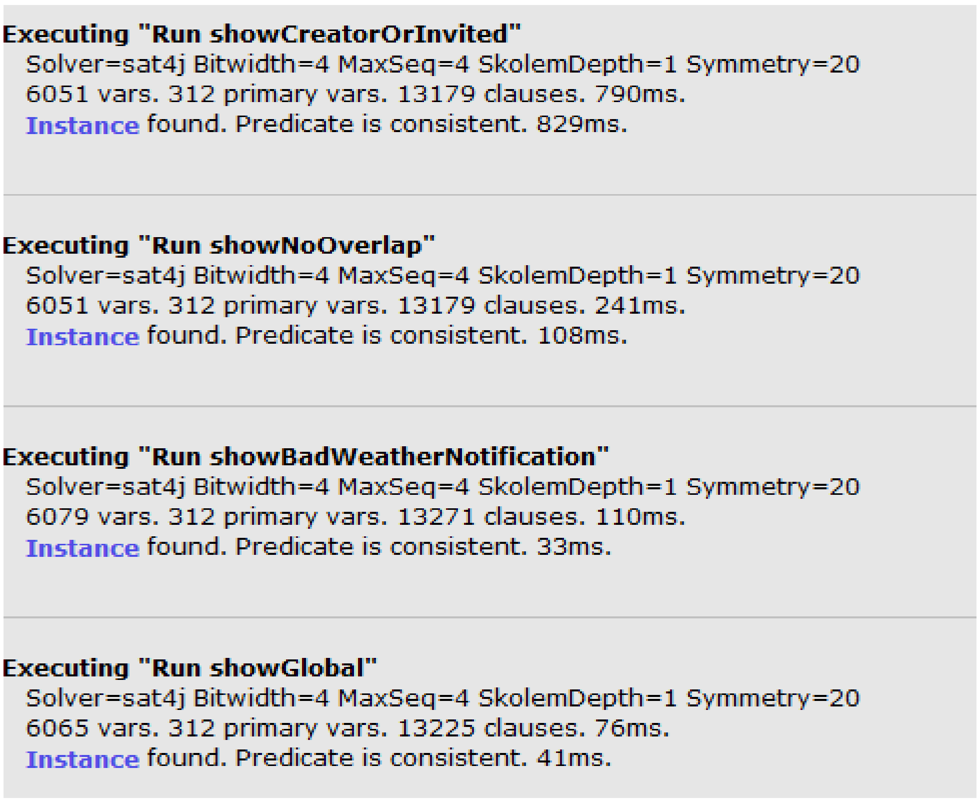
```
pred showCreatorOrInvited{

#User=2

#Invitation=1

#Event=2

}

run showCreatorOrInvited
```

This is the result:



8.2.5 Result

In this section we report the result of Alloy Analyser:



```
Executing "Run showCreatorOrInvited"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   6051 vars. 312 primary vars. 13179 clauses. 790ms.
   Instance found. Predicate is consistent. 829ms.

Executing "Run showNoOverlap"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   6051 vars. 312 primary vars. 13179 clauses. 241ms.
   Instance found. Predicate is consistent. 108ms.

Executing "Run showBadWeatherNotification"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   6079 vars. 312 primary vars. 13271 clauses. 110ms.
   Instance found. Predicate is consistent. 33ms.

Executing "Run showGlobal"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   6065 vars. 312 primary vars. 13225 clauses. 76ms.
   Instance found. Predicate is consistent. 41ms.
```

9 Tools

In order to complete this documentation, we have used the following programs:

- ***Microsoft World 365*** to redact and format this document
- ***Modelio*** to create use case, class diagram and sequence diagram
- ***Alloy Analizer 4.2*** to test the consistency of the class diagram