Requirements Analysis and Specifications Document <MeteoCal>

MATTEO GAZZETTA   ALESSANDRO FATO

837853   *838218*

November 16, 2014

Status: Final

Version: 1.0

Contents

1 Introduction

1.1 Purpose

The following document represents the Requirement Analysis sand Specification Document (in short RASD) of the project <MeteoCal>(in short MC). The document was written to meet the needs of all the stakeholders of the project, which are:

1.2 The Present System

Until this moment, all the major web calendars are done without forecast weather information for the calendar events. The users have to go to on external web services to find forecast weather information for their events.

- The clients, that are the teachers of the course "Software Engineering 2" at Politecnico di Milano
- The developers, that are Matteo Gazzetta and Alessandro Fato, students of the course
- The testers, that are other students of the same course that will be chosen by the teachers

The goal of the RASD is to documenting the process of Requirements Analysis & Specification, by describing the choice made in terms of functional and not functional requirements, constraints, goals and how the system is meant to be used.

1.3 Scope

The intent of the project is delivering a web application which allows the users to have a personal calendar in which manage own events or appointments based on weather conditions.

1.4 Domains

The application to realize is a web application for a calendar with weather forecast information.

For a first analysis of the application's domain we used the method "The World & Machine" of M. Jackson & P. Zave. This approach show us the entities that interact with the system to build ("The World"), the entities of the system to develop ("The Machine") and the shared phenomena within the world and the application, which is the part of the world that is known or directly managed by the application to be developed.
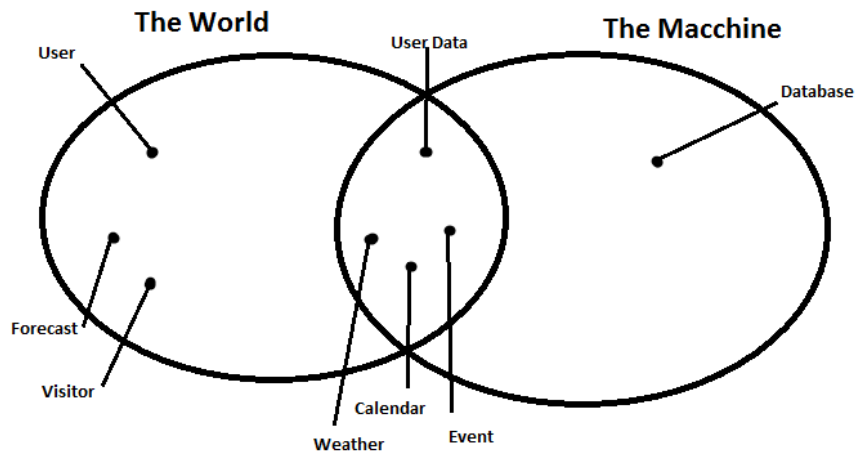
Figure 1: The World and The Machine of MC

1.5 Principals objectives

- Management of the calendar's events (creation, deletion, update and visibility)
- Management and notifications of outdoor events based on updated forecast weather information
- Management of invitations to events between registered users

1.6 Applications functionality

1.6.1 System function

- Users registration on the website
- Log in of the user
- System must be able to fetch updated weather forecast from external forecast services
- In case of bad weather conditions for outdoor events, three days before the event, the system should propose to its creator the closest (in time) sunny day (if any)

1.6.2 Actors function

- **Users** are <MeteoCal>'s users and the primary users of the system.
  - Add/Modify/Delete events on the calendar
  - Invite any Users to the events
  - Accept or decline invitations from other users
- **Visitors** are unregistered users
  - Registration on the application

1.6.3 Application Goals

[G1] MC allows visitors to register and log into the system

[G2] MC allows users to create, modify, update and delete owned events

[G3] MC allows users to view public calendars, public events and owned events

[G4] MC notifies EO (on log in) three days before owned outdoor events in case of of bad weather conditions

[G5] MC notifies all EPs (on log in) one day before the outdoor event in case of bad weather conditions

[G6] MC proposes to EOs the closest (in time) sunny day (if any), three days before owned outdoor events in case of of bad weather conditions

[G7] MC allows EOs to invite friends to owned events

[G8] MC allows EPs to accept or decline invitations

[G9] MC allows EPs to cancel confirmed participation to event

[G10] MC allows Users to change personal information and setting

[G11] MC allows Users to Add/Remove a calendar to the preferred list.

1.7 Application limits

- The system doesn't have email notification in case of bad weather conditions
- Import and Export of user calendar
- Periodically (every x hours) check of bad weather conditions for the events
- The Users cannot delete its account from the system

1.8 Document History

Version	Change Description	Author	Date	Released
1.0	Initial Document Creation	MG AF	01.11.2014	16.11.2014

1.9 Definitions, Acronyms, Abbreviations

1.9.1 Definitions

Definition	Explanation
User	Registered and authenticated member of the application
Event	Information about location, date and participants of users appointment
Event Organizer	The User who owns the event
Event Participant	User who participates the event created by the Event Organizer
Invitation	Request of participation to an event send by the Event Organizer to the Event Participants
Calendar	The set of events scheduled for the user
Visibility	The privacy setting (Public or Private) of a calendar or an event of the system
Weather Conditions	Information about weather condition related to event
Notification	A message send from the system to the users
Weather Notification	A notification of the bad weather condition related to an event
Event Rescheduling	The activity of rescheduling an event, done by the Event Organizer in case of bed weather condition, eventually after system notification and suggestion

1.9.2 Acronyms

Acronym	Explanation
G	Goal
FR	Functional Requirements
NFR	Non Functional Requirements
UC	Use Case
EO	Event Organizer
EP	Event Participant
WC	Weather Conditions
WN	Weather Notification
ER	Event Rescheduling
MG	Matteo Gazzetta
AF	Alessandro Fato
RASD	Requirements Analysis and Specification Document
DBMS	Database Management System
API	Application Programming Interface
OWM	OpenWeatherMap

1.9.3 Abbreviations

Abbreviation	Explanation
MC	MeteoCal
DB	Database

1.10 Document References

Documents below, related to the current initiative, have been created prior to or in conjunction with the Functional Requirements document and can be referenced for further detail:

Name	Date	Version	Author	Location
MeteoCal Prj 14-15.pdf	24.10.14	1.0	Professors	Deliveries folder
IEEE Std 830-1998.pdf	20.10.98	1.0	IEEE	URL
Alloy-Fato-Gazzetta.als	16.11.14	1.0	MG AF	Appendix

1.11 Overview

This document is composed by the following parts:

Introduction Gives a general description of the software to be realized

Overall Description Shows the general aspect of the project, which are interface, constraints, assumptions and dependencies of the software and the characteristics of the users

Specific Requirements In this section there are scenarios, use cases and other diagrams useful to represent the functions of the application

Appendix In this section there is the Alloy model of system to be realized

2 Overall Description

2.1 Product Perspective

The software to be realized is a web application for managing public and private events schedule and participation. In case of bad weather forecast during scheduled outdoor activities, the application provides a basic recommender system to choose the closest possible in time, sunny day (if any).

2.1.1 System Interfaces

MC does not provide any external system interface.

2.1.2 Hardware Interfaces

MC does not provide any external hardware interface.

2.1.3 User Interfaces

The software product will present a web-based user interface, with three different layouts based on page functions:

- **Home Page**: Page for registration and log in of the users
- **Calendar View**: Page for viewing and managing calendars
- **Event View**: Page for viewing and managing events
- **Settings View**: Page for system settings personal configuration

A brief functional description of each view correlated by mock-up will follow.

The **home page** is the landing page for non authenticated users and presents two principal options:

- Visitors can register himself in to the system
- Users can authenticate in to the system
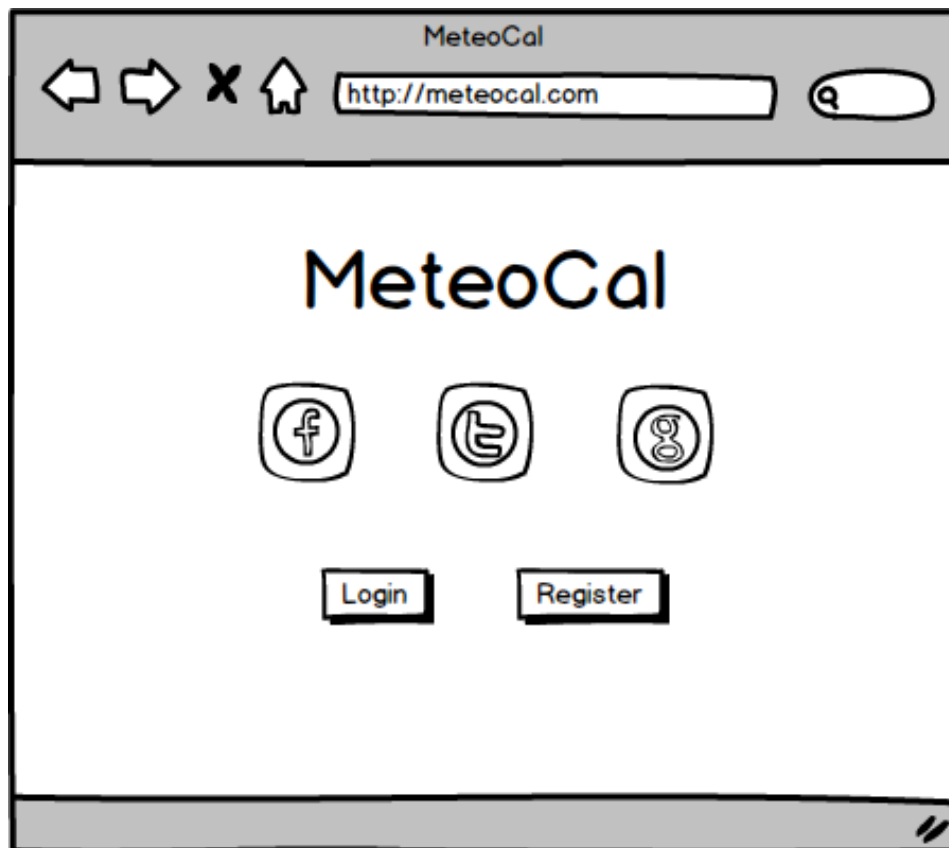
Figure 2: An ideal mock-up of the home page of the application

After log in, the landing page of the user interface is the Calendar View, which permits the following actions:

- View events scheduled in the calendar range specified
- Create/Modify/Delete an event
- Searching inside owned, participating and public events events or other user public calendars
- View notification messages
- View the preffered calendars
- Open settings page
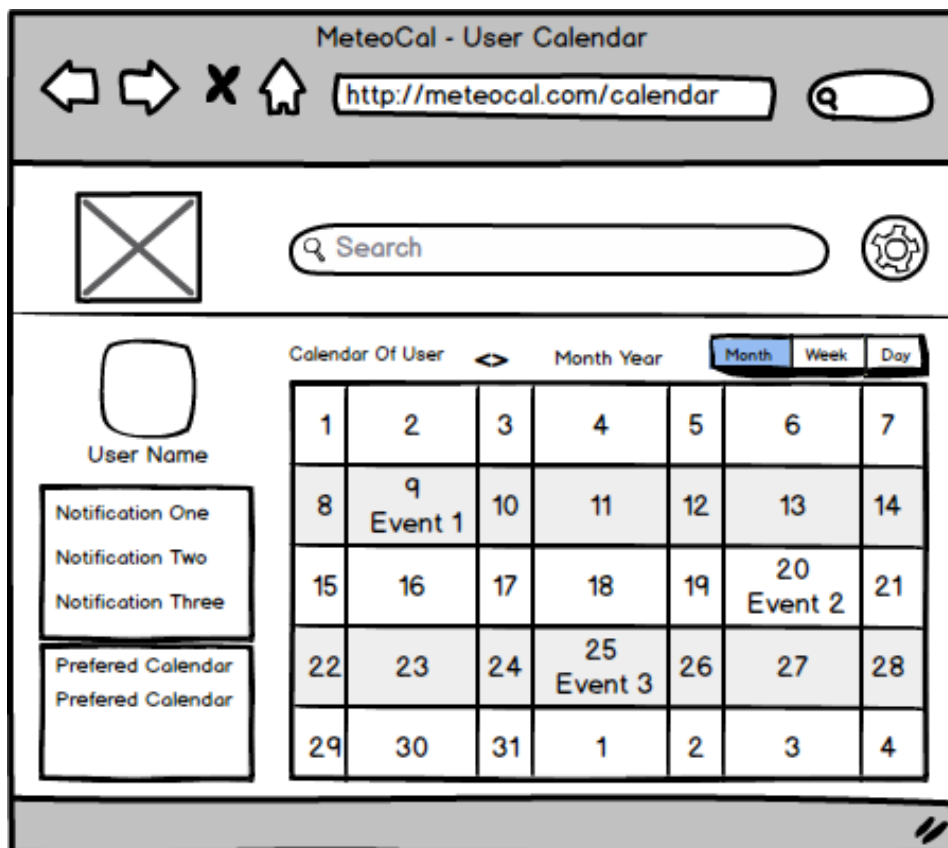
Figure 3: An ideal mock-up of the Calendar View

The **Event View** permits the user to manage owned or participating event
- View event details
- View participants and pending invitation status
- Add participants (if EO)
- Edit or delete the current event (if EO)
- Cancel participation in the current event (if EP)

Figure 4: An ideal mock-up of the Event View for the EO

Figure 5: An ideal mock-up of the Event View for the EP

In the **Settings View** the users can configure is own settings for the application, in particular they can:

- Change the visibility of their own calendar
- Change account information (email, username and password)
- Change personal information
- Change time zone
- Change Date/Time format

Figure 6: An ideal mock-up of the Settings View

2.1.4 Software Interfaces

The application will run on GlassFish Application Server and it will use an external database. For this project it will be used the Oracle MySQL Community Server.

DataBase Server	
Nome	MySQL Community Server
Versione	5.6.21
Site	http://www.mysql.com/

Application Server	
Nome	Glassfish
Versione	4.1
Site	https://glassfish.java.net/

The application runs in every operating system for which is available the application server and the database server.

2.1.5 Communication Interfaces

Protocol	Application	Port
TCP	HTTP	80
TCP	MySQL DB	3306

2.1.6 Memory

For the correct execution of the application on the server the system must respect the minimal requirements for the primary memory of the application server and the database server. Also there is required at least 5GB of secondary memory available on the system.

2.1.7 Operations

All the needed operations are described in the product functions section, in exception of the ordinary maintenance operations that the administrator will do periodically.

2.1.8 Site adaptation requirements

Requirements for the server:
- Java Virtual Machine
- DBMS and Application Server
- Memory requirements

Requirements for the user system:
- Web Browser (Chrome, Firefox, Internet Explorer 10+)
- Internet Connection

2.2 Product Functions

It was identified two different types of actors interacting with the system. Following we define the identified actors with a description of the system functionality that the system must provide.

Visitor

> *[FR1]* Registration and Log in
>
> *[NFR1]* The users data must be saved in a secured way and respectfully to privacy policy
>
> *[NFR2]* Changes to the system must be permanent
>
> *[NFR3]* The system must be reliable and available even in case of heavy loads
>
> *[NFR4]* The system must be available 24h/7d
>
> *[NFR5]* All the visitor's input must be checked

User

> *[FR2]* Views personal calendars
>
> *[FR3]* Views public calendars
>
> *[FR4]* Views public events
>
> *[FR5]* Views joined events
>
> *[FR6]* Configure system settings
>
> *[NFR1]* The users data must be saved in a secured way and respectfully to privacy policy
>
> *[NFR2]* Changes to the system must be permanent
>
> *[NFR3]* The system must be reliable and available even in case of heavy loads
>
> *[NFR4]* The system must be available 24h/7d
>
> *[NFR6]* The weather forecast should be updated at least daily

The User is also divisible in different specific actors that inherit the User functionality:

EO

> *[FR7]* Creates events
>
> *[FR8]* Configure the event visibility (public or private)
>
> *[FR9]* Modifies events
>
> *[FR10]* Deletes events
>
> *[FR12]* Invites friends to the owned events
>
> *[FR11]* Be notified three day before in case of bad weather condition for an owned outdoor event
>
> *[FR13]* Three day before in case of bad weather condition for an owned outdoor event

EP

> *[FR11]* Accept or decline invitations to events
>
> *[FR12]* Cancel accepted participation to an event
>
> *[FR13]* Be notified one day before in case of bad weather condition for an outdoor event

2.3 User Characteristics

Expected users should meet the following characteristic:

- Knowledge in using a browser
- Knowledge in how a digital calendar works

2.4 General Constraints

2.4.1 Regulatory policies

MC doesn't have to meet any regulatory policies

2.4.2 Hardware limitations

MC doesn't have to meet any hardware limitations if not the minimum request memory limitations described in the previous sections and a computational power needed to handle all the Users.

2.4.3 Interfaces to other applications

MC does need to meet the interfaces of OpenWeatherMap services. Also the application communicate with the social interfaces services for doing the registrations and log ins of, respectively, the Visitors and the Users.

2.4.4 Parallel operation

MC must support parallel operations from different users, especially when working with data and connections

2.4.5 Audit functions

The system isn't obliged by constraint of certification on the user data.

2.4.6 Control functions

The system doesn't control any external device or system.

2.4.7 Higher-order language requirements

The maintenance of the application requires knowledge of HTML, Java, JEE technologies, SQL and Administration of information system.

2.4.8 Reliability requirements

The system doesn't requires any particular constraints about reliability of the service. For the reliability of the system is sufficient to replicate the DB to avoid data loss.

2.4.9 Criticality of the application

The system doesn't perform critical operations, nevertheless it must be accessible 24/7.

2.4.10 Safety and security considerations

The system isn't obliged to any specific security constraint, if not about the conservation of the sensible data of the user inside the DB.

2.5 Assumptions

The following are the assumptions made to specify some unclear aspects of the specification document given

Registration strategy In the specification document is not mentioned any particular registration strategy or mandatory user information. Therefore it is assumed that there are no mandatory personal information, but is still possible to register with existing social network accounts

Notification In the specification document there is no mention on system notifications details, thus it is assumed to be adequate a textual message shown in the personal page

Public Events The specification document says about public events *"If an event is public, all the registered users can see its details, including the corresponding participants"*. Therefore we assume that if an event is public, any user can participate to it

Events Geographical Data In the specification document is not mentioned any particular geolocation strategy: it is assumed that EOs must provide valid city name or GPS coordinates for the event created

Weather Forecast The specification document says about events save *"Whenever an event is saved, the system should enrich the event with weather forecast information (if available)"* and doesn't specify any specific behaviour in case of no weather forecast available. Therefore we assume that the system do create the event without adding weather condition

Bad Weather The specification document doesn't says anything about which weather conditions are consider a *'bad weather condition'.*Therefore we assume that a bad weather condition means a raining, snowing weather and a very high/low temperature.

Recommender System The specification document says about events rescheduling for bad weather condition: *"In case of bad weather conditions for outdoor events, three days before the event, the system should propose to its creator the closest (in time) sunny day (if any)."*. If the recommender system fails to suggest any valid alternative date, we assume that the EOs can keep or delete the scheduled event.

Rescheduling Activities In the specification document there isn't any description of the rescheduling phase of an event. Therefore it is assumed that if the event is rescheduled successfully, all the EPs of that event are notified about the reschedule and must confirm again their participation

2.6 Apportion of requirements

Future releases of the software product may provide support for:

- Events chat system
- Email Notification
- Import and export of user's calendar
- Periodically update of weather conditions
- HTTPS connection
- Friendship/Following social patterns

3 Specific Requirements

3.1 External Interfaces

3.1.1 OpenWeatherMap

Weather forecast information for users events, both 5 day forecasts and 16 day forecasts are available at any location or city. 5 day forecasts include weather data every 3 hours and 16 day forecasts include daily weather. Forecasts are available in JSON, XML, or HTML format. OWM API Documentation

Parameter	Description
city	
id	City identification
name	City name
country	Country (GB, JP etc.)
coord	
lat	City geo location, lat
long	City geo location, long
dt	Data receiving time, unix time, GMT
main	
temp	Temperature, Kelvin (subtract 273.15 to convert to Celsius)
temp_min	Minimum temperature at the moment
temp_max	Maximum temperature at the moment.
weather	See Weather condition codes table below
rain	
3h	Precipitation volume for last 3 hours, mm
snow	
3h	Snow volume for last 3 hours, mm

Parameters of API call for 5-day/3 hour forecast

Parameter	Description
city	
id	City identification
name	City name
country	Country (GB, JP etc.)
coord	
lat	City geo location, lat
long	City geo location, long
dt	Data receiving time, unix time, GMT
temp	
day	Day temperature, Kelvin (subtract 273.15 to convert to Celsius)
min	Min daily temperature, Kelvin
max	Max daily temperature, Kelvin
night	Night temperature, Kelvin
eve	Evening temperature, Kelvin
morn	Morning temperature, Kelvin
weather	See Weather condition codes table below
rain	
3h	Precipitation volume for last 3 hours, mm
snow	
3h	Snow volume for last 3 hours, mm

Parameters of API call for 16-day/daily forecast

Parameter	Description
weather	
id	Weather condition id
main	Group of weather parameters (Rain, Snow, Extreme etc.)
description	Weather condition within the group
icon	Weather icon id

<div align="center">Weather condition codes</div>

3.1.2 Social

For the log in and registration the system use the OAuth protocol in particular the system use the:

- Facebook API
- Twitter API
- Google Plus API

From each social network in the registration procedure we take this information from the Socials API:

- Name Surname
- Gender
- Email
- Url of the profile image
- Date of Birth
- Current Location (For time zone setting)

3.2 Functional Requirements

Will follow a detailed analysis of the functional requirements, supported by scenarios and use cases descriptions.

3.2.1 Scenarios

In this section there are some possible scenarios of MC.

Scenario 1	*Visitors register himself in the system*

Alice really needs a web app that could help here to handle personal appointments in a fashioned and enjoyable way. In particular, she frequently organizes trips and pick nick with her friends but, sadly for her, she lives in cold, rainy place. Surfing the Internet she finds the MeteoCal App website. Loving it, she decides to register into the system. From the public homepage, goes through the registration procedure, giving username, her email address and her beloved and world-wide used password. At the end of the procedure, if the data are correct, the system create a new account for Alice.

Scenario 2	*Visitors register himself in the system with an existing social network account*

Alice really needs a web app that could help here to handle personal appointments in a fashioned and enjoyable way. In particular, she frequently organizes trips and pick nick with her friends but, sadly for her, she lives in cold, rainy place. Surfing the Internet she finds the MeteoCal App website. Loving it, she decides to register into the system. From the public homepage, goes through the registration procedure and, instead of the usual boring method, decides to register with here Facebook account. At the end of the procedure, if the system-Facebook transaction is valid, the system create a new account for Alice.

Scenario 3	*User logs into MC*

Alice would like to know if here next pick nick at the city park is going to be delayed for bad weather or not. So goes on the public home page of MC and logs into the system with her username and password.

Scenario 4   *User logs into MC with an existing social network account*

Alice would like to know if here next pick nick at the city park is going to be delayed for bad weather or not. So goes on the public home page of MC and try to logs into the system. But she is lazy and forgetful and prefers to use her Facebook account instead of the her trying to remember the username and password.

Scenario 5   *User adds a new event*

Alice, an authentic pick nick fun, would like to organize a pick nick with her friends at the city park on next Monday. So she logs into MC and creates a new event. During the event creation inserts the title and a brief description of the event. She doesn't want that her ex boyfriend Bob (who is an MC user too) could try discover the event details, so she sets visibility as private. Then, she inserts location and date details for the pick nick: her city, next Monday, at the Green Park. Filled out the last fields, the system gives Alice a feedback on the weather forecast currently available for the occasion. Alice isn't fine with the forecast and decide to accept the system suggestion to anticipate the pick nick of one day. Then, she starts to invite her friends and when finished, save the event into the system.

Scenario 6   *EO modifies an existing event*

Alice, an authentic pick nick fun, organized a pick nick with her friends at the Green Park on next Monday. Suddenly, her boss tells her that next Tuesday there is an important software release and, on Monday, she will have to work till late night. So Alice logs into MC and modifies the pick nick event. After the date modification, the system gives her a feedback on the weather forecast currently available for the occasion. Alice isn't fine with the forecast and decide to accept the system suggestion to anticipate the pick nick of one day. Finally she saves the event yet updated: the system will then send invitations to all the current participant previously invited to event.

Scenario 7   *EO deletes an existing event*

Alice, for her first anniversary, organized a ride on the Ferris wheel with her boyfriend Bob on next Friday. She is an MC faithful user and created the event on the system. Sadly, she found out that Bob is faithless and they broke up. So, just after they broke up, Alice logs into MC and deletes the event she created.

Scenario 8   *EO sends a participation invitation to an owned (existing) event*

Alice, an authentic pick nick fun, organized a pick nick with her friends at the Green Park on next Monday. The day after the event creation, she met at the grocery store her childhood friend Carl and after a little chat she decides to invite him at the pick nick. So she logs into MC, open the pick nick event details and sends an invitation to her friend just found.

Scenario 9   *EP accepts a participation invitation*

Carl, a childhood friend of the pick nick maniac Alice, was invited to an event organized by Alice. After the login inside MC, the system notifies Carl of the invitation; he is very happy to join her good friend Alice for a sandwich in the Green Park so accept the invitation a become one of the participant of the event.

Scenario 10   *EP declines a participation invitation*

Carl, a childhood friend of the pick nick maniac Alice, was invited to an event organized by Alice. After the login inside MC, the system notifies Carl of the invitation; he would be very happy to join her good friend Alice, but looking at the events details he found out that his ex girlfriend Dorotea accepted Alice invitation and was going to come at the party. So sadly and with great regret he declines the invitation and, bothered, logs out from the website.

Scenario 11   *EP modify the participation to the event*

Carl, a childhood friend of the pick nick maniac Alice, accepted an invitation to an event organized by Alice, on next Tuesday. Suddenly, his boss tells him that next Wednesday there will be an important software release and, on Tuesday, he will have to work till late night. So, a bit bothered, Carl logs into MC and open the pick nick event organized by Alice and cancels the partecipation to the event.

Scenario 12   *EP manages a participation invitation of a rescheduled event*

Carl, a childhood friend of the pick nick maniac Alice, was invited to an event organized by Alice. After the login inside MC, the system notifies Carl of the reschedule day of the event because a rainy day was forecasted on the previous scheduled day; So he is very eager to meet her good friend Alice for a sandwich in the Green Park so confirm the partecipation at the new scheduled day for the event.

Scenario 13   *User searches a public calendar in the system*

Carl, a childhood friend of the pick nick maniac Alice, would like to go out for romantic date with her. Alice is a very sociable woman and set her personal calendar visibility to public. Carl, knows this, and want to view her personal calendar to fit her current events with their possible romantic date. So he logs into the system and searches in the search bar her calendar and view it.

Scenario 14   *User searches a public event in the system*

Carl, a childhood friend of the pick nick maniac Alice, would like to make a pass at her. Alice is a very sociable woman and often participates to many flash mobs organized in her city. Carl, knows this, and want to look for the next flash mobs in their city to try to be available for the event (he is going to pass for the event . . .). So he logs into the system and searches in the search bar about the next public flash mob event.

Scenario 15   *User searches an owned/participating event in the system*

Alice, an authentic pick nick fun, organized a pick nick with her friends at the Green Park on next Monday. A few days after the event creation, she doesn't remember exactly when the pick nick was scheduled but she is in a hurry and wants to find it with just a few clicks. So she logs into the system and searches between her events, writing in the search bar about the pick nick.

Scenario 16   *User modifies personal information or system settings*

Dorotea, the Carl ex girlfriend, after the broke up moved in another street. But knowing that Carl is stalking her, really would like to share publicly her new address. So she logs into MC, goes in the account settings and deletes from the system the information related to her new address.

Scenario 17   *User modifies personal information or system settings*

Carl, an MC beloved user, after breaking up with Dorotea, would like to make a pass at our pick nick fun Alice. She is a very sociable woman, often participates to many flash mobs organized in her city and chose to set her calendar public. Carl, knows this, and want to look for the next flash mobs in she is going to attend (he is going to pass for the event . . .). So he searches her calendar and simply stars it, so that every time he will log into the system with just one click will see all her personal appointments and could be there for hardly trying to make a pass at her.

Scenario 18   *User modifies personal information or system settings*

Dorotea, the Carl ex girlfriend, after the broke up decided she doesn't want to have any more news about Carl. She also don't want to read anymore his name. So logs into the system, go the Carl's personal calendar and unstars it. Know his calendar will be no more in her preferred calendars list.

3.2.2 Use Cases

Figure 7: Visitor Use Case

3.2.2.1 [UC1] - *<Registration>*

1. *Brief Description*
   The use case details the steps to be performed by visitors to register in the system

2. *Actors*

   - Visitor

3. *Entry condition*

   - The visitor is connected to the <MeteoCal>'s homepage and is not registered into the system

4. *Basic Flow of Events*

Step	Description
1	The visitor clicks on the registration button
2	The system shows required account fields
3	The visitor enters required account information and clicks the register button
4	The system register the new user and redirect him to his personal calendar web page

5. *Exit condition*

   • The user is successfully registered and logged into the system.

6. *Exceptions*

   • The visitor doesn't fill correctly all the required fields or inserts an invalid email; the system notifies the error and doesn't proceed with the registration

7. *Goal*

   • **[G1]**
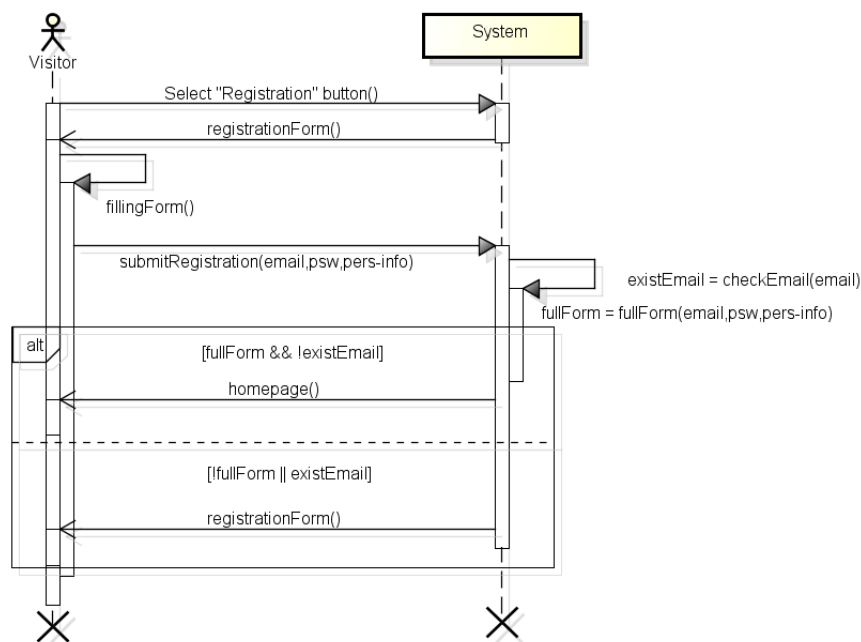
Figure 8: Registration Use Case

3.2.2.2 [UC2] - <Registration Social>

1. *Brief Description*
   The use case details the steps to be performed by visitors to register in the system via an external valid social identity

2. *Actors*

   • Visitor

3. *Entry condition*

   • The visitor is connected the <MeteoCal>'s homepage and still not registerd in website

4. *Basic Flow of Events*

Step	Description
1	The visitor clicks on the desired social network button, for registration via external social ID
2	The system redirects the user for the external social network authentication
3	If the visitor fails to authenticate in the external social network, the procedure goes back to point 2
4	The external social website redirects the visitor to MC
5	The system asks for the missing information and for the password to log in

5. *Exit condition*

   • The user is successfully registered and logged into the system

6. *Exceptions*

   • If the visitors fails to authenticate in the external social networks, the procedure fails
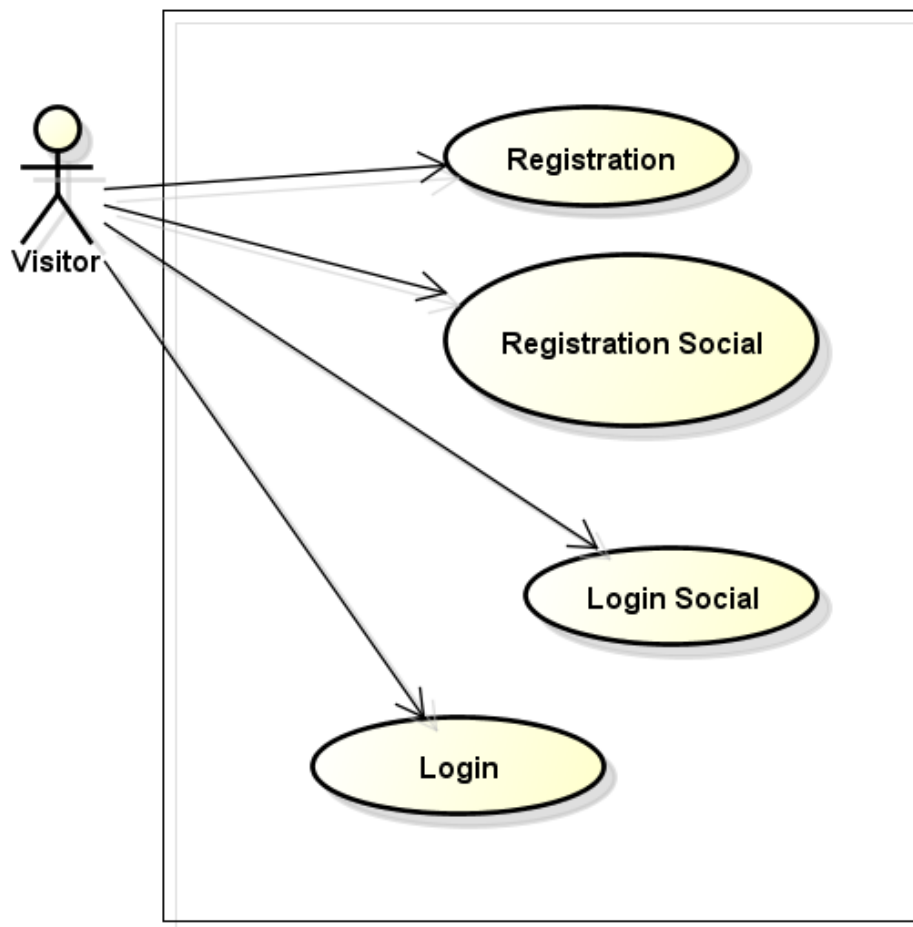
7. *Goal*

   • **[G1]**
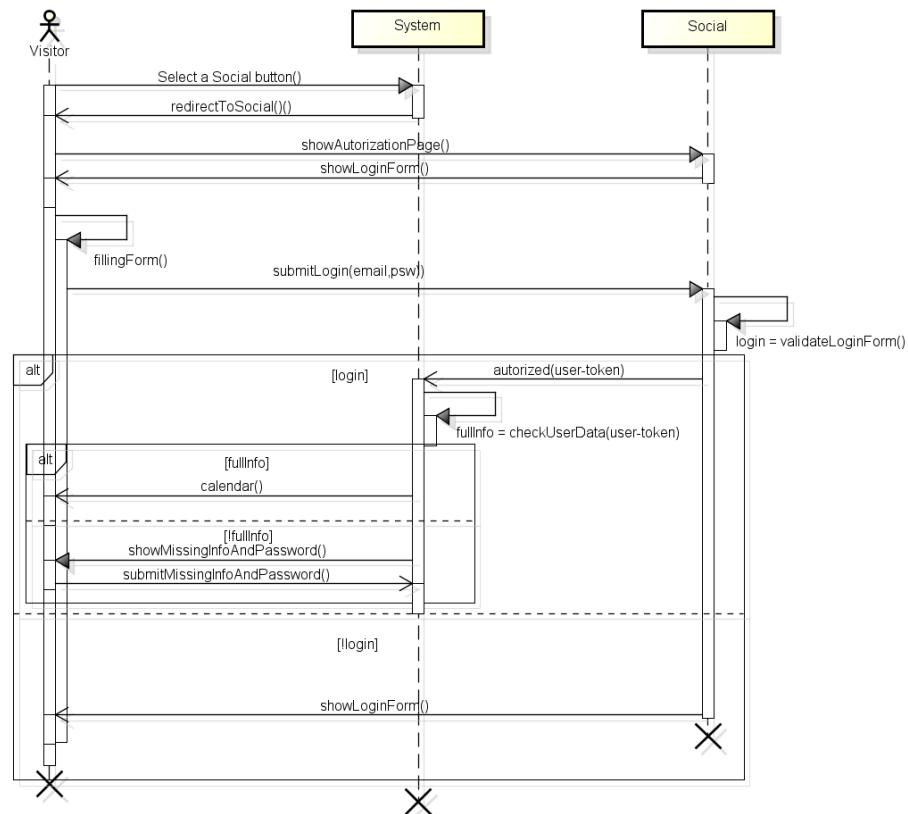
Figure 9: Registration Social Use Case

3.2.2.3 [UC3] - *<Login>*

1. *Brief Description*
   The use case details the steps to be performed by visitors to log into the system

2. *Actors*

   - Visitor

3. *Entry condition*

   - The visitor is connected to the <MeteoCal>'s homepage and is already registered in the system

4. *Basic Flow of Events*

Step	Description
1	The user clicks on the log in button
2	The system shows to the visitor the fields for the insertion of email and password
3	The visitor inserts email and password and clicks on confirm button
4	The system verifies the visitor identity and redirects the user to his personal calendar webpage

5. *Exit condition*

   - The user is successfully logged into the system

6. *Exceptions*

   - If email or password are incorrect the login procedure fails

7. *Goal*

   - **[G1]**
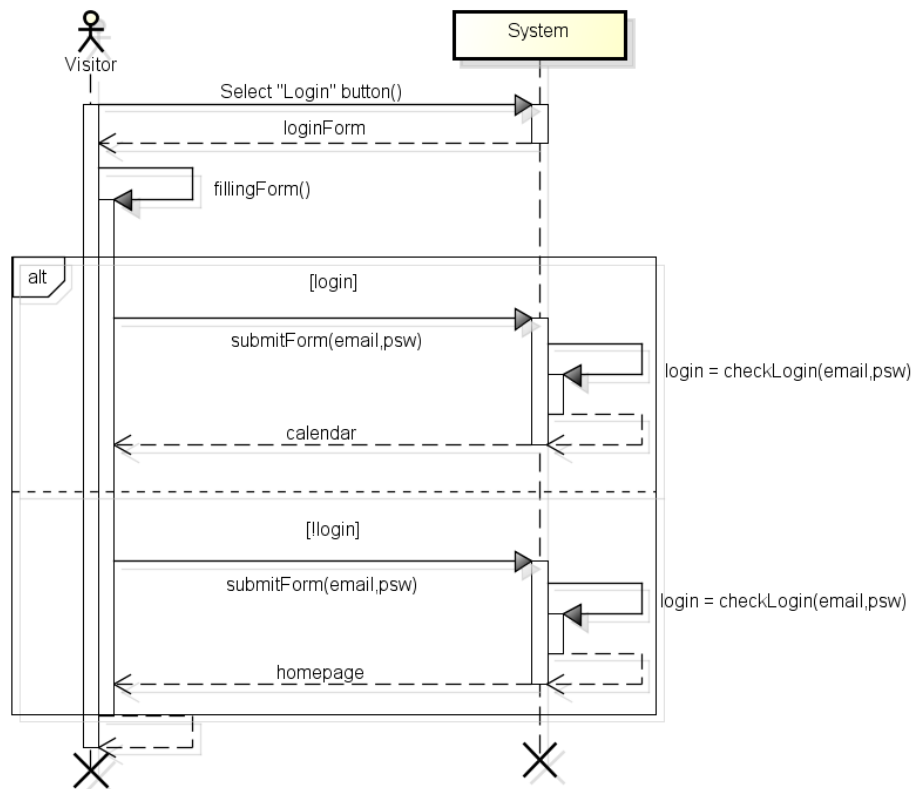
Figure 10: Login Sequence Diagram

3.2.2.4 [UC4] - *<Login Social>*

1. *Brief Description*
   The use case details the steps to be performed by visitors to log in the system via an external social identity

2. *Actors*

   - Visitor

3. *Entry condition*

   - The visitor is connected to the <MeteoCal>'s homepage and owns a valid external social identity

4. *Basic Flow of Events*

Step	Description
1	The visitor clicks on the desired social network button, for log in in via external social ID
2	The system redirects the user for the external social network authentication
3	If the visitor fails to authenticate in the external social network, the procedure goes back to point 2
4	After the successful authentication, the external social website redirects the visitor to the <MeteoCal>'s website

5. *Exit condition*

   - The visitor is logged into the system

6. *Exceptions*

   -

7. *Goal*

   - **[G1]**
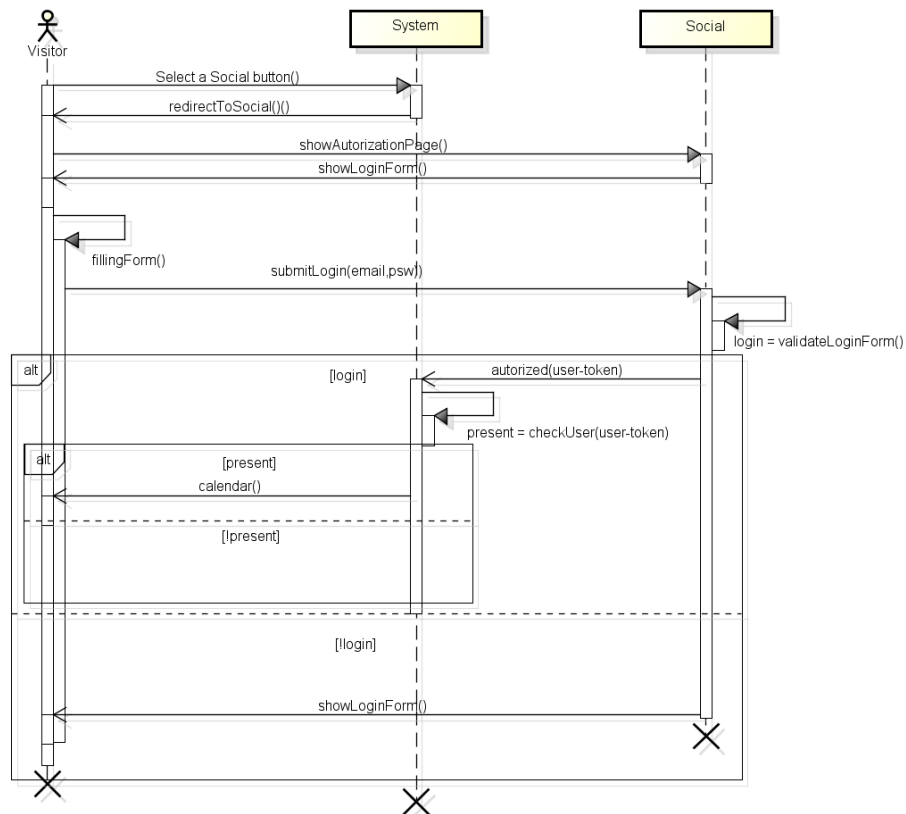
Figure 11: Login Social Sequence Diagram
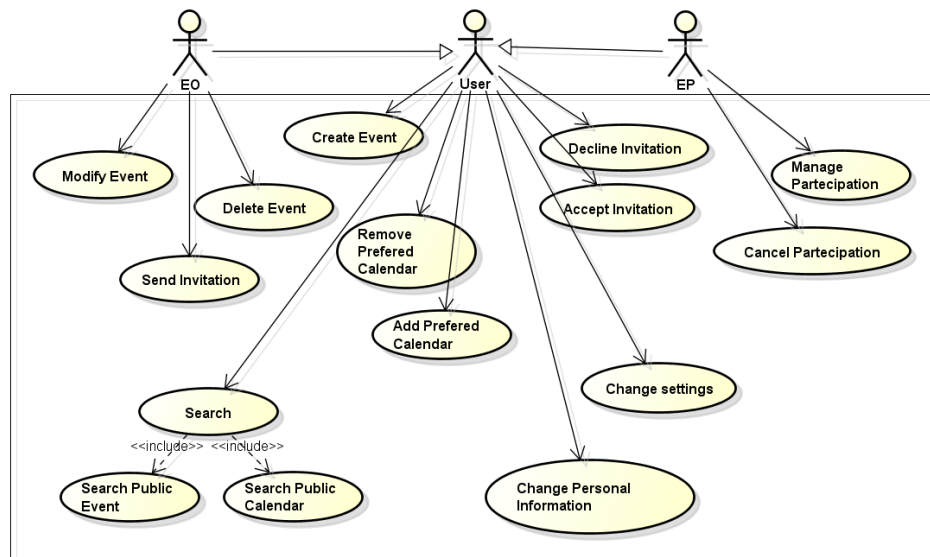
Figure 12: User Use Case

3.2.2.5 [UC5] - *<Creation of a new event>*

1. *Brief Description*
   The use case details the steps to be performed by users to create new events
2. *Actors*

   - User
3. *Entry condition*

   - The User is connected to the personal calendar web page
4. *Basic Flow of Events*

Step	Description
1	The user selects the date desired in the calendar
2	The user enters mandatory event information (name, location, Indoor/Outdoor, visibility, participants if any)
3	The system checks location information and if they are valid, correlate the event with weather information (if forecast available)
4	In case of bad weather conditions, the system suggests an alternative in time, sunny day to the user (if forecast available)
5	Eventually, the User modifies the event location or date and the procedure goes back to point 4
6	The User clicks on the confirm button to confirm the event creation

5. *Exit condition*

   - The new event is successfully created in the user's calendar into the system.

6. *Exceptions*

   - If it is not possible to download valid forecast information for the event, no weather information are added by the system
   - If the User doesn't fill with valid information all the mandatory fields the event creation fails

7. *Goal*

   - **[G2]**

Figure 13: Create Event Sequence Diagram

3.2.2.6 [UC6] - <Modify an existing event>

1. *Brief Description*
   The use case details the steps to be performed by EOs to modify existing events

2. *Actors*

   - EO

3. *Entry condition*

   - The EO have at least an existing event

4. *Basic Flow of Events*

Step	Description
1	The EO selects the event to modify
2	The system shows current information of the event
3	EO changes the desired information of the event
5	The system checks location information and if they are valid adds weather information to the event (if available)
6	In case of bad weather conditions, the system suggests an alternative in time, sunny day to the user (if forecast available)
7	Eventually, the User modifies the event location or date and the procedure goes back to point 5
8	EO selects save button to confirm the event modifications

5. *Exit condition*

   - The event is successfully modified in the system

6. *Exceptions*

   - If it is not possible to download valid forecast information for the event, no weather information are added by the system
   - If the User doesn't fill with valid information all the mandatory fields the event modification fails

7. *Goal*

   - **[G2]**
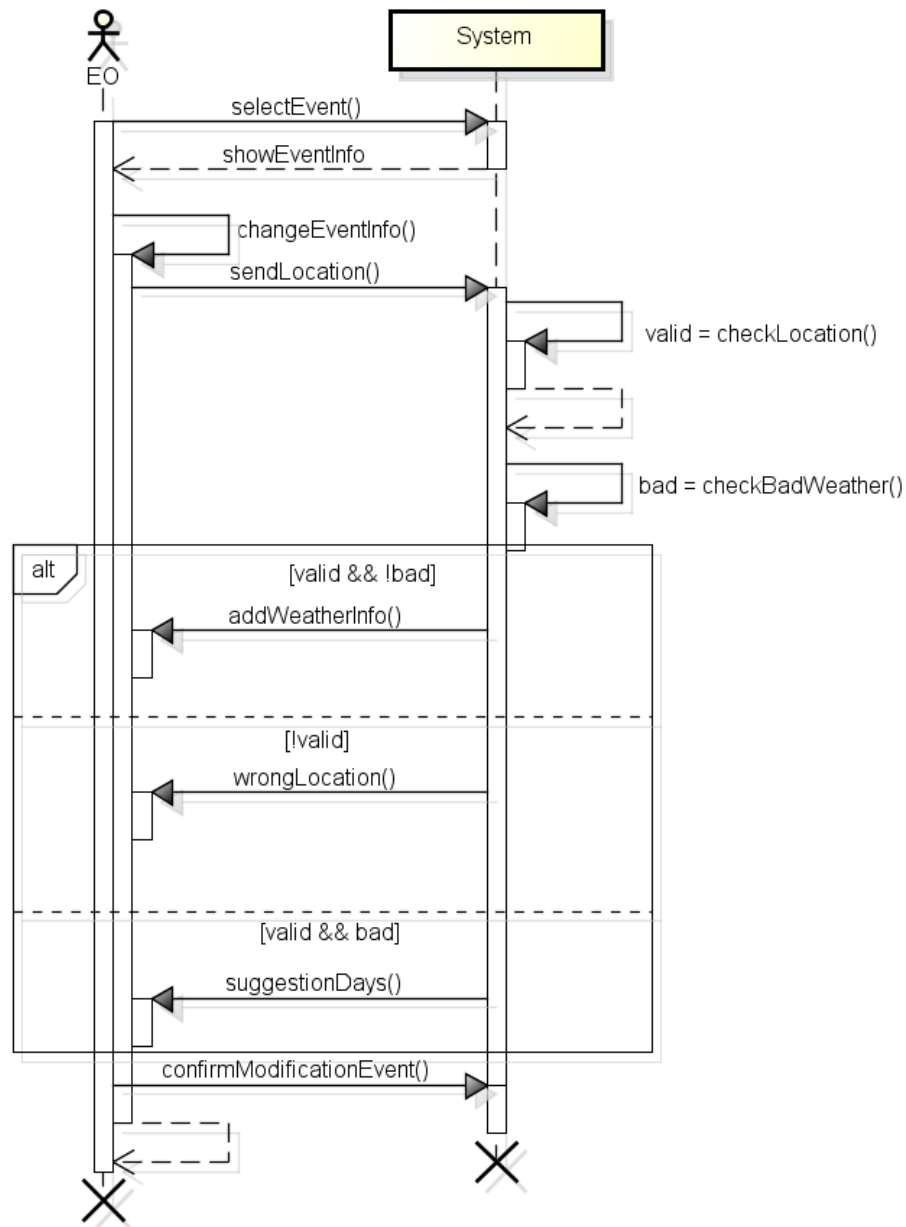
Figure 14: Modify Event Sequence Diagram

3.2.2.7 [UC7] - *<Delete an existing event>*

1. *Brief Description*
   The use case details the steps to be performed by EOs to delete existing events
2. *Actors*

   - EO

3. *Entry condition*

   - The EO have at least an existing event

4. *Basic Flow of Events*

Step	Description
1	The EO selects the desired event
2	The system shows the current information of the event
3	EO selects the delete button
4	EO confirms the delete operation

5. *Exit condition*

   - The event is successfully deleted from the system

6. *Exceptions*

   -

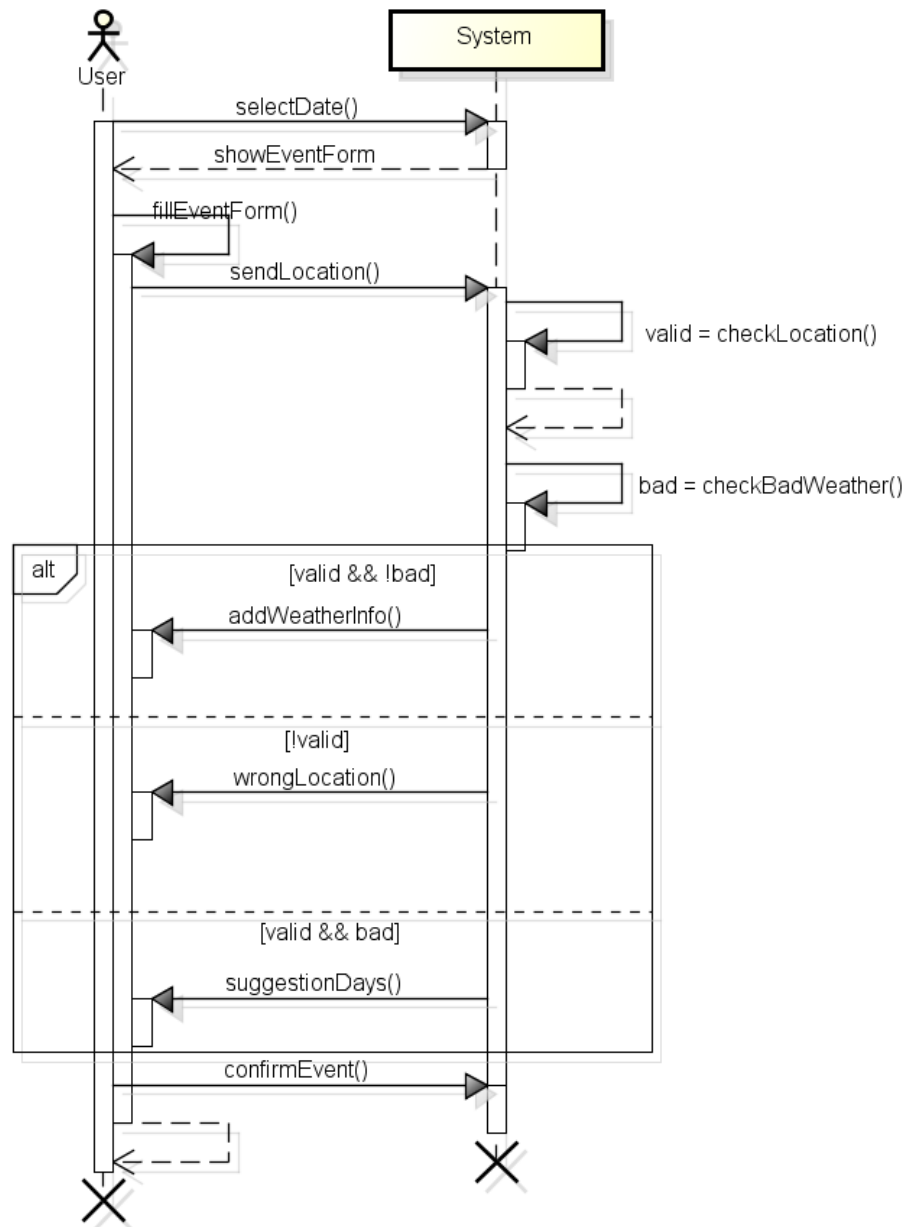7. *Goal*

   - **[G2]**
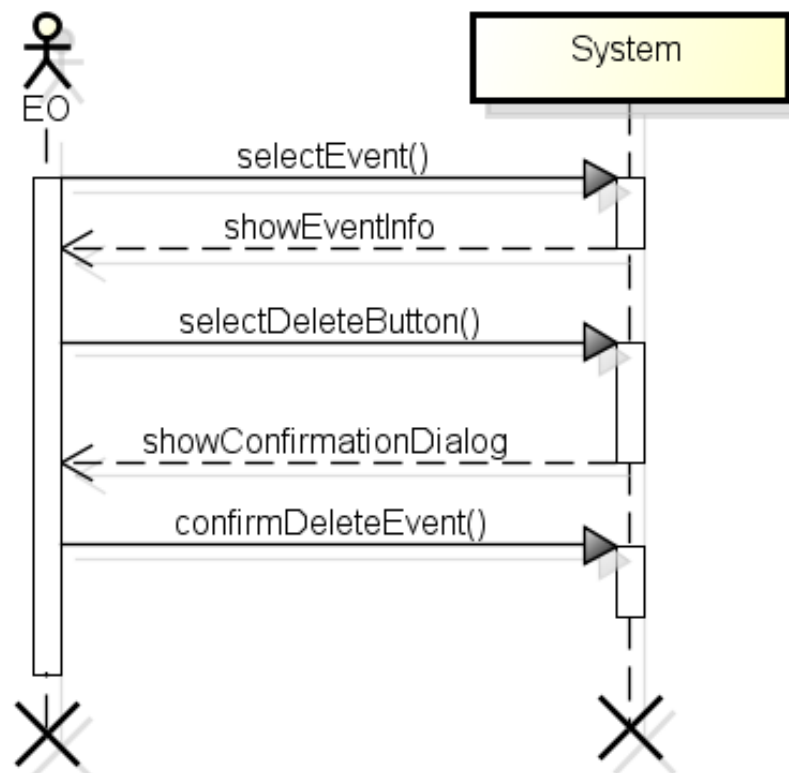
Figure 15: Delete Event Sequence Diagram

3.2.2.8 [UC8] - *<Sends an invitation to an event>*

1. *Brief Description*
   The use case details the steps to be performed by EOs to invite friends to existing events

2. *Actors*

   - EO

3. *Entry condition*

   - The EO have at least an existing event

4. *Basic Flow of Events*

Step	Description
1	The EO selects the desired event
2	The system shows the current information of the event
3	EO selects the add participant button
4	EO enters the friend information
5	The system confirm to the user that the invitation is been sent to the EO's friend.

5. *Exit condition*

   - The invitation is successfully send to the EO's friend

6. *Exceptions*

   - Friend doesn't exist or its unreachable.

7. *Goal*

   - **[G7]**
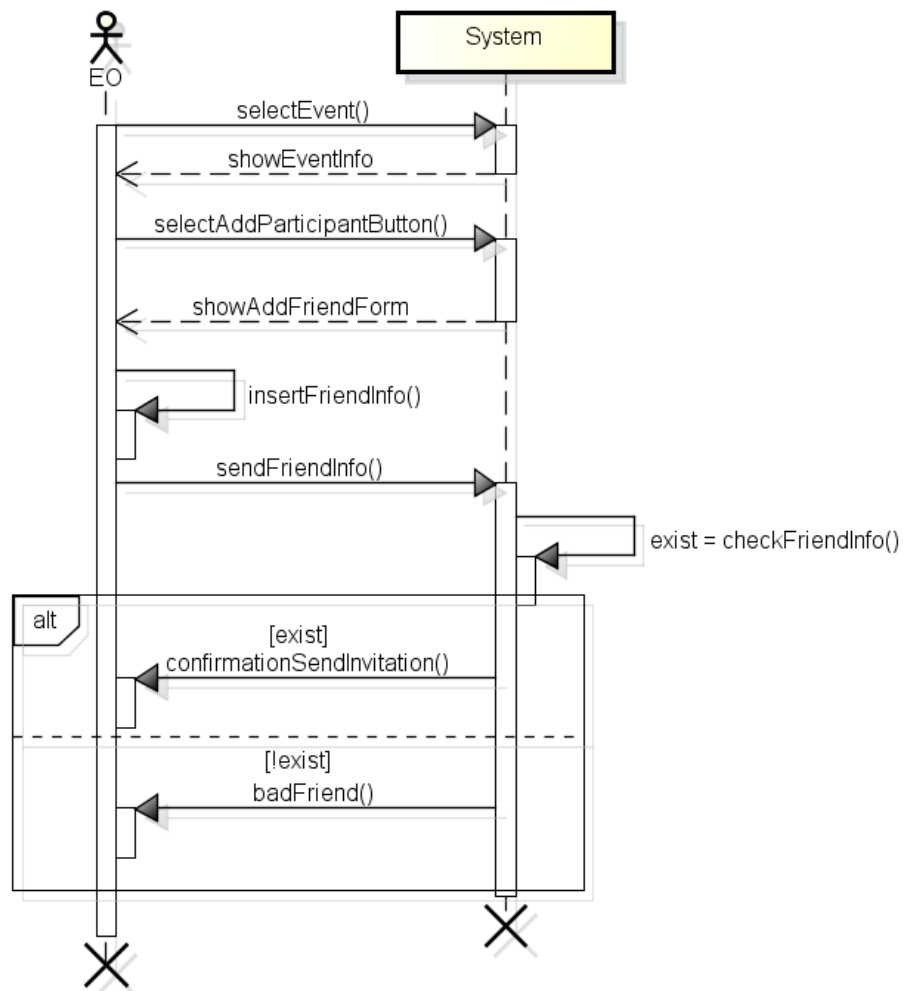
Figure 16: Send Invitation Sequence Diagram

3.2.2.9 [UC9] - <Accept invitation to an event>

1. *Brief Description*
   The use case details the steps to be performed by Users to accept an invitation to an existing event

2. *Actors*

   • User

3. *Entry condition*

   • The User have at least a pending invitation request

4. *Basic Flow of Events*

Step	Description
1	The User selects the desired invitation request notification
2	The system shows the current information of the event
3	User selects the accept button
4	The system modifies the event list participant adding the user to event EPs list

5. *Exit condition*

   • The user is successfully added to the event from the system.

6. *Exceptions*

   •

7. *Goal*

   • **[G8]**
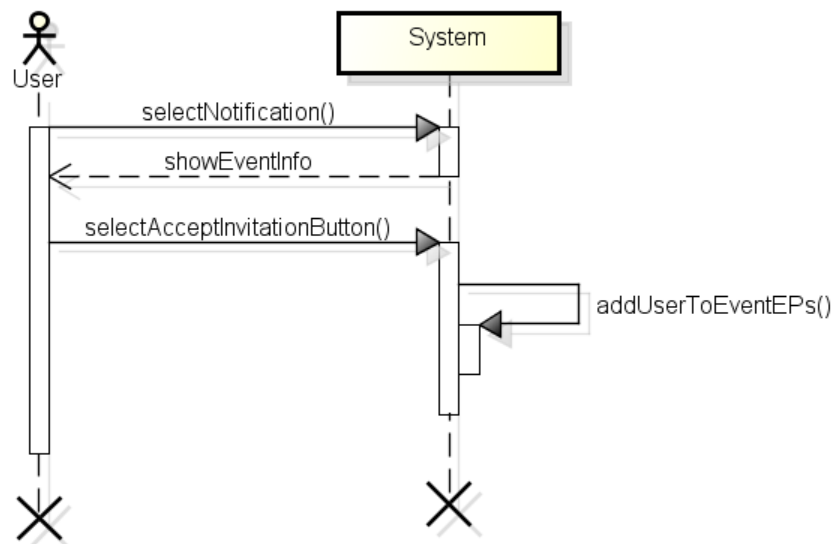
Figure 17: Accept Invitation Sequence Diagram

3.2.2.10 [UC10] - <Decline invitation for an event>

1. *Brief Description*
   The use case details the steps to be performed by a User to decline an invitation to to an existing event

2. *Actors*

   - User

3. *Entry condition*

   - The User have at least a pending invitation request

4. *Basic Flow of Events*

Step	Description
1	The User selects the desired invitation request notification
2	The system shows the current information of the event
3	User selects the decline button
4	The system registers the declined invitation

5. *Exit condition*

   - The event invitation status is declined

6. *Exceptions*

   -

7. *Goal*

• **[G8]**
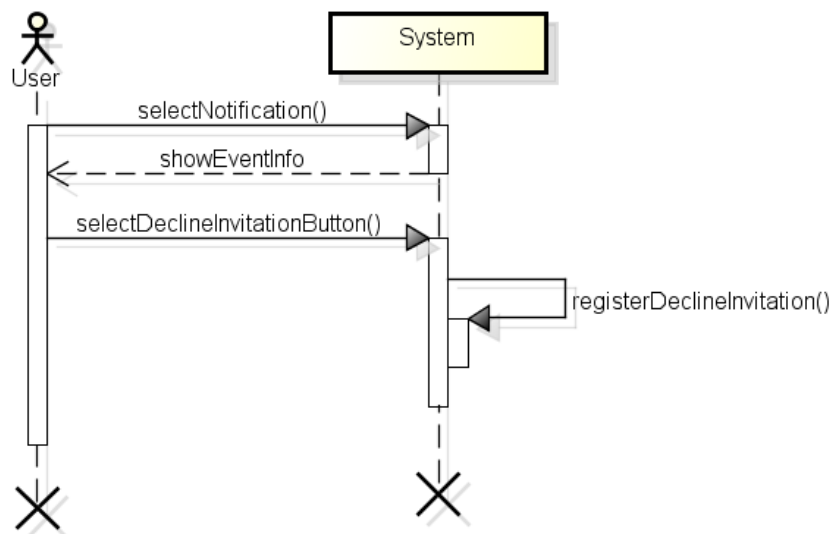
Figure 18: Decline Invitation Sequence Diagram

3.2.2.11 [UC11] - *<Cancel participation to an event>*

1. *Brief Description*
   The use case details the steps to be performed by a EPs to cancel the partici-
   pation to existing events

2. *Actors*

   - EP

3. *Entry condition*

   - The EP is participating at least to one event
   - The EP is on his personal calendar

4. *Basic Flow of Events*

Step	Description
1	The EP selects the desired event
2	The system shows the current information of the event
3	User selects the cancel participation button
4	The system asks confirmation for the canceling participation
5	The system cancel the user from the list of event participants

5. *Exit condition*

   - The User is no more an EP of the described event

6. *Exceptions*

   -

7. *Goal*

   - **[G9]**
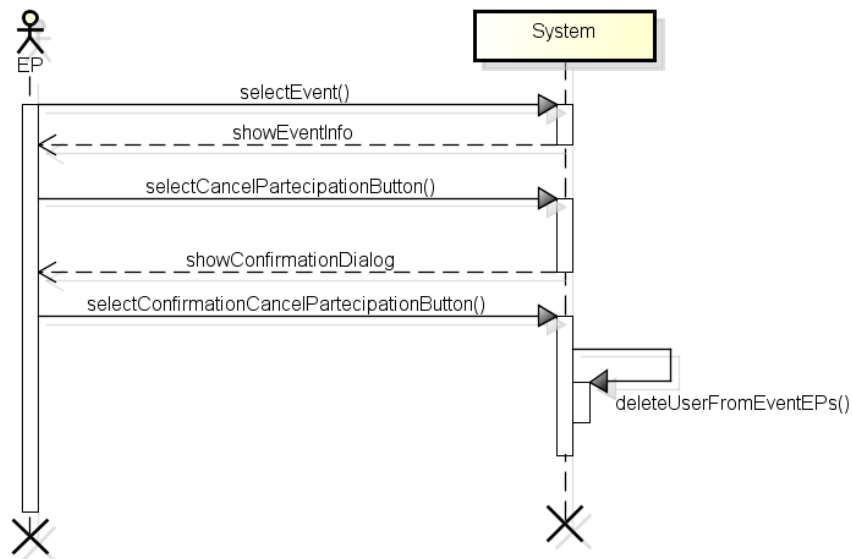
Figure 19: Cancel Partecipation Sequence Diagram

3.2.2.12 [UC12] - *<Manage participation to a rescheduled event>*

1. *Brief Description*
   The use case details the steps to be performed by EPs to manage a rescheduled event

2. *Actors*

   - EP

3. *Entry condition*

   - The EP is participating at least to one event that is rescheduled
   - The EP is on his own calendar

4. *Basic Flow of Events*

Step	Description
1	The EP selects the notification of a rescheduled event
2	The system shows the new current information of the event
3	User accept or decline the new schedule of the event
4	The system if the user accepts keep it in the EPs list of the event otherwise remove it

5. *Exit condition*

   - The User if decline the reschedule is no more an EP of the described event

6. *Exceptions*

-

7. *Goal*

- **[G8]**
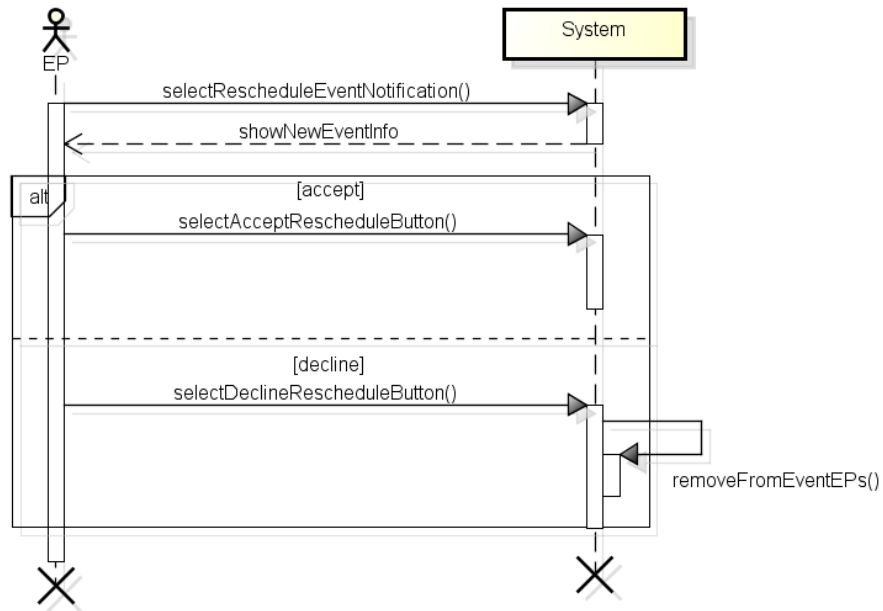
Figure 20: Manage Partecipation Reschedule Sequence Diagram

3.2.2.13 [UC13] - *<Search a public calendar>*

1. *Brief Description*
   The use case details the steps to be performed by Users to search through public calendar

2. *Actors*

   - User

3. *Entry condition*

   -

4. *Basic Flow of Events*

Step	Description
1	The User selects the search bar
2	The User enters calendar information
3	The system shows to the User result(s) for the query (if any)
4	The User selects the desired calendar
5	The system redirect the User to the selected calendar page.

5. *Exit condition*

   - The User is successfully viewing the desired calendar

6. *Exceptions*

   - No calendar can be found based on the user input

7. *Goal*

   - **[G3]**

Figure 21: Search Public Calendar Sequence Diagram

3.2.2.14 [UC14] - <Search a public event>

1. *Brief Description*
   The use case details the steps to be performed by Users to search public events in the system
2. *Actors*

   - User
3. *Entry condition*

   - The event must be part of a public calendar
4. *Basic Flow of Events*

Step	Description
1	The User selects the search bar
2	The User enters event information
3	The system shows to the User result(s) for the query (if any)
4	The User selects the desired event
5	The system shows to the User the event information
5. *Exit condition*

   - The User is successfully viewing the desired event

6. *Exceptions*

   • No event can be found based on the user input

7. *Goal*

   • **[G3]**
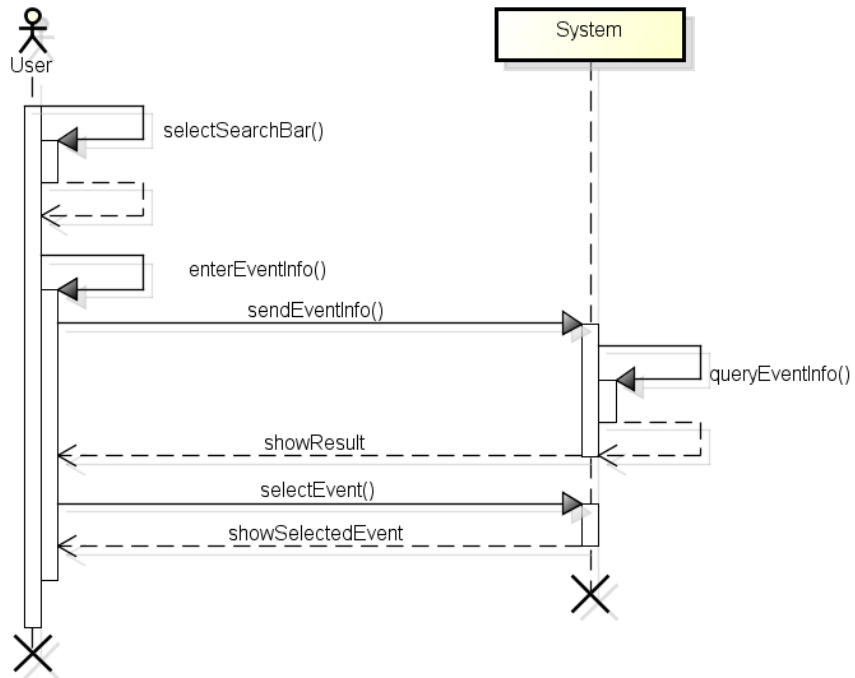
Figure 22: Search Public Event Sequence Diagram

3.2.2.15 [UC15] - *<Change personal information and settings>*

1. *Brief Description*
   The use case details the steps to be performed by User to change their personal information and settings in the system

2. *Actors*

   - User

3. *Entry condition*

   -

4. *Basic Flow of Events*

Step	Description
1	The User selects the settings button
2	The system shows the settings page of the User
3	The User changes the desired personal information and settings
4	The User selects the save button on the bottom
4	The system saves permanently the changes on the system.

5. *Exit condition*

   - The User personal information and settings are permanently and successfully changed

6. *Exceptions*

   - If all mandatory fields doesn't contain valid information, the procedure fails

7. *Goal*

   - **[G10]**
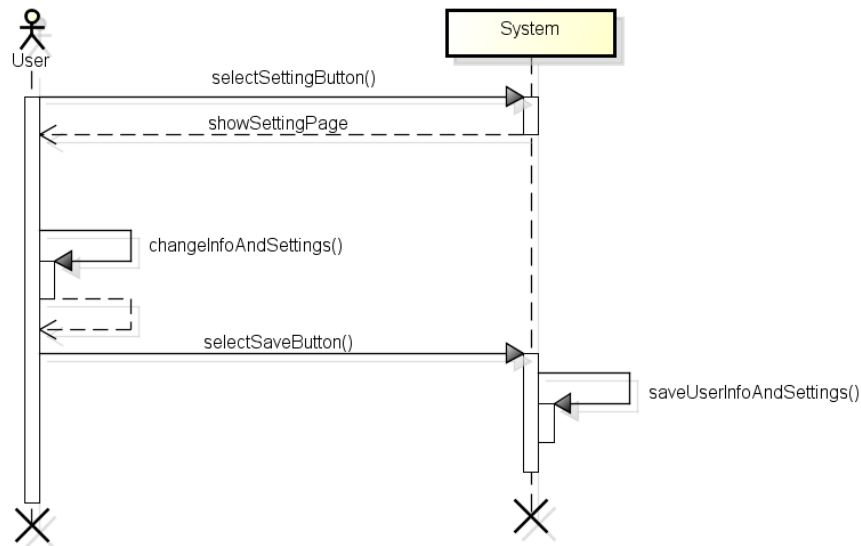
Figure 23: Change Informartion & Settings Sequence Diagram

3.2.2.16 [UC16] - <Add preferred public calendar>

1. *Brief Description*
   The use case details the steps to be performed by User to add public calendars into the preferred calendars

2. *Actors*

   - User

3. *Entry condition*

   - The User is viewing a public calendar

4. *Basic Flow of Events*

Step	Description
1	The User selects the start button of the current calendar
2	The calendar is added to the starred list

5. *Exit condition*

   - The calendar is successfully added from the User preferred calendar in the system.

6. *Exceptions*

   -

7. *Goal*
   - **[G11]**

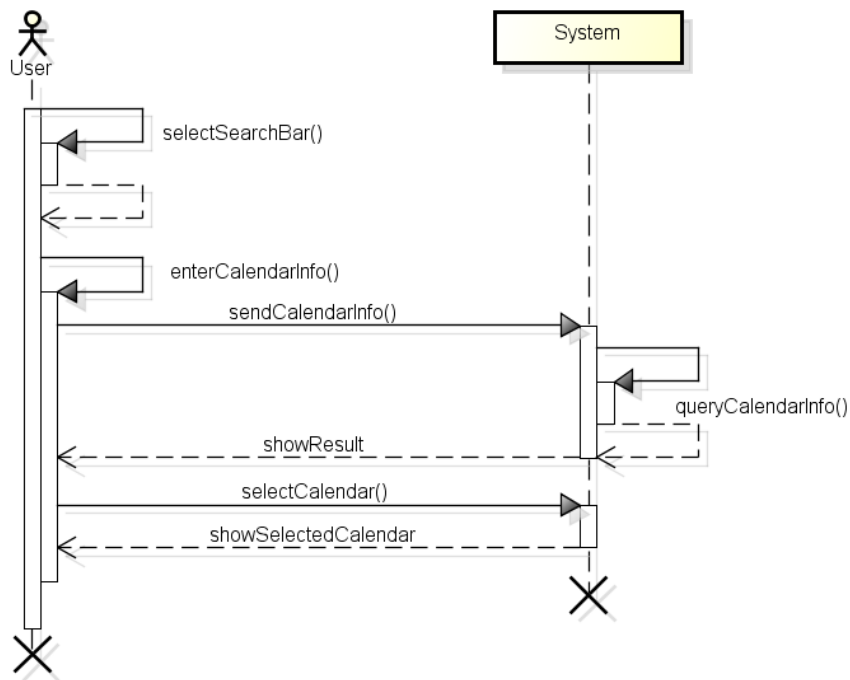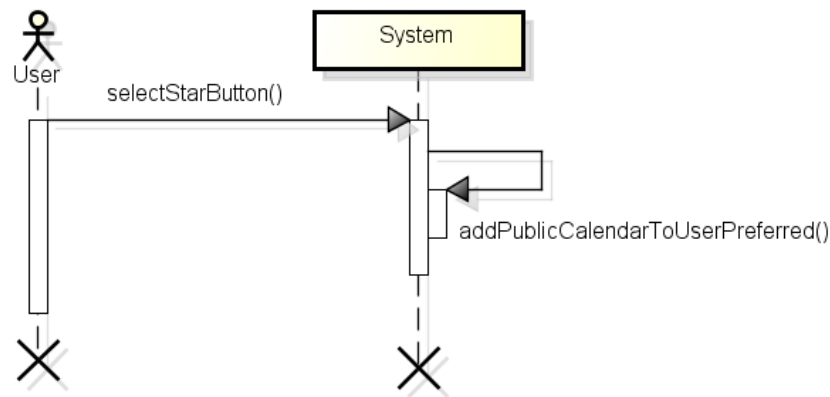Figure 24: Add Preferred Calendar Sequence Diagram

3.2.2.17 [UC17] - <Remove preferred public calendar>

1. *Brief Description*
   The use case details the steps to be performed by User to remove public calendars from the preferred calendars

2. *Actors*

   - User

3. *Entry condition*

   - The User is viewing a public calendar

4. *Basic Flow of Events*

Step	Description
1	The User selects the start button of the current calendar
2	The calendar is removed from the starred list

5. *Exit condition*

   - The calendar is successfully removed from the User preferred calendar in the system.

6. *Exceptions*

   -

7. *Goal*
   - **[G11]**
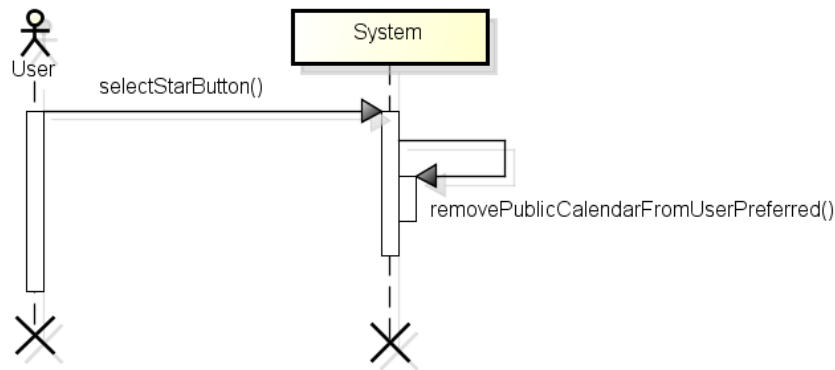
Figure 25: Remove Preferred Calendar Sequence Diagram

3.3 Class Diagram
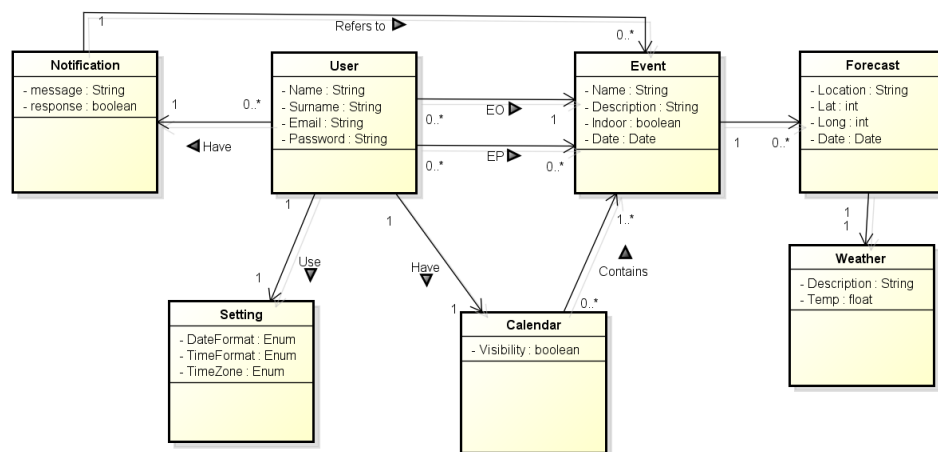
Figure 26: Class Diagram <MeteoCal>

3.4 Performance requirements

We assume the response time of the system is close to zero so the performance are essentially bounded by the Customer's Internet connection and interaction.

3.5 Design constraints

The software will be developed following the Java Enterprise Edition characteristics.

3.6 Software system attributes

3.6.1 Reliability

The system to keep safe from failure the User data will provide the replication of the DB like for example a RAID0 or a RAID10 configuration and suggested to external backup to minimize the data loss in case of failure.

3.6.2 Availability

The proposed system will be active all the time. The server in which the system will be installed must allow ordinary maintenance operation without reduce the availability of the system. Anyway the system isn't critical so unavailability for rather short time is acceptable.

3.6.3 Security

The system implements authentication of the Users to protect personal information (Calendar,Event . . .) with email and password or by the authorization with the Social Login. The User's passwords are saved using a hash function with salt and key stretching (PBKDF2). Also the system because permit the user input, that will be used in queries on the DB or in other User's pages, will be necessary an opportune filtering/escaping mechanism to prevent security vulnerability like SQL Injection and Cross-Site Scripting (XSS).

3.6.4 Maintainability

The system doesn't need complex maintainability if not adding new secondary memory space for the new Users if the space capacity is low.

3.6.5 Portability

The proposed system will be installed in any OS with Java Virtual Machine (JVM).

4 Appendix

4.1 Alloy

```
/*---------------------------------------------------------------------------------------*/
/*                                     # IMPORTS #                                       */
/*                                                                                        
*/
/*---------------------------------------------------------------------------------------*/
open util/boolean


/*---------------------------------------------------------------------------------------*/
/*                                   # NEW DATA TYPES #                                   */
/*                                                                                        
*/
/*---------------------------------------------------------------------------------------*/

enum Gender {MALE, FEMALE}
enum InvitationStatus {PENDING,ACCEPTED,DECLINED}
enum Visibility {PUBLIC, PRIVATE}
enum EventType {OUTDOOR, INDOOR}

sig Text{
        /* Used for comparing emails */
        value: Int
}


/*---------------------------------------------------------------------------------------*/
/*                                     # ENTITIES #                                      */
/*                                                                                        
*/
/*---------------------------------------------------------------------------------------*/


/* Entity used to model an event */
sig Event{

        id: one Int,
        eo: one User, // Event Organizer
        name: one Text,
        description: lone Text,
        location: one Text,
        date:  one Int,
        outdoor: one EventType,
        visibility: one Visibility,

        duration: lone Int,
        forecast: lone Forecast,

        /* List of Participating Users */
        lEPs: set User,
        /* List of Invited to Users */
        lINVs: set User,

}{

}
// EVENT FACTS
fact eventFact {

        // There aren't two events with the same ID
        no disj e1:Event, e2:Event • e1≠e2 and e1.id = e2.id

        // There aren't identical events with different IDs
        no disj e1, e2: Event • e1 ≠ e2 and e1.id  ≠ e2.id and e1.eo = e2.eo and e1.name = e2.name and e1.for
e2.forecast and e1.date = e2.date and e1.visibility  = e2.visibility
```

```
        // EO isn't in the set of EPs
        no e1: Event ● e1.eo in e1.lEPs

        // EO isn't in the set of lINVs
        no e1: Event ● e1.eo in e1.lINVs

        // No event whitout EO
        ∀ e: Event ● one u: User ● e.eo = u

        // No event whitout Calendar
        ∀ e: Event ● one c: Calendar ● e in c.lOEs

        // ∀ participated event in EPs list calendar
        ∀ e: Event ● ∀ u: User ● one en: EventNotification ●  u in e.lEPs iff (e in u.calendar.lPEs and en.ev
e and en.status = ACCEPTED and en in u.lNTs)

        // ∀ organized event in EOs list calendar
        ∀ e: Event ● ∀ u: User ● u = e.eo iff e in u.calendar.lOEs

        // No user at the same time in lEPs and lINVs
        ∀ e: Event ● no u: User ● u in e.lEPs  and u in e.lINVs

        // No user at the same time EO and in lINVs or in lEPs
        ∀ e: Event ● no u: User ● u = e.eo and ( u in e.lINVs or u in e.lEPs )

        // Forecast information of the same location of the event
        ∀ e: Event ● e.forecast.location = e.location

        // Forecast information of the same date of the event
        ∀ e:Event ● e.forecast.date = e.date

}

/* The user calendar.
 * This entity models a calendar as two set of different kind of events.
 */
sig Calendar{

        /* List of Organized Events */
        lOEs: set Event,

        /* List of Participating Events */
        lPEs: set Event,

        /* Calendar visibility */
        visibility: one Visibility,

}{

}
// CALENDAR FACTS
fact calendarFacts{
        // No calendars whitout users
        ∀ c: Calendar ● one u:User ● c = u.calendar

        // No event at the same time in lOE and lPE
        ∀ c: Calendar ● no e: Event ● e in c.lOEs and e in c.lPEs
}


/*User entity */
sig User {

        /* Mandatory user information */
```

```
            id: one Int,
            name: one Text,
            surname: one Text,
            email: one Text,
            password: one Text,

            /* User non mandatory information */

            gender: lone Gender,
            dateBirth: lone Int,
            avatar: lone Text,

            /* User lists and settings */

            /* User calendar */
            calendar: one Calendar,
            /* User personal system settings */
            setting: one Setting,

            /* List of Prefered Calendar */
            lPRs: set Calendar,
            /* List of Notification */
            lNTs: set Notification,

}{

}
// USER FACTS
fact userFact{

            // There aren't two users whit the same ID
            no disj u1, u2: User ● u1≠u2 and u1.id = u2.id

            // There aren't two users whit the same Email
            no disj u1, u2: User ● u1≠u2 and u1.email = u2.email

            // The user is EO of ∀ the events in its own calendar.lOEs
            ∀ u: User ● ∀ e: Event ● e in u.calendar.lOEs iff u = e.eo

            // The user is EP of ∀ the events in its own calendar.lPEs
            ∀ u: User ● ∀ e: Event ● e in u.calendar.lPEs iff u in e.lEPs

            // All calendars in prefered are public
            ∀ u: User ● ∀ c: Calendar ● c in u.lPRs iff c.visibility = PUBLIC

            // Users cannot bookmark their own calendars
            no u: User ● u.calendar in u.lPRs

            // Users cannot auto invites owned event
            no u:User ● ∀ en:EventNotification ● en in u.lNTs and en.event in u.calendar.lOEs

}


/* User setting */
sig Setting{

            dateFormat: one Int,
            timeFormat: one Int,
            timeZone:  one Int,

}{

}
// SETTINGS FACTS
fact settingFact{
```

```
            // No setting whitout users
            ∀ s:Setting ● one u:User ● s = u.setting

}


/* Notification entity
 *
 * A notification is a message sent to a user from the system about events or other sources of
 * user related information
 */
sig Notification{

            id: one Int,
            /* This is the text contained by the notification */
            msg: one Text,

}{


}
// NOTIFICATION FACTS
fact notificationFact{

            // There aren't two notifications with the same ID
            no disj n1:Notification, n2:Notification ● n1 ≠ n2 and n1.id = n2.id

            // No notifications whitout users
            ∀ n: Notification ● some u: User ● n in u.lNTs

}


/* Entity used to model event-related notification, for example invitation or
 * rischeduling
 */
sig EventNotification extends Notification{

            event: one Event,
            status: one InvitationStatus,

}{


}
// EVENT NOTIFICATIONS FACTS
fact EventNotification{

            // No event notification without event
            ∀ en: EventNotification ● one e: Event ● en.event = e

            // There aren't two Enotifications with the same ID
            no disj en1: EventNotification, en2: EventNotification ● en1 ≠ en2 and en1.id = en2.id

            // There aren't two event notifications for the same event to the same user with the same text
            ∀ u: User ● no disj en1, en2: EventNotification ● en1 ≠ en2 and en1.event = en2.event and ( (en1 in u

            // User is invited iff he has event notification about the event
            //∀ u: User ● ∀ e: Event ● one en: EventNotification ● (u in e.lINVs) iff ( en in u.lNTs and en.event
e )

            // The user is participating iff the eventNotification is accepted
            //∀ u: User ● ∀ e: Event ● one en: EventNotification ● (u in e.lEPs) iff (en in u.lNTs and en.event =
e and en.status = ACCEPTED)


}
```

```
/* Entity used to model weather forecast for outdoors events */
sig Forecast{

        /* A topografic reference, usu∀y a city name */
        location: one Text,

        /* GPS coordinate of the given forecast */
        lat:  one Int,
        long: one Int,

        /* Date for the forecast */
        date:  one Int,

        /* Date when the forecast was given by the web service API */
        creationDate: one Int,

        /* Weather condition for the forecast */
        weather: one Weather,

}{

        /* No forecast for the past */
        date ≥ creationDate

}
// FORECAST FACTS
fact forecastFact{

        // There aren't two forecasts whit the same location in the same date
        no disj f1:Forecast, f2:Forecast ● f1 ≠ f2 and f1.location = f2.location and f1.date =
f2.date

        // There aren't two forecasts whit the same lat-long in the same date
        no disj f1:Forecast, f2:Forecast ● f1 ≠ f2 and f1.lat = f2.lat and f1.long = f2.long and f1.date =
f2.date

        // No forecast without at least one event
        ∀ f: Forecast ● some e: Event ● f = e.forecast
}


/* Entity used to model a weather condition
 * NOTE: this is entity will probably have many  more fields
 */
sig Weather{

        weatherConditionCode: one Int,
        description: one Text,
        temp: one Int,

}{

}
// WEATHER FACTS
fact weatherFact{

        // No weather whitout forecasts
        ∀ w: Weather ● one f: Forecast ● w = f.weather

}



/*------------------------------------------------------------------------------------------*/
/*                                        # PREDICATES #
*/
```

```
/*---------------------------------------------------------------------------*/

pred addEventToEP(u, u': User, e: Event){

            u' in e.lEPs and

            (u'.calendar.lPEs = u.calendar.lPEs + e) and

            e in u'.calendar.lPEs

}

run addEventToEP

/*-------------------

pred addEventToEO(u, u': User, e: Event){

            u' = e.eo and

            (u'.calendar.lOEs = u.calendar.lOEs + e) and

            e in u'.calendar.lOEs

}

run addEventToEO

/*-------------------

pred deleteEvent(u, u': User, e: Event){

            u = e.eo and

            (u'.calendar.lOEs = u.calendar.lOEs - e) and

    //      ∀ u2,u3:User • u2 in e.lEPs and (u3.calendar.lPEs = u2.calendar.lPEs - e) and e not in u3.cal
and

            e not in u'.calendar.lOEs

}

run deleteEvent

/*-------------------

pred acceptPartecipation(u,u': User, e:Event){

        u in e.lEPs and

        (u'.calendar.lPEs = u.calendar.lPEs + e) and

        e in u'.calendar.lPEs

}

run acceptPartecipation

/*-------------------

pred declinePartecipation(u: User, e,e':Event){

        u in e.lEPs and

        (e'.lEPs = e.lEPs - u) and
```

67

```
                e' not in u.calendar.lPEs

}

run declinePartecipation
```
                                                                              /*-------------------
```
pred cancelPartecipation(u,u': User, e:Event){

        u in e.lEPs and

        (u'.calendar.lPEs = u.calendar.lPEs - e) and

        e not in u'.calendar.lPEs

}

run cancelPartecipation
```
                                                                              /*-------------------
```
pred sendEventNotification (u,u':User, e:Event){

        (u in e.lEPs or u= e.eo) and

        {one n:EventNotification ● n.event = e and (u'.lNTs = u.lNTs + n) }

}

run sendEventNotification
```
                                                                              /*-------------------
```
pred rescheduleEventByEO(e:Event, n:Notification){

                n.event = e and

                {∀ u:User ● u in e.lEPs implies {some u':User ● u'.lNTs = u.lNTs + n }}

}

run rescheduleEventByEO



/*------------------------------------------------------------------------------------------*/
/*                                       # ASSERTIONS #
*/
/*------------------------------------------------------------------------------------------*/

// event only in eo calendar and eps calendar
assert eventConsistency{

        ∀ e:Event ● ∀ u:User  ● u in e.lEPs iff e in u.calendar.lPEs
        ∀ e:Event ● ∀ u:User  ● u = e.eo iff e in u.calendar.lOEs

}

check eventConsistency
```
                                                                              /*-------------------
```
// no calendar private in prefered calendar user
assert noPrivateCalendarPreferred{
```

```
        ∀ u:User ● no c:Calendar ● c in u.lPRs and c.visibility = PRIVATE

}

check noPrivateCalendarPreferred

                                                                    /*--------------------

// ∀ u in e.lPEs must have e in own u.calendar.lPEs
assert checkEPsListPartecipatedEvents{

        ∀ e:Event ● ∀ u:User ● u in e.lEPs iff e in u.calendar.lPEs

}

check checkEPsListPartecipatedEvents

                                                                    /*--------------------

// check that if a user declined an invite the event isn't in the partecipatedevents
assert noPartecipatedEventIfDeclinedOrPending{

        ∀ u:User ● ∀ en:EventNotification ● en in u.lNTs and (en.status = DECLINED or en.status =
PENDING) implies en.event not in u.calendar.lPEs

}

check noPartecipatedEventIfDeclinedOrPending


pred show{
#User > 3
#Event > 3
#Event.lEPs > 2
#Event.lINVs > 2
#Notification > 4
#EventNotification > 0
#Forecast > 3
}

run show for 10
```



```
13 commands were executed. The results are:
    #1: Instance found. addEventToEP is consistent.
    #2: Instance found. addEventToEO is consistent.
    #3: Instance found. deleteEvent is consistent.
    #4: Instance found. acceptPartecipation is consistent.
    #5: Instance found. declinePartecipation is consistent.
    #6: Instance found. cancelPartecipation is consistent.
    #7: Instance found. sendEventNotification is consistent.
    #8: Instance found. rescheduleEventByEO is consistent.
    #9: No counterexample found. eventConsistency may be valid.
    #10: No counterexample found. noPrivateCalendarPreferred may be valid.
    #11: No counterexample found. checkEPsListPartecipatedEvents may be valid.
    #12: No counterexample found. noPartecipatedEventIfDeclinedOrPending may be valid.
    #13: Instance found. show is consistent.
```
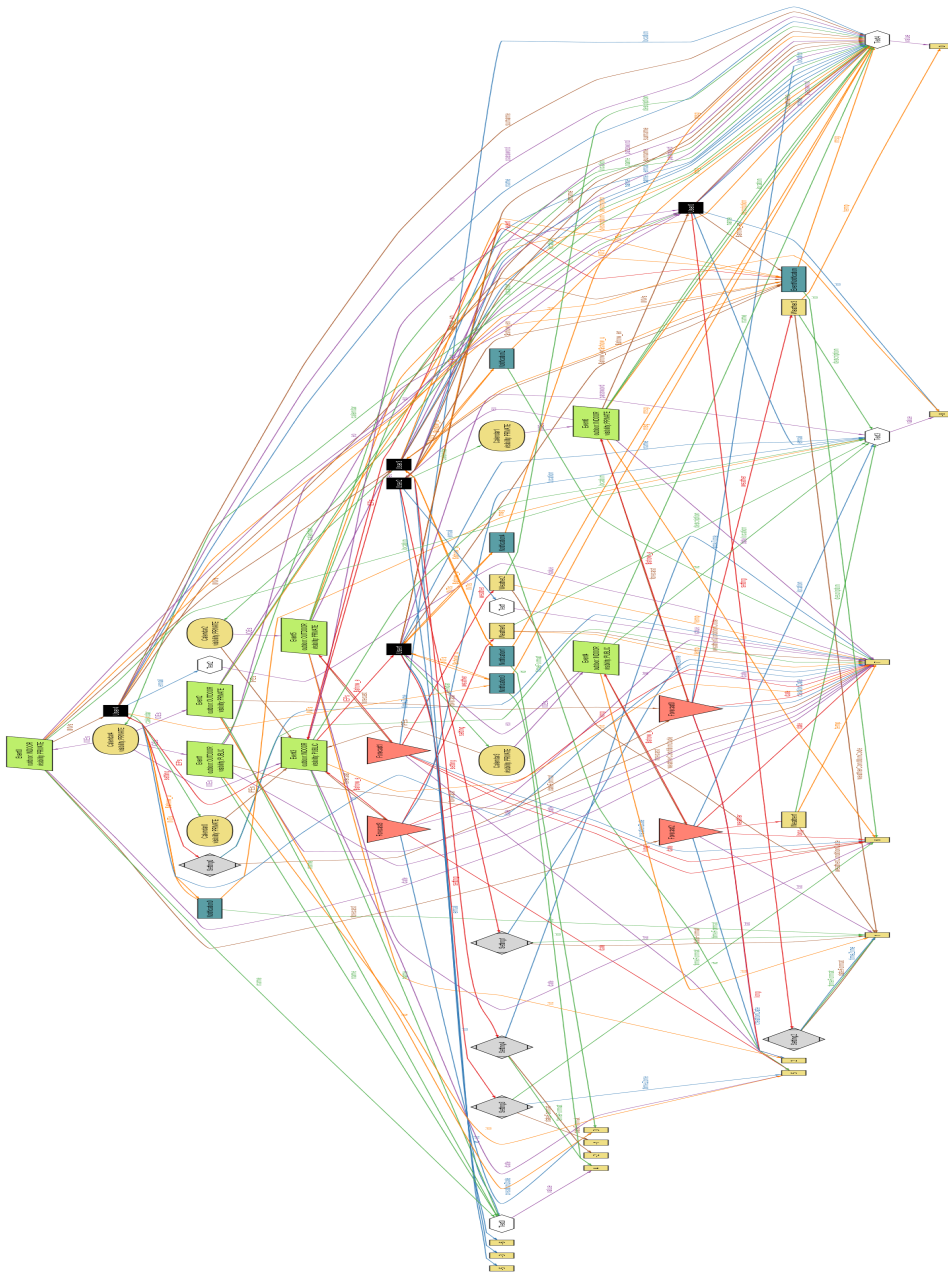
Figure 27: Result of Alloy Analizer

Figure 28: The Alloy World