Politecnico di Milano

MeteoCal

Requirements Analysis and Specifications Document

Authors:
DAVERIO Matteo
DE MARIA Valerio

November 16th 2014

TABLE OF CONTENT

1. Introduction

1.1. Description of the given problem

We will project MeteoCal which is a weather based online calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor activities.
Users, once registered, are able to create, delete and update events.
An event contains information about when and where the event will take place, whether the event will be indoor or outdoor. During event creation, any number of registered users can be invited. Only the organizer will be able to update or delete the event. Invited users can only accept or decline the invitation.
Whenever an event is saved, the system enriches the event with weather forecast information (if available). Moreover, it notifies all event participants one day before the event in case of bad weather conditions for outdoor events. Notifications are received by the users when they log into the system.

1.2. Glossary

First of all we have to define some words that will be used in our documents.

- **User:** for user we mean a person already registered in the system, so that has his own calendar with his own events.
- **Guest:** a guest is a person that probably for the first time accesses the system or that has not already signed up. Guest has limited access rights into the system than a user, his functionalities are reduced and can only see public calendars of the users.
- **Event:** every kind of activity that a user wants to do in a specific day, having a starting and ending time.

- **Invitation:** a request of participation sent from the creator of the event to another user .

1.3. Goals

We think that MeteoCal has to provide these main features:

- to register a new user into the system.
- to schedule the personal events of registered users.
- to provide (when it is possible) weather forecast to the event.
- to advertise and propose a re-scheduling in case of bad weather conditions for outdoor activities.
- to invite other registered users to organized events.
- to notify the interested users about event's changes.
- to make available public calendars and public events for consultation.

1.4. Domain properties

We suppose that these conditions hold in the analyzed world:

- Once invited, users can only accept or decline the invitation.
- The organizer of the event will take part in it.
- If a user accepts an event invitation then he will participate to it.
- An user has in is calendar the events that has created or in which participate.
- An user cannot participate to more than one event simultaneously.
- An user cannot have more than one invitation for a certain event.
- If user U participates to event E, U cannot have an invitation for E.
- Users can have only one calendar.
- The private events and calendars cannot be seen by other users.

1.5. Assumptions

We assume that:

- An user can have a private calendar with public event (supposing that in the future he can convert his calendar from private to public), in that case the guests and other users cannot see either public and private events.
- Guests can see public calendars.
- An user can make invitations for event “E” also after that he creates “E”.
- An user cannot invite himself.
- Users with private calendar cannot be searched.
- More than one event of a certain day can have the same time scheduling.
- Users can accept invitations of events that aren’t be done yet.

2. PROPOSED SYSTEM

2.1. Overview

We propose a web platform that allows a guest to register into the system or to find a public calendar and see its public events. A user can log in and use, in addition, all the other utilities of MeteoCal system: schedule events (create, update, delete), change privacy settings of his calendar and events, accept or decline invitations from other users events.
The system will add weather forecast to the events interacting with meteo service provider.

We will create a notification system in order to update users about invitations, bad weather conditions and other changings related to events which they have to take part in.

2.2. Actors

The actors of our system are:

- **Guest:** a person that has not registered and so can only see public calendars of the users.
- **User:** a person that has registered and so has his own calendar and events.
- **Meteo service provider:** an external system that can provide weather forecast.

3. REQUIREMENTS

3.1. Functional requirements

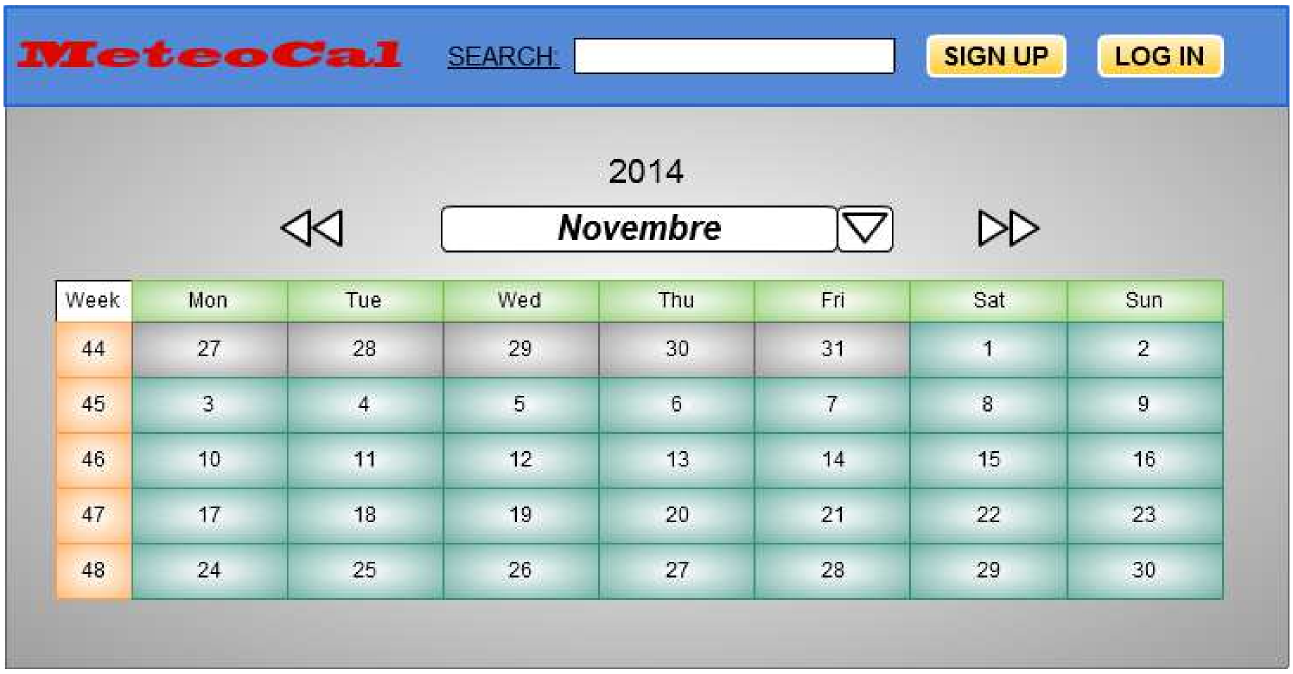
- Guest: he can
    - Sign up.
    - See public calendars.
- User: he can
    - Log in.
    - Create events.
    - Update or delete his events.
    - Invite other users to his events.
    - Accept or decline invitations from other users.
    - Receive notifications about changes in events where he has to take part.
    - Change privacy settings.

3.2. Non functional requirements

3.2.1. User interface

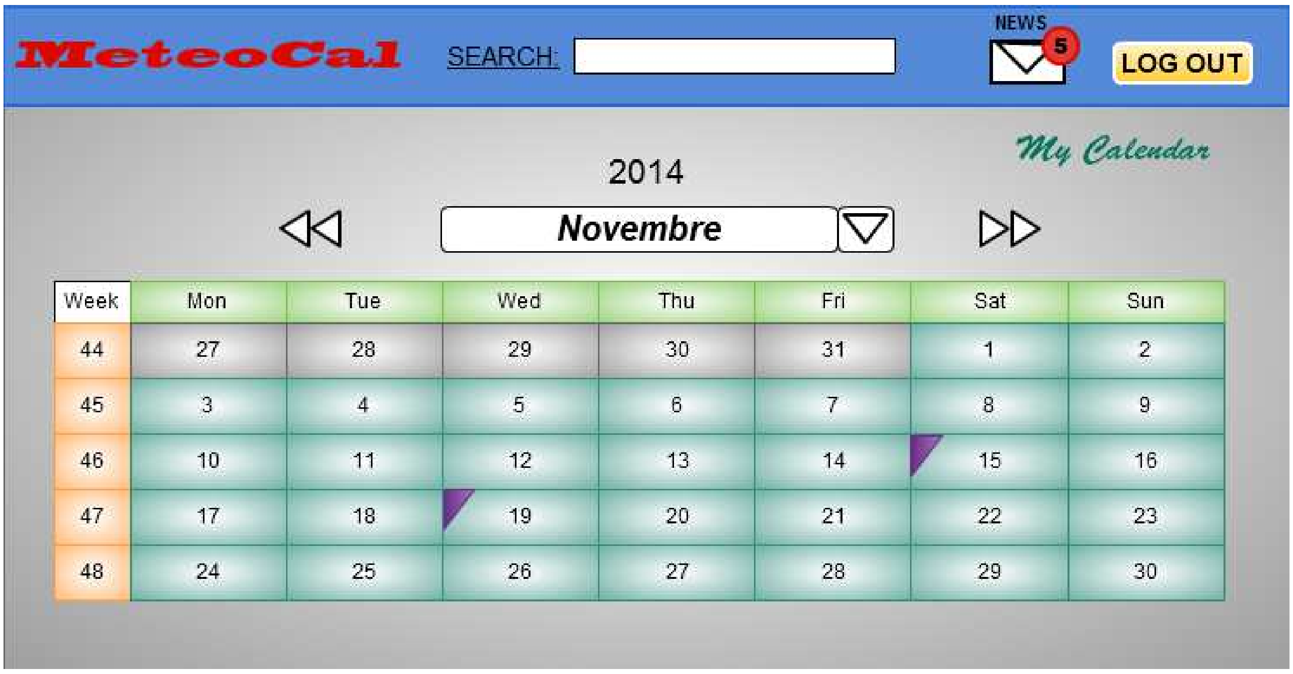
The interface of our application is thought to be used via web. We decided to have the same theme for all the pages of our system and also the same top screen bar (with MeteoCal's logo, search area and navigation buttons). This choice is done to make the system easy to use so that could be implemented also as a mobile application, this could be really useful for users that want to schedule their events while they are traveling to/from work.

The first page that appears once entered in MeteoCal system gives the possibility to sign up (in case of guest) or log in (in case of user). Guests can also search for public calendars and see them on this page. Initially the calendar placed in this page will be a general one set on the actual date.



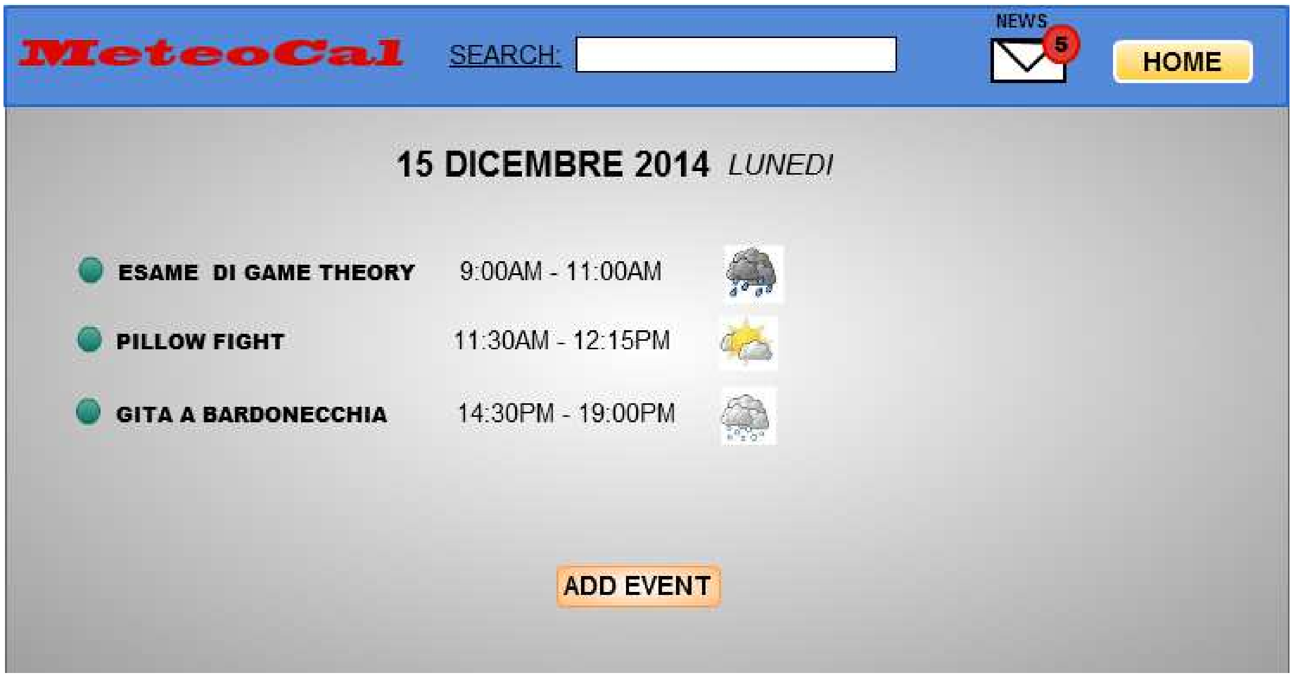
Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
44	27	28	29	30	31	1	2
45	3	4	5	6	7	8	9
46	10	11	12	13	14	15	16
47	17	18	19	20	21	22	23
48	24	25	26	27	28	29	30

Once logged into the system the profile page (let us call it "home") has an icon to click on to see the notifications. User can navigate his calendar and click on a day to see the events scheduled on it or create a new one. If there are events scheduled on a day a small triangle will be placed on that day.



Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
44	27	28	29	30	31	1	2
45	3	4	5	6	7	8	9
46	10	11	12	13	14	◤ 15	16
47	17	18	◤ 19	20	21	22	23
48	24	25	26	27	28	29	30

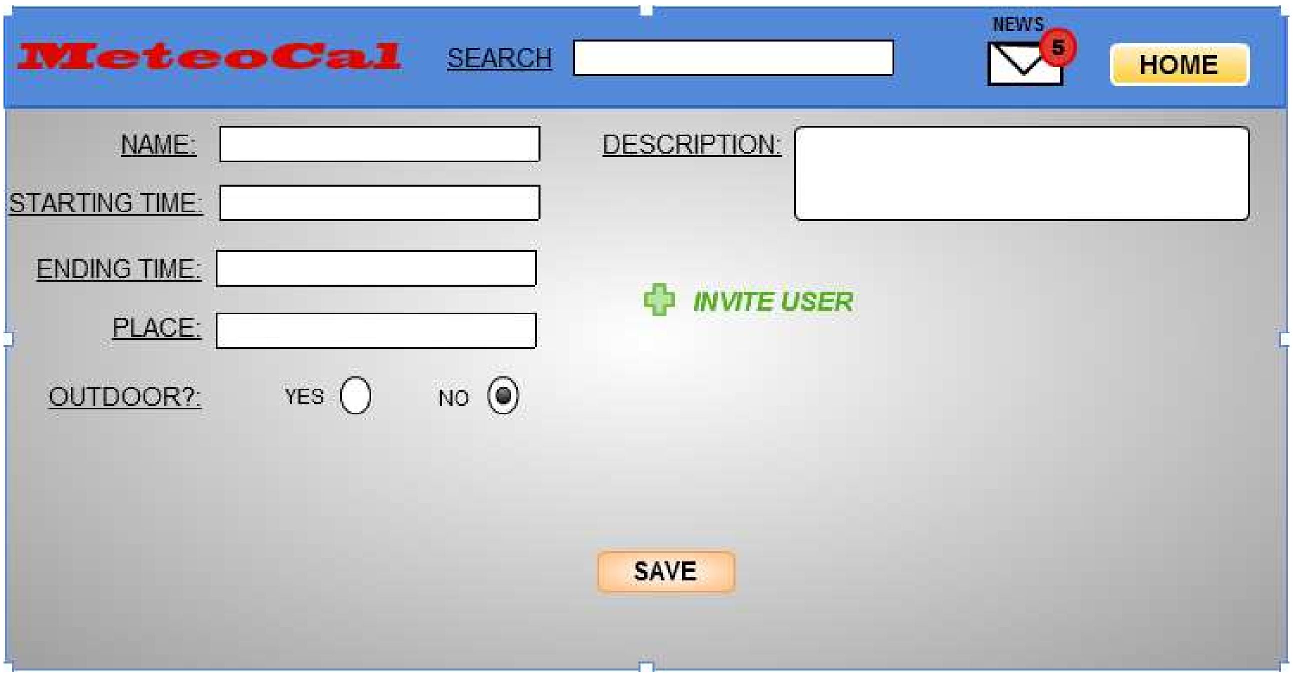
By clicking on a day user can see the events that he has on that day and also add another event.



15 DICEMBRE 2014 *LUNEDI*

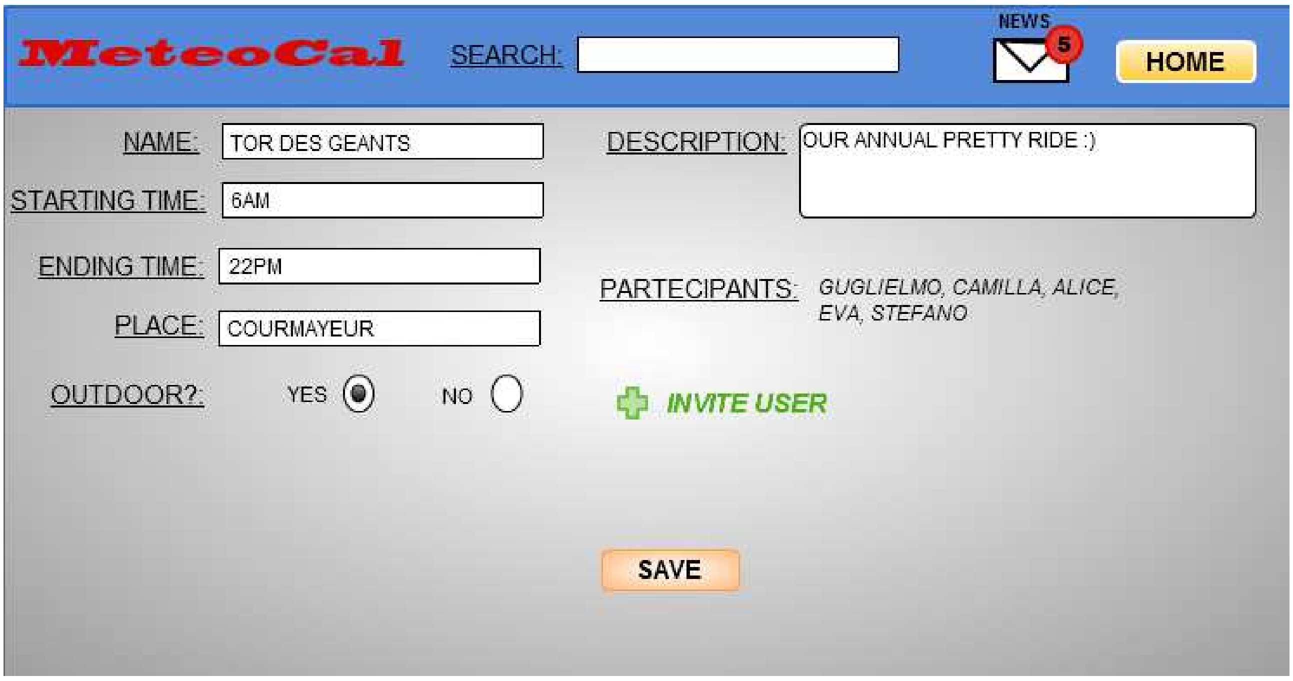
● ESAME DI GAME THEORY	9:00AM - 11:00AM
● PILLOW FIGHT	11:30AM - 12:15PM
● GITA A BARDONECCHIA	14:30PM - 19:00PM

The page for the event creation gives the possibility to set the attributes of the event, send invitations and then save it.



NAME:
STARTING TIME:
ENDING TIME:
PLACE:
OUTDOOR?: YES [✓ NO]
DESCRIPTION:

The page to update the event is the same of the page of event creation but already fulfilled with actual data and event's participants



NAME: TOR DES GEANTS

STARTING TIME: 6AM

ENDING TIME: 22PM

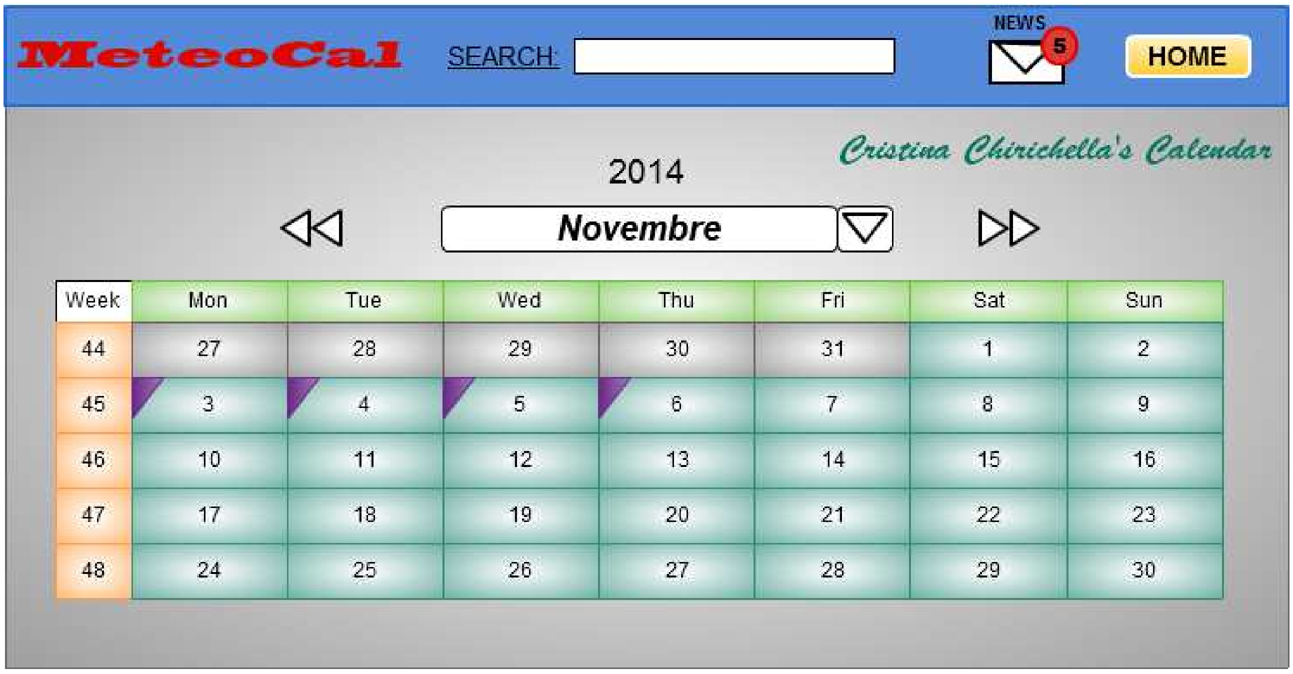
PLACE: COURMAYEUR

OUTDOOR?: [✓ YES] NO

DESCRIPTION: OUR ANNUAL PRETTY RIDE :)

PARTECIPANTS: GUGLIELMO, CAMILLA, ALICE, EVA, STEFANO

User can obviously also search for public calendars and so the calendar shown in the page will be replaced by the calendar of the other user. Clicking on the "Home" button user can return on his calendar.



Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
44	27	28	29	30	31	1	2
45	3	4	5	6	7	8	9
46	10	11	12	13	14	15	16
47	17	18	19	20	21	22	23
48	24	25	26	27	28	29	30

3.2.2. Documentation

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

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

3.2.3. System architecture

We will use J2EE platform with a database in which all information about users and their calendars will be stored.
We need a central server to manage client's requests so an Internet connection is needed to use MeteoCal and also a recent web browser.

4. Specifications

We list here some of the MeteoCal's specifications about how we can reach the goals listed above:

- An user can see public calendar and public event from the all world.
- Only a user can create an event.
- Event will be well structured, it means that event must have a starting time and an ending time, a day, and a description of the event.
- *Users can set the bad weather conditions related to the events that they create.*
- *An user can request to participate to a public event.*
- *Once received an invitation a user can let it pending and decide to participate or not on a later stage.*

It is important to say that the points in italic are advanced functionalities that we would like to implement to have a more complete system, but, before, we want to carry out all the basic set of functionalities in order to have a complete product.

5. Scenarios

- Leo wants to organize a football match with his friends, so he visits MeteoCal, watches his friends' calendars(which are public) and finds a day when all of them are free. He creates then an outdoor football match's event inviting his friends. A day before the event, he sees a notification on the website, informing him of bad weather for the football’s day, so he can change his organization in time finding an indoor football field.

- Luciano wants to organize the best barbecue of his life. He finds a day in his calendar on MeteoCal when he would like to do it, and creates the event. Three days before the barbecue, MeteoCal gives him a notification of bad weather, proposing him to change the event date. It proposes two days after. Luciano, who doesn’t want to ruin his barbecue, changes the day following the MeteoCal’s suggestion.
- Valerio is logged to the platform and looks if there is something new on his friends’ calendar. So he goes in the search bar and inserts the name of a friend, selects him, and sees his calendar if it is public. Doing it, he finds on Leo’s calendar a swimming event for tomorrow. Valerio is very interested, because in that time he is free, so he sends a participation request to his friend, because he wants to go swimming with him. Leo sees the request and accept it. After that, the event is added on Valerio’s calendar.
- Matteo has discovered that his professors use MeteoCal. Their calendar is public and is fulfilled with public events referring the classes in which they teach. So that even if Matteo is not a MeteoCal user can see the scheduling of classes that attends. In a raining Friday morning, Matteo visits MeteoCal and discovers that today is scheduled only a class in which he is not really interested in

so he decides to stay at home and study the topics of that lesson by himself in front of a cup of tea.

- Stefano logs into the system and receives a notification saying that Valerio invites him to participate on “Tor des Geants” challenge the 7th September 2015. Stefano thinks that is too hard for him and decides to let the invitation pending. Two days later he logs into the system again and sees that also Alice participates to the Valerio’s event so Stefano decides to accept the invitation, clicks the button “Accept” and the event is saved on his calendar.

6. UML MODELS

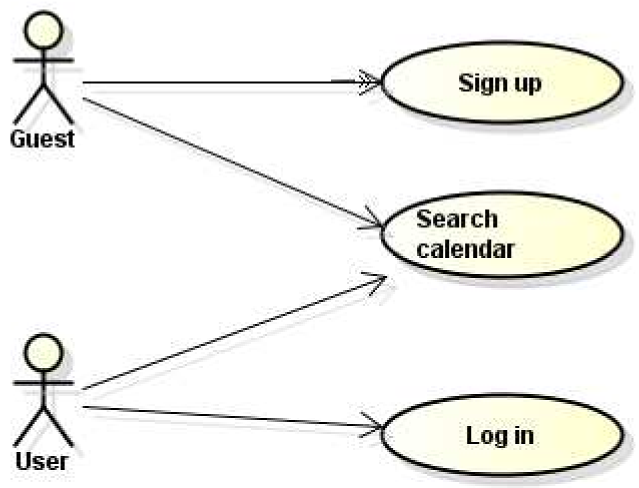
6.1. Use case diagram

Use cases:

- Sign up
- Log-In
- Search for a calendar
- Create event
- Update event
- Delete event
- Invite user
- Weather forecast
- Accept invitation
- Decline invitation

We decided to split the Use Case Diagram into smaller ones because we wanted to make the situation clearer.

All references to “pages”, “buttons” or “input forms” are only hypothesis to make the situation clearer and to help the reader to draw a visual picture in his mind of what we are talking about. Real pages and page structures will be well defined in the Design Document.



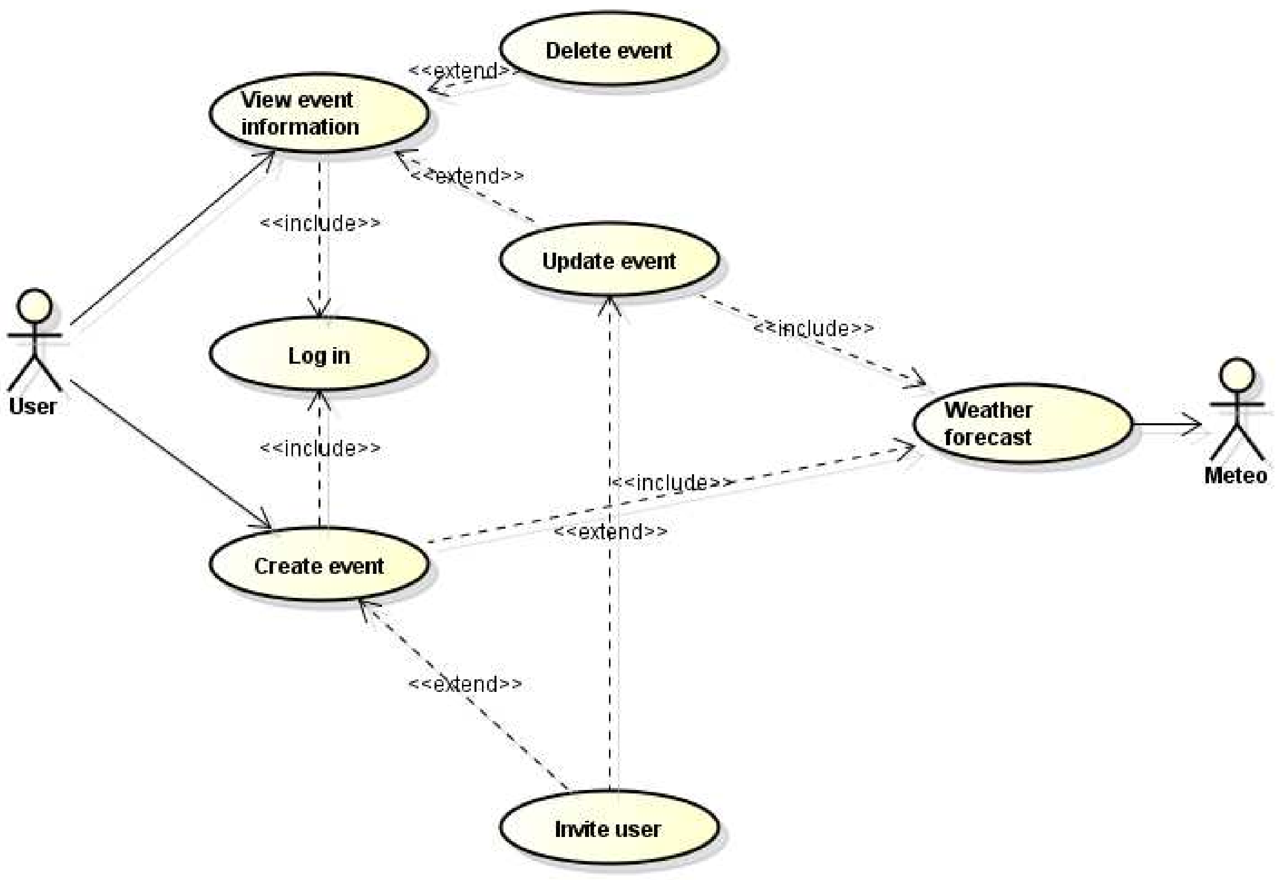
Name	**Log in.**
Actors	User.
Entry conditions	The user has successfully signed up to the system.
Flow of events	• The user opens the home page of the platform;

	• The system shows him the page; • The user enters his e-mail address and password in the input form provided; • The user clicks the button "log in"; • The system shows the profile page.
Exit conditions	There are no exit conditions.
Exceptions	E-mail address or password inserted are wrong. An error message is shown.

Name	**Sign up.**
Actors	Guest.
Entry conditions	The guest hasn't already signed up and he views MeteoCal's home page.
Flow of events	• The user clicks on the button

	"sign up"; • The system shows a page which contains an input form. The input form asks for some personal information and an e-mail address; • The guest fulfills the input form and clicks "sign up"; • The link redirects him to MeteoCal, the system confirms the registration.
Exit conditions	The information about the guest is stored into the database in a way to grant his log in. The log in is now possible.
Exceptions	The guest enters a not valid password, or doesn't fill some mandatory fields. In this case the page is reloaded showing an error message.

Name	**Search for a calendar.**
Actors	Guest or User.
Entry conditions	The guest is in the home page. The user is in the home page or in his profile page.
Flow of events	• The guest\user types the name of a user in a search field to have a quick search; • The system shows the results (name and e-mail).
Exit conditions	There are no exit conditions.
Exceptions	The research gives no results. The user\guest clicks “search” without entering any information.



Name	**Create event.**
Actors	User.
Entry conditions	The user has selected a day in which he wants to create the event.
Flow of events	• The user fulfill the input forms that are in the event creation

	page; • The user clicks the button "Save event"; • The system shows to the user the page of the day enriched with the event created.
Exit conditions	The event is stored into the database.
Exceptions	The starting or ending time is not a correct time (E.g. letters are written instead of numbers). One of the input forms is empty. In that case a message error is shown.

Name	**Update event .**
Actors	User.
Entry conditions	The user is in the page containing

	the information about the event that he has created.
Flow of events	• The user click the button "Update"; • The system shows the same page for the creation of an event already fulfilled with the event's data; • The user modify the data that wants to change and clicks the button "Save event"; • The system shows the page of the event with the updated data;
Exit conditions	The data related to the event changed are modified in the database. The update is done also in the users invited to the event. A notification is sent to users invited to the event.

Exceptions	The starting or ending time is not a correct time (E.g. letters are written instead of numbers). In that case an error message is shown.

Name	**Eliminate event .**
Actors	User.
Entry conditions	The user is in the page containing the information about the event that he has created.
Flow of events	• The user click the button "Eliminate"; • The system shows a confirmation window; • The user confirms the intention to eliminate the event; • The system shows the page of the day related to the event

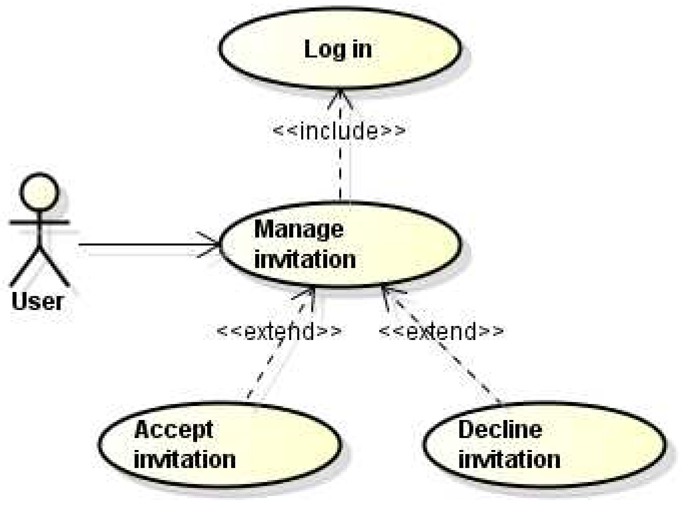
	deleted;
Exit conditions	The data related to the event are no more present in the database. The event is no more present in the calendar of user that has created it and in the calendar of his participants. A notification is sent to users invited to the event.
Exceptions	There are no exceptions.

Name	**Weather forecast.**
Actors	Meteo
Entry conditions	There are no entry conditions
Flow of events	• Meteo receives a request for a weather forecast from the system; • Meteo sends to the system the information required.
Exit conditions	Weather forecast are added into the database.

Exceptions	There are no weather forecast available.

Name	**Invite user.**
Actors	User.
Entry conditions	The user is in the page containing the information about the event that he has created.
Flow of events	• The user clicks the button "Invite"; • The system shows a page with an input form; • User types the name of the user that wants to invite; • The system lists users matching to the search input; • User clicks on the name of the user that wants to invite; • The system shows the event page update with the name of the invited person.

Exit conditions	The data related to the user invited to the event are saved in the database. A notification is sent to the user invited.
Exceptions	User invites himself.



Name	**Accept invitation.**
Actors	User.
Entry conditions	User is logged into the system.

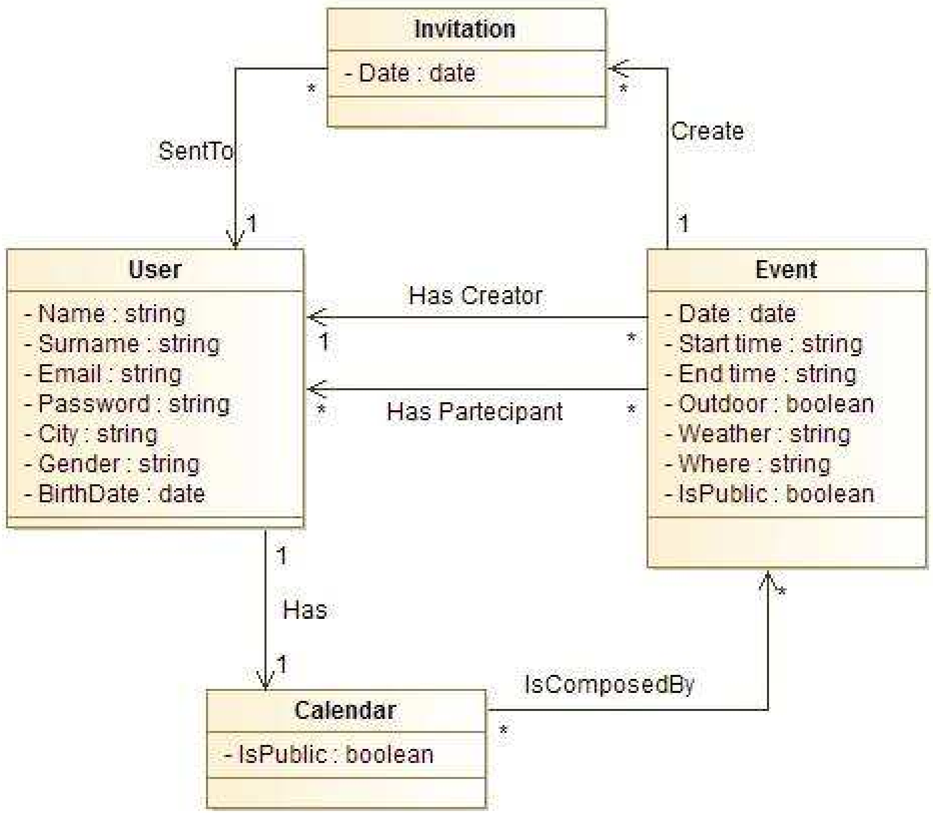
Flow of events	• System notifies the user about the invitation; • User clicks on the notification’s icon; • System shows to the user the information about the invitation and the buttons “Accept” and “Decline”; • User clicks the button “Accept”; • System shows to the user the previous page.
Exit conditions	Data about the event are saved into the database. A notification is sent to the user that has created the event.
Exceptions	There are no exceptions.

Name	**Decline invitation.**
Actors	User.
Entry conditions	User is logged into the system.

Flow of events	• System notifies the user about the invitation; • User clicks on the notification's icon; • System shows to the user the information about the invitation and the buttons "Accept" and "Decline"; • User clicks the button "Decline"; • System shows to the user the previous page.
Exit conditions	A notification is sent to the user that has created the event.
Exceptions	There are no exceptions.

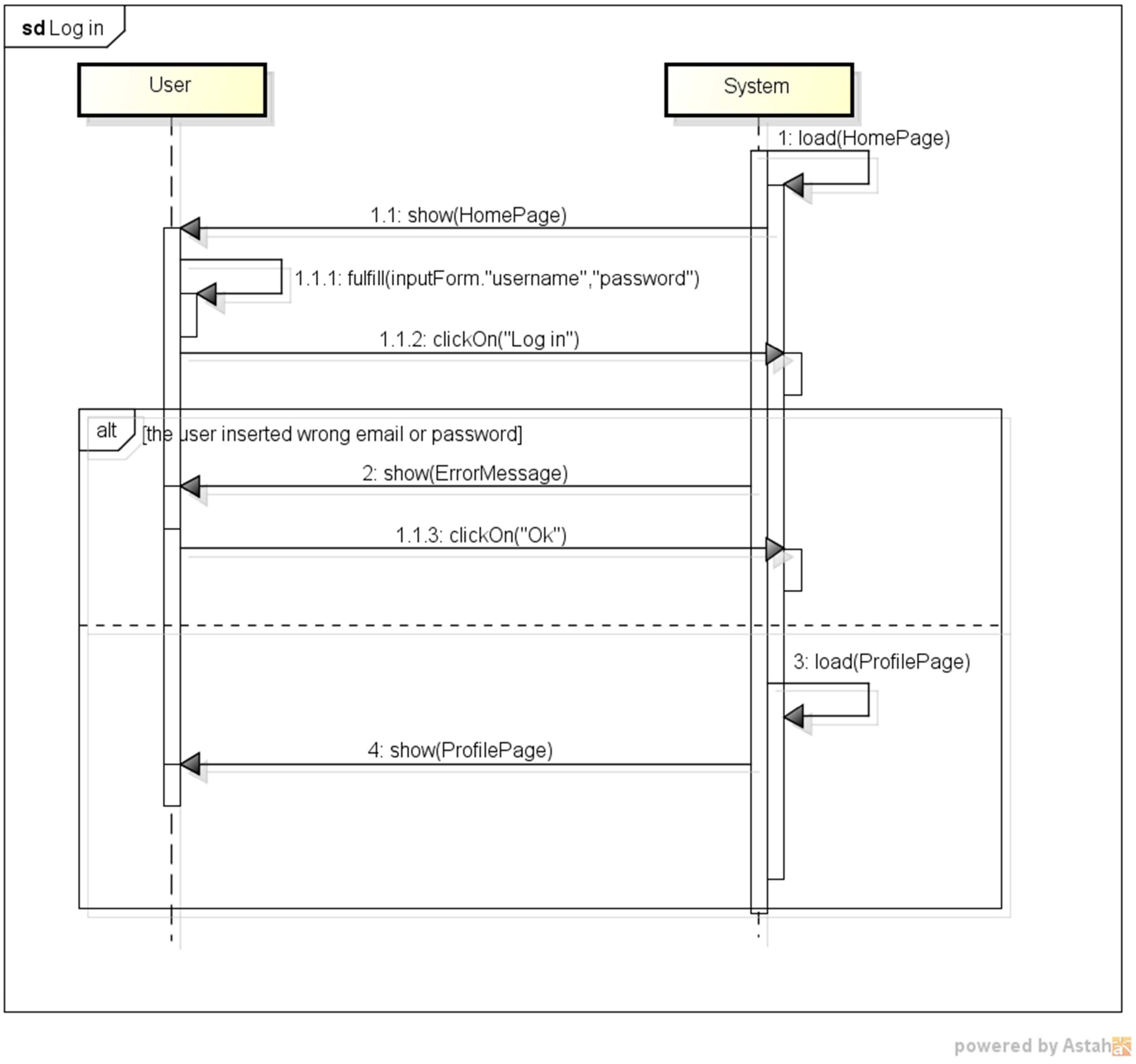
6.2. Class diagram

Now we can draw a class diagram of our system:

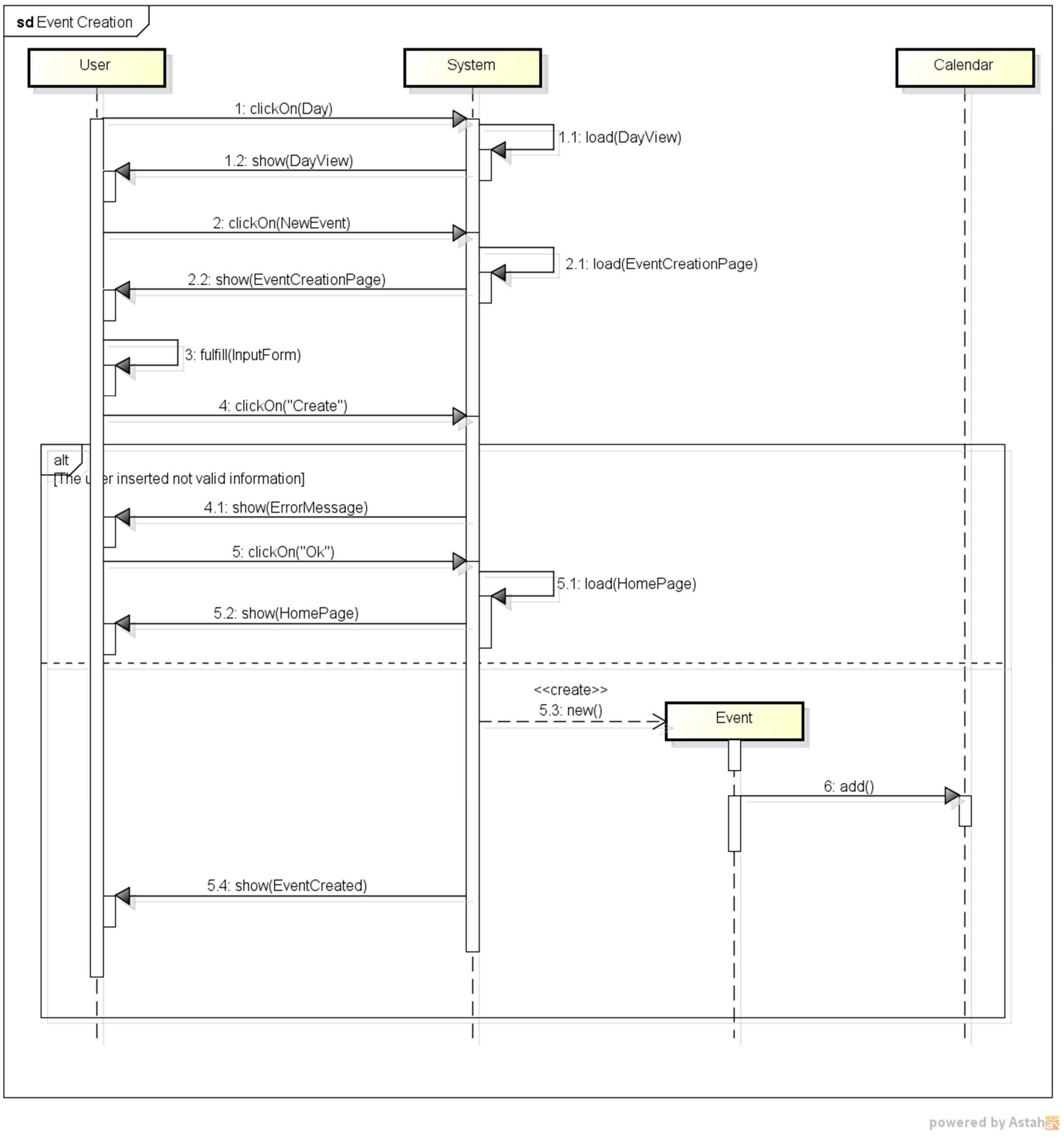


6.3. Sequence diagram

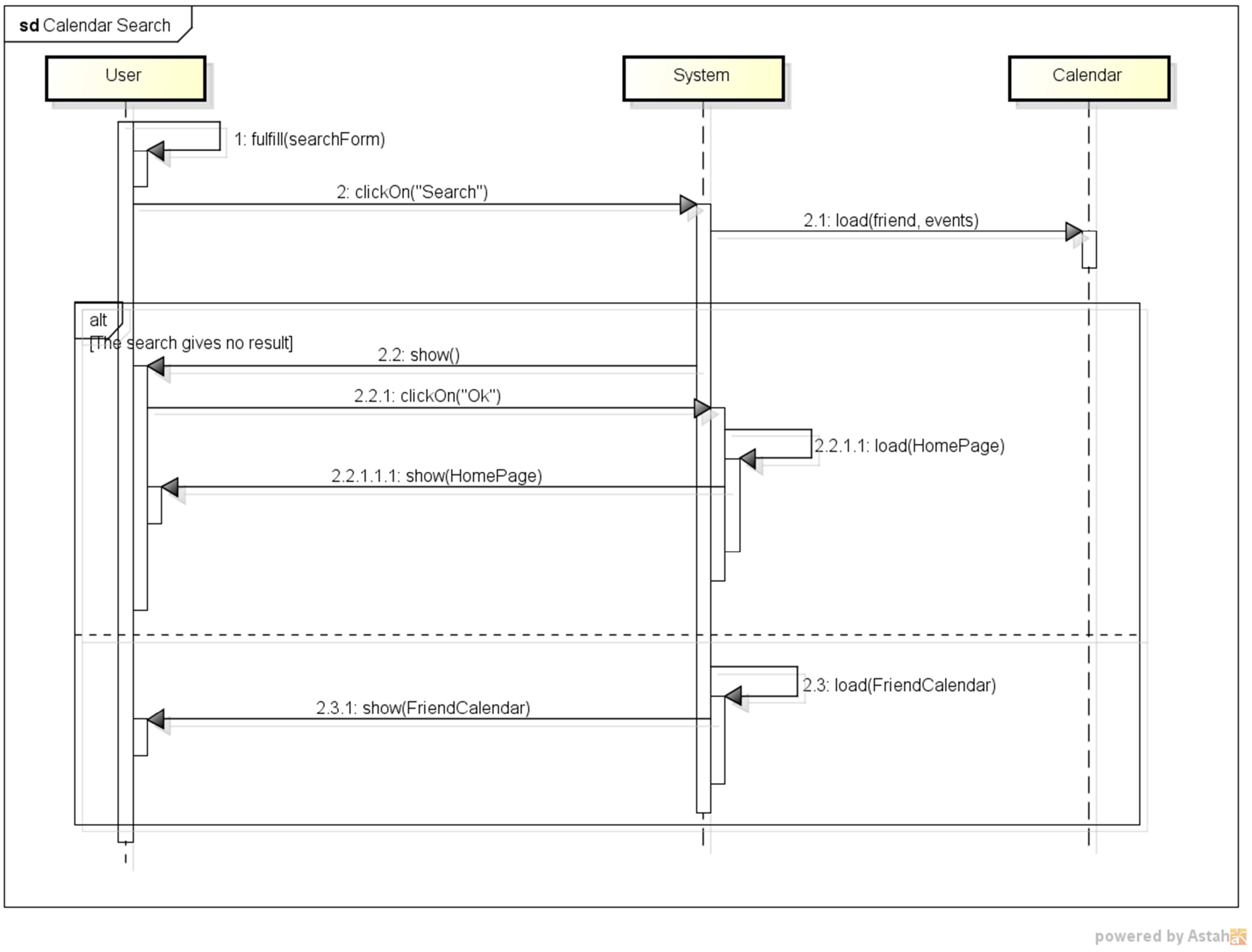
6.3.1. Log in



6.3.2. Event creation

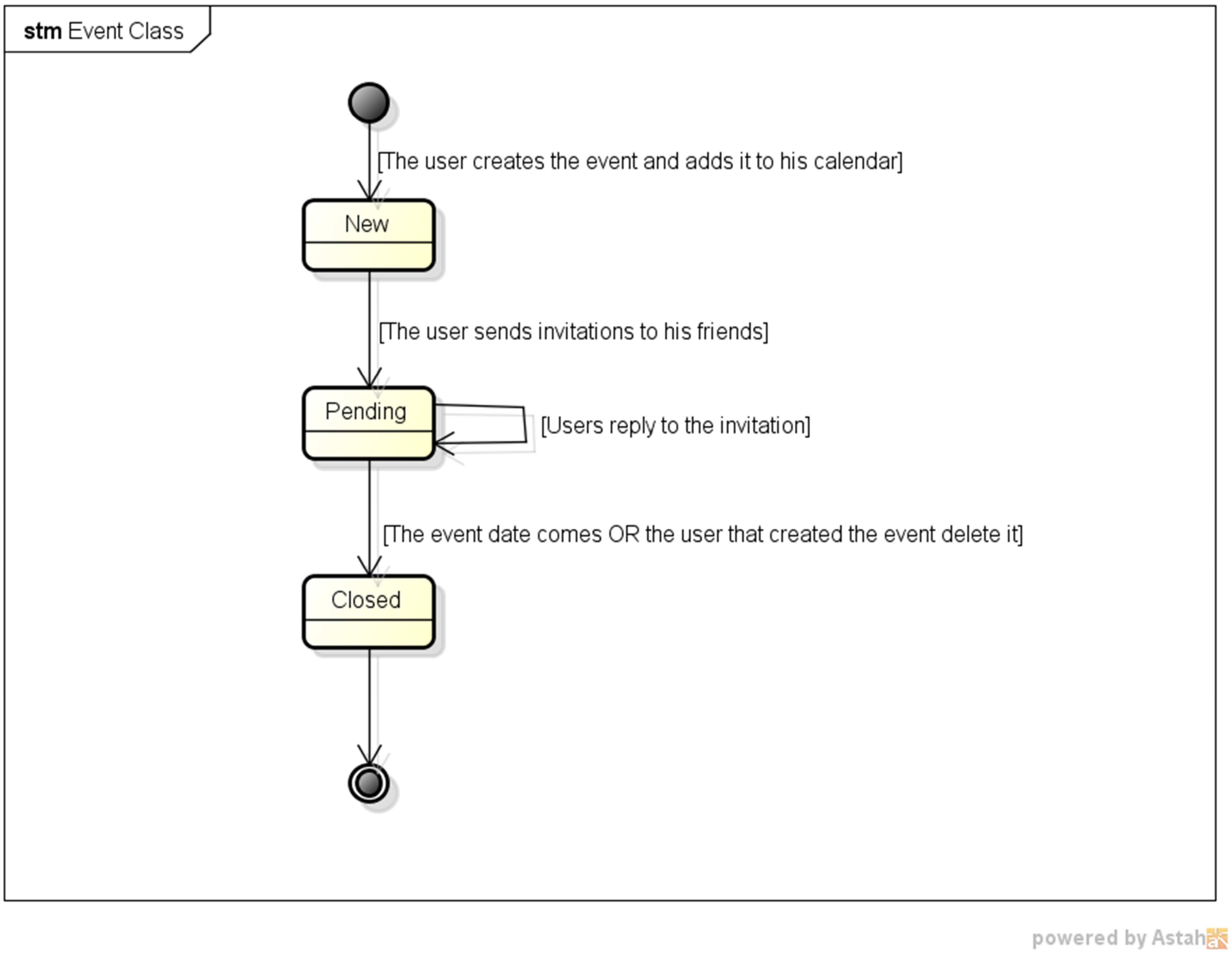


6.3.3. Calendar search

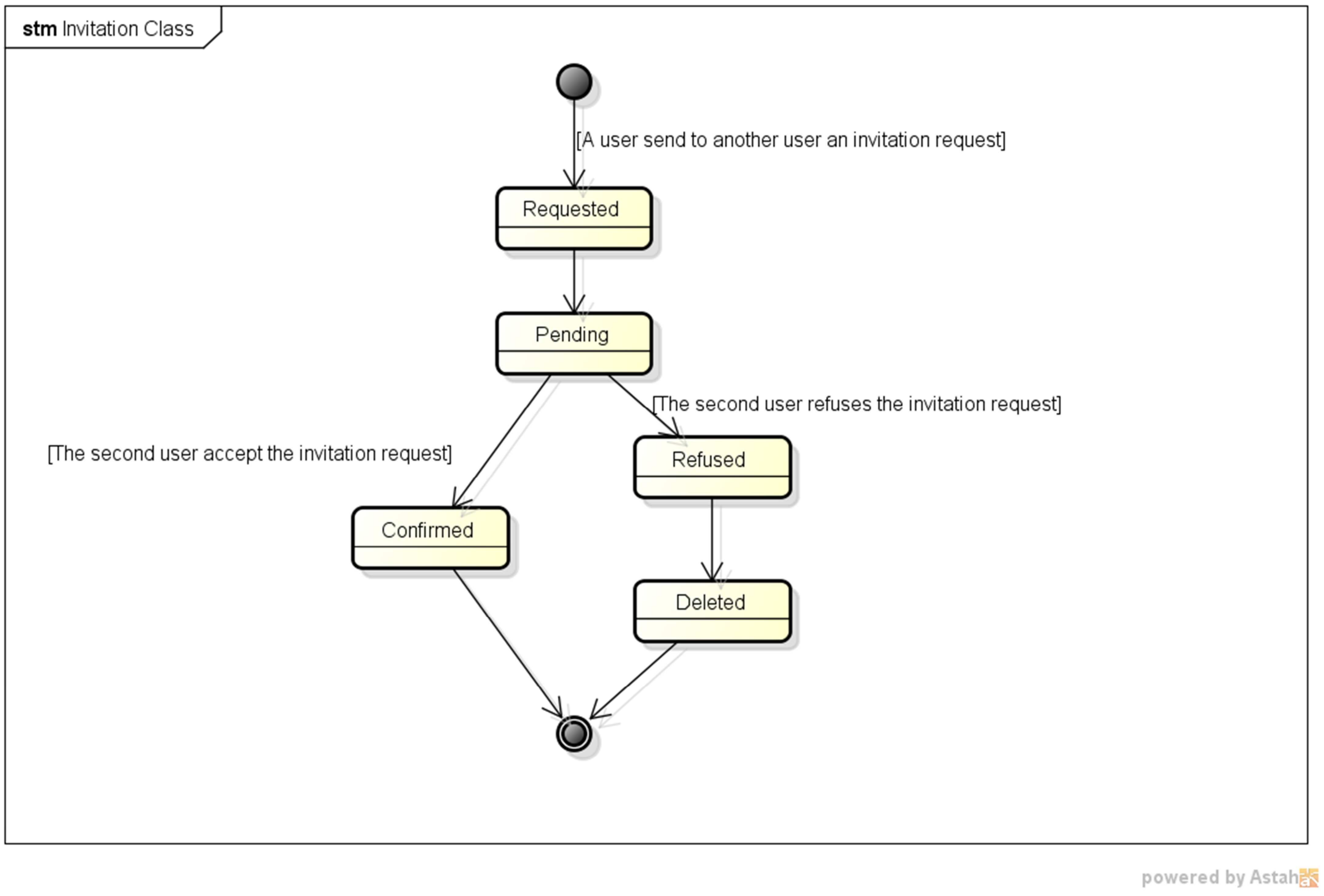


6.4. State chart diagram

6.4.1. Event class


stm Event Class
[The user creates the event and adds it to his calendar]
New
[The user sends invitations to his friends]
Pending
[Users reply to the invitation]
[The event date comes OR the user that created the event delete it]
Closed
powered by Astah

6.4.2. Invitation class


stm Invitation Class
[A user send to another user an invitation request]
Requested
Pending
[The second user accept the invitation request]
[The second user refuses the invitation request]
Confirmed
Refused
Deleted
powered by Astah

7. ALLOY MODELING

7.1. Model

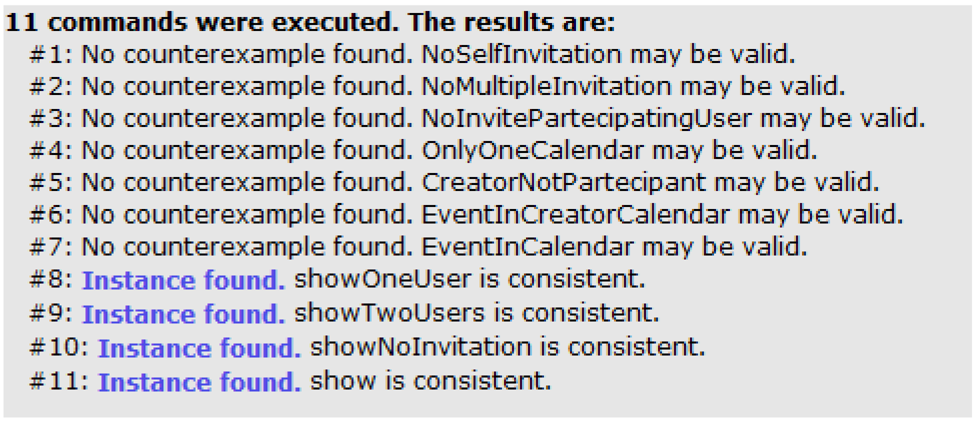
In this paragraph we try to understand if our class diagram can be consistent using Alloy Analyzer. We report below the code used.

```
module MeteoCal

//SIGNATURES

sig Calendar {
   events: set Event
}

sig User {
   hasCalendar: one Calendar
}

sig Event {
   hasCreator: one User,
   hasPartecipant: set User
}

sig Invitation {
   toEvent: one Event,
   sentTo: one User
}
```

```
//FACTS

fact invitationProperties {
    //A cannot send a invitation to himself
    no req: Invitation | req.sentTo=req.toEvent.hasCreator
    //no multiple request
    no disj req1,req2: Invitation | req1.toEvent=req2.toEvent && req1.sentTo=req2.sentTo
    //if A partecipate to event E, A cannot be invited again to the same event
    no req: Invitation | req.sentTo in req.toEvent.hasPartecipant
}

fact calendarProperties {
    //calendars are associated to exactly one user
    all cal: Calendar | one user: User | user.hasCalendar=cal
}

fact eventProperties {
    //if a user is the event creator, he must not compare in the event's partecipant
    no ev: Event | ev.hasCreator in ev.hasPartecipant
    //an event must compare in his creator's calendar
    all ev: Event | ev in ev.hasCreator.hasCalendar.events
}

fact userProperties {
    //if an event is in a user's calendar, the user have to be his creator or a partecipant, and viceversa
    all us: User | us.(~hasCreator+~hasPartecipant)=us.hasCalendar.events
}
```

```
//ASSERTIONS

assert NoSelfInvitation {
	no req: Invitation |  req.sentTo = req.toEvent.hasCreator
}
check NoSelfInvitation

assert NoMultipleInvitation {
	no disj req1,req2: Invitation | req1.toEvent=req2.toEvent && req1.sentTo=req2.sentTo
}
check NoMultipleInvitation

assert NoInvitePartecipatingUser {
	no req: Invitation |  req.sentTo in req.toEvent.hasPartecipant
}
check NoInvitePartecipatingUser

assert OnlyOneCalendar{
	all cal: Calendar | one user: User | user.hasCalendar=cal
}
check OnlyOneCalendar

assert CreatorNotPartecipant{
  no ev:Event| ev.hasCreator in ev.hasPartecipant
}
check CreatorNotPartecipant

assert EventInCreatorCalendar{
  all ev:Event| ev in ev.hasCreator.hasCalendar.events
}
check EventInCreatorCalendar

assert EventInCalendar{
  all us: User | us.(~hasCreator+~hasPartecipant)=us.hasCalendar.events
}
check EventInCalendar


//PREDICATES

pred showOneUser(){
}

run showOneUser for 5 but 1 User

pred showTwoUsers(){
}

run showTwoUsers for 5 but 2 User

pred showNoInvitation(){
}

run showNoInvitation for 5 but 0 Invitation

pred show() {
}

run show for 3
```

And here a report of Alloy Analyzer:



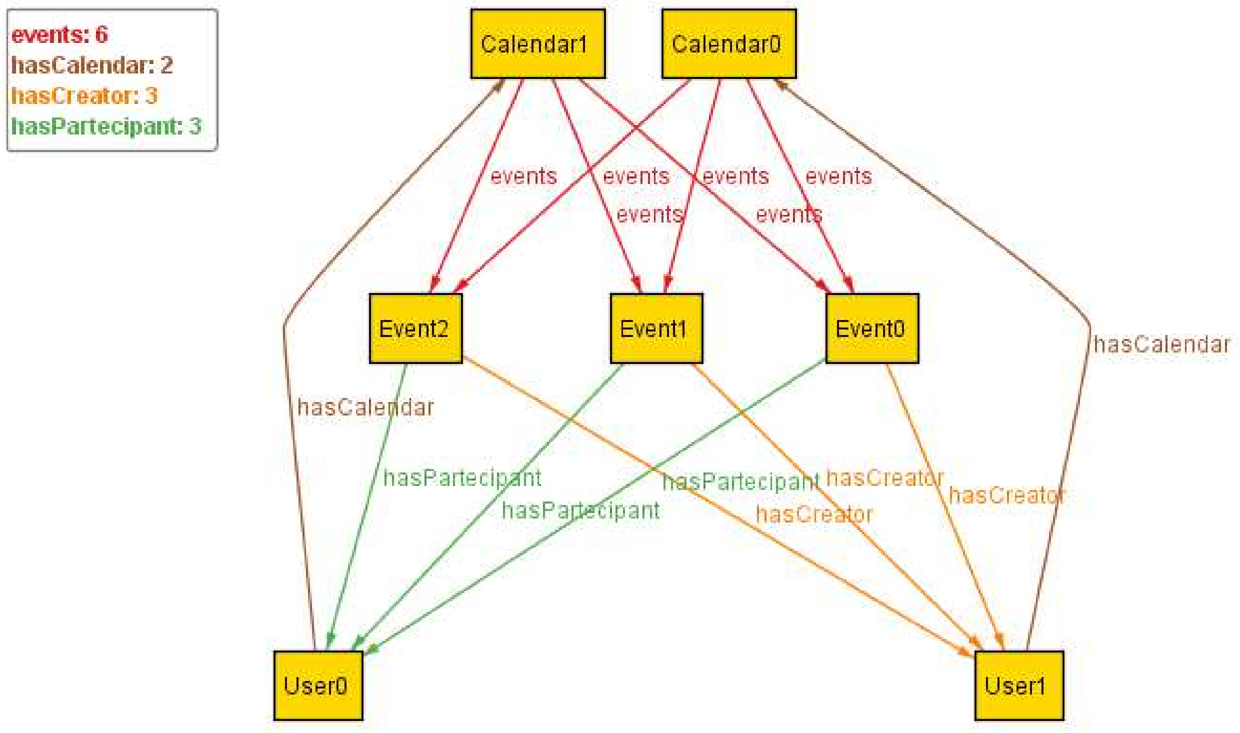
```
11 commands were executed. The results are:
  #1: No counterexample found. NoSelfInvitation may be valid.
  #2: No counterexample found. NoMultipleInvitation may be valid.
  #3: No counterexample found. NoInvitePartecipatingUser may be valid.
  #4: No counterexample found. OnlyOneCalendar may be valid.
  #5: No counterexample found. CreatorNotPartecipant may be valid.
  #6: No counterexample found. EventInCreatorCalendar may be valid.
  #7: No counterexample found. EventInCalendar may be valid.
  #8: Instance found. showOneUser is consistent.
  #9: Instance found. showTwoUsers is consistent.
  #10: Instance found. showNoInvitation is consistent.
  #11: Instance found. show is consistent.
```

7.2. Worlds generated

In this paragraph we report some worlds generated by Alloy Analyzer in order to make understand that our model is consistent.

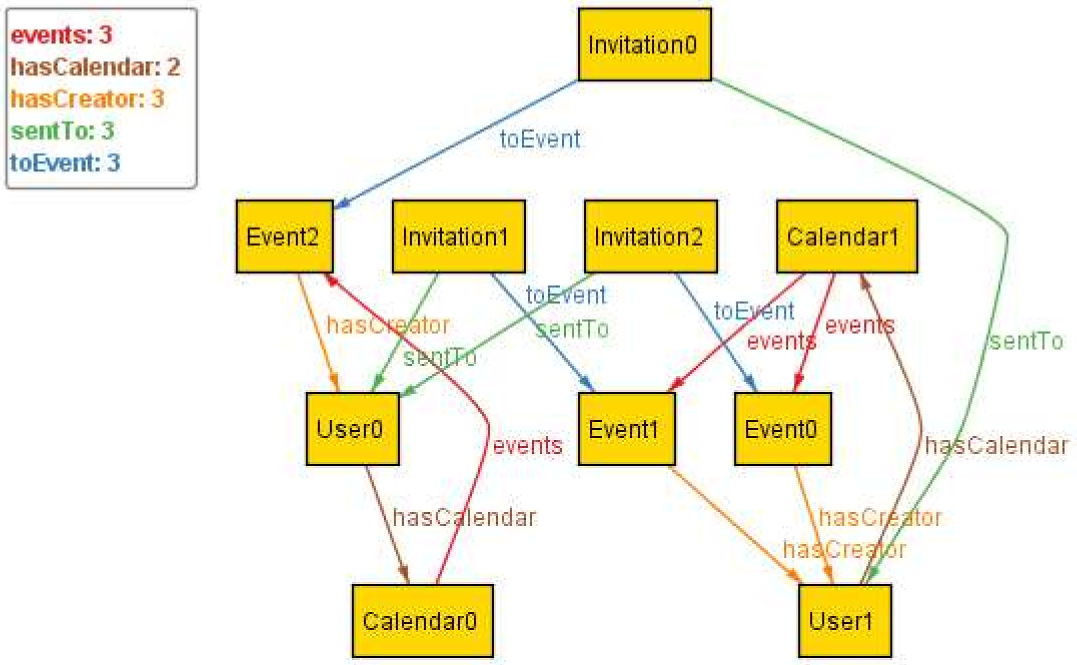
ALL INVITATIONS ACCEPTED

User1 creates three events and sends for each one an invitation to User0. User0 has already accepted, he is saved as “participant” in all the events and so events are saved in his calendar.

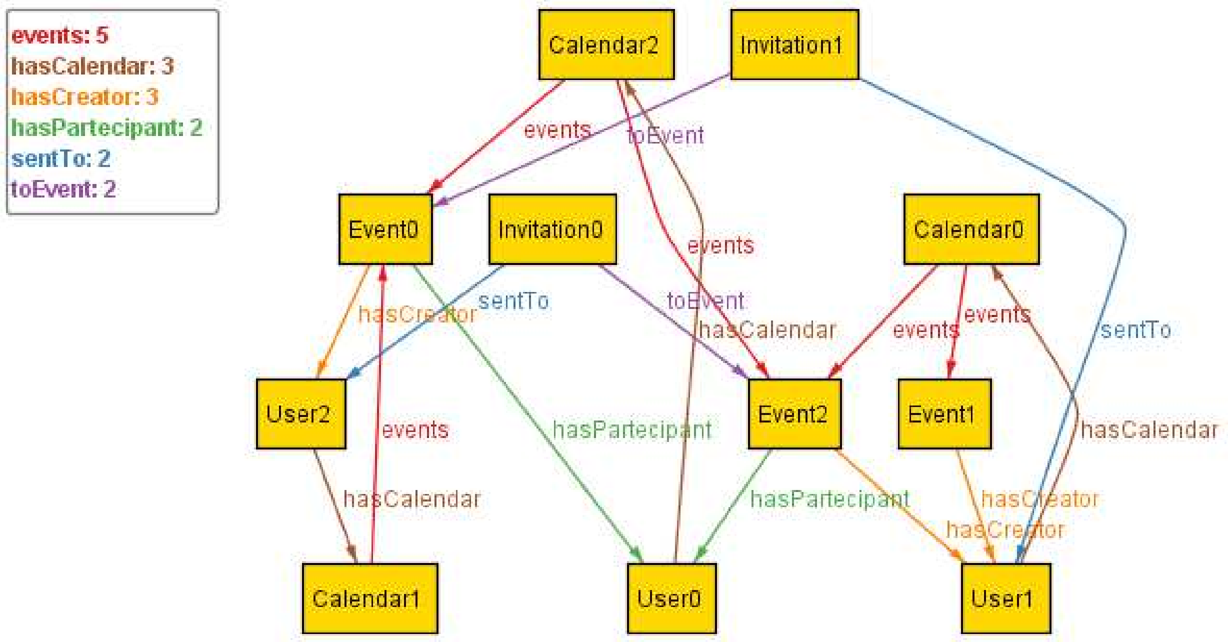


ALL INVITATIONS PENDING

Here we have User0 and User1 that create at least one event and send an invitation to the other user. No one has yet accepted the invitation so all invitations are pending and they refer to the event and the user invited.



SOME INVITATIONS ACCEPTED



8. USED TOOLS

The tools that we used to create this RASD document are:

- Microsoft Office Word 2010: to redact and to format this document;
- Astah Professional: to create UML diagrams;
- Cacoo: to create the UI sketches;
- Alloy Analyzer 4.2: to prove the consistency of our model;
- Paint: to report the Alloy model and the worlds generated.