Politecnico di Milano

Master of science in computer science and engineering

MeteoCal

A WEATHER BASED ONLINE CALENDAR

RASD Document

Requirements Analysis and Specification Document

Teachers

Raffaela Mirandola
Elisabetta Di Nitto
Marco Miglierina

Students

Alessandro Negrini 836806
Andrea Gulino 836681
Paolo Guglielmino 837055

Academic Year 2014/2015

Contents

1 Introduction

This document aims to provide a detailed analysis of the requirements and specifications related to the system we are going to develop.
Together with the Project Planning document, this will be the starting point of our work.
The Project Planning document allows us to better organize tasks in order to avoid loss of time and improve efficiency.
This document goes the same way, but doing it from a more technical perspective.
It is the first significant stage of the software lifecycle (planning, development, design, testing, ...)

The following is addressed to the team members and to every stakeholder involved in the project.
We will use it through all the development process in order to keep coherence with the specifications and satisfy all the requirements. Stakeholders, instead, will use it as a reference point to interpret our work.
Thus RASD is one of the most important gears of the entire engine and every defect of design could seriously affect the project success.
This is just the first version, changes and integrations could follow in the next deadlines.

We start with a description of the problem in terms of the Jackson and Zave approach (The world and the Machine) and then we will go across the analysis of both functional and non-functional requirements.
We will make use of the usual UML formalisms, that include class diagrams, use cases, sequence diagrams and so on.
The final part will be devoted to the formalization and testing of our model made using Alloy.

1.1 Description of the given problem

The goal is to project and implement MeteoCal, a weather based online calendar.
This software must help people to schedule their activities or events avoiding bad weather conditions if the activity is outdoor.
There are two categories of users, those who are registered and those who are not. In order to use the functionalities provided by the software, the user must be registered.
A non registered user can receive an invitation by a registered user, but in order to accept the invitation and take part to the event the user must register into the system.

Registered users can use the software to create, update and delete events (providing all the related information including where and when the events will take place, wether the event is indoor or outdoor,...).
The peculiarity of this application is that every event will be automatically enriched with weather forecast and the system will be able to notify the participants of the event in case of bad weather conditions, helping the users to find a proper solution.

In addition, there are some other tasks that the system must accomplish:

- it has to provide a mechanism that allows people to make their own calendar public, so that it is visible to every other registered user. Users can decide whether to make public the entire calendar , or only the single event. In case of private event other users can only see that in those hours the user is busy.Otherwise if the events is public users also see events details.
- a first way to propose a solution is the one in which the system, in case of bad weather, proposes to its creator three days before the closest sunny day when event can be organised.
- the system must also provide a mail notification system (both for invitation and bad weather condition)
- the system must provide a mechanism that allows users to export and import their calendar.
- the system must avoid conflicts between events
- the system must periodically update weather conditions in order to keep track of weather changes and eventually notify users

1.2 Definitions, acronyms, abbreviations

Listed below are some definitions of those terms used inside the document that may cause ambiguity.

Dictionary	
Keyword	**Definition**
Guest	It refers to the general guest of an event, both registered and not registered
World Phenomena	requirements engineering concerning with phenomena occurring in the world.
Machine Phenomena	requirements engineering concerning with phenomena occurring in the machine.
Shared Phenomena	phenomena that can be controlled by the world and observed by the machine, or controlled by the machine and observed by the world.
Canceled Event	event that will not take place and is no more visible on the calendar
Deleted Event	event deleted from system
Shared Event	event in which more than a user will take part

In addition we think that it can be useful to make having a glossary of abbreviations and acronyms:

Glossary	
Acronym or Abbreviation	**Definition**
[WP]	World Phenomena
[MP]	Machine Phenomena
[SP]	Shared Phenomena
[G]	Goals
[A]	Actors
[D]	Domain Properties
[R]	Requirements
[USC]	USer Case
JEE	Java Enterprise Edition
EJB	Enteprise Java Beans
CSS	Cascading Style Sheet
HTML	HyperText Markup Language
JSP	Java Server Pages
JSF	Java Server Faces
DD	Design Document
RASD	Requirements Analysis and Specification Document
AJAX	Asynchronous JavaScript And XML
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
WAI	Web Accessibility Initiative
W3C	World Wide Web Consortium

1.3 Overview

What above was just a brief introduction to the project. In the following chapters things will be explained more deeply. However here we provide in advance a global view of document organisation :

- **Chapter 2**
  The second chapter starts giving a first idea of the proposed system, even if the detailed architecture will be shown in the Design Document. Then risks that can occur during the development are taken into account jointly with constraints.
  Follows the identification of stakeholders and their relations with the system.
  Another important part is represented by the Jackson and Zave approach, and problem is explored also under that perspective.
  A fundamental part follows, one of the most important of all document and it is represented by preliminary considerations.
  The chapter ends with a collection of goals and domain properties, and thus the requirements derived from them.

- **Chapter 3**
  This section concerns with actors identifying, so that we point out everyone (or everything) who has to deal with system. In order to find actors, we ask ourselves which groups

or user are supported by MeteoCal to perform their work. Eventually we show external hardware or software interact with the system.

- **Chapter 4**
  Non functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviours.
  We will define what our system is supposed to be (not to do, covered by functional requirements).

- **Chapter 5**
  "Scenario is a narrative description of what people do and experience as they try to make use of computer system and applications"[1].
  So in this part some significative informal descriptions of single system feature of the system used by a single actor are provided

- **Chapter 6**
  In this chapter we make use of UML diagrams in order to build system use cases and to express the flow of events.
  Furthermore, a global static view of the system will be given using a Class Diagram.

- **Chapter 7 and 8**
  Provide the alloy model contributed to the requirement analysis and analysis model.

- **Chapter 9 and 10**
  Used tools and references

[1] M.Carrol, Scenario-Based Design, Wiley, 1995

2 Overall Description

Concerning the development of the front-end of this application we were allowed to choose between a java application and a web application.
We opted for the latter, convinced that this choice will bring benefits both to the development process and to user experience, especially considering the context of our application.

First of all, regarding the developing process, the application will have to interact with one or more external services through their APIs.
Most of modern APIs, such as Google APIs and forecast API we're going to use, are designed to easily fit the HTML5 paradigm. Moreover, JavaScript and its extensions (JQuery, AJAX) extremely simplify the integration with those services, reducing the code for an API request to just a bunch of lines.
Thus, the communication between the front-end and the back-end of our application will be connectionless, and the front-end will interact with the back-end in the same way it interacts with the other external services, that is through the use of an API.
Data exchange will be based on HTTP requests and XML data exchange; this will make the back-end completely independent from the front-end technology.

The second motivation regards the user experience. Users, should be able to use the application in any kind of situation, either when they are using their laptop/desktop computers but also, and probably mainly, when they are outdoor using their personal tablets or smartphones. Portability is nowadays one of keys to success for any kind of application. Even if java has been designed to be cross-platform, its portability is mainly limited to laptops and desktop computers. Web apps, instead, can be executed on any kind of device running a web browser. Moreover, they are always up-to-date and they behave almost the same way no matter the device, without being affected by system or software updates. Finally, the large availability of open source templates and UI frameworks for web app development will allow us to build an awesome and scalable user interface, providing the best look and feel for any context of use.

2.1 Risk Analysis

During the problem analysis, we identified some risks that could compromise system development. First, requirements can be misunderstood and this could lead to errors or delays that could compromise the entire project . Therefore we have to be really careful to the requirements collection phase.

In addition, some stakeholders may not understand or feel confused reading them, thus a very important part is covered by formal modelling that allows readers to learn requirements in a better way, reducing ambiguities. Another challenge consists on learning JEE jointly with the project development; this could involve possible delays and additional efforts.

2.2 Constraints

Constraints to be satisfied can be divided in two categories:

- **Software constraints:** system must be developed using JEE platform, and for business logic we are forced to use EJB. Other programming languages are not fixed in stone, but developing a web application means that we are implicitly asked to use languages such as HTML, CSS, Javascript, JSP, JSF, ...
  Users must have an up to date browser and an Internet connection. A more detailed description about software and hardware will be found further and in the next document, concerned with Design.

- **Time delivery constraints:** we are asked to meet some milestone and to deliver some documents for each of them :

  - 2 November 2014: Group registration and presentation
  - 16 November 2014: RASD Document (we will attach also a not requested document, the Project Plan Document)
  - 7 December 2014 : DD, that must contain a functional description of the system
  - 25 January 2015 : Implementation following requirements and design specified in RASD. Source code and executable must be delivered, including installation document and user manual. In addition a detailed document where are indicated the number of hours spent by each group member
  - 10 February 2015 : acceptance testing documentation realised on an other project developed by another group
  - Date to be defined : final presentation, we can consider it like a deployment of the complete system

2.3 Identifying stakeholders

Our main stakeholder is the Professor, who wants to have a complete and working system that meets every specification provided with the problem. Her main goal is to check that we have understood the development process in all its parts and that we can carry out a project following each development phase taking care of milestones.

So we want to show that we can accomplish all these things, trying to provide a good quality documentation and of course a good quality software.
We will provide all base actions trying to extend the software with some extra functionality.

Another figure that can have a meaningful interest in the application is the end user. The user should be the most generic possible, so our software will be able to meet all kinds of usage. Relating stakeholders to their interest into a table we can get the so call stakeholders matrix. Since we have just few stakeholders we get a non complete table.

POWER	*HIGH*	-	Professor
	LOW	-	Final User
		LOW	*HIGH*
		INTEREST	

In order to satisfy the customer with a well developed application we pointed out three main aspects that we consider important:

- **Usability :** we want to put the user, rather than the system, in the centre of the process. Our system must be easy to use, but at the same time it must provide all functionalities. In order to reach this goal, our system will display its contents in a concise and clear way, building a simple and user friendly UI.

- **Nice User interface :** even if design isn't our main objective, we think that a good looking and endearing UI is essential to affect users.

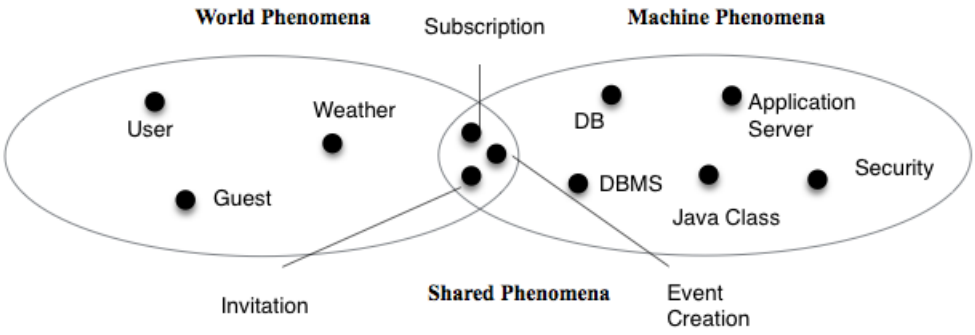
- **Stability :** system must be always available, and able to offer all its services. For example we should avoid possible system failures during registration process, event creation and so on.

2.4 The World and the Machine(Jackson and Zave)

Up to now, we have presented the project from a general perspective, taking care of general aspects without using any formal model or language.
From now on, we start taking care of details, beginning with an high level abstraction: the World and the Machine model.
It was presented in 1995 by Jackson and Zave and it helps us to collect requirements, dividing them into three categories: World, Machine and Shared Phenomena.
We attached also an instance of the approach, even if this is not complete.

Its main goal is to show how these phenomena are related each others.

Just an instance of Jackson and Zave model



A more detailed list of phenomena

- **World Phenomena**

  WP1 : a general user decides to register into the system

  WP2 : weather changes

  WP3 : a user decides to remove his account

  WP4 : the event takes place

  WP5 : a user decides to organise a new event

- **Machine Phenomena**

  MP1 : system deals with user registration

  MP2 : system stores user information in a secure DB

  MP3 : system stores user events into a specific data structure

  MP4 : system takes care of privacy policies

  MP5 : system finds a solution in case of bad weather for an outdoor event

  MP6 : system provides a DBMS in order to manage DB

  MP7 : every software that is part of the system (Application Server, ...)

- **Shared Phenomena**

SP1 : user fills in the registration form and signs up

SP2 : user creates a new event

SP3 : user invites some guest to one of his events

SP4 : guest receives an invitation

SP5 : guest decides to accept/decline invitation

SP6 : user removes his account from the system

SP7 : user sets event preferences

SP8 : user chooses a solution proposed by the system in case of bad weather

2.5 Preliminary Considerations

Before listing domain properties, since there are some points that are not so clear inside problem specifications, in this section we will make some assumption that will be respected during the overall project. In particular we assume that :

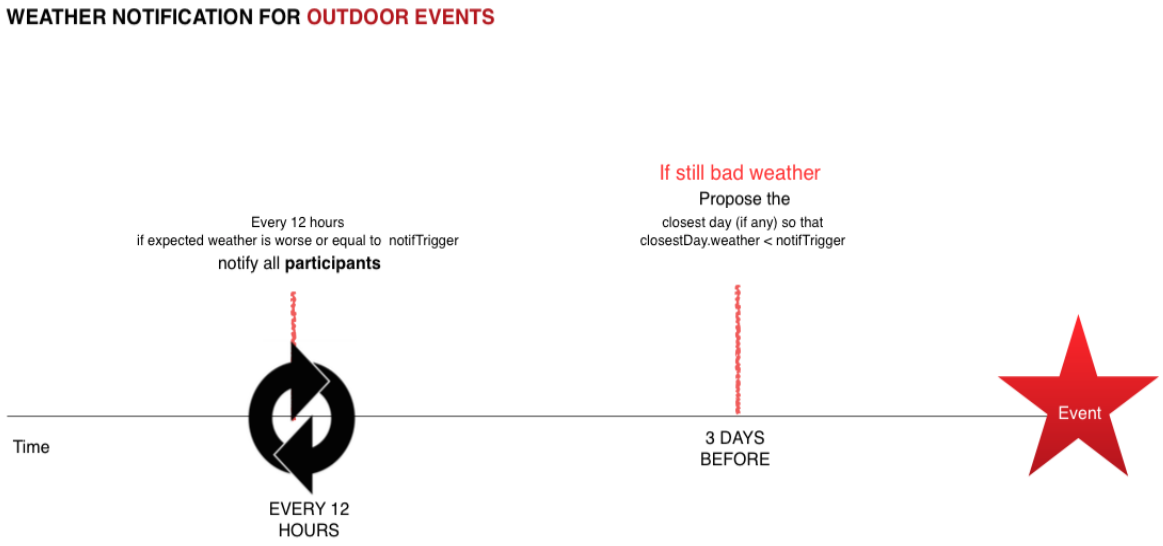
- An user can have **only one calendar**. This choice was driven by two reasons.
  The first one is because we think that giving the user the opportunity to create more than one calendar does not make sense: a physical user can't be in two different place at the same time.
  The second is to simplify calendar management. In fact, if we suppose that a user can have more than one calendar (let us suppose 2, one for work and one for home), each time he adds an event the system should perform a double check: either verifying that there are no time conflicts within the same calendar, but also checking for time conflicts with events belonging to the other calendar.

- We assume that weather notifications are sent to the user only for outdoor events with an additional constraint. When users create an event, in addition to specifying wheter it is outdoor or not, they can set a notification trigger.
  The user receives notifications if and only if the event is outdoor and the expected weather is equal or worse than the weather specified setting the notification trigger.
  The relation between API weather status and MeteoCal weather status id the following:
  - 0 : clear sky, few clouds = SUN
  - 1 : scattered clouds, broken clouds = CLOUDS
  - 2 : shower rain, rain = RAIN
  - 3 : thunderstorm = THUNDERSTORM
  - 4 : snow, mist = SNOW

- Non-registered users can be invited by registered users through their email address. In order to accept and use system functionalities they must register into the system with the same email address used for the invitation.

- In order to develop a more complete and interactive software, we will allow guests to leave a comment to the event they have been invited to.

- Concerning with the way how the software will find the best solution in case of bad weather during an outdoor event, we have identified different policies to deal with this problem:
  - starting from event creation moment, the system periodically updates the related weather information and whenever
  a weather change is detected it notifies the user accoarding to what said before about the trigger;
  - three days before the event date, in case of bad weather, the system
  notifies the user with a list of available time slots in which the weather is expected to be better.
  - in any case, the user can change the event time slot, cancel the event or simply ignore the notification.
  Visual representation of weather notifications :



- We will introduce some social networking functions: users will be able to follow other users (like in Twitter), comment an event, view other users' profile and calendar (if public)

- The system will avoid time confilicts: not two events at the same time.

- Making calendar public, does not imply to have all events in it public.

- If event creator cancels a shared event all participants receive a notification followed by a note explaining the reason of cancellation

- If event creator changes date,time or place of an event, all participants receive a notification.
  In this case all the users who had accepted the invitation are asked again to confirm their partecipation.

- Notifications are received only for outdoor events.

- All time references in DB are expressed as GMT+0. Time in user calendar is represented according to the user's default location.

2.6 Goals

The minimum set of features that the system should provide to its users is the following:

Goals Table	
Goal	**Definition**
[**G1**]	Registration of a generic user
[**G2**]	User must be able to create a new calendar
[**G3**]	User must be able to schedule its events inside his calendar avoiding bad weather conditions
[**G4**]	CRUD operations on events must be supported
[**G5**]	Users must be able to invite some guests to their events
[**G6**]	Guests must be able to receive invitations by other users
[**G7**]	The system must provide a mechanism in order to allow users to publish their calendar
[**G8**]	System must allow users to import and export their calendars
[**G9**]	Users can either accept, decline or ignore an invitation
[**G10**]	Users must be allowed to update and remove their accounts
[**G11**]	System provides a solution in case of events problems (bad weather conditions)
[**G12**]	In order to manage events in a better way, system must update weather condition periodically
[**G13**]	Users must be allowed to view other users' public calendars
[**G14**]	System must provide a way to specify weather preferences
[**G15**]	For every modification to a shared events, all participants must receive a notification

2.7 Domain Properties

The following is a list of domain conditions that should always be satisfied:

Domain Properties Table	
Domain	**Definition**
[**D1**]	A user can invite both registered and not registerd users
[**D2**]	A user doesn't have two overlapped events in the same hours. Even an overlap of a minute generates a conflict
[**D3**]	A user doesn't invite the same guest at the same event more than once.
[**D4**]	Weather forecasts provided by the open source weather data are correct
[**D5**]	Users are uniquely identified by their email address
[**D6**]	Users that attend an event have previously accepted an invitation to that event
[**D7**]	Users that create an event or accept an invitation to a event participate to it
[**D8**]	The system notifies participants to an event in case of bad weather at maximum one day before the event
[**D9**]	The time slot in which a user decides to place his event is free and available
[**D10**]	A user can't schedule an event for a past date/time
[**D11**]	A user can't create events that end before they start, or events that last less than a minute. The same constraints must hold also when events are modified
[**D12**]	A user can't invite himself
[**D13**]	A user can invite both followed users and other users
[**D14**]	A user can't follow himself
[**D15**]	A public calendar doesn't imply that events inside it are public as well
[**D16**]	In order to make an event public, the calendar in which it is placed must be public
[**D17**]	Only the user who created the event can update, cancel or delete it

2.8 Functional Requirements

Starting from domain properties, written in the previous section (section 2.7), supposing that they hold and taking in consideration goals (section 2.6), we can derive the following requirements.

The table in the next page is designed so that for each goal (assumed that domain properties hold) we derive the most important requirements that the system must meet.

Requirements Table	
	Derived requirements
[**G1**]	-[**R1**] The system has to provide a sign up functionality -[**R2**] The system updates the DB with correct data
[**G2**]	-[**R3**] The system creates, for each registered user, one associated calendar -[**R4**] The system can allow user to delete and recreate a new calendar from scratch
[**G3**]	-[**R5**] The system provides an interface in order to allow user scheduling their events -[**R6**] The system provides a mechanism that is able to avoid events during bad weather conditions -[**R7**] The system updates weather forecasts periodically -[**R8**] The system provides weather information during the event creation process
[**G4**]	-[**R9**] The system provides CRUD operations for events
[**G5**]	-[**R10**] The system allows users to find other user's profiles -[**R11**] The system offers the possibility to invite guests to events -[**R12**] The system proposes a list of followed users when a user wants to make invitations
[**G6**]	-[**R13**] The system collaborates with a SMTP server in order to send email notifications -[**R14**] The system offers an ad hoc internal mechanism to notify users in addition to email
[**G7**]	-[**R15**] During event creation, the system provides an interface that allows users to choose whether the calendar/event is public/private
[**G8**]	-[**R16**] During events creation system must check that the new event doesn't overlap with another
[**G9**]	-[**R17**] The system must be able to upload and import external calendars -[**R18**] The system must be able to export the user calendar in different formats
[**G10**]	-[**R19**] The system provides a way to accept or decline invitation
[**G10**]	-[**R20**] The system has a module that manages user settings and allows users to remove or update their account.
[**G11**]	-[**R21**] The system provides different policies to find a solution in case of bad weather conditions for outdoor events. As explained in section 2.5, system allows user to change date (so the system looks for the closest good weather day presenting to the user the possible choices), or to change place. As for changing place, the user can choose a new place manually, or can rely on a system tool that suggests him the most suitable place .
[**G12**]	-[**R22**] The system allows all the partecipants to an event to leave a comment on that event
[**G14**]	-[**R24**] During event creation, the system lets the user to choose a custom notification trigger
[**G15**]	-[**R25**] The system includes a mail notification mechanism -[**R26**] The system also allows users to display notifications within the application UI

3 Actors Indentifying

The actors of the system represent all those entities that interact in an active way with the system. We found three different kind of actors that interact with the system:

- **A1] NOT REGISTERED USER** : not yet registered user.
  - He can register into the system, so he becomes a registered user
  - He can receive invitation from other registered user
- **A2] USER** : registered user
  - He can schedule his calendar
  - He can create/update/cancel/delete his events
  - He can browse other user public calendars
  - He can follow other users
  - He can accept or decline invitations
  - He can comment events he is invited to
  - He can update or delete his profile
  - He can log into the system
  - He can logout
- **A3] SYSTEM ADMINISTRATOR** : the system manager.
  - He can have a complete access to the system

- He can manage user profiles
- He can manage the overall system
- He can manage preferences regarding weather forecasts update routines

If we agreed with the Use Case formalism that considers as an actor any external hardware or software interacting with the system, we should have considered as an actor also the external service that provides weather forecasts to the MeteoCal.
Anyway, we opted not to consider this system as an actor because, at this level of abstraction, we think that we can consider it as a part of our system, and not as an external component.

4 System Quality

4.1 Non functional Requirements

Non functional requirements refer to "what the system shall be". They describe the interaction between the system and its environment independently from implementation.
We have pointed out some non functional requirements that system must meet, and also some qualities that our development will guarantee.
We already mentioned some of them in section 2.3, here we provide a complete detailed list.

4.1.1 Usability and Portability

As already told our system will provide an easy to use graphic interface. Moreover, we want that our software can be used on the largest possible number of devices. Thus, our platform must be well designed both for desktop and smartphones. Thus, the front-end must be developed in a way that allows to fit all screen sizes.
In addition, we expect to provide a mobile implementation of this application. This extra project is carried on within another university course : Design and Implementation on Mobile Application.
It won't be a web app application, but an application developed using ad hoc mobile programming language in order to improve and offer some functionalities that a simple web app wouldn't be able to offer.

4.1.2 Accessibility

Accessibility is the degree to which our application is available to as many people as possible. We can consider it as the 'ability to access'.
Often, this concept focuses on people with disabilities or special needs (blind people, ...) .
A member of our group last year, within a international context (Athens programme), attended a course of "Accessible Web Design" and he will put in action all the things he learned.
Overall we will try to meet the rules imposed by WAI and W3C standards

4.1.3 Efficiency

Within software development framework, efficiency means to use as less resources as possible. Thus, system will provide data structures and algorithms aimed to maximize efficiency.

We will also try to use well known patterns reusing as many pieces of code as possible, taking care of avoiding any anti-pattern.

4.1.4 Extensibility

Our system must provide a design where the implementation takes future growth into account. It will be developed in a way such that the addition of new functionalities won't require strong changes to the internal structure and data flow.

4.1.5 Maintainability

As in the section above, when a developer will have to fix something inside the code, the implementation will try to make the maintenance operation easy to accomplish.

4.1.6 Availability and Stability

Our system must be able to offer its services whenever the user asks for them. It must be available 24/7.
In case of malfunctioning, administrator will provide maintenance in order not to affect service availability

4.1.7 Security/Privacy

Without any doubt it is one of the most important requirements.
Every online service that deals with personal data should ensure this property. Supposed that data is permanent on the disk, system must concern with privacy of data. In order to meet this requirement, it will provide an authorization system that filters only registered users.
Users' Passwords are collected in a secure and encrypted way on a database. In this way the user will be the only person knowing his passwords.
Not only the database must be secure, but also every transaction between front-end and back-end should rely on secure protocols (Actually, since building a secure protocol costs money, we suppose that classical protocols will ensure security).

4.1.8 Reliability

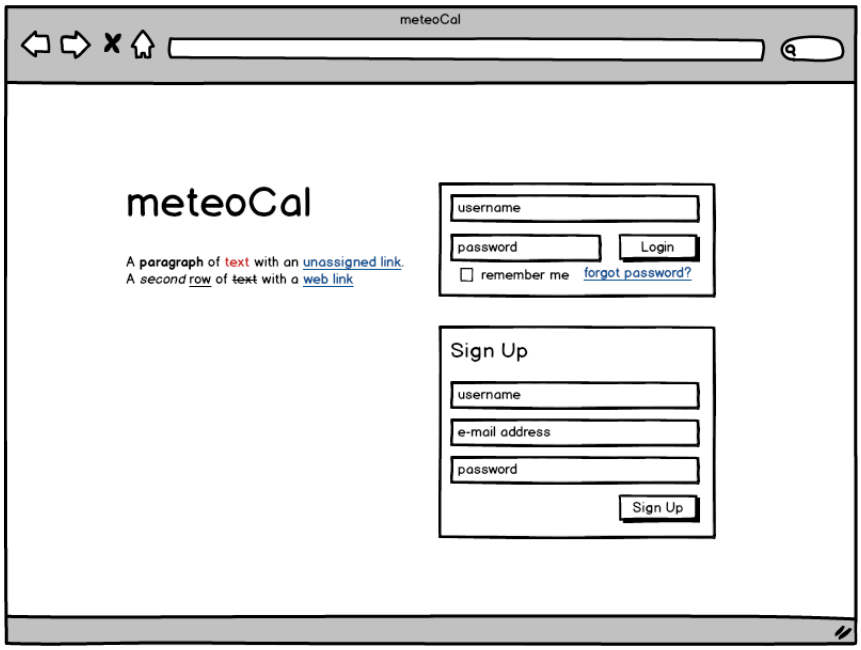
Since data are exchanged among users, data reliability is essential. Users can base their actions on other users' data. Moreover, we suppose that the memory where database is stored is stable.

4.2 User Interfaces

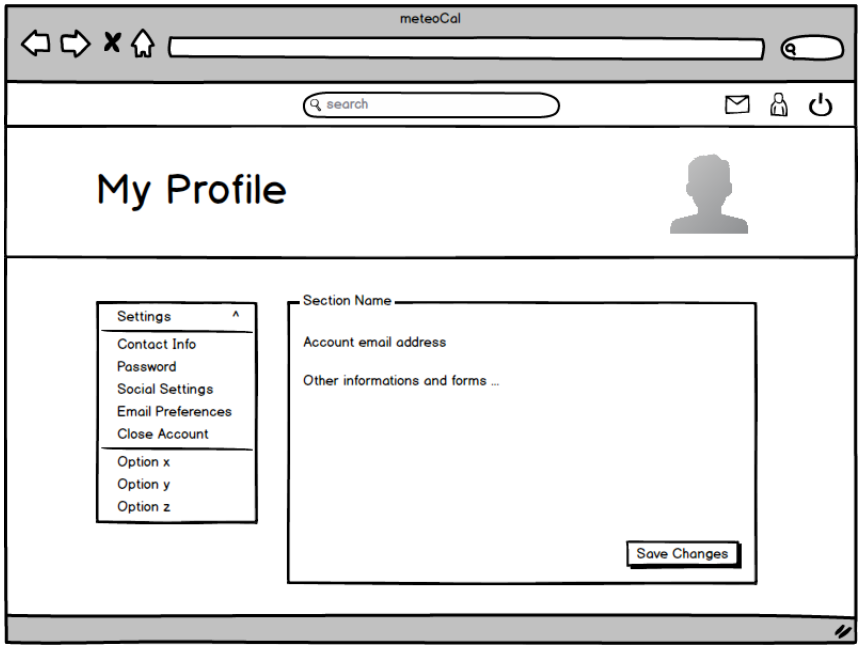
The interface of our application is though to be used via internet, in order to reach the most part of the users in every way.
As yold before, MeteoCal will provide a simple and complete graphic structure that makes it easy to find main functionalities by users.
What follows are some page layouts according to which our UI will be designed.

Our platform contains two main environments, one is about user profile management, and the other one is about event management within which the user can have a look to his calendar and perform actions like adding new event, delete, update, and so on.

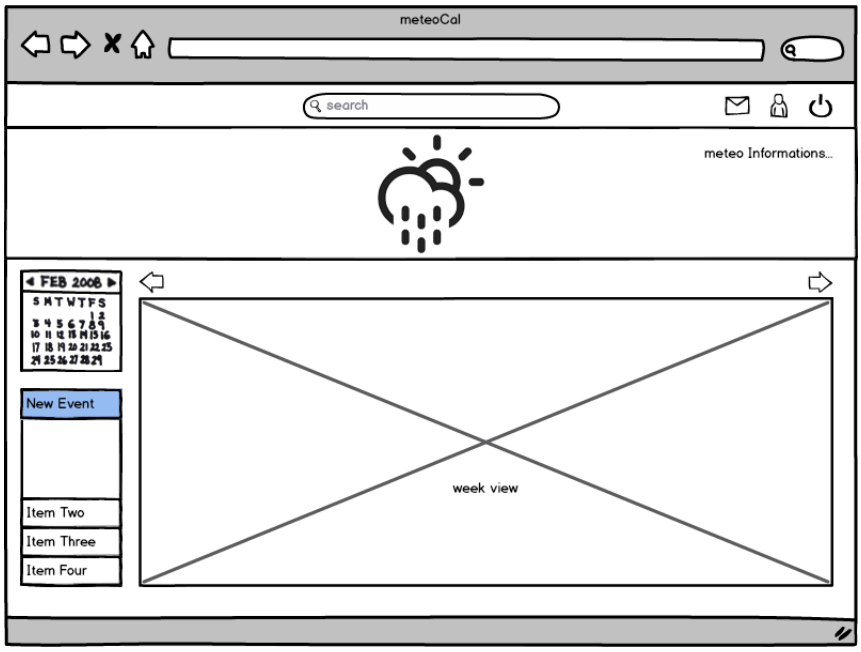
4.2.1 Login interface



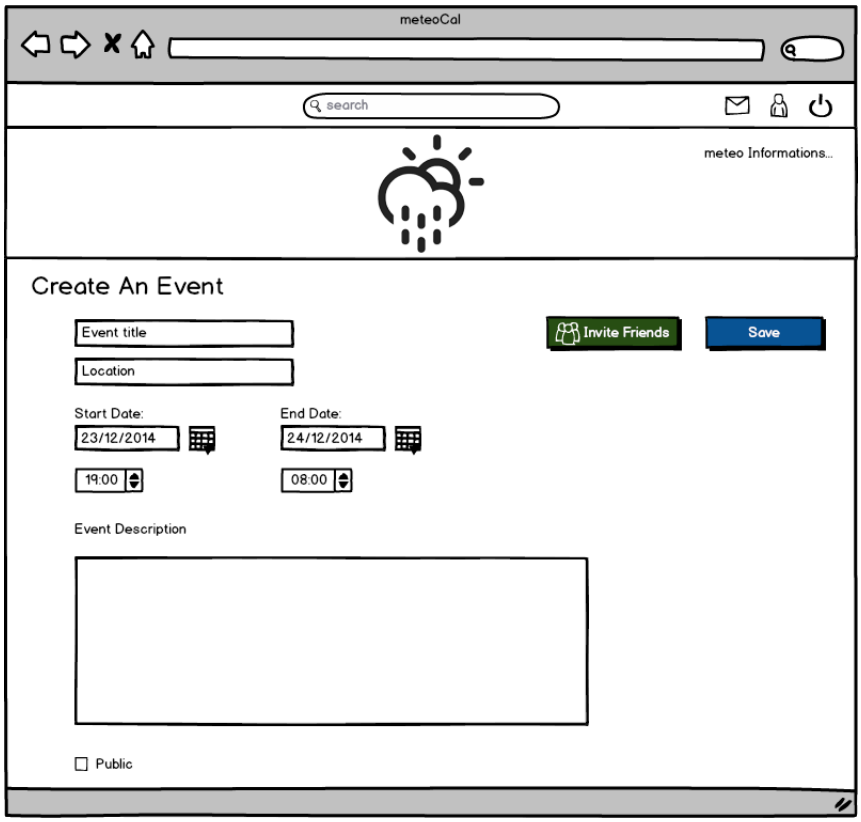
4.2.2 User Profile interface



4.2.3 Calendar interface



4.2.4 Create Event interface



4.3 Documentation

In order to show project development and also to offer a documentation to the final user, we will write the following documents:

- **Project Plan** : defines team members and team organization. A brief time estimation will be done.
- **RASD** : addressed to final users, clients, analysts and developers. Its objective is to understand the given problem defining its goals, requirements and specification
- **DD** : addressed to developers and analysts. It aims to define real structure of our web application and its tiers.
- **JavaDoc comments in the source code**: addressed to other developers in order to understand and maintein the code.
- **Installation Manual**
- **User Manual**
- **Testing Document**: report of a project developed by another group

4.4 System Architecture

We have already given a quite detailed explanation of our system . Here we point out the main components used.
From the server side point of view, since our platform is developed over a JEE platform, JSP and Servlets are fundamentals. As for the client side, the final user only needs to have a browser.

5 Scenarios Identification

In this part of the document we will provide, starting from requirements specified in an a natural language, partially detailed in previous chapters, a brief description of the system using some scenarios. Afterwards, we will come up with a higher level description through use cases, using also a semi-formal notation: UML diagrams.

Known the size of our project, we could list hundreds of scenarios. The list below states just some of them.

5.1 Alice decides to register to the system

Alice's life is very busy and she looks for a system that allows her to schedule her activities. Moreover, she wants the system to provide a mechanism of notifications that is able to keep her up to date on event changes.
So she googles for "Agenda" and she gets a wide list of results. Scrolling down she finds the result "MeteoCal" and she wonders why that agenda has that name. She decides to click on it and the home page, containing a brief description of the app functionalities, appears. She reads the description and she gets captured by the fact that the application joins the agenda with weather forecasts. She thinks that in that way whenever she wants to do something outside she can rely on it. Thus, she decides to register to the system, she fills-in and submits the registation form with her personal information, including also a default location, and confirms her email address by clicking a link eclosed in the received email that finally redirects her to the app login page.

5.2 Alice starts scheduling an event with her children to the sea

After loggind-in, Alice starts scheduling her activities trying to create a new event for the next Sunday afternoon. So she presses "New event" and an empty form is shown.
She has to provide the name of the event, when it starts and it ends, a brief description, and choose between the "outdoor" and "indoor" options. Her event is an outdoor one, and when she sets the "outdoor" option, she notes with pleasure that, differently from other agenda software that she previously tried, she can also add her weather preferences.
Since she expects to go to the seaside with her children, she specifies that she wants to be

notified only in case of rainy weather. In case of clouds, she will go to sea anyway.
Finally she presses the "Add" button and the event is correctly added to her calendar.

5.3 Bob plans a shared event with three friends

Bob is a man who has already discovered MeteoCal before Alice did. He uses it very often and today he wants to plan Saturday night outside with other three friends. He decides to create an event with guests on MeteoCal. So he creates an event and he specifies all event information and weather preferences. Before pressing the "Add" button, in order to confirm event, he invites his friends, so he types two of his friends' names and he adds the suggested users to the event, and adds the email of a friend who is not registered to MeteoCal. Finally, he presses the "Add" button and the event is correctly added to his calendar.

5.4 Alessandro, a registered user, receives the invitation from Bob and accepts

Alessandro, the first of the three persons that Bob invited to his event, is scheduling a picknic with his family for next Friday afternoon.
He is logged into the system and at a certain point he ears the MeteoCal notification sound, it is Bob's invitation.
He opens the notification and gets redirected to the invitation. After looking at the event details he clicks on the 'Acceot' button and the event is finally added to his calendar.

5.5 Andrea unfortunately can' t attend the event

Andrea, the last of the three persons that Bob invited to his event, is working hard on his PhD thesis of Philosophy and he considers ephimeral going outside with other friends. While he is writing his thesis on his 'brand new' Windows 98, he receives an invitation email by MeteoCal. Opening the link enclosed in the email and logging in into the system, he reads about Bob's event invitation but, for the reasons explained above, he is is forced to decline the invitation. When he declines the invitation, he has also the opportunity to explain the reason why he has declined it, and he writes a brief note apologizing with Bob, Alessandro and Paolo.

5.6 Paolo, a non registered user, receives the invitation from Bob and accepts

Paolo, the third of the three persons that Bob invited to his event, is surfing on internet when he receives an email from MeteoCal. He isn't registered to the application.
He is interested to Bob invitation and he wants to accept after checking the event date and time. He pressed the "Accept " button on the received mail, and the MeteoCal registration page appears.
Paolo fill in the form with his information, and finally he can accept and confirm his participation to the event. At the end his calendar has a scheduled event for the saturday night.

5.7 Bob, the event creator, two days before the event, finds out a problem

Two days before attending the event with his friends, Bob finds out that on Saturday night he isn't free and he must attend an appointment that he can't miss.
So he decides to postpone the event to the next week, using updating options provided by the system.
The System will notify his friends (the invited ones), including Andrea, the one that refused, because in this new date he could be available, sending them a notification .
The event is moved from the original Saturday, to the next one.

5.8 Andrea gets available for the new date

Andrea receives the notification that the date of the event has changed.
On that date he will have already discussed his thesis, and so he will be willing to attend the meeting with Bob, Alessandro and Paolo. After logging-in and accepting the invitation he is finally added to the invitation partecipants and a new event appears on its calendar.

5.9 Paolo confirms the changes

Also Paolo confirms changes and the event is rescheduled for the new date .

5.10 Alessandro can't attend the modified event

On the contrary, Alessandro, who works in the city hospital, has to watch over his patients on that specific date time and so he is forced to decline the invitation.
He writes a note explaining the reason why he can't be present.

5.11 Alice updates her profile

Alice suddenly remembers that her profile can be enriched with some more information, and she also wants to upload a profile photo. So she logs into the system and accessing to personal information manager section, she has the possibility to change all her data.

5.12 Alice receives bad weather notification and reschedules he revent

Two days before the event, Alice receives a bad weather notification from MeteoCal. MeteoCal provides her a list of possible time slots for which she can reschedule the event. The first solution proposed is for the following Tuesday, but she can't do that because she is at work. Another proposed time slot is for the following Saturday. She's free and she chooses this option. Her calendar is updated.

5.13 Marco receives bad weather notification and he ignores

Marco is a user that has planned an outdoor event for Monday morning. The day before, he receives a bad weather notification, and as usual MeteoCal provides him some solutions.
Marco decides to keep his event anyway, so he ignores the notification.
The calendar remains unchanged.

6 UML Models

6.1 Use Case Diagram

Starting from scenarios collected in the previous section and the general analysis done at the beginning of this document, we could derive many use cases. However, we will do a detailed description, not for every possible use case, as done for scenarios, but only for the most significative ones .

Use cases are flows of events initiated by an actor.

First, we design a **Use Case Model**, that is the set of all use cases specifying the complete functionality of the system .

Then, each single use case is handled on its own. For each use case, information like use case name, participating actors, exceptions, flow of events and thus sequence diagram associated are provided. We describe in a detailed way the main use cases.
Some use cases that extend other use cases are omitted because they are really similar to others.

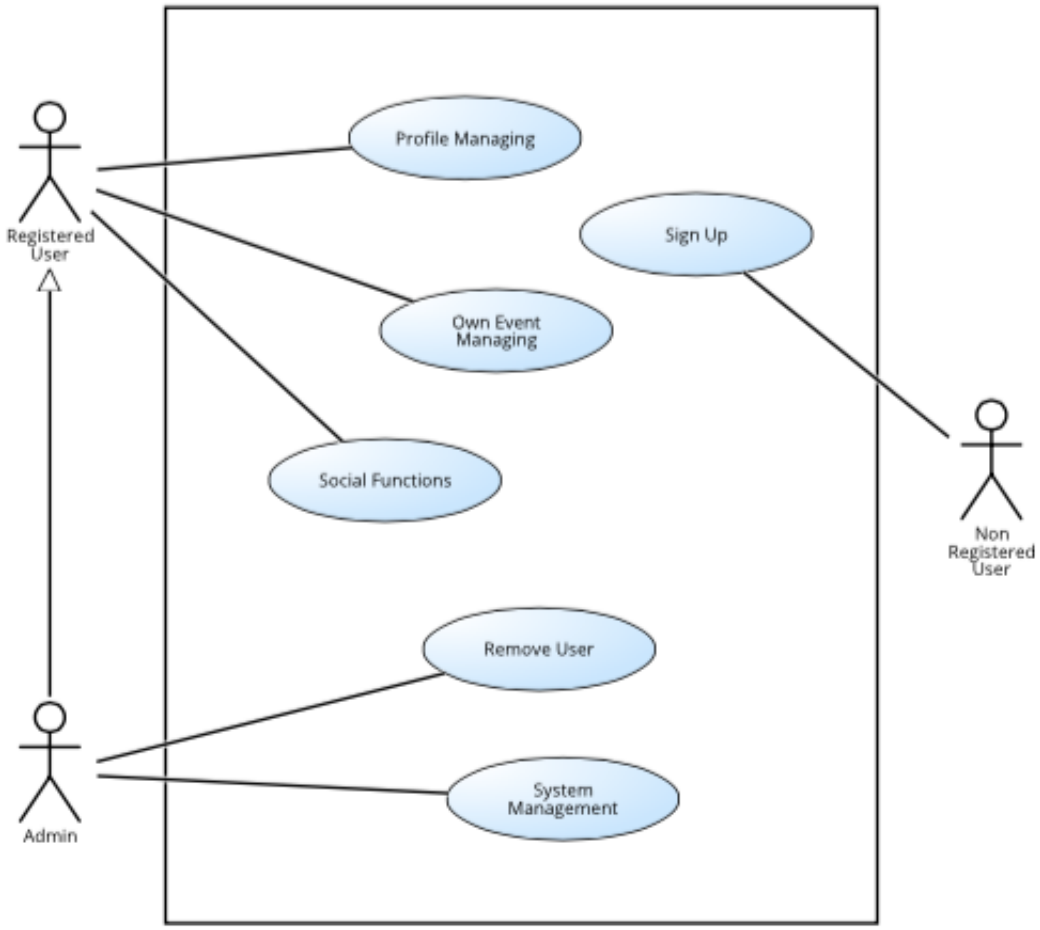
The use cases we are going to present are the following:

- Sign up
- Log in
- Creation of new event
- Receive event request
- Make public calendar
- Updating event information
- Deletion event
- Updating personal profile
- Follow another user
- Event invitation
- Search a registered user and see his public events/calendar

- Account remove
- Calendar export
- Calendar import
- Provide a feedback to an event
- Add a new system functionality

6.1.1 Use Case Model

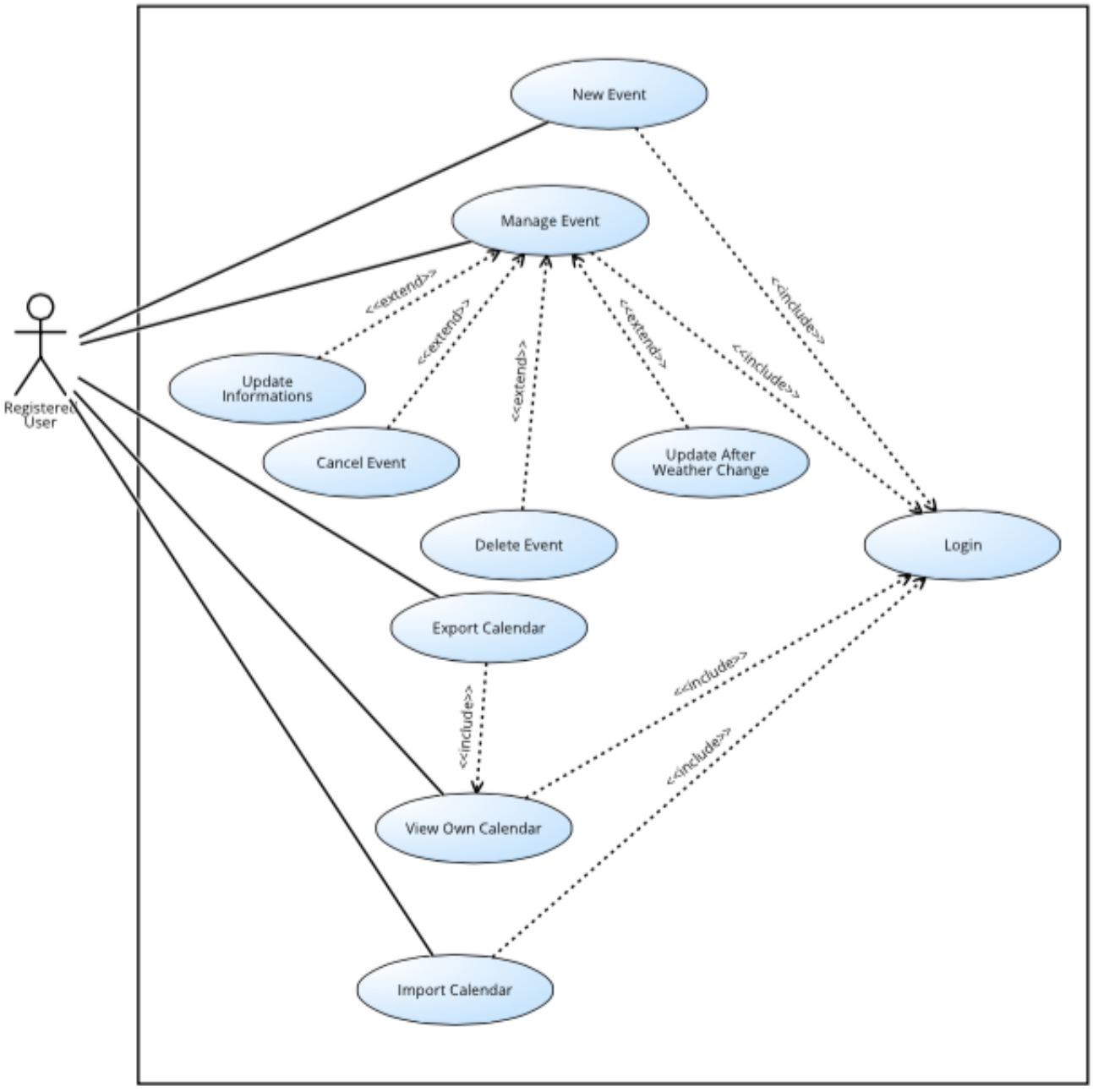
Below the general use case is depicted, in it we include our main actors. It offers just a very high level vision of the system.



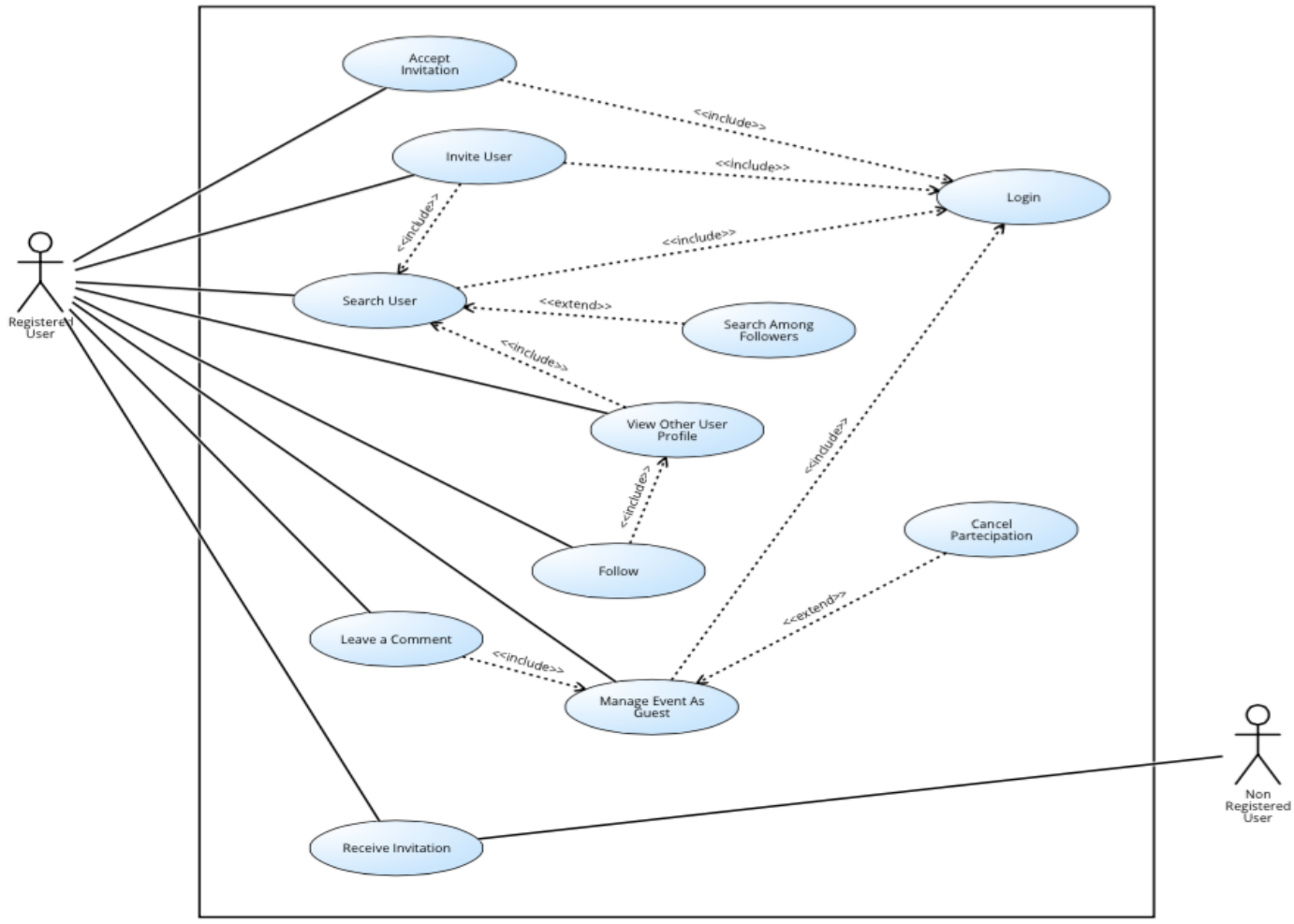
We pointed out three main categories of use cases. We call them :

- Event Management : it collects functionalities like creation of a new event, event deletion, update event details, export calendar and import calendar
- Profile Management : it collects functionalities like login, registration, updating personal info, and account deletion
- Social : it collects functionalities like following new user, searching for user, viewing user's profile and calendars, rate and comment, receive event invitation.

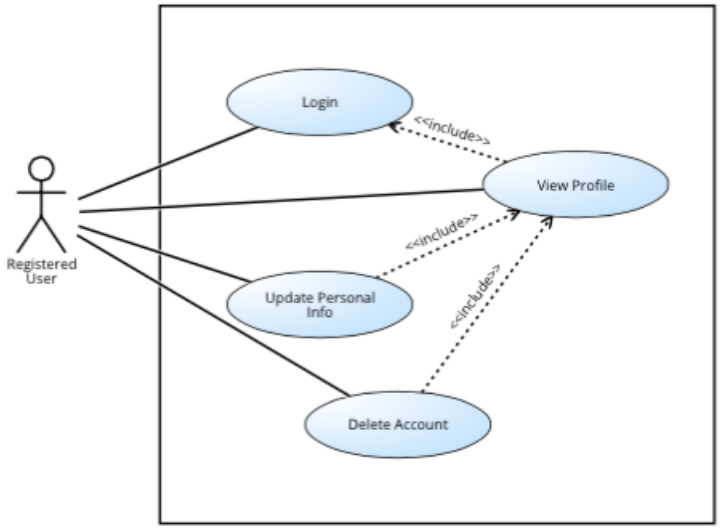
6.1.2 Event Management Use Case



6.1.3 Social Use Case

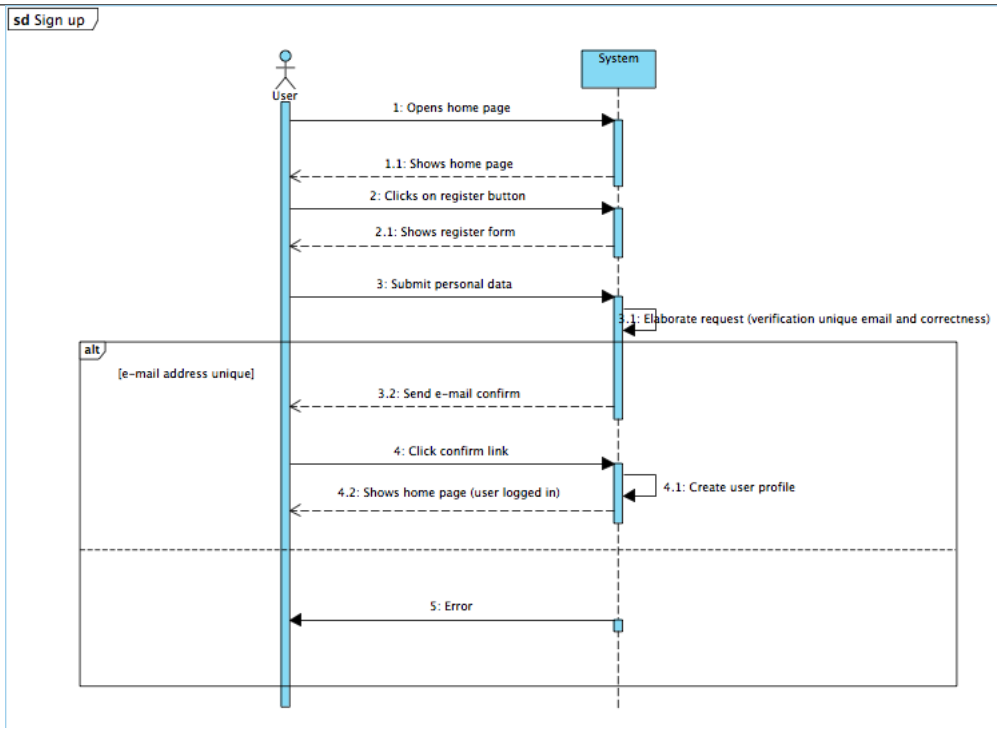


6.1.4 Profile Management Use Case

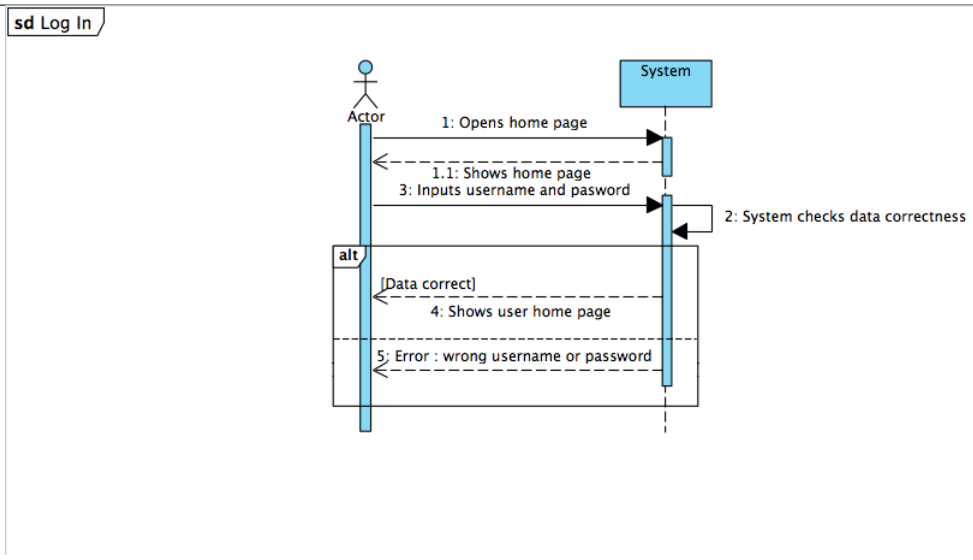


6.2 Sequence Diagram

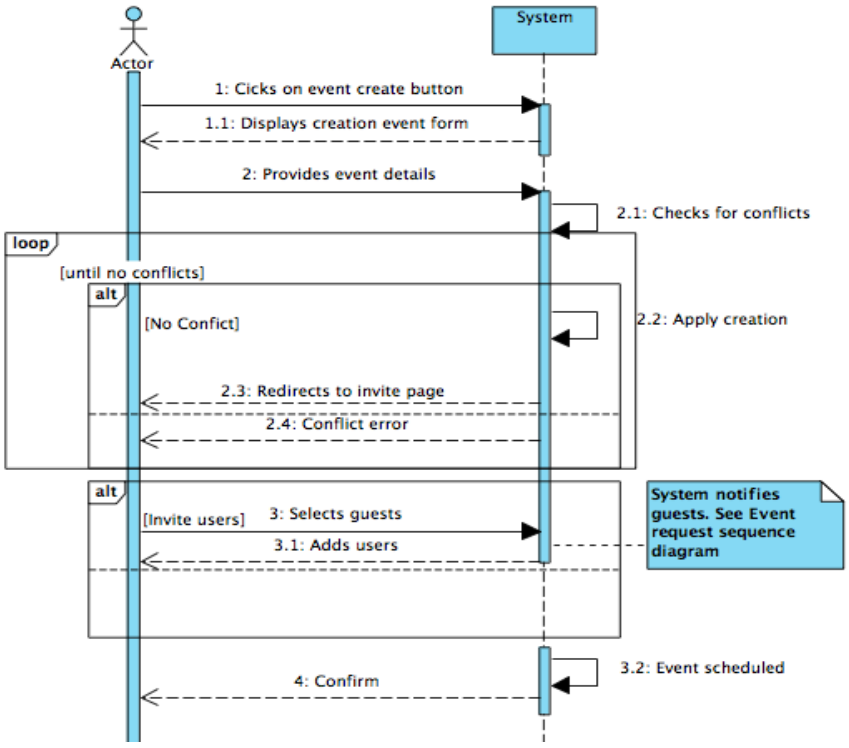
6.2.1 Sign up

Sign up	
Code	USC01
Description	The unregistered user registers in the system
Assumptions	The user is not registered yet
Actors	Non-registered user
Entry conditions	The user navigates to the homepage of MeteoCal
Exit conditions	Profile information and data are successfully saved. User is registered
Exceptions	User types something wrong, or misses to fill something. Email address can be not unique. Users doesn't click on email confirm
Flow of events	1. The User opens the home page of MeteoCal 2. The System shows him the form to be filled in 3. The User clicks on register button 4. The System shows him the form to be filled in 5. The user inputs his personal information and click on confirm button 6. The System checks data validity and uniqueness of the email address. 7. The System sends the confirmation email 8. The User clicks on confirm 9. The System creates profile and opens the log in page Also a user calendar
Sequence Diagram	

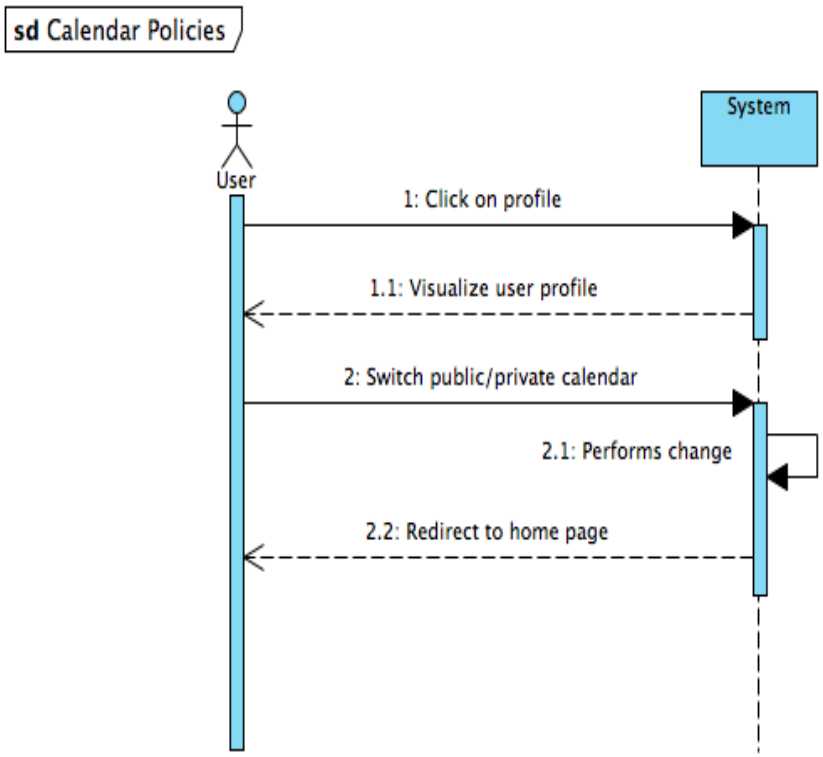
6.2.2 Log in

Log in	
Code	USC02
Description	The registered user logs into the system
Assumptions	The user is registered to the system
Actors	Registered user or Administrator
Entry conditions	The user has successfully signed up to the system
Exit conditions	None
Exceptions	Username or password are wrong
Flow of events	1. The User or administrator opens the home page of MeteoCal 2. The system shows him the page 3. The User or administrator inputs his username and password and clicks on log in button 4. The System checks data correctness 5. The System shows the profile or administrator page
Sequence Diagram	

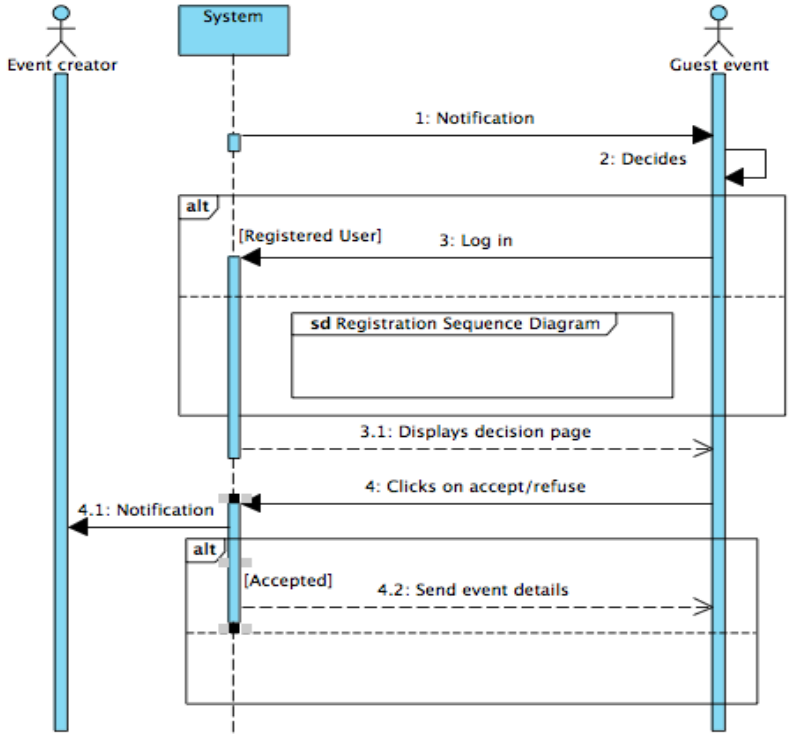
6.2.3 Creation new event

Creation new event	
Code	USC03
Description	User creates an event
Assumptions	The user must be logged in, and he wants to create a new event that doesn't already exist
Actors	Registered user
Entry conditions	The user has successfully signed up to the system
Exit conditions	The event is correctly created and scheduled on the calendar
Exceptions	New event generates a conflict with another event.
Flow of events	1. The User clicks on event create button 2. The system shows him the section in which user fill in event information 3. The User inserts starting and ending time and date, the name of the event, a note 4. The User confirms clicking on confirm button 5. The system checks for conflicts 6. The System schedule in the correct way the event on user's calendar 7. The Systems redirects user to invite user page 8. The User selects user to invite 9. The System adds them to events and send a notification [see receive event request]
Sequence Diagram	

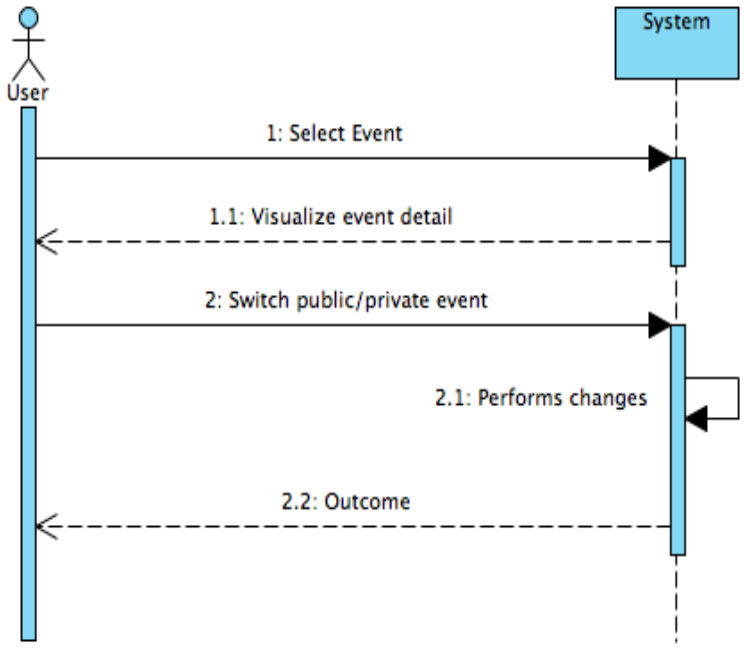
6.2.4 Make Public Calendar

Make Public Calendar	
Code	USC04
Description	A user wants to make public his own calendar
Assumptions	The user must be logged in, and he wants to make his calendar public Making calendar public, does not imply to have all events inside it public as well.
Actors	Registered user
Entry conditions	The user has successfully signed up to the system
Exit conditions	The calendar is visible to everyone
Exceptions	None
Flow of events	1. The User clicks on his profile 2. The System displays his information 3. The Users switches the visibility settings of his calendar 4. The System performs the action and allows everyone to view it 5. The System redirects user to home page
Sequence Diagram	

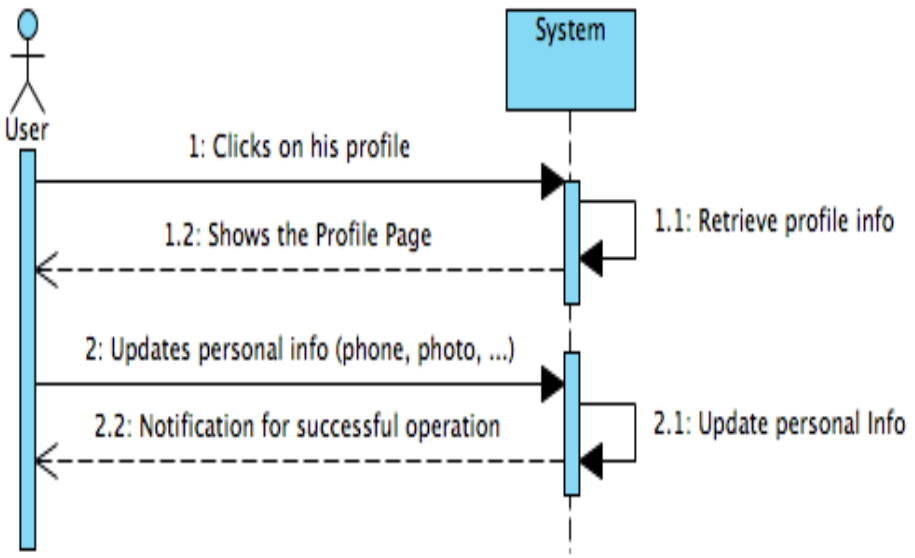
6.2.5 Receive event request

Receive event request	
Code	USC05
Description	User receives a invitation by an other user. This use case describes only the relation between system and the invited user, when he receives an invitation.
Assumptions	User has a mail address
Actors	Registered or non-registered User
Entry conditions	An user has invited a user to attend his event
Exit conditions	User decides whether accept or decline the invitation
Exceptions	User ignores email invitation
Flow of events	1. The System sends an email that invite the user to attend a particular event 2. The User decides whether accept/decline 3. The User logs into the system if he is already registered [User signs up if he is not yet registered (see sign up use case and sequence diagram)] 4. The System provides him an interface that allow user to choose whether accept/decline 5. The User clicks on accept/decline button 6. The System informs event creator user whether he accepted or not
Sequence Diagram	

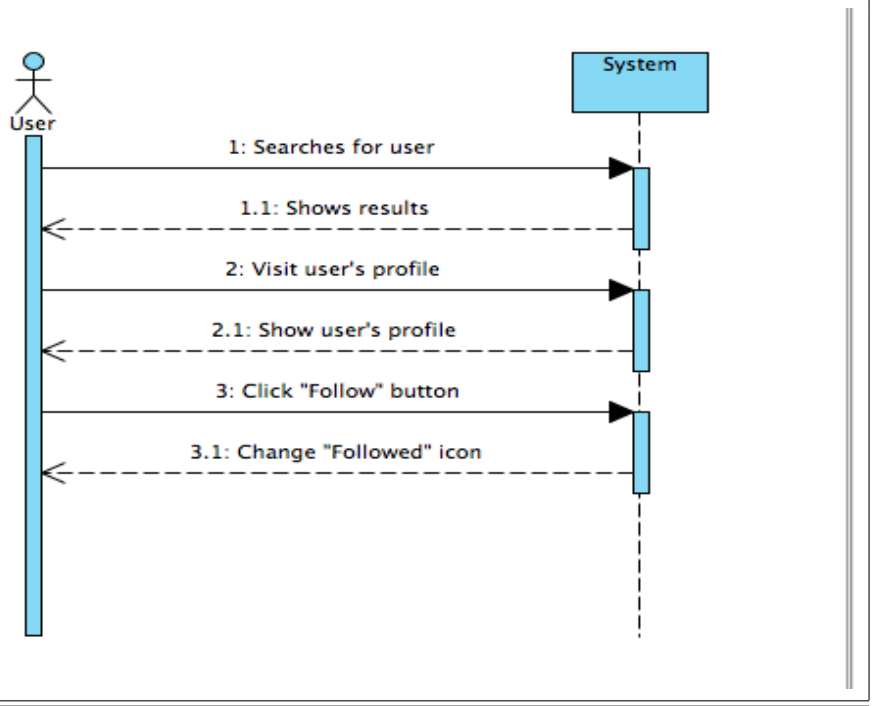
6.2.6 Make event public

Make event public	
Code	USC06
Description	A user wants to make public his own event
Assumptions	User has at least an event scheduled in calendar
Actors	Registered User
Entry conditions	User is logged into the system
Exit conditions	The event details is visible to everyone
Exceptions	None
Flow of events	1. The User clicks on one of his events 2. The System shows events detail 3. The User switches event visibility setting 4. The System performs changes
Sequence Diagram	

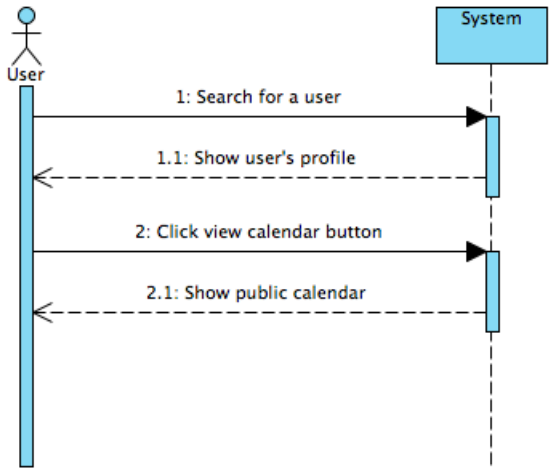
6.2.7 Update User profile

Update User profile	
Code	USC07
Description	The user wants to update his profile (photo, add tel.number, ...)
Assumptions	User is registered
Actors	Registered User
Entry conditions	User is registered and he is logged into the system
Exit conditions	User profile is updated
Exceptions	None
Flow of events	1. The User clicks on update profile button 2. The System retrieves profile info 3. The System shows him the update form 4. The User changes or add some fields 5. The System updates the new profile 6. The System sends notification to the User
Sequence Diagram	

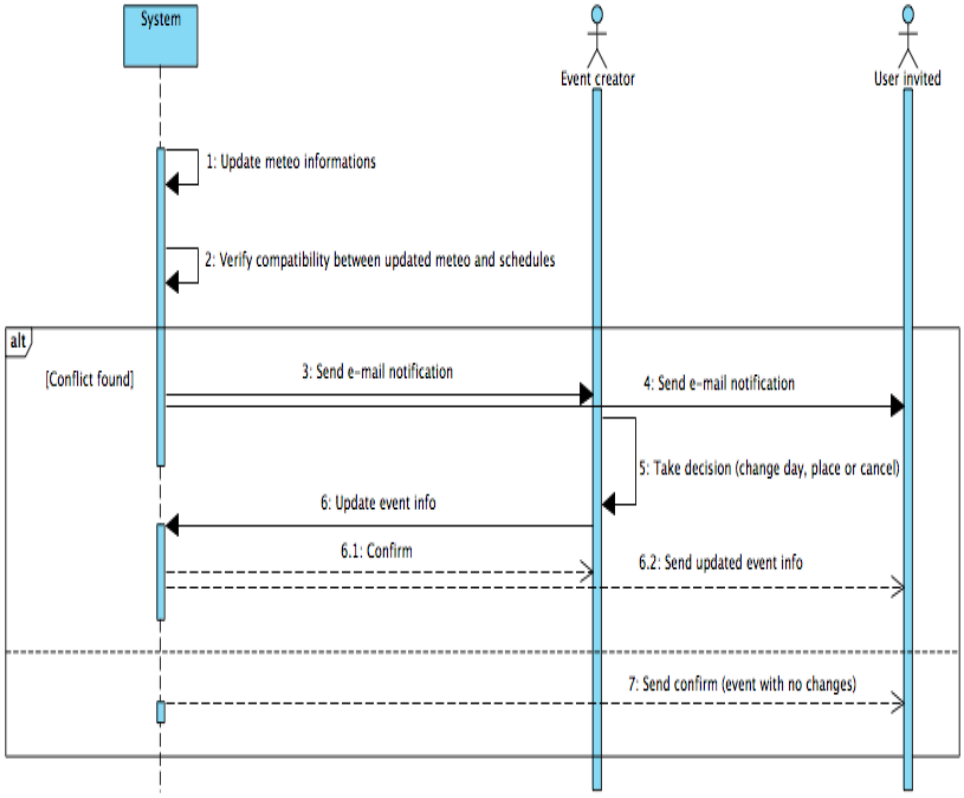
6.2.8 User starts following an other user's profile

User starts following an other user's profile	
Code	USC08
Description	User wants to add to his MeteoCal friends another registered user
Assumptions	User is registered
Actors	Registered User
Entry conditions	User is registered and he is logged into the system
Exit conditions	Users now is following that profile
Exceptions	None
Flow of events	1. The Users search for a particular user searching him on search bar 2. The System shows him his profile 3. The User clicks on follow button 4. The System adds a new profile to the ones that user already follows
Sequence Diagram	

6.2.9 User views another user's public calendar

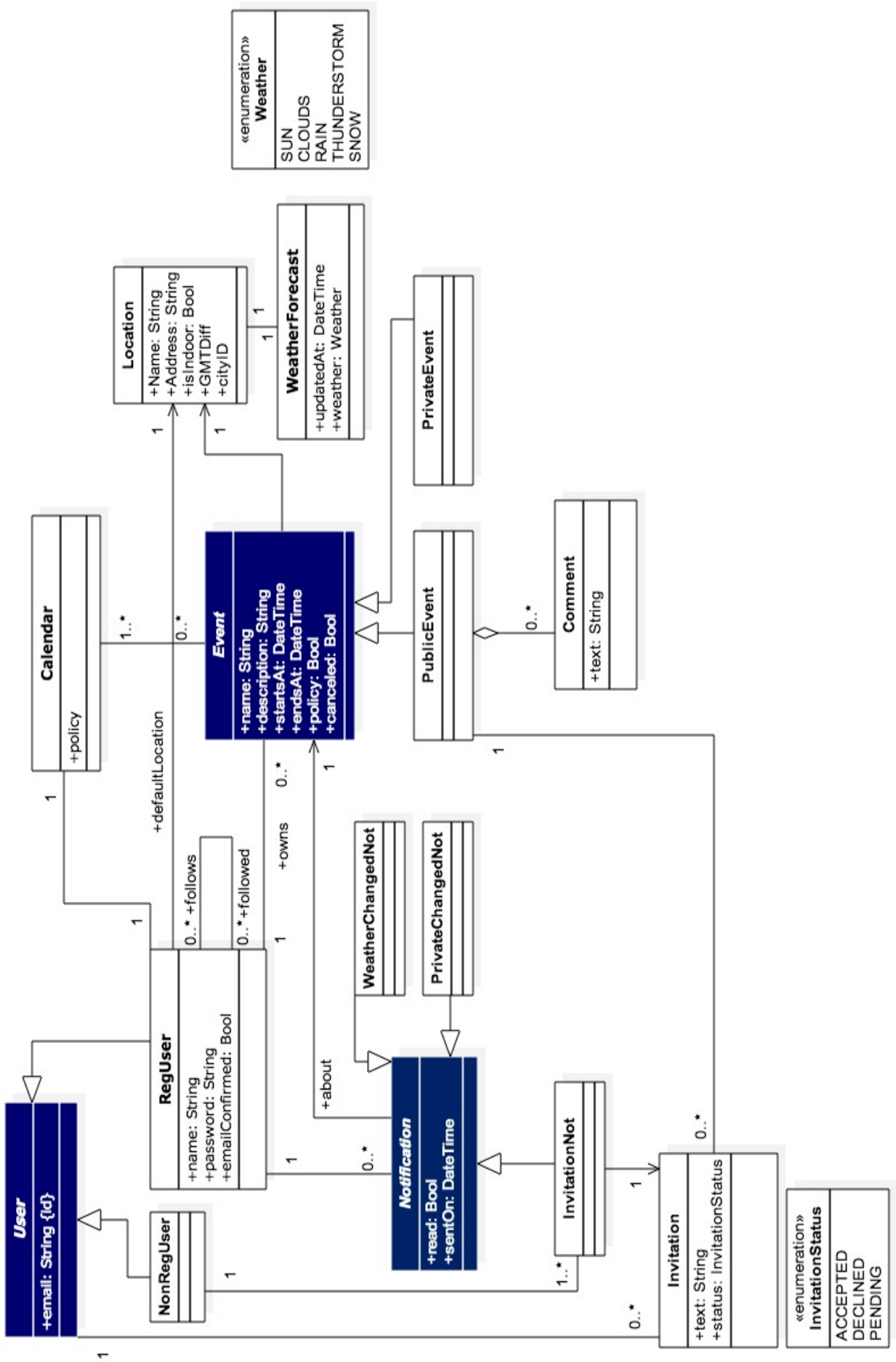
User views another user's public calendar	
Code	USC09
Description	User wants another registered user's calendar
Assumptions	User is registered and the other has a public calendar
Actors	Registered User
Entry conditions	User is registered and he is logged into the system
Exit conditions	User has a view on public calendar
Exceptions	None
Flow of events	1. The Users search for a particular user searching him on search bar 2. The System shows him his profile 3. The User clicks on button that allows him to view his calendar 4. The System shows him the complete calendar with eventually also public events
Sequence Diagram	

6.2.10 Bad Weather Notification

Bar Weather Notification	
Code	USC10
Description	System updates weather condition, and notifies user in case of bad weather
Assumptions	System receives weather updates
Actors	Registered and Non Registered User
Entry conditions	None
Flow of events	1. The System updates weather conditions 2. The System checks if there are outdoor events scheduled in bad conditions 3. The System sends email notification to event creator and any invited users 4. The User takes a decision 5. The System reschedules the event with new details, if changed 6. The System sends confirmation to any user
Sequence Diagram	

6.3 Class Diagram

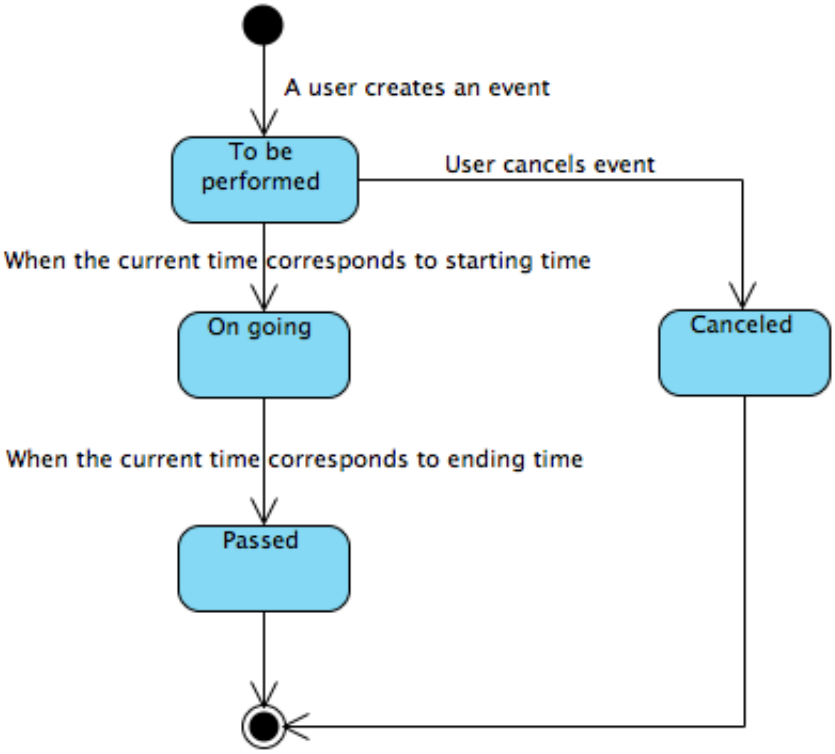
Now it's time to derive, starting from requirements detailed in the previous chapter and flow of events, our project class diagram. It presents project specification under a static point of view of main entity involved in, and their relationship.
The following diagram introduces the conceptual classes that we have decided to include in the software product.


«enumeration»
Weather
SUN
CLOUDS
RAIN
THUNDERSTORM
SNOW
Location
+Name: String
+Address: String
+isIndoor: Bool
+GMTDiff
+cityID
WeatherForecast
+updatedAt: DateTime
+weather: Weather
PrivateEvent
Calendar
+policy
Event
+name: String
+description: String
+startsAt: DateTime
+endsAt: DateTime
+policy: Bool
+canceled: Bool
PublicEvent
Comment
+text: String
+defaultLocation
+owns
+follows
+followed
WeatherChangedNot
PrivateChangedNot
RegUser
+name: String
+password: String
+emailConfirmed: Bool
+about
User
+email: String {id}
Notification
+read: Bool
+sentOn: DateTime
InvitationNot
NonRegUser
Invitation
+text: String
+status: InvitationStatus
«enumeration»
InvitationStatus
ACCEPTED
DECLINED
PENDING

6.4 State Charts Diagram

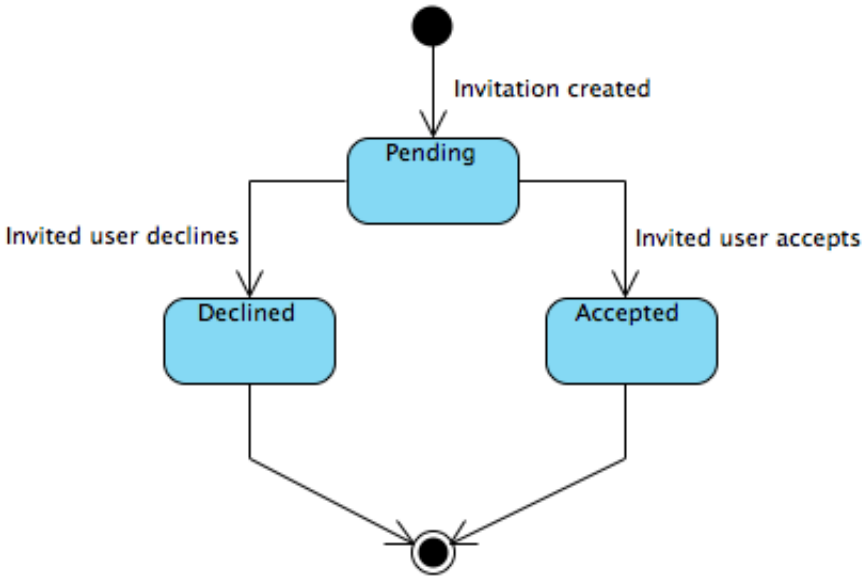
6.4.1 Lifecycle of an event

At the beginning, when a user creates an event, the event is in "To be performed" state. Then if he doesn't cancel it, when the current time corresponds to the event starting time the events starts and so it goes into "On going state", until it ends and goes into "Passed" state.
Eventually, before the event starts, the user can cancel the event, and event goes into "Cancelled" state.



6.4.2 Lifecycle of an invitation

At the beginning, when a user creates a shared event, system sends a notification to a given user. The first state is the "Pending" one, that means that the invitation hasn't received an answer yet by invited user.
If invite is accepted/declined it goes into "Accepted"/"Declined" state.


Invitation created
Pending
Invited user declines
Invited user accepts
Declined
Accepted

7 Alloy Modelling

Our main goal is not be inconsistent, and now, using this tool we try to understand if our Class Diagram is well designed using Alloy Analyzer.
The following Alloy code models a system abstract representation.

```
--MeteoCal MODEL
-----------------------------------
-- SIGNATURES

--Generic User identified by email address
abstract sig User {
   email: one String,
   invitations: set Invitation,
   notifications: set Notification
}
--Non Registered User
sig NotRegUser extends User {
}{ #invitations >= 1
  some inot: InvitationNot |  inot in notifications
}

--Registered User
sig RegUser extends User{
   emailConfirmed:  one String,
   -----------------
   calendar: one Calendar,
   events: set Event, //owned events
   followed: set RegUser,
   followers: set RegUser,
}{ emailConfirmed = "true" || emailConfirmed =  "false"}

--Event
sig Event {
   owner: one RegUser,
   location: one Location,
   startsAt: one Int,
   endsAt: one Int,
   notifTrigger: one Int, //0 = SUN |  1=CLOUD | 2 = RAIN | 3 = THUNDERSTORM | 4= SNOW
   canceled: one String,
   invitations: set Invitation,
   comments: set Comment
}{
   startsAt >= 0 && endsAt >= 0
   canceled = "true" || canceled = "false"
   notifTrigger >= 0 && notifTrigger=< 4
}
```

```
--Comment
sig Comment {
   author: one RegUser,
   toEvent: one Event
}

--Calendar
sig Calendar {
   owner: one RegUser,
   events: set Event
}

--Location
sig Location {
   isIndoor: one String,
   weatherForecast: one WeatherForecast
}{ isIndoor = "true" || isIndoor = "false" }

--WeatherForecast
sig WeatherForecast{
   value: Int,
   toString: String
} { (value=0 && toString="SUN") ||
   (value=1 && toString="CLOUDS") ||
   (value=2 && toString="RAIN") ||
   (value=3 && toString="THUNDERSTORM") ||
   (value=4 && toString="STORM")
}

--Notification
abstract sig Notification {
   relatedToEvent: one Event,
   addressedTo: one User
}

--Notification of weather changes
sig WeatherChangedNot extends Notification {}

--OtherChanged: event in which the user was invited was canceled/modified
sig NotMyEventChangedNot extends Notification {}

--InvitationNot
sig InvitationNot extends Notification {}
```

```
--Invitation
sig Invitation {
    status: one String,
    sentTo: one User,
    toEvent: one Event
}{ status="ACCEPTED" ||
   status="DECLINED" ||
   status="PENDING" ||
   status="CANCELED"
}

-----------------------------------
--PREDICATES

//Force good weather and event not canceled
pred goodWeatherNotCanceled [] {
    #WeatherChangedNot = 0
    all e:Event | e.canceled = "false" && e.notifTrigger = 1
}

//WORLD1: A NRU is invited
pred world1 [] {
    goodWeatherNotCanceled []

    #NotRegUser = 1
    #Comment  = 0
}

// WORLD2: NRU invites another NRU, NRU follows NRU, invited NRU leaves a comment
pred world2 [] {

    all u:User | u.email = "user1@me.com" || u.email = "user2@me.com"
    goodWeatherNotCanceled []

    #User = 2
    #Comment = 1
    #Event = 1
    #Invitation = 1
    #NotMyEventChangedNot = 0

    some u1,u2:User | u1.followers = u2
    all e:Event | e.comments.author != e.owner

}
```

```
//WORLD3: Declined invitation
pred world3 [] {
   all u:User | u.email = "user1@me.com" || u.email = "user2@me.com"
   goodWeatherNotCanceled []

   #User = 2
   some u:User | u.invitations.status  = "DECLINED"

}

//WORLD4: Multiple events, multiple users
pred world4 [] {

   #User = 2
   #Event = 3
   #Notification = 0
   #Invitation = 0
   #Comment = 0

   all u:User | #u.events >= 1

}


//WORLD5: WeatherChanged, owner and guest notified
pred world5[] {
   #User = 2
   #Invitation = 1
   #WeatherChangedNot > 1
   #Comment = 0
}

//True iff the set of followed users, after have followed a new user, is equal to the
//previous set plus the user just followed
pred followFriend[ru1,ru1',ru2:RegUser]{
   ru1'.followed=ru1.followed+ru2
   ru1'.followers=ru1.followers
   ru1'.emailConfirmed=ru1.emailConfirmed
   ru1'.calendar=ru1.calendar
   ru1'.events=ru1.events
}
```

```
//True iff the set of followed users, after have unfollowed a new user, is equal to the
//previous set minus the user just followed
pred unfollowFriend[ru1,ru1',ru2:RegUser]{
   ru1'.followed=ru1.followed-ru2
   ru1'.followers=ru1.followers
   ru1'.emailConfirmed=ru1.emailConfirmed
   ru1'.calendar=ru1.calendar
   ru1'.events=ru1.events
}

//True iff the set of events in the user's calendar and set of user's events, after have added a new event, are equal to the
//previous sets plus the event just created
pred addEvent[ru1,ru1':RegUser,e:Event]{
   ru1'.followed=ru1.followed
   ru1'.followers=ru1.followers
   ru1'.emailConfirmed=ru1.emailConfirmed
   ru1'.calendar.events=ru1.calendar.events+e
   ru1'.events=ru1.events+e
}
//True iff the set of events in the user's calendar, after have canceled a new event, are equal to the
//previous sets minus the event just created. Instead, the number of user's events set remains equal, the only thing that
//changes is the attribute canceled that switches from "no" to "yes"
pred cancelEvent[ru1,ru1':RegUser,e:Event]{
   ru1'.followed=ru1.followed
   ru1'.followers=ru1.followers
   ru1'.emailConfirmed=ru1.emailConfirmed
   ru1'.calendar.events=ru1.calendar.events-e
   ru1'.events=ru1.events
   ru1'.events.canceled="yes"
}

-----------------------------------
--FACTS (22)

//RU and NRU  complete the set Users
fact userSet {
   User = NotRegUser + RegUser
}

//Not two users with the same email
fact  emailUnique {
   not some disj u1,u2:User | u1.email = u2.email
}
```

```
//Consistency of relation invitation<->user
fact invitationUserConsistency {
   all i:Invitation, u:User | i.sentTo = u iff i in u.invitations
}

//Consistency of relation notification<->user
fact notificationUserConsistency {
   all n:Notification, u:User | n in u.notifications iff n.addressedTo = u
}

//Consistency of relation RegUser<->Event
fact eventUserConsistency {
   all u:RegUser, e:Event | e in u.events iff e.owner = u
}

//Consistency of relation RegUser <-> Calendar
fact calendarUserConsistency {
   all u:RegUser, c:Calendar | u.calendar = c iff c.owner = u
}

//Consistency or relation Event<->Invitations
fact invitationEventConsistency {
   all i:Invitation, e:Event | i.toEvent = e iff i in e.invitations
}

//Consistency or relation Comments<->Events
fact commentsEventsConsistency {
   all e:Event, c:Comment | c in e.comments iff e in c.toEvent
}

//Followers/Followers
fact followingRules {
   all disj u1,u2: RegUser | u2 in u1.followed  iff u1 in u2.followers
   not some u: RegUser | u in u.followed
   not some u: RegUser | u in u.followers
}

//A NRU or a RU with not accepted email can have only PENDING or CANCELED invitations
fact notFullyRegisteredConstraints {
 all u:NotRegUser, i:Invitation | i in u.invitations implies (i.status = "PENDING" || i.status= "CANCELED")
 all u:RegUser, i:Invitation |
       (i in u.invitations && u.emailConfirmed = "false")  implies (i.status= "PENDING" || i.status= "CANCELED")
}
```

```
//A RU  with not confirmed email has not own events and no events in his calendar
fact notAcceptedEmail {
   all u:RegUser | u.emailConfirmed = "false" implies #u.events = 0
}

//Constraints on Calendar
fact calendarContents {
   //Calendar contains all not canceled owned events
   all e:Event, u:RegUser | (e in u.events && e.canceled = "false") implies
      e in u.calendar.events
   //Calendar contains all events to which the user is invited with status ACCEPTED
   all i:Invitation, u:RegUser | (i.sentTo=u && i.status = "ACCEPTED" )  implies
      i.toEvent in u.calendar.events
   //Calendar does not contain any other event
   not some e:Event, u:User |
      ( e in u.calendar.events &&
         (e not in u.events && e not in u.invitations.toEvent)
      )
   not some e:Event, u:User, i:Invitation|
      e in u.calendar.events && i.toEvent = e && i.sentTo = u&& i.status != "ACCEPTED"

   not some e:Event, u:User | e in u.calendar.events && e.canceled = "true"
}


//A user cannot invite itself to one of his events
fact notSelfInvitation {
   all e:Event | e.owner not in e.invitations.sentTo
}

//If the event is canceled all the invitations must have CANCELED as status
fact canceledEvent {
   all e:Event, i:Invitation | (i.toEvent = e && e.canceled="true") implies
       i.status = "CANCELED"
   all e:Event, i:Invitation | (i.toEvent = e &&  i.status = "CANCELED") implies
      e.canceled = "true"
}
```

```
//A notification can be related to events owned by the user or that have the user as guest with status = PENDING | ACCEPTED | CANCELED
fact notificationEventDomain {
   all n:Notification, u:User, i:Invitation |
      ( n in u.notifications implies (
         ( n.relatedToEvent in u.events  ) || (n.relatedToEvent in u.invitations.toEvent)
         )
      ) && (
         (n.relatedToEvent = i.toEvent && i in u.invitations) implies
            ( i.status="PENDING" ||  i.status ="ACCEPTED" ||  i.status ="CANCELED")
      )
}

//Not two invitations for the same event to the same user
fact invitationUnique {
   all disj  i1,i2:Invitation | (i1.sentTo = i2.sentTo) implies i1.toEvent !=i2.toEvent
}

//Invitation notifications constraints
fact invitationNotConstraints {
   //Exists the related invitation
   all inot: InvitationNot | inot.relatedToEvent in inot.addressedTo.invitations.toEvent

   //Not two invitation notifications for the same event
   all disj inot1,inot2 :InvitationNot | inot1.relatedToEvent != inot2.relatedToEvent

}

//Weather notifications can exist only if (EVENT IS OUTDOOR)  && (ACTUALWEATHER equal or worse NOTIFTRIGGER)
fact weatherNotExistence {
   all w:WeatherChangedNot
      | w.relatedToEvent. location.weatherForecast .value>=  w.relatedToEvent.notifTrigger
      && w.relatedToEvent.location.isIndoor = "true"
   //Reverse condition
   all e:Event | e.notifTrigger  =< e.location.weatherForecast.value implies (some n:  WeatherChangedNot | n.relatedToEvent = e)
}

//NOTE1: We should say that there must be as many WeatherChangedEvents as are the times the weather status
//          goes above trigger .. but we should store the weather changes somewhere and this would add unuseful
//          complexity to the model.

//NotMyEventChangedNot only for events to which the user is invited
fact notMyEventNots {
   all n: NotMyEventChangedNot | n.relatedToEvent in n.addressedTo.invitations.toEvent
}
```

```
//NOTE2: We should say that there must be as many NotMyEventChangedNot as are the times the event date/location
//          change. We don't do it for the same reason of NOTE1

//NOTE3: We should also say that if a WeatherChangedNotification or a NotMyEventChangedNot is sent to one guest
//          all other guests with not declined invitation should receive the same notification.
//          We could also write complex constraints on the number of notifications received by two guests of the same event
//          In order to simplify we only write the folllowing 'weak' constraints:
fact notificationsConstr {
   //If a user has a weather notification for an event e, the owner and all the other guests of the event should have at least
   //one weather notification regarding the same event
   all u:User, n:WeatherChangedNot, e:Event |(n in u.notifications && u.notifications.relatedToEvent = e
   && e.canceled = "false") implies (
         all u:User | (u in e.invitations.sentTo || u = e.owner) implies
               (some t:WeatherChangedNot| t.addressedTo = u && e in t.relatedToEvent)
   )

   //If a user has a NotMyEventChanged notification for an event e, all the other guests should have at least
   //one NotMyEventChanged notification regarding the same event
   all u:User, n:NotMyEventChangedNot, e:Event |(n in u.notifications && u.notifications.relatedToEvent = e
   && e.canceled = "false") implies (
         all u:User | u in e.invitations.sentTo  implies
               (some t:NotMyEventChangedNot| t.addressedTo = u && e in t.relatedToEvent)
   )
}

//Time constraints
fact timeConstraints {
   //Positive duration of events
   all e:Event | e.endsAt > e.startsAt

   all disj e1,e2:Event,  u: RegUser| (e1 in u.calendar.events && e2 in u.calendar.events) implies
            not (
            ( e2.startsAt >= e1.startsAt && e2.startsAt  <e1.endsAt) ||
            ( e2.endsAt  > e1.startsAt  && e2.endsAt  <= e1.endsAt )
            )
   //The constraint above allows an event to start in the same instant when another event finishes
}

//Only guests and owner of an event can comment that event
fact commentsAuthorsConstraint {
   all c:Comment | c.author in c.toEvent.invitations.sentTo || c.author in c.toEvent.owner
}
```

```
//Not independent  Location | WeatherForecast (To avoid atoms not envolved in a relation)
fact notDateTimeAndLocationIndependent {
   all l:Location | l in Event.location
   all wf:WeatherForecast | (wf in Location.weatherForecast )
}

-----------------------------------
--ASSERTIONS

run world1 for 2
run world2 for 2
run world3 for 2
run world4 for 10
run world5 for 3

//Two users can't own the same event
assert ownerTest {
   not some e: Event, disj u1,u2:RegUser | e in u1.events && e in u2.events
}
check ownerTest for 5

//If a User does not confirm his email he cannot have any event in his calendar
assert calVoid {
   not some u:RegUser | u.emailConfirmed = "false" && #u.calendar.events > 0
}
check calVoid

//A User cannot have an invitation notification related to one of his events
assert invitationNotification {
   not some u:RegUser, n:InvitationNot | n in u.notifications && n.relatedToEvent.owner = u
}
check invitationNotification

//When I follow and the unfollow a user the state doesn't change
assert followUnfollowUserDoesNotChangeState{
   all ru1,ru2,ru1',ru1'': RegUser | ru2 not in ru1.followed and followFriend[ru1,ru1',ru2]
and
   unfollowFriend[ru1,ru1',ru2] implies ru1.followed = ru1''.followed
}
check followUnfollowUserDoesNotChangeState

//When I create an event and then I delete the event, the calendar does not change. Instead, in the sets of user events the added event
// will be kept, but its attribute canceled is switched from "no" to "yes"
assert addCancelEventDoesNotChangeState{
   all ru1,ru1',ru1'': RegUser,e:Event | e not in (ru1.calendar.events) and ru1.events.canceled="yes" and addEvent[ru1,ru1',e]
and
   cancelEvent[ru1,ru1',e] implies ru1.calendar.events = ru1''.calendar.events
}
check addCancelEventDoesNotChangeState
```

```
//check if a user can invite both register and not register users
assert inviteBothRegisteredAndNotRegistered{
   all i:Invitation | i.sentTo in NotRegUser || i.sentTo in RegUser
}

check inviteBothRegisteredAndNotRegistered
```

And here the report of the Alloy Analyzer :

```
Executing "Run world1 for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  5148 vars. 362 primary vars. 11912 clauses. 44ms.
  Instance    found. Predicate is consistent. 120ms.

Executing "Run world1 for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  5148 vars. 362 primary vars. 11912 clauses. 37ms.
  Instance    found. Predicate is consistent. 33ms.

Executing "Run world2 for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  5296 vars. 390 primary vars. 12307 clauses. 54ms.
  Instance    found. Predicate is consistent. 118ms.

Executing "Run world3 for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  5276 vars. 388 primary vars. 12254 clauses. 40ms.
  Instance    found. Predicate is consistent. 27ms.

Executing "Run world4 for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  94410 vars. 3410 primary vars. 201922 clauses. 6319ms.
  Instance    found. Predicate is consistent. 373ms.

Executing "Run world5 for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 SkolemDepth=1 Symmetry=20
  9480 vars. 603 primary vars. 21427 clauses. 70ms.
  Instance    found. Predicate is consistent. 52ms.

Executing "Check ownerTest for 5"
  Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
  23161 vars. 1220 primary vars. 50593 clauses. 208ms.
  No counterexample found. Assertion may be valid. 8ms.

Executing "Check calVoid"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9552 vars. 606 primary vars. 21546 clauses. 62ms.
  No counterexample found. Assertion may be valid. 13ms.

Executing "Check invitationNotification"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9542 vars. 609 primary vars. 21518 clauses. 51ms.
  No counterexample found. Assertion may be valid. 14ms.
```

```
Executing "Check followUnfollowUserDoesNotChangeState"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9802 vars. 615 primary vars. 22228 clauses. 85ms.
  No counterexample found. Assertion may be valid. 4ms.

Executing "Check addCancelEventDoesNotChangeState"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  10044 vars. 633 primary vars. 22778 clauses. 104ms.
  No counterexample found. Assertion may be valid. 5ms.

Executing "Check inviteBothRegisteredAndNotRegistered"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9493 vars. 606 primary vars. 21403 clauses. 78ms.
  No counterexample found. Assertion may be valid. 7ms.
```

8 Worlds Generated

8.1 Not registered User World

In this world, NotRegUser, who is not registered, is invited to an Event created by RegUser, who is registered.
NotRegUser has an invitation to that event with status 'PENDING' and a Invitation notification related to the same event.
Some main constraints are satisfied:
- Not two users with the same email
- A NotRegUser can only have 'PENDING' or 'CANCELED' invitations
- A NotRegUser has no calendars / own events
- When there?s an Invitation there must be an associated InvitationNotification
- A not-canceled event appears in its owner's calendar
- If the event is not canceled there's no invitation to that event with status 'CANCELED'

8.2 Invitation World

In this world, RegUser1 follows RegUser0.
RegUser1 has created an Event and has invited RegUser0 to it.
Therefore RegUser0 receives an InvitationNotification and decides to ACCEPT the invitation.
Now the event appears on its Calendar1.
Eventually, RegUser0 leaves a Comment to the event. Some important constraints are satisfied:
- If RegUser1 follows RegUser0, RegUser0 must be in RegUser1's followers
- Only users invited to an event can leave a comment to that event
- If an invitation is ACCEPTED, the related event must appear in the guest's calendar
- Not two invitations for the same event addressed to the same user

8.3 Decline Invitation World

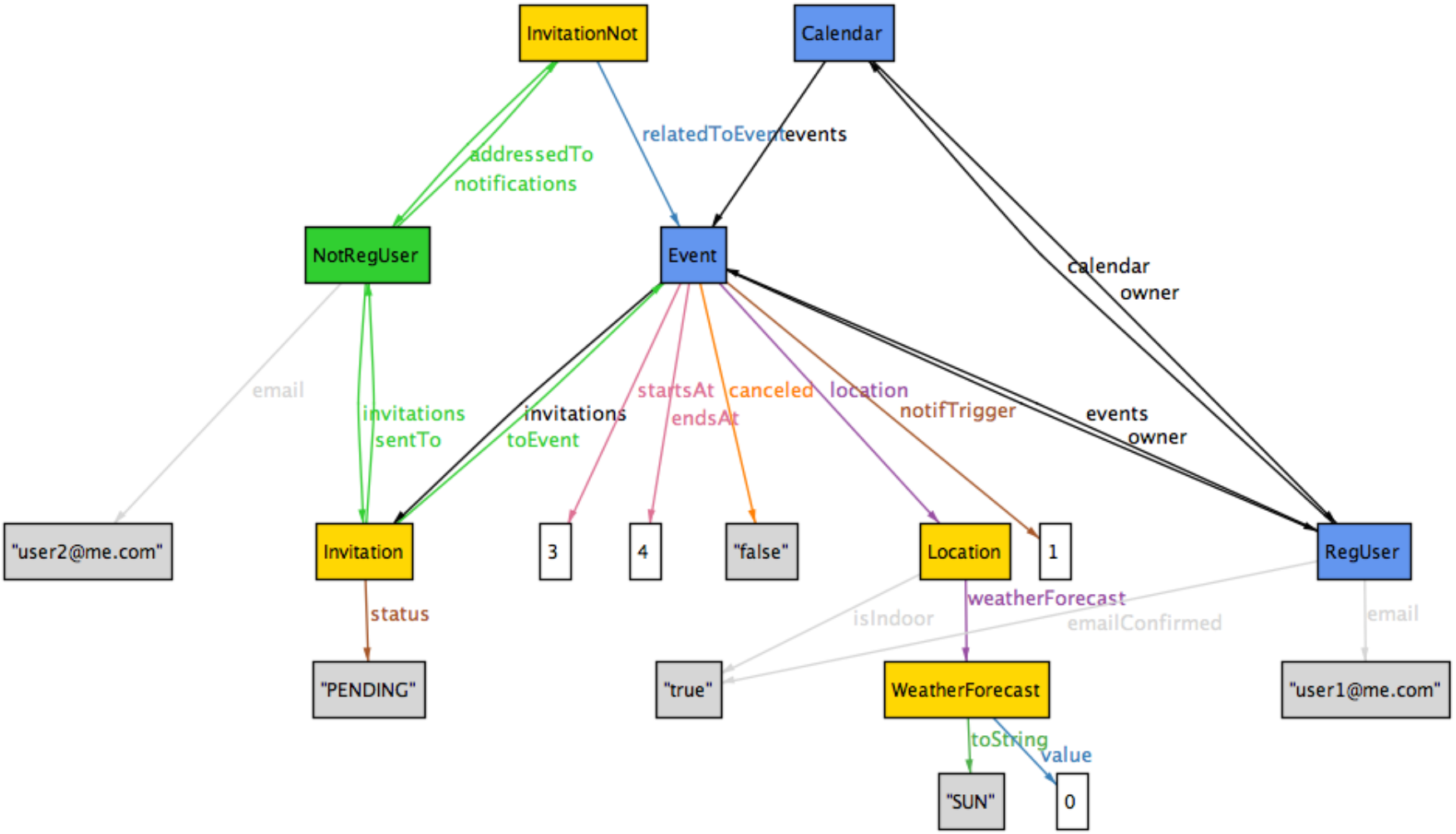
In this world, RegUser0 invites RegUser1 to one of his events.
RegUser1 decides to DECLINE the invitation. Thus, the event won't appear in RegUser1's Calendar. Remarkable constraints:
- If an invitation is DECLINED, the related event must not added to the invited user's calendar

8.4 Events World

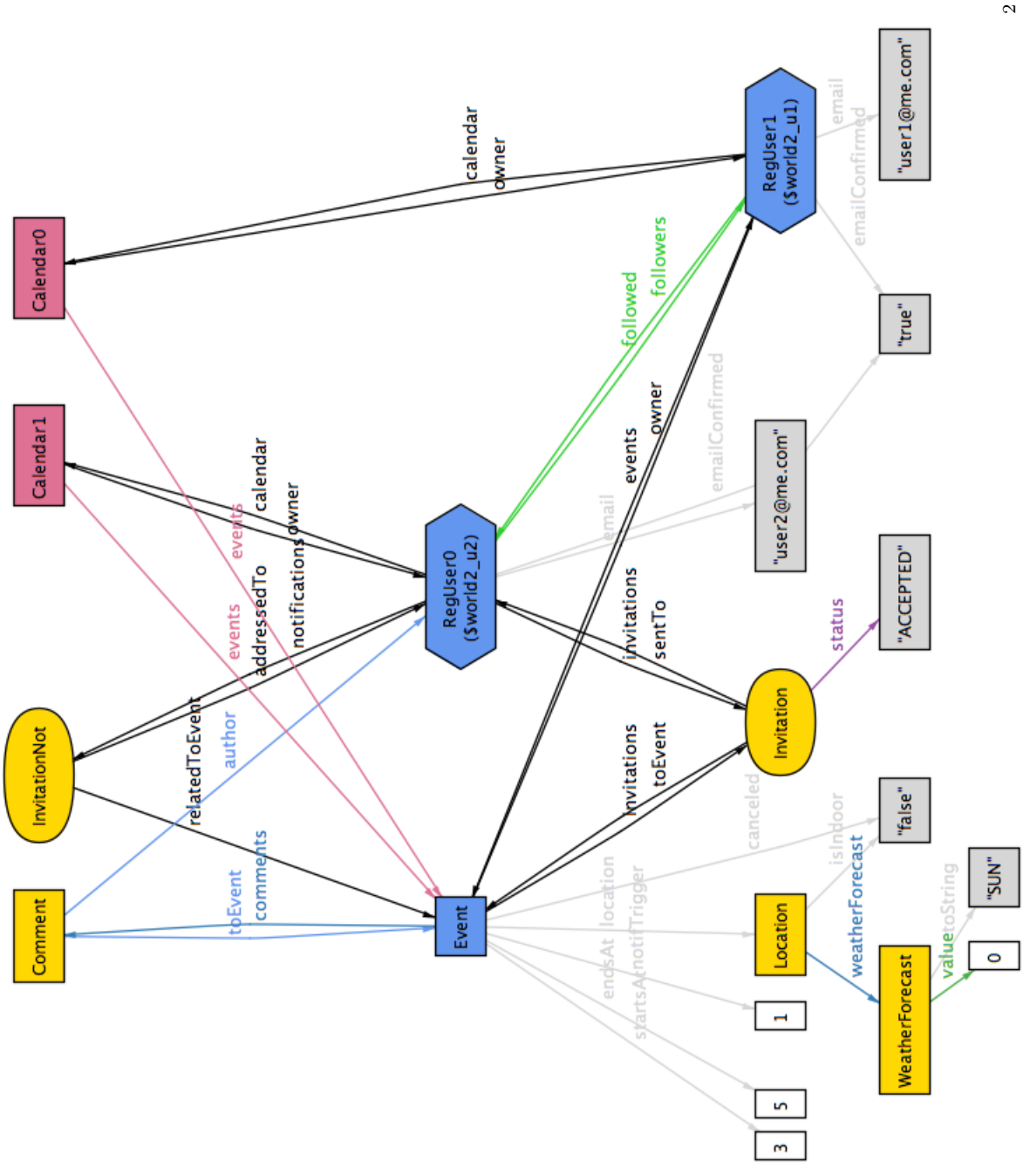
In this world RegUser0 has created two events:
- Event1 starts at 5 and ends at 6
- Event2 starts at 6 and ends at 7
Both events are in RegUser0's calendar.
Another user, RegUser1 had an event Event0 starting at 0 and ending at 3, but he decided to cancel it. Satisfied constraints:
- An event can be owned exactly by one user.
- The only possible time superposition for two not-canceled events e1 and e2 is e1.endsAt = e2.startsAt
- All not canceled events appear on their owner's calendar

8.5 Weather Changed World

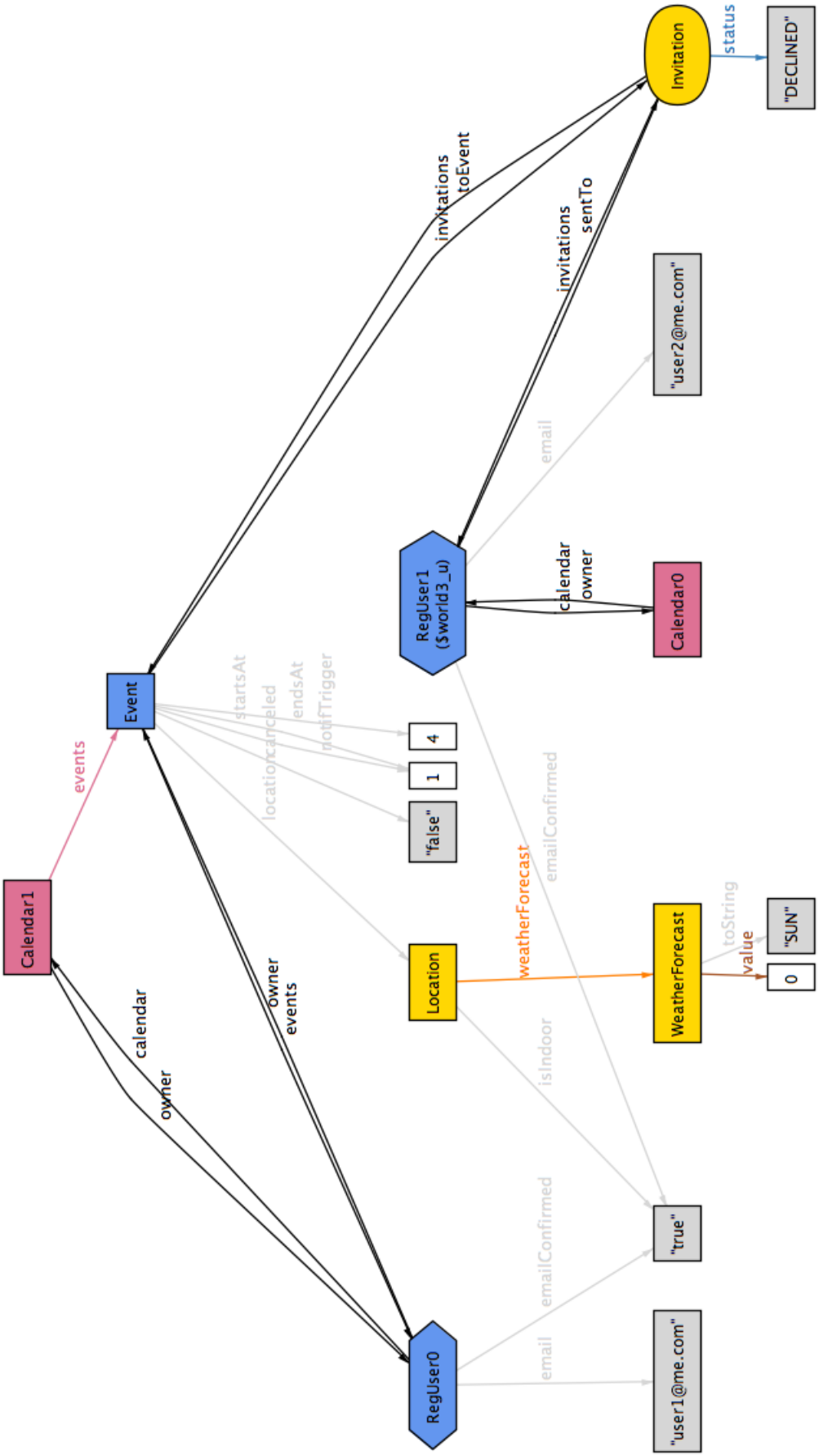
In this world RegUser0 has invited RegUser1 to his outdoor Event and RegUser1 has accepted. RegUser0 has set the notifTrigger to 'RAIN' (= 2). That means that both the owner and the guests will receive a notification if the weather turns to 'RAIN', 'THUNDERSTORM' or 'SNOW'.
In this case the weather turned to 'THUNDERSTORM' and both the owner and the only guest received a WeatherChanged notification.
Satisfied constraints:
- WeatherChangedNot sent only for outdoor events
- WeatherChangeNot only if weather is equal or worse than notifTrigger
- Owner and all guests (invitationStatus != DECLINED) receive the same kind of notifcation



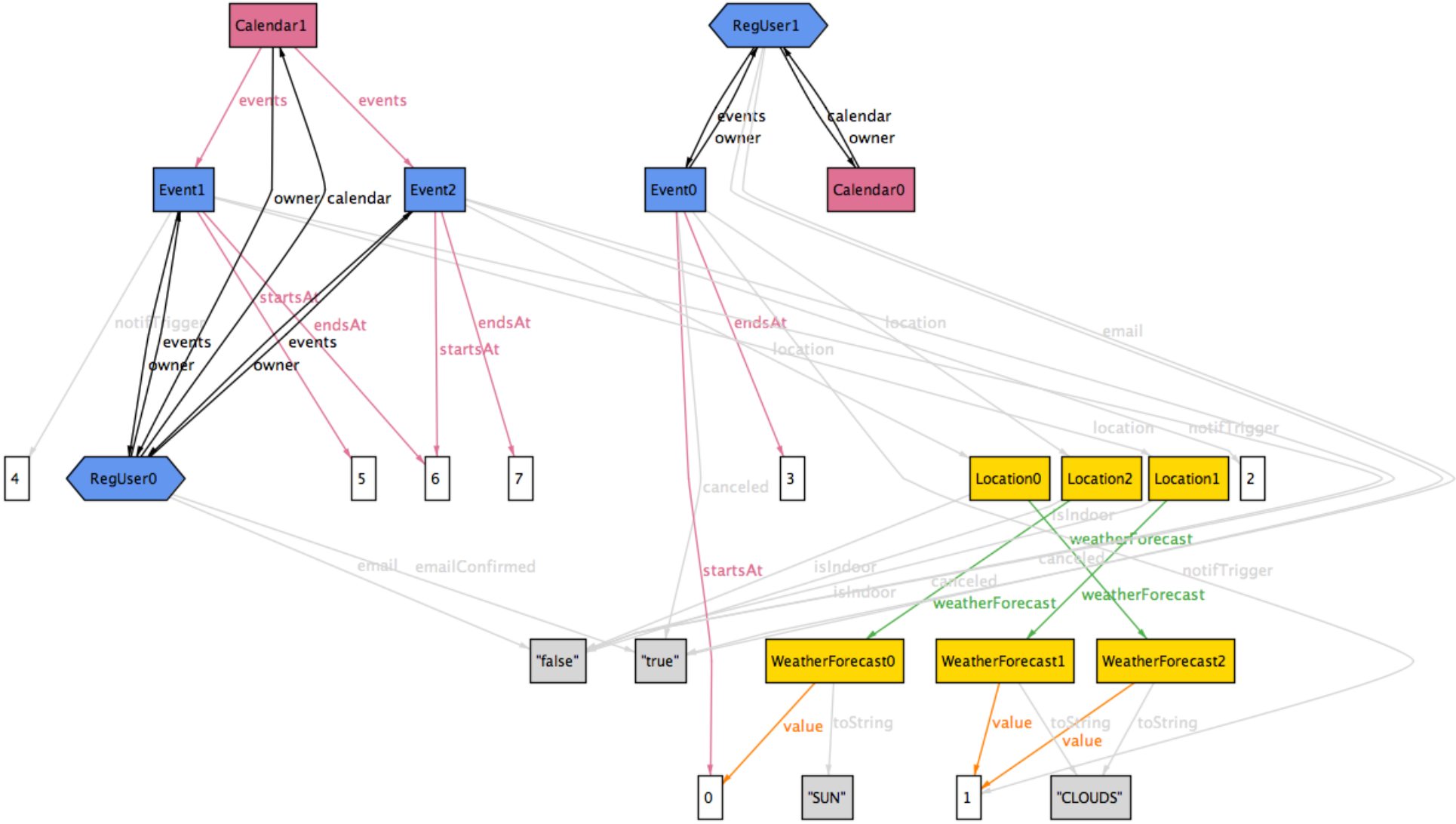
[1]Figure 1 : not registered user world



[2]Figure 2 : invitation world

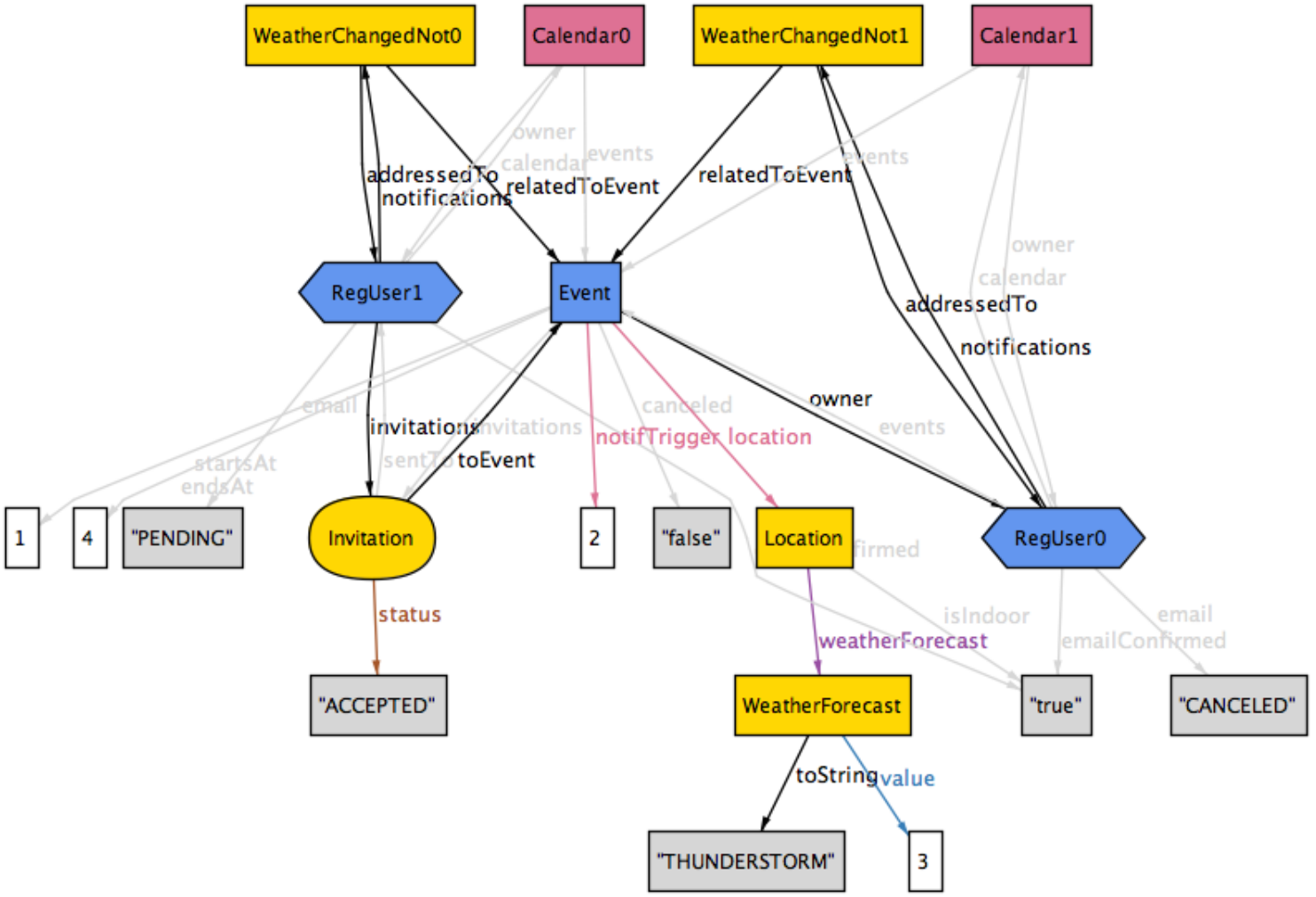


[3]Figure 1 : decline invitation world



[4]Figure 2 : concurrent event world

GENERATED



[5]Figure 1 : weather changed world

9 Used Tools

Tools we used to create this RASD document are the following :

- LaTeX: to write and format this document
- Pages : to draw extra figures and graphs
- Signavio Platform : to design Use Cases
- starUML : to design Class Diagrams and State Charts Diagram
- Visual Paradigm : to design Sequence Diagrams
- Blasmiq Mockup : to design web pages' mockup

10 References

- SE2 teacher Mirandola's lecture slides
- The Meaning of Requirements:
http : *//www.uml.org.cn/requirementproject/pdf/jackson_annals*97.*pdf*
- Non-functional Requirements :
http : *//en.wikipedia.org/wiki/Non* − *functional_requirement*