Requirements Analysis and Specification Document

Edoardo Scibona
Mat. 836884

November 17, 2014

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, acronyms and abbreviations . . . 3
- 1.4 Overview . . . 5

2 Overall description **6**
- 2.1 Product perspective . . . 6
- 2.2 Product functions . . . 6
- 2.3 User characteristics . . . 6
- 2.4 Constraints . . . 7
- 2.5 Assumptions and dependencies . . . 7
  - 2.5.1 Domain properties . . . 7
  - 2.5.2 Assumptions . . . 8
- 2.6 Proposed system . . . 8
- 2.7 Stakeholders identification . . . 9
- 2.8 Actors identification . . . 9

3 Requirements **10**
- 3.1 Functional requirements . . . 10
- 3.2 Non functional requirements . . . 11

4 Scenarios **12**

5 UML models **15**
- 5.1 Use cases . . . 15
  - 5.1.1 Overview . . . 15
  - 5.1.2 Sign up . . . 16
  - 5.1.3 Login . . . 17
  - 5.1.4 Create event . . . 18
  - 5.1.5 View event . . . 19
  - 5.1.6 Update event . . . 20
  - 5.1.7 Delete event . . . 21
  - 5.1.8 Accept invite . . . 22
  - 5.1.9 Decline invite . . . 23
- 5.2 Class diagram . . . 24

6 Alloy model **25**
- 6.1 Model code . . . 25
- 6.2 Generated worlds . . . 28

7 Document informations **29**
7.1 Effort 29
7.2 Changelog 29
7.3 Tools used 29

1 Introduction

1.1 Purpose

This document will serve as a reference point regarding the requirements, goals and objectives pertaining the MeteoCal project. It will be useful in delineating key aspects of the project before the start of the development phase.
One part of the intended audience for this document is represented by the customer, who will need to verify that his desires and requests have been correctly understood, check what assumptions were made and understand in what the delivered project will consist.
The other part of the intended audience are the developers, who prepared the document interacting with the customer, and that are now ready to use it as a reference during the development phase.
Apart from the main audience this document may be of interest to future users of the service or researchers who may be interested in the planning phase to gain a better understanding of the system.

1.2 Scope

We have to realize a system on behalf of the X software house.
The system, called MeteoCal, consists in a new kind of online calendar aimed at helping users plan their activities and events.
The registered users should be able to create, update and delete events.
While a user is creating an event he should add informations about the event such as the location, the date and whether the event will be held indoors or outdoors; the user should also be able to invite other users to the event in the event creation phase. Updating and deleting the event are operations that can only be performed by the event owner.
Invited users are able to accept or decline an invitation to an event but may also leave it pending.
The system, after an event is created, should enrich it, if possible, with weather informations and, in case of bad weather coinciding with an outdoor event, it should notify all event's participants one day in advance.

1.3 Definitions, acronyms and abbreviations

Guest a person using the service that isn't either logged in or registered.
A guest may only login or register.

User a person using the service that has logged in with his credentials and that is able to create and manage events.

Event user created activities that will be held in a specific place at a specific time and that are either indoor or outdoor.
Other users can be invited to events.

Event type the type that better describes the event.
There are only two types: outdoor or indoor.

Indoor event an event that will be held indoors and as such not affected by bad weather.

Outdoor event an event that will be held outdoors and as such may be affected by bad weather. The system should signal, if possible, this inconvenience to participants.

Event Owner the user that has created the event and the only one that can update or delete the event.

Invited user a user that has been invited to an event but has not yet accepted or declined the invitation.

Participant a user that has been invited to an event and has accepted the invitation signalling his intention to be present at the event.

Invitation request sent by a user to other users asking for their participation to an event.

Notification short message displayed by the system to users to signal something.

Bad weather notification message automatically displayed on day in advance to users invited to an outdoor event with bad weather.

Service the part of the system used by the users, namely the front-end

System the whole MeteoCal application, composed of front-end and back-end. This is the full product that needs to be developed.

Login the procedure through which a guest, entering his credentials, authenticates as a user.

Sign up the procedure through which a guest registers to the service creating a new account.

1.4 Overview

The structure of this document for the first two sections follows mostly the standard *IEEE Std 830-1993* but deviates from it in the last sections to better achieve the custom requests received for its structure and contents.

2 Overall description

2.1 Product perspective

The product which will be developed is a new system in itself. It will have its requirements but will not be subject to those of a legacy system.
The only interaction with external systems will be through the use of a weather API to enrich events' informations.

2.2 Product functions

We list here the major functions that the product will need to perform, these can be considered as goals for the system.

- A person should be able to register to the service becoming a user.
- A user can create an event becoming the event owner.
- The event owner can update an event.
- The event owner can delete an event.
- The event owner should add informations to an event.
- The event owner can invite other users to an event.
- Invited users should be notified of pending invitations.
- A user can accept an invitation to an event.
- A user can decline an invitation to an event.
- Users participating to an outdoor event with bad weather should be notified a day in advance.

2.3 User characteristics

There will not be a restricted or specialized category of users for this service, thus we expect an adoption by people from all over the world with different backgrounds, education, experience and technical expertise.
With this consideration in mind we think the service's ease of use and intuitive understanding will be of fundamental importance for its growth.

2.4 Constraints

The development of the product will be subject to the following constraints.

- The usage of the weather API
- The system must be realized using Java Enterprise Edition

2.5 Assumptions and dependencies

We illustrate in this section the assumptions that were made while thinking about the development of the product.
We subdivide this section into two subsections: domain properties, assertions that we think hold in the analysed world, and general assumptions about the system introduced to resolve unspecified behaviours.

2.5.1 Domain properties

- A user creating an event inputs the event informations correctly.
- An event is held in a specific place.
- An event is held at a specific time.
- A user cannot create different events in the same time frame.
- A user prioritizes his own events.
- Events are held in the future with respect to the time of creation.
- An event is held either indoors or outdoors.
- Users know the usernames of the people they want to invite.
- A user will not invite himself to his events.
- A user always participates to his events.
- Users can be invited to multiple concurrent events but will only accept one invitation among the possible ones.
- Weather forecast is reliable.
- Users will take note of the notification signalling bad weather.

2.5.2 Assumptions

- Newly registered users have no events and no invitations pending.
- Pending invitations will be removed after the event has ended.
- Once accepted invitations cannot be declined and vice versa.
- Deleted events are automatically removed from participants' calendars.
- Past events become read only but may still be deleted.
- Weather forecasts are checked periodically until the start of the event.
- Wrong usernames among the invited users are discarded.
- Usernames can contain only alphanumeric characters and underscores and are at most 20 characters long.
- Passwords must be at least 8 characters long with no restriction on the characters.
- The following parts of the events are unmodifiable after the creation: date, location, invited users, participants.
- The following parts of the events are modifiable by users after the creation: name, description.
- The event's name must not be empty.

2.6 Proposed system

We propose a web service with the following characteristics.

The home page will be displayed to guests shortly describing the service and inviting them to either register or login.
Users will have a calendar reporting the events that they will attend, quick access to event creation, a list of events that they have been invited to and still waiting for response and eventual notifications from the system.
Events in the calendar will lead to event pages which will feature the event's name, a short description, the event's type, location, date, participants and weather forecast if available.
The event page will also display the update and delete options to the event owner.

Users will be able to create events in the event creation page adding the aforementioned informations, except weather forecast, optionally inviting their friends and then confirming the creation of the event.

2.7 Stakeholders identification

We identify the following stakeholders.

Professor the professor commissioned the project and is interested not only in the final piece of software but also in the development process that includes both the documentation part and the coding part.
The professor will follow the project in order to evaluate the abilities shown by the student in the role of a developer.

Users the users are interested in a simple way to create and manage their events and also keep track of those of their friends.
They will also benefit from the weather forecasts provided automatically by the service and the eventual notifications warning them of bad weather in case of outdoor events.

Developer the developer is interested in completing the project in the given time frame and needs to demonstrate his abilities in doing so.
He is also concerned in realizing a system that caters effectively to users and their needs in managing events.

2.8 Actors identification

We identify the following actors that will interact with the system.

Guest the guest is a person that may want to use the service but in order to do so he must either login, if he already has an account, or register in case he is new to the service.

User the user is a person that is registered to the service and has logged in.
The user can create and manage his own events, view events he is participating in and be invited to other users events

3 Requirements

We will now identify the requirements that together with the domain properties described in the section 2.5.1 will permit the achievement of the goals described in the section 2.2.

3.1 Functional requirements

We list here again the goals for the system in bold and for each goal we list the functional requirements that the system needs to satisfy in order to achieve the goal.

- **A person should be able to register to the service becoming a user.**
  The system must provide a sign up feature that enables guests, through the input of a unique username and password to create a new user account for the service.
- **A user can create an event becoming the event owner.**
  The system must provide an event creation feature that enables users to create a new event based on informations such as name, description, location, date and participants.
- **The event owner can update an event.**
  The system must let the event owner use the update feature on his events in case he wants to modify some informations regarding an event.
- **The event owner can delete an event.**
  The system must let the event owner use the delete feature on his events in case he wants to delete an event.
- **The event owner should add informations to an event.**
  The system must require at the moment of the event creation some crucial informations from the user such as: the place where the event will take place, the time when the event will take place, the event's name and the event's type.
- **The event owner can invite other users to an event.**
  The system must implement a feature that lets users invite other users to the events they are creating.

- **Invited users should be notified of pending invitations.**
  The system must alert, through a notification, the users that they were invited to a certain event.

- **A user can accept an invitation to an event.**
  The system must implement a feature that lets users accept the invitations that they receive.

- **A user can decline an invitation to an event.**
  The system must implement a feature that lets users decline the invitations that they receive.

- **Users participating to an outdoor event with bad weather should be notified a day in advance.**
  The system should check the weather forecasts, if available, and for outdoor events it should notify the participants in case of bad weather one day before the event.

3.2 Non functional requirements

We list here some non functional requirements that the system should satisfy.

Ease of use Given the description of the users' characteristics in the section 2.3 and considering the global audience of the service we think that users should be able to navigate and use the service with ease.
They should readily understand what the service offers and how they can accomplish those functions.

Stability The system should be stable and always accessible offering thus a seamless experience to its users and guests. The data and calendars of the users should be well preserved and not at risk of loss.

Maintainability Developers should be able to maintain the system with ease.

Portability Given today's ubiquitous computing and growing mobile's market we think that the service should be easily accessible from any type of device. Being this a web based service any device with a web browser should be able to access the service although further optimizations may be required for some platforms.

Security The users' passwords should not be stored in plain text but using cryptographically secure methods.

4 Scenarios

We list here some example scenarios regarding MeteoCal usage.

- **Alice discovers the service through a friend's referral**
  Alice wants to throw a party in three weeks and is undecided on how she is going to get all her friends together.
  She asks Bob for advice and he tells her about the MeteoCal service, recommending that she should check it out.
  Alice goes on the website, reads the informations displayed in the home page and takes into account the option of using MeteoCal to plan her event.
  Having more urgent things to do she does not immediately register and leaves the homepage.

- **Alice registers as a new user**
  Alice has evaluated different alternatives but found MeteoCal to be the best one also because many of her friends, such as Bob, are already users.
  She returns to the service's homepage and clicks the sign up button to register, a page loads asking for a user name and a password and Alice inputs her user name, *anna_sterling89*, and her password and completes the sign up procedure.
  She is now registered to the service and can fully use its functions.

- **Alice creates a new event**
  Alice creates her birthday event using the *New event* page. She inputs the event name, *Alice super birthday party*, a brief description, the event date, and her house address as a location; since the party will be held in her house she chooses the indoor option as the event type.
  She then proceeds to invite her friends adding their user names to the list of invited users. Finally she confirms the event creation.

- **Alice updates the event**
  Alice, to spice things up, decided that her birthday will be a costume party, she needs thus to inform her friends.
  She goes to her birthday's event description page, chooses the update option and updates the description recommending everyone to come masked and finally confirms the update.

- **Bob accepts an invitation**
  Bob has been invited to Alice's party, he is eager to go to it and ac-

cepts the invitation that is displayed in his personal page, he is now a participant and this event is added to his calendar.

- **Bob inputs wrong login informations**
  Bob wants to see what events he will be attending next week and goes to the service to refresh his memory.
  He needs to login and inputs his username correctly but makes a mistake while typing his password.
  After confirming the login action the system displays a message stating that either the username or the password are invalid.
  Bob proceeds then to login again, this time successfully, and views his calendar.

- **Lori has to choose an available username**
  Lori discovers the MeteoCal service and, excited about the future opportunities, decides to register.
  In the registration page she chooses as her username simply *lori*, unfortunately this username is already taken and the page displays a brief error message stating so.
  Lori is a little disappointed but notwithstanding this she quickly comes up with the username *loriWHamer*, which is available, inserts her password and completes the registration.

- **Lori creates an outdoor event**
  Lori wants to go run with Carl at the city's central park. She creates the event, adds the information, sets its type to outdoor and invites Carl to it.

- **Carl is notified of bad weather**
  Carl has received the invite to Lori's running session, he willingly accepted the invitation, and got this event added to his calendar.
  A day before the running session with Lori the system notifies him of bad weather in his city the next day.
  Carl is thankful for the system notification service because he is coming back from another city and would not have thought to check the weather before the event.

- **Lori deletes an event**
  Apparently Lori was struck by some bad luck and her running session will have to wait.
  She decides to delete the event hoping to schedule a new one in the future, she thus proceeds to the event page and selects the deletion

option, she confirms her intentions and deletes the event. The event was deleted and removed from the calendars of Lori and Carl.

- **Lori and Carl are not notified of bad weather**
  Lori plans a new event for the following week to go hiking in the mountains and invites Carl to it.
  During the event creation she adds the location of the event but unfortunately the system cannot get a weather forecast for the event. Lori still remains confident that the system will get a weather forecast before the event and will notify them as needed.
  The mountains she chose are a remote region and no weather data becomes available in the week before the event and thus the system is not able to perform the weather check.
  Lori and Carl confident that the weather will be good reach the mountains for their hiking activity but sadly the sky is overcast and, hearing rumbles, they decide to spend the rest of the day in their cabin.

5 UML models

5.1 Use cases

Starting from the scenarios described in section 4 we are going to generalize them and distil a list of use cases that delve deeper into the event flows of the system.

5.1.1 Overview

In the following figure we show the general use case diagram that contains the use cases that we are going to specify in the following sections.

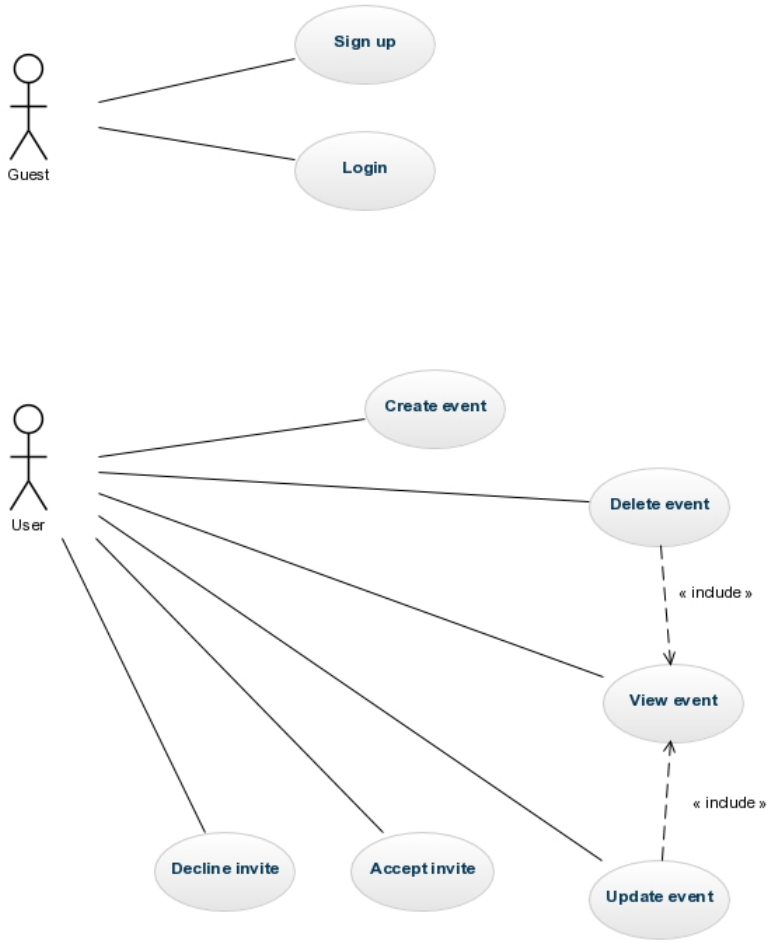


Figure 1: Use cases overview

5.1.2 Sign up

Actors	Guest
Pre-conditions	The guest is not a returning user.
Events flow	1. The guest arrives on the home page of the MeteoCal service. 2. He chooses to sign up which requires a username and a password. 3. He inputs a username and the password two times to verify it. 4. He confirms his will to sign up to the service. 5. The system checks whether the username is available and the passwords coincide. 6. If the system does not find any errors the procedure terminates correctly.
Post-conditions	The guest now has an account on the service and has become a user. A new calendar and user page were created from him.
Exceptions	If the username is not available the system will prompt the guest to choose a new one. If the two passwords do not coincide the system will prompt the guest to type them again.

5.1.3 Login

Actors	Guest
Pre-conditions	The guest is a returning user.
Events flow	1. The guest arrives on the home page of the Meteo-Cal service. 2. He chooses to log in and is asked for his username his password. 3. He inputs the username and the password. 4. He confirms his will to log in to the service. 5. The system checks if the combination of username and password are valid. 6. If the system does not find any errors the user is brought to his personal page and the procedure terminates correctly.
Post-conditions	The guest is now authenticated as a user on the service and can fully use the service's features.
Exceptions	If the combination of username and password are not valid the system will prompt the guest to input his credentials again.

5.1.4 Create event

Actors	User
Pre-conditions	The user is logged in and wants to create a new event.
Events flow	1. The user chooses to create a new event. 2. The service asks for the following event's informations. The forms with "*" are required. • Name* • Description • Date* • Location* • Type* • Invited users. 3. The user fills the required forms and the optional forms that he is interested in. 4. He confirms his will in creating the event. 5. The system checks if the event is in conflict with previously scheduled events and if the event's information is good. 6. If the system does not find any errors the event is created and added to the user's calendar. The user is brought back to his personal page. 7. The system sends the invitations to eventual participants.
Post-conditions	The event is now created and added to the user's calendar. The user becomes the event owner. Eventual participants were notified
Exceptions	If creating the event would lead to a conflict the system alerts the user and cancels the procedure.

5.1.5 View event

Actors	User
Pre-conditions	The user is logged in and wants to view a previously scheduled event.
Events flow	1. The user chooses from his calendar the event that he wants to see. 2. The service displays the event's page. 3. If the user is also the event's owner the service displays the update and delete options. 4. The user looks at the informations presented in the event's page. 5. The user may return to his personal page.
Post-conditions	The user has reviewed the event's informations.
Exceptions	None.

5.1.6 Update event

Actors	User
Pre-conditions	The user is logged, he is the event owner and wants to modify a previously scheduled event.
Events flow	1. The user chooses from his calendar the event that he wants to update. 2. The service displays the event's page with the update and delete options. 3. The user chooses the update option. 4. The user updates the modifiable informations. 5. The user confirms the update. 6. The system checks for errors in the modified forms and, if no errors are found, saves the event. 7. The system brings the user to the event's page displaying the updated event's information.
Post-conditions	The user has updated some of the event's informations.
Exceptions	If the system finds some errors it does not save the updated informations and notifies the user.

5.1.7 Delete event

Actors	User
Pre-conditions	The user is logged, he is the event owner and wants to delete a previously scheduled event.
Events flow	1. The user chooses from his calendar the event that he wants to delete. 2. The service displays the event's page with the update and delete options. 3. The user chooses the delete option. 4. The user confirms that he wants to delete the event. 5. The system proceeds to delete the event. 6. The system removes the event from the user's calendar and from the eventual participants' calendar. 7. The system removes any pending invitation. 8. The system notifies the eventual participants of the deletion. 9. The system brings the user back to his personal page.
Post-conditions	The user has deleted the event. The event is removed from the user's calendar and from the eventual participants' calendar, the participants are also notified.
Exceptions	None.

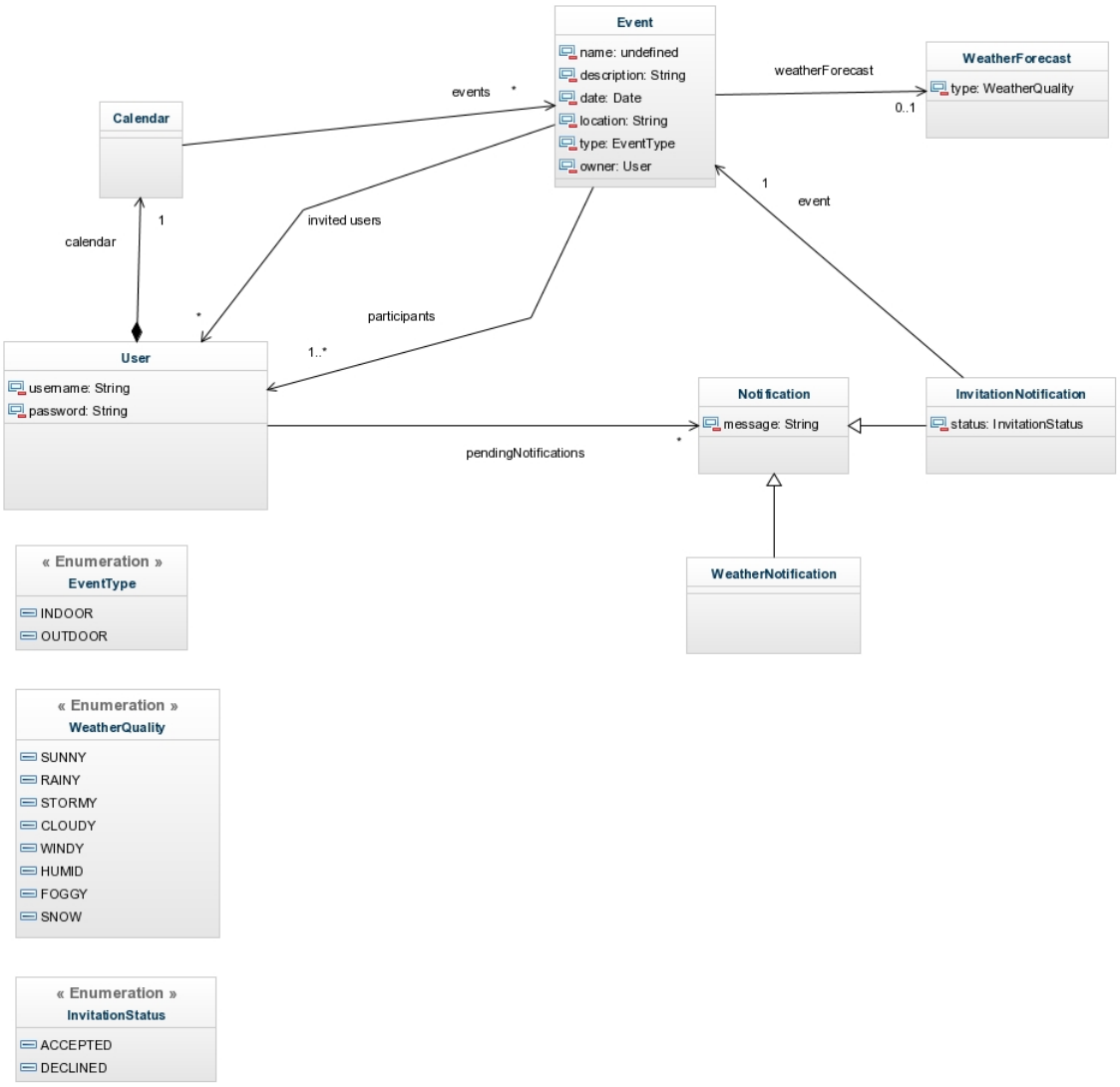
5.1.8 Accept invite

Actors	User
Pre-conditions	The user is logged in and has an event's invitation pending.
Events flow	1. The user sees a pending invite to an event. 2. The user decides to go to the event. 3. The user chooses the accept option. 4. The system checks for conflicting scheduling for the user. 5. If it does not find any conflicts the system lists the user as a participant to the event, removes the pending invite and adds the event to the user's calendar.
Post-conditions	The user has been added as a participant to the event. The invite was removed. The event was added to the user's calendar.
Exceptions	If the system finds some conflicts in the scheduling it alerts the user that he cannot participate to the event.

5.1.9 Decline invite

Actors	User
Pre-conditions	The user is logged in and has an event's invitation pending.
Events flow	1. The user sees a pending invite to an event. 2. The user decides not to go to the event. 3. The user chooses the decline option. 4. The system removes the pending invite.
Post-conditions	The invite was removed.
Exceptions	None.

5.2 Class diagram



6 Alloy model

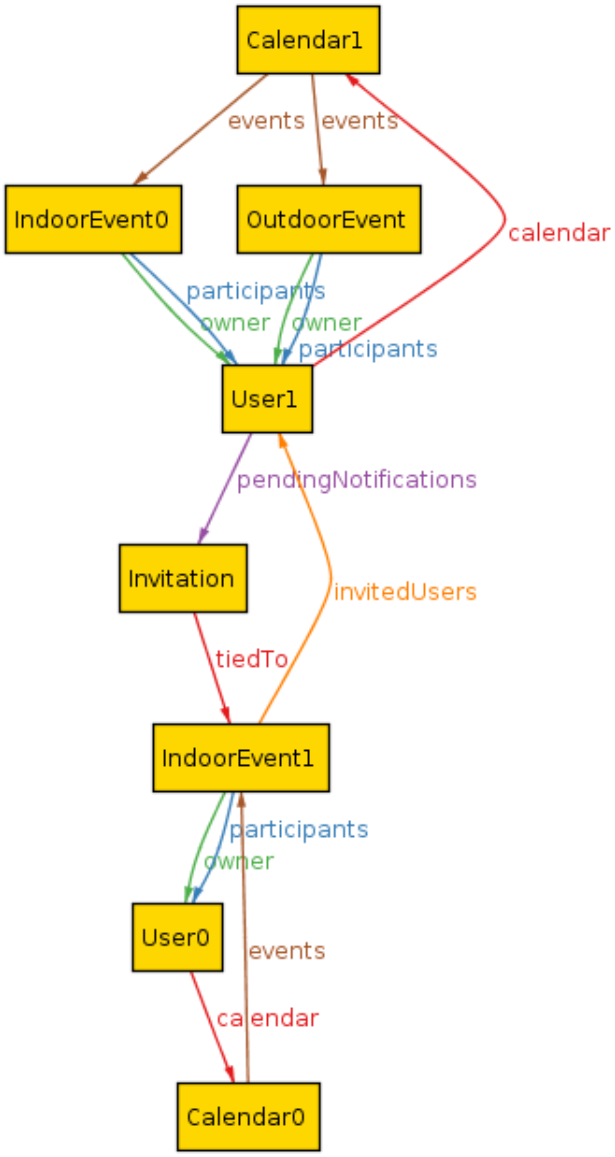
6.1 Model code

```
module meteocal

//Signatures

sig User {
        calendar: one Calendar,
        pendingNotifications: set Notification
}

sig Calendar {
        events: set Event
}

sig Notification {

}

sig Invitation extends Notification {
        tiedTo: one Event
}

sig WeatherAlert extends Notification {
        tiedTo: one OutdoorEvent
}

abstract sig Event {
        owner: one User,
        invitedUsers: set User,
        participants: some User
}

sig IndoorEvent extends Event {

}

sig OutdoorEvent extends Event {
```

```
}

//Facts

fact userProperties {
        //Users cannot have the same calendar
        all disj u1, u2: User | u1.calendar  != u2.
           calendar
        //Users cannot have the same notifications
        all disj u1, u2: User | no n: Notification | n
           in u1.pendingNotifications and n in u2.
           pendingNotifications
}

fact calendarProperties {
        //Every calendar belongs exactly to one user
        all c: Calendar | one u: User | u.calendar = c
}

fact eventProperties {
        //The event owner is not an invited user
        all u: User, e: Event | e.owner = u implies (u
           not in e.invitedUsers)
        //The event owner is always a participant
        all u: User, e: Event | e.owner = u implies u
           in e.participants
        //Participants are no more invited
        all u: User , e: Event | u in e.participants
           implies u not in e.invitedUsers
        //If a calendar contains an event the user is a
            participant
        all e: Event, u: User | e in u.calendar.events
           iff u in e.participants
}

fact notificationProperties {
        //Every notification belongs to a user
        all n: Notification | one u: User | n in u.
           pendingNotifications
        //A user is shown an invitation only if invited
```

```
    all u: User, e: Event | some i: Invitation | u
        in e.invitedUsers implies (i in u.
        pendingNotifications and e = i.tiedTo)
    //If a user is not an invited to an outdoor
        event he is not shown weather alerts
    all u: User, e: Event, w: WeatherAlert | w.
        tiedTo = e and w in u.pendingNotifications
        implies u in e.participants
    //There is at most one Invitation per event per
         user
    no disj i1, i2: Invitation | one u: User | i1
        in u.pendingNotifications and i2 in u.
        pendingNotifications and i1.tiedTo = i2.
        tiedTo
    //There is at most one WeatherAlert per Outdoor
         event per user
    no disj w1, w2: WeatherAlert | one u: User | w1
         in u.pendingNotifications and w2 in u.
        pendingNotifications and w1.tiedTo = w2.
        tiedTo
    //Weather alerts are displayed to all
        participants
    all disj w1, w2: WeatherAlert, disj u1, u2:
        User | w1 in u1.pendingNotifications iff w2
        in u2.pendingNotifications and w1.tiedTo =
        w2.tiedTo
}
```

6.2 Generated worlds



7 Document informations

This section does not pertain the MeteoCal system or its requirements but it is intended as a short recap and log for the document itself.

7.1 Effort

As of November 17, 2014 approximately **20 hours** have been spent making this document and refining it.

7.2 Changelog

The last time that the document was updated was November 16, 2014.

7.3 Tools used

Texmaker 4.1 to write the document.
http://www.xm1math.net/texmaker/

GenMyModel to create the UML diagrams.
http://www.genmymodel.com/

Alloy Analyzer 4.2 to create and study the Alloy model.
http://alloy.mit.edu/alloy/