Department of

Computer Science and Engineering

MeteoCal

Project development for the 2014 Software Engineering 2 Course

Supervisor: **Prof. Di Nitto**

Authors

Andrea Bignoli
Matr. 837493
andrea.bignoli@gmail.com

Leonardo Cella
Matr. 838074
leonardocella@gmail.com

Requirement

Analysis

Specification

Document

Digest

RASD is the acronymous of Requirement Analysis and Specification Document. In this document we are listing and analysing the requirements and the models of MeteoCal service. We are also going to include a description of the sceneries of usage, a use cases analysis. This document is brought to the attention of the stakeholders and the development team.

Problem description

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

Table of Contents

1. Introduction

1.1. Main Purpose

The main goal of the MeteoCal project is the creation of an information system to allow registered users to manage events they decide to take part in. In particular all the events have associated information concerning where and when the event will take place. The system will grant the creator the possibility of inviting other users to the event and alloy him to manage the participation. Moreover the system will also provide the participants with weather forecasts for their events, and also warn them in case of bad weather conditions.

MeteoCal has to provide the following functionalities to the end user:

1. Access into the system (registration of an account, login, logout).
2. Personal calendar management.
3. Participation to an event.
4. Creation of an event, specifying details such as: weather conditions to avoid, invited users, the location, whether the event is outdoor or indoor.
5. Alert participants about changes to their events. Modifications of this kind can be made only by the creator of the event.
6. Notify the participants in case of bad weather conditions.
7. In case of bad weather forecasts three days before an event's start, the system will suggest the creator a better date to reschedule the event.
8. More functionalities specified later in the document.

1.2. Domain Properties

We suppose that the following conditions hold in the analysed world:

1. The system won't assume that the participation to an event prevent the user from choosing to give his participation also to an other event, since this would limit the possibilities of usage for the end user.
2. According to the previous property, a user may add to his calendar parallel events.
3. Events can be only outdoor or indoor.

1.3. Glossary

Here are listed the most used keywords that occur in this document, in order to ease the comprehension :

Event	An event concerned is an activity, created by one user, that may involve the participation of other users. Each event is either public, or private.
Public Event	It's a kind of event where all the registered users can see the event details, including the corresponding participants. This will allow every user on the system to find the event using a search bar. An invitation isn't required to participate to the event.
Private Event	An event only invited user can participate in. Non invited user can't even see the event details or find the event using the search bar.
Calendar	To each user is associated one and only one calendar. The calendar collects all the events the user decided to participate in. Each calendar is either public or private.
Public Calendar	This kind of calendar is visible to all registered users and can be accessed by typing the owner's username in a search bar provided by the system. These last ones will see all the time slots in which a user is busy but without seeing the details of the corresponding events, unless they have been defined as public.
Private Calendar	This kind of calendar can be seen exclusively by the owner.
Creator	The user that created the event. His username will be shown in the event details.
Participant	It's a user who decided to go to an event, and so he receives notifications related to the event. The participants are also listed in the event details.
Invitation	An invite from an event creator to participate in the event sent to an user. All invited users are displayed in the event details.
User	A person logged in an account on the system.
Guest	Is a not yet registered user, who can see only general informations about the application and create a new account.
Login	Is the first operation that everyone has to do each time they want to access their own user-page.
Registration	It's the operation that a guest has to do if he wants to use and access the MeteoCal application. This operation has to be done only one time. During this operation a guest have to choose his username and his password.
Weather Conditions	It's a field of the event, are necessary to give advices and notifications to the users who participate to an event.

Notification	Notifications appear each time that something meaningful occur to an event. Notifications are sent only to participants and are either change notifications or weather notifications.
Change Notification	A change notification is sent to all participants each time the creator of an event modifies the event details.
Weather Notification	A weather notification is sent to all participants one day before the event's start in case the forecast weather condition belongs to the set of conditions listed as bad for the event.
Username	An univocal identifier for each user registered in the system.
Password(psw)	A secret combination of characters which is required to access a specific account.

1.4. Assumptions

There are some unclear points in the problem presentation document, so we have to assume some facts.

The main issue is about the contrast of an event being public and a participant having set his calendar as private. For instance, we decided to show the participants to an event in the event page, as many social networks do at the current state. But what about the users that decided to have a private calendar? We decided that the privacy of the user should have precedence. For this reason those usernames won't be listed in the participants list. Those usernames will instead be listed in the invited users list, if invited, since this doesn't harm their privacy. The creator of an event will also be displayed in every case in the event details.

2. Current System

At the current state-of-the-art there isn't a common and widespread application, used to manage events and appointments of a person strictly that grants also management options related to the weather forecasts.

The kind of service we are going to provide is offered in a limited way by social networks, even if they hardly ever offer informations about the environment where an event is going to take place.

3. Proposed System

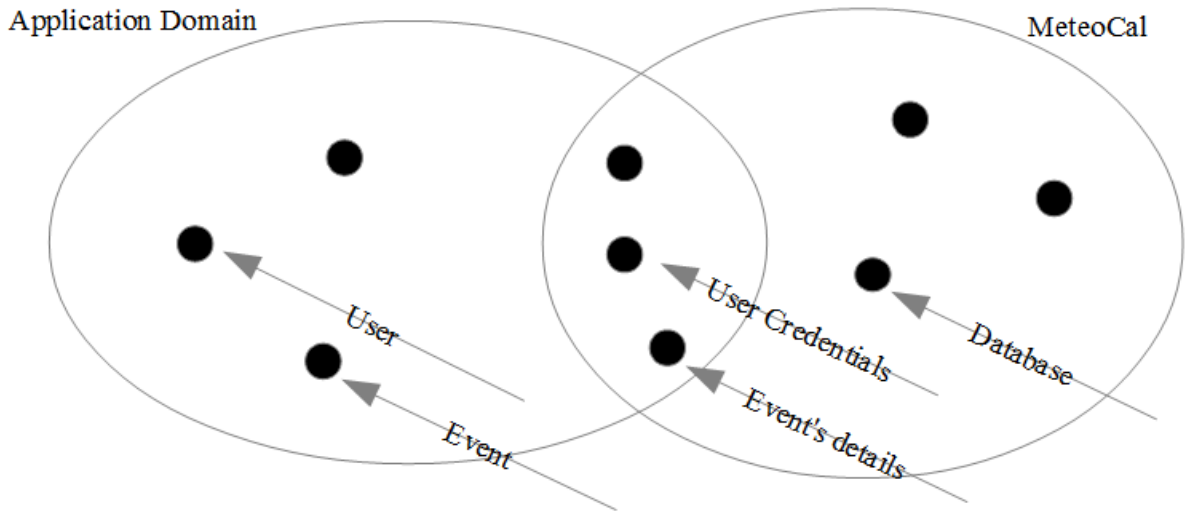
Our system introduce some innovations in the management of the events, in particular it gives useful information and help you to avoid unexpected weather issues .

In our system each user(*guest*) has to login in the site, if has already registered; in order to access to his personal page (*user*).

We offer to our users the possibility of creating *private* and *public* events. The public ones are visible to all other registered users, while the private ones are visible only to invited users.

3.1. Jackson-Zave approach

For the design part we decide tu use an approach which is known as Jackson-Zave. This approach let we able to define the intersection (shared phenomenons) between the environment of the application, that consists of the activities and the appointments of users, and the software MeteoCal that we are going to develop. Below there is a graph with an only demonstrative purpose.



4. Domain

The field of use of the system is quite general. The service offered can be useful in both casual and personal domain, as there is no particular restriction on the type of events created by the users and managed by the system.

4.1. Entities

The analysis of the problem showed that the main elements the system will handle are “users” and “events”. The relationship between the two will be ruled by a number of factors, including, but not limited to, weather condition constraints and notification of different types. The set of the main entities will hence be defined as such:

1. Users
2. Events
3. Calendars
4. Weather condition constraints
5. Invites
6. Notifications (regarding both: weather issues and schedule changes)
7. The weather forecast service

4.2. Stakeholders

The people having interest in the development of the system are the following:

1. The Software Engineering 2 course's supervisors, including the teacher and the laboratory team who committed the development and provided the informal requirements for the system in the first place.
2. The X company who will actually provide our service to the broad public.
3. The final users, interested in having access to the cloud services for managing their personal events. In particular the keen interest provided by the system is the possibility to invite other users, and keeping in check the weather conditions for each particular event if needed.

4.2. Relationships

In the domain application context were discovered several relationships among entities, which are going to be better specified later in this document. Some of the more simple and meaningful ones are specified here:

- Each event has one and only one user as creator
- Each user has exactly only one calendar
- Only the creator can send invites to the other users
- Each event is characterized by only one weather condition
- Notifications are send to all the users that still participate to the corresponding event

5. Requirements

5.1. Functional Requirements and Actors

The actors of the system are all the users that will interact with the system. In our case both active and passive roles are present. The active actors are those who use the system functionalities, according to what is described below. The same holds for the passive ones, that are necessary for the system to work.

1. *Guest:*
    - *Sign up:*
      register a new account on the system, specifying username and password.
2. *User:*
    - *Login:*
      access an account on the system.
    - *Calendar management:*
      access to a personal calendar, with the option to make it visible to all other users.
    - *Event creation:*
      create a new event on his calendar (with access to all the functionalities granted to an event creator). The following parameters will be required:
        - Title
        - Location
        - Date and time
        - Invited users
        - Indoor / Outdoor
        - Set of weather conditions to avoid
        - Public / Private
    - *Reply to an invite:*
      choosing whether he will participate or not to the event he's invited to.
    - *Receive notifications:*
      about bad weather conditions for upcoming events, or schedule change for any event the user chose participate in.

3. *Event creator (on the created event):*
    - *Invite users to the event:*
      the invited users will be able to reply to the invitation.
    - *Change scheduling of the event:*
      A notification will be sent to all the users that chose to reply positively to an invitation to the event.
    - *Change event description*

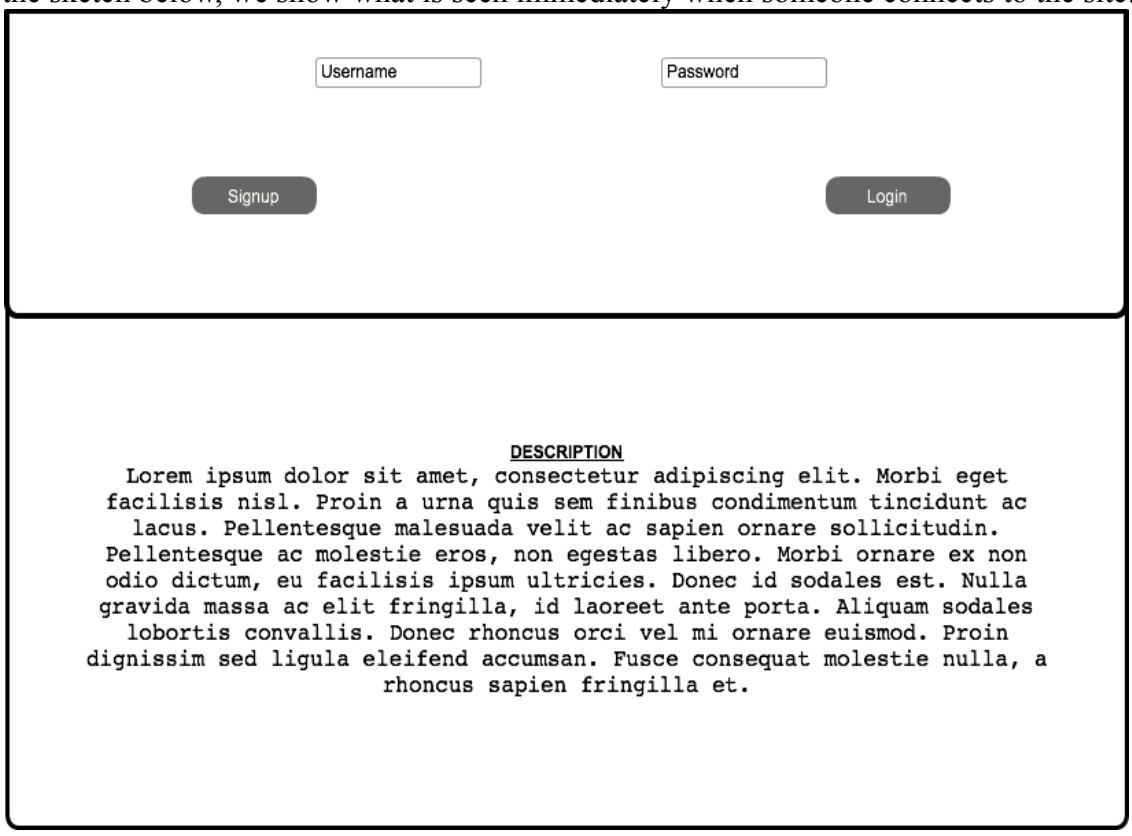
The weather forecasts will be provided by an external service.

5.2. Non Functional Requirements

The application interface will be a website, to grant maximum accessibility and reduce the effort by the final user to set up the system. In fact, the user will be able to access all our services throw a web page. At the same time we intend to create a light user interface, like modern websites tend to do. Following those standards, will not only arguably improve the aesthetics of the website, but also augment its usability.

The first step will be the registration of a new account This will be accomplished easily by fulfilling the form on the page shown by the system to the guests with the required data. The required registration information will be kept at the minimum possible to grant the fastest and easiest access possible to the system. The final user will be able to register a new account specifying only username and password. The username will be visible to the other users.

In the sketch below, we show what is seen immediately when someone connects to the site:



Once the user successfully signed in the system, he will access a front page showing his calendar. On the top of the page a bar of tools will allow the user to access all the offered functionalities. The bar will be present in each page to make the navigation between our services easier for the final user.

From the calendar the user will be able to visualize all the events he decided to participate in, and quickly glance on basic event information. Clicking on an event will bring the user to a page describing all its details, including the author.

If the user is also the creator of the event he will be able to modify all the event information and invite other users to participate. On this page the creator of the event will also be able to change the event visibility between public and private. A public event will be visible to all the users.

Clicking on a username on any page will bring the user to his calendar if one of the following conditions holds:

- The current user and the clicked one are the same person. In this case this behaviour is equivalent to a “Home” link that will be always present in the bar at the top.
- The calendar of the clicked user is declared as public.

A user can see the calendar of another user if this is declared as public. In this case he will be able to see the scheduling on his calendar without being able to see the event details, unless the event itself is public.

The bar will also include notification and invitation icons, granting the user an immediate glance over them. These icons will also allow the user to quickly get to pages to visualize the details of each notification or invitation.

The notifications page, for instance, will allow the user to view all current notifications, regarding weather condition issues and schedule changes.

The invitations page will show all the events the user has been invited to, and will allow him to quickly get to the page of the events he has been invited to.

The link “Create event” on the bar at the top, will lead the user to a page dedicated to the creation of a new event. The user will be asked the parameters specified in the previous section.

If the user chooses “outdoor” during event creation, the system will suggest him an automatic set of weather conditions to avoid (including “Rain” and “Snow”). The user will still be able to specify a different set of unwanted weather conditions.

During the creation of the event, other users may be invited to participate by the creator. Even after the creation of the event has been finalized the author will be able to modify event information and invite more people.

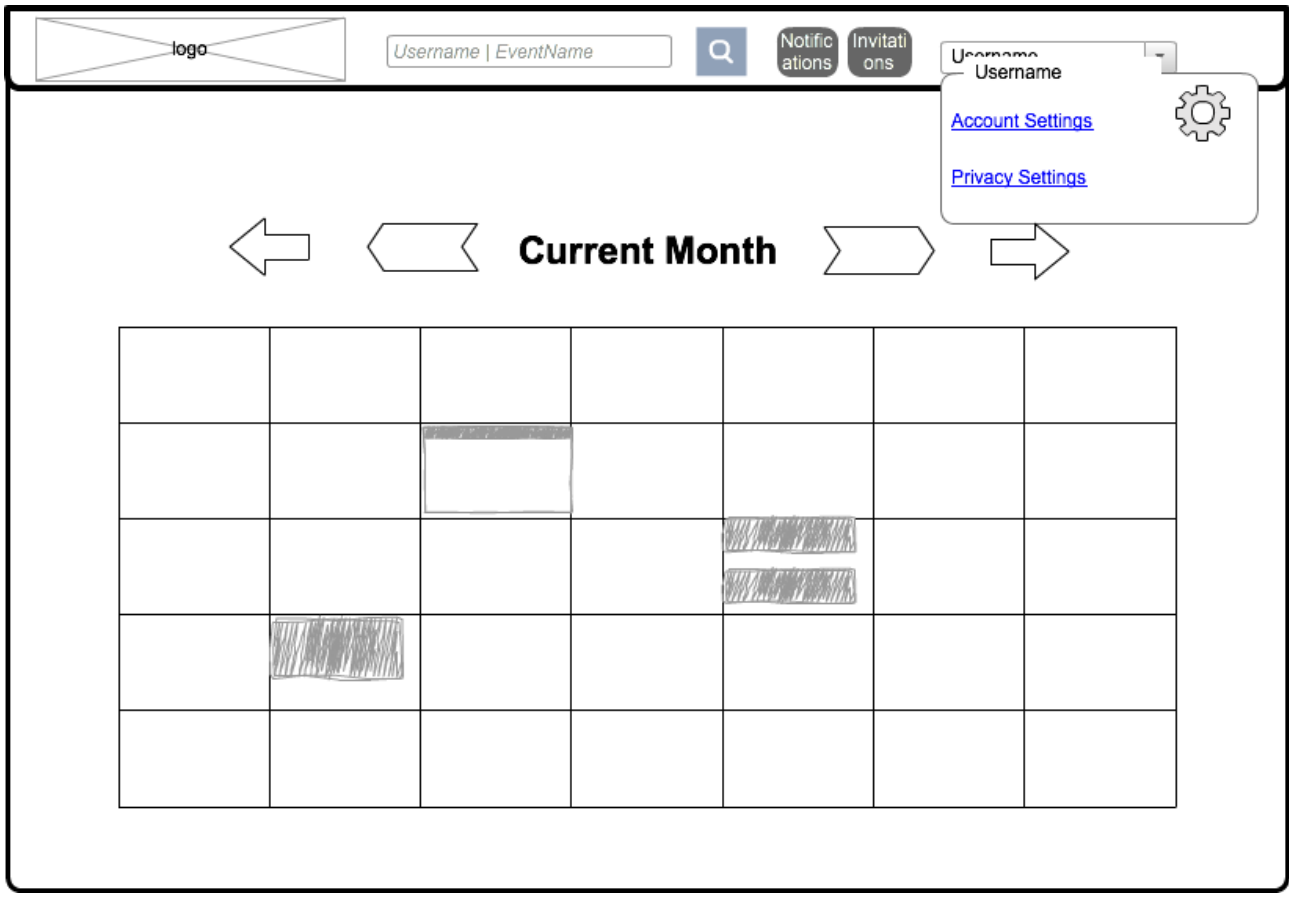
On the bar at the top the user will find also a link to an account settings page that will allow him to toggle the public visibility of his personal calendar and also to change his login password.

The bar at the top will also include a link to a page where the user will be able to search for user calendars and public events. He will be able to search a username and the system will show him the the desired calendar if the owner declared it as public, or a message informing that the calendar associated to the given username is private otherwise.

A user will also be able to search for public events by name. The given keywords will be used to filter between the titles of the public events on the system. Once the search is completed, the user will be shown the list of compatible public events.

As mentioned before, we'll try to convey all the main functionalities in the bar at the top of every page. This should be possible while also keeping a light user interface easy to operate with.

A sketch of the user personal page, in order to explain functionally our solution:



6. Documentation

The implementation of the system will be covered by a complete documentation in every stage of he project, to grant maximum readability to our work and allow update options.

The documents to be produced are the following:

- *Project Plan:*
  deadlines establishments to improve the efficiency of our progress. Has been given to us in the problem presentation document.
- *Requirement Analysis and Specification Document:*
  to clarify the informally given problem to a state where we can effectively derive the desired concrete product.
- *Design Document:*
  to better define the technical implementation of our system.
- *JavaDoc Comments:*
  to grant readability for maintenance and update purposes.
- *Installation Manual:*
  a guide describing how to set up our system and start using it.
- *User Manual:*
  a guide on how to use our services efficiently.
- *Testing Document:*
  a concise report describing the testing process for the system.

7. Architectural considerations

To implement the system we are going to make use of J2EE and a database to store our data.

We will describe our data organization in the Design Document. Some of it will be clarified partially by the following points of this document.

The end user will need an Internet connection to access our system and an updated browser following the current standards.

8. Specifications

This section will clarify the relationships between different elements of the system that are effectively relevant on the actual service offered to the user.

- The events can be concurrent. The fact that a user may not be actually able to participate to both the events has been taken into account, but we decided that allowing concurrent events offers more flexibility to our service, and grants the user a better usability in a variety of cases.
- A user has visibility over the details of an event if and only if one of the following holds:
    1. The user is the creator of the event.
    2. The user has been invited to the event.
    3. The event is public.
- A user can answer to an invite only by yes or not.
- Deciding to participate to an event isn't irreversible: the invited user will still be able to withdraw his participation on the event page. In this case he will also stop receiving notifications about that event. He will be still able to see the previously received notifications about that event in his past notifications page.
- Discarding an invite doesn't make the invited user lose visibility on the event details.
- Users using the search bar will be able to find also the private events they are invited to.
- Discarding an invite isn't irreversible: the invited user will still be able to decide to participate on the event page, or looking into his discarded invitations.
- The event details include also a list the invited users.
- The event details include also a list the participants to the event.
- The users that set their calendar as private will not be displayed in the participants list, as this would harm their privacy.
- Only the creator of an event can modify its details.
- The creator of an event cannot invite himself to the event.
- All and only the attendees to an event will receive notifications relative to:
    1. Weather conditions issues.
    2. Event details change (e.g. the event's scheduling).
- The invitations to an event will be automatically discarded once that event has ended.
- The creator of an event won't be able to change event's details once that event has ended.

9. Alloy

9.1. Alloy Code

The following Alloy code models our system proving its coherency and allowing a better understanding of its logic. In the following sections the reader will find instance diagrams to quickly glance at some of the characteristics of the model presented.

```
//---------------------------------- SIGNATURES ----------------------------------


// The user entity
sig User {
	access: one AccessData,
	calendar: one Calendar
}

// Just to represent parameters used as unique identifier for the user. In this case,
// just the username is enough
sig AccessData {}

// The calendar associated to a user
sig Calendar {
	eventsInCalendar: set Event
}

// Just to represent parameters used as unique identifier for the event
sig EventData{}

// The location where an event can take place
sig Location{}

// Indoor/Outdoor event
abstract sig LocationType {}
one sig Outdoor extends LocationType {}
one sig Indoor extends LocationType{}

// A value describing a specific point in time
sig DateTime{}

// A generic order relationship set of values
abstract sig OrderRelation {}
one sig Smaller extends OrderRelation {}
one sig Equal extends OrderRelation {}
one sig Greater extends OrderRelation {}
// The chronological order of DateTime values
one sig ChronologicalOrdering {
	order: DateTime -> DateTime -> OrderRelation
}
```

```
// The entity representing a generic event
sig Event {
	creator: one User,
	invited: set User,
	participants: some User,
	data: one EventData,
	location: one Location,
	locationType: one LocationType,
	start: one DateTime,
	end: one DateTime,
	badWeatherConditions: set WeatherCondition,
	weatherForecasts: set WeatherForecast
} {
	ChronologicalOrdering.order[end,start] = Greater
}

// A private event is different from an event only due to the fact that only invited
// users can participate (not considering the creator)
sig PrivateEvent extends Event {}

// An invitation to an event
sig Invitation {
	seen: one boolean,
	event: one Event,
	invited: one User,
	invitationDateTime: one DateTime
}{
	ChronologicalOrdering.order[invitationDateTime,event.end] = Smaller
}

// A generic notification for an event
abstract sig Notification {
	recipient: one User,
	seen: one boolean,
	aboutEvent: one Event,
	notificationDateTime: DateTime
}{
	ChronologicalOrdering.order[notificationDateTime,aboutEvent.end] = Smaller
}

// Will be automatically generated only if any substancial event data changes
sig ChangeNotification extends Notification {}

// Will be automatically generated only if the weather forecast for a specific event changes  // between
good and bad, to warn the user
sig WeatherNotification extends Notification{}

// The entity representing the external weather forecast service we are going to use
// This service will offer, via its APIs, a function taking as input a DateTime value and
// a location to return a forecast if avaible
sig WeatherForecastService {
	forecast: Location -> DateTime -> lone WeatherCondition
}
```

```
// A weather forecast for the time interval (forecastStart, forecastEnd), in the location
// of the event this weather forecast is associated to
sig WeatherForecast {
        weatherCondition: lone WeatherCondition,
        forecastStart: one DateTime,
        forecastEnd: one DateTime
} {
        ChronologicalOrdering.order[forecastEnd,forecastStart] = Greater
}

// A utility object representing the last weather forecast observed by the user on a
// specific event
sig WeatherView {
        lastSeen: one WeatherCondition,
        event: one  Event,
        viewer: one User
}

// The different weather conditions, more may be added in the actual implementation
abstract sig WeatherCondition {}
one sig Sun extends WeatherCondition {}
one sig Clouds extends WeatherCondition {}
one sig Rain extends WeatherCondition {}
one sig Snow extends WeatherCondition {}

// Standard boolean value
abstract sig boolean {}
one sig True extends boolean {}
one sig False extends boolean {}
```

```
// ---------------------------------------- FACTS ----------------------------------------


// -- USER --


//        No duplicate user
fact noDuplicateUser {
        no disj u1,u2: User | u1.access = u2.access
}

// No access data without any user
fact accessDataBelongsToUser {
        no ad: AccessData | no u: User | u.access = ad
}


// -- CALENDAR --



// Calendars not to be shared between users
fact calendarIsNotShared {
        all c: Calendar | one u: User | c = u.calendar
}

// Conformity between calendar and events its owner decided to participate in
fact calendarConformity {
        all e: Event, u: User | e in u.calendar.eventsInCalendar iff u in e.participants
}

// -- EVENT --


// No shared event data
fact noSharedEventData {
        (no disj e1,e2: Event | e1.data = e2.data)
}

// No event data not associated to an event
fact noUnlinkedEventData {
        no ed: EventData | no e: Event | e.data = ed
}

// -- INVITATION --


// All invited users receive an invitation
fact invitationForInvited {
        all e: Event, u: User | u in e.invited implies one i: Invitation | i.event = e
        and i.invited = u
}

// No invitation is sent to non invited users
fact noInvitationForNonInvited {
        (no i: Invitation | i.invited not in i.event.invited)
}
```

```
// The aren't multiple invitation to the same event for the same user
fact noMultipleInvitations {
	(no disj i1, i2: Invitation | i1.event = i2.event and i1.invited = i2.invited)
}

// The creator of an event can't invite himself to the event
fact creatorCannotInviteHimSelf {
	(no e: Event | e.creator in e.invited)
}

// -- PARTICIPANT --


// All the participants to a private event must be invited first, exception made for the creator
fact participantsMustBeInvitedAtPrivateEvent {
	(no e: PrivateEvent | some u: User | u in e.participants and !(u in e.invited)
	and e.creator != u)
}

// The creator of an event is always in participants's list
fact creatorInParticipantsList {
	all e: Event | e.creator in e.participants
}

// Notifications should be sent only to participants
fact notificationsOnlyToParticipants {
	(no n: Notification | n.recipient not in n.aboutEvent.participants)
}

// -- WEATHER FORECAST --


// We are going to use only one weather forecast service
fact oneWeatherForecastService {
	#WeatherForecastService = 1
}

// All weather forecasts are relative to only an event
fact weatherForecastBelongToAnEvent {
	all wf: WeatherForecast | one e: Event | wf in e.weatherForecasts
}

// All weather forecasts and are taken by the forecast service
fact weatherForecastFromForecastService {
	all wf: WeatherForecast | one e: Event | wf in e.weatherForecasts and
	wf.weatherCondition =		WeatherForecastService.forecast[e.location,wf.forecastStart]
}
```

```
// All weather forecasts relative to an event form a chronological sequence, starting
// at the event's starting time and ending at the event's ending time
fact weatherForecastInSequence {
	all e: Event | one wf: e.weatherForecasts | wf.forecastStart = e.start
	all e: Event | one wf: e.weatherForecasts | wf.forecastEnd = e.end

	all wf1: WeatherForecast | one e: Event | wf1 in e.weatherForecasts and (wf1.forecastStart !
	= e.start iff (one wf2: WeatherForecast | wf2 in e.weatherForecasts and wf2.forecastEnd
	= wf1.forecastStart and wf2.weatherCondition != wf1.weatherCondition))

	all wf1: WeatherForecast | one e: Event | wf1 in e.weatherForecasts and (wf1.forecastEnd !
	= e.end iff (one wf2: WeatherForecast | wf2 in e.weatherForecasts and
	wf2.forecastStart = wf1.forecastEnd and wf2.weatherCondition != wf1.weatherCondition))
}

// One weather view for (user,event) pair
fact univoqueWeatherViewForUserEvent {
	no disj wv1,wv2: WeatherView | wv1.viewer = wv2.viewer and wv1.event = wv2.event
	all e: Event, u: User | (u in e.participants) implies one wv: WeatherView | wv.event = e
	and wv.viewer = u
}

// No weather view on events the user is not participating in
fact noWeatherViewForNonParticipants {
	no wv: WeatherView | wv.viewer not in wv.event.participants
}

// -- ADDITIONAL CONSTRAINTS --


// No location not associated to an event
fact noUnlinkedLocation {
	no l: Location | no e: Event | e.location = l
}

// No dateTime not associated to something
fact noUnlinkedDateTime {
	no dt: DateTime | (no e: Event | e.start = dt or e.end = dt) and (no wf: WeatherForecast |
	wf.forecastStart = dt or wf.forecastEnd = dt) and (no n: Notification |
	n.notificationDateTime = dt) and (no i: Invitation | i.invitationDateTime = dt)
}

// Chronological ordering properties (general total ordering relationship)
fact chronologicalOrderingProperties {
	all dt: DateTime| ChronologicalOrdering.order[dt,dt] = Equal
	all disj dt1, dt2: DateTime | (ChronologicalOrdering.order[dt1,dt2] = Smaller) iff
	(ChronologicalOrdering.order[dt2,dt1] = Greater)
	// Transitivity
	all dt1,dt2,dt3: DateTime | (ChronologicalOrdering.order[dt2,dt1] = Greater and
	ChronologicalOrdering.order[dt3,dt2] = Greater) implies
	(ChronologicalOrdering.order[dt3,dt1] = Greater)
}
```

```
// ---------------------------------- ASSERTIONS ----------------------------------------


// No invitation for user that don't belong to the invited list
assert invitationConsistency {
	no i: Invitation | i.invited not in i.event.invited
	#Invitation = (sum e: Event | #e.invited)
}

check invitationConsistency for 5

// Since user that haven't been invited to a private event cannot partecipate to the
// event they won't receive any notification from it
assert noNotificationsAboutPrivateEventsForNonInvited {
	no n: Notification | n.recipient not in n.aboutEvent.invited and n.recipient !=
	n.aboutEvent.creator and n.aboutEvent in PrivateEvent
}

check noNotificationsAboutPrivateEventsForNonInvited for 5

// The association between user and access must be bi-directional
assert equalNumberAccessDataUser {
	#User = #AccessData
}

check equalNumberAccessDataUser for 8

// Each event data is associated to one and only one event, and each event has
// associated data
assert equalNumberEventDataEvent {
	#Event = #EventData
}

check equalNumberEventDataEvent for 8

// The locations represented on the system are no more than the events
assert locationNumberSmallerThanEvents {
	#Location <= #Event
}

check locationNumberSmallerThanEvents for 8

// Checks weather forecasts consistency
assert weatherForecastConsistency {
	all wf: WeatherForecast | one e: Event | wf in e.weatherForecasts and
	WeatherForecastService.forecast[e.location,wf.forecastStart] =	wf.weatherCondition
}

check weatherForecastConsistency for 8
```

```
// ------------------------------------ PREDICATES ----------------------------------------


// Shows public event participation constraints
pred showPublicEventParticipationConstraints() {
        #Event = 1
        #PrivateEvent = 0
        #User = 4
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2
        #Invitation = 2

        all e: Event | #e.participants = 3 and some u: User | u in e.invited and u not in
        e.participants
}

run showPublicEventParticipationConstraints for 5


// Shows private event participation constraints
pred showPrivateEventParticipationConstraints() {
        #Event = 1
        #PrivateEvent = 1
        #User = 4
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2
        #Invitation = 2

        all e: Event | #e.participants = 2 and some u: User | u in e.invited and u not in
        e.participants
}

run showPrivateEventParticipationConstraints for 5

// Show weather forecasts consistency
pred showWeatherForecastConsistency() {
        #Event = 1
        #PrivateEvent = 0
        #User = 1
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2

        all e: Event | #e.badWeatherConditions = 0
}

run showWeatherForecastConsistency for 5
```

```
// Shows weather forecasts sequence properties
pred showWeatherForecastSequenceProperties() {
	#Event = 1
	#PrivateEvent = 0
	#User = 1
	#Notification = 0
	#WeatherForecast = 3
	#Location = 1

	all e: Event | #e.badWeatherConditions = 0
}

run showWeatherForecastSequenceProperties for 5
```

All the assertions have been checked, and no predicate has been identified as inconsistent by Alloy Analyzer 4.2.

9.2. Alloy Diagrams

The following alloy diagrams will show instances of specific parts of the system to better clarify the model constraints over its instances. Some components in the system not relevant to each particular case will be intentionally omitted. It's important to notice that those components are present in the system and still instantiated even if not represented.

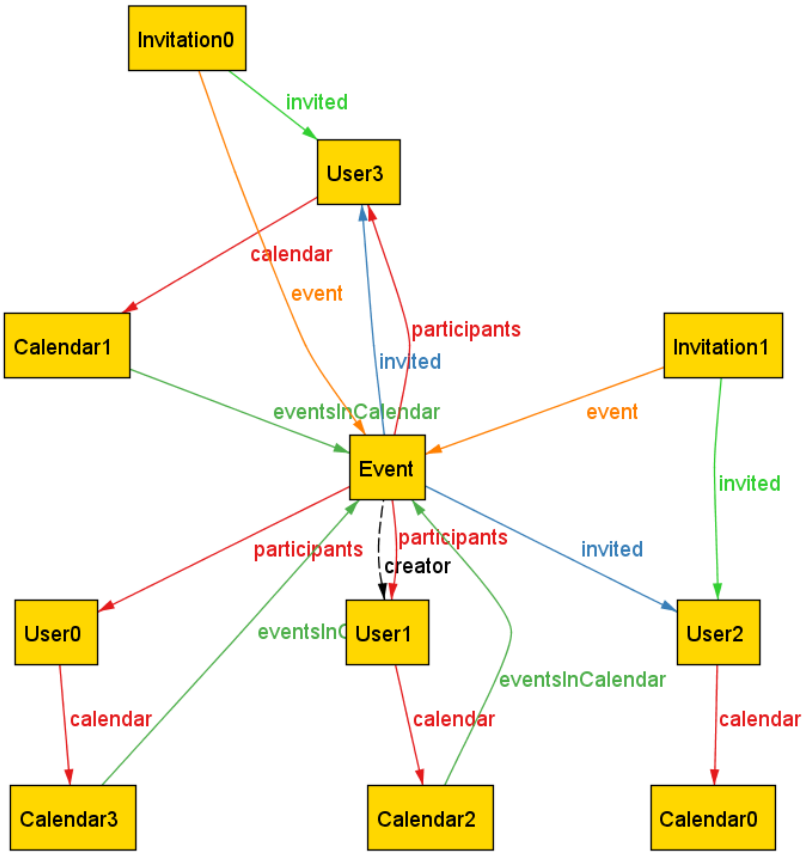
9.2.1. Public event participation constraints and calendar conformity

The diagram on the following page has been produced by the following code:

```
// Shows public event participation constraints
pred showPublicEventParticipationConstraints() {
        #Event = 1
        #PrivateEvent = 0
        #User = 4
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2
        #Invitation = 2

        all e: Event | #e.participants = 3 and some u: User | u in e.invited and u not in
        e.participants
}

run showPublicEventParticipationConstraints for 5
```



The diagram shows that:

1. User1, being the creator of the event, is also a participant.
2. One invitation has been sent to each invited user (2 and 3).
3. The participation is obviously not compulsory: user 2 has been invited but he's not participating to the event.
4. The event is public, so user 0, even if not invited, can participate to the event.
5. Calendar 1,2,3, associated to each one of the participants to the event include the event.
6. Calendar 0, associated to the only user that doesn't participate to the event, doesn't include the event.

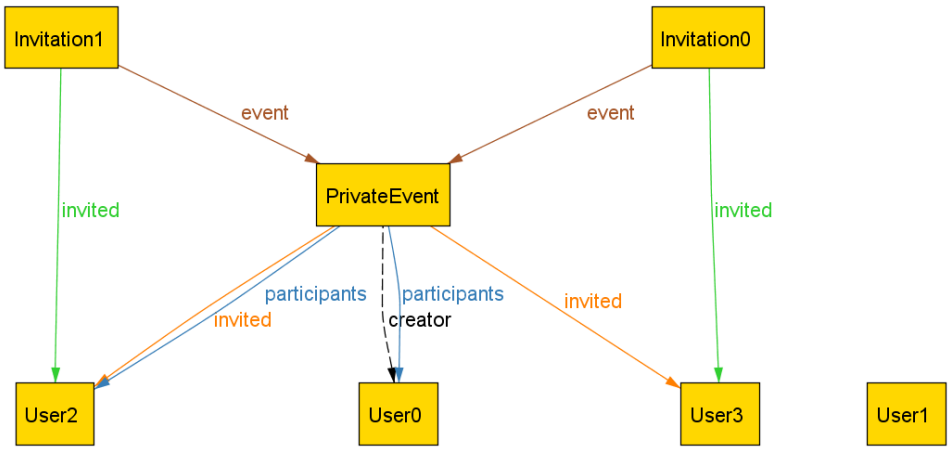
9.2.2 . Private event participation constraints

The diagram below has been produced by the following code:

```
// Shows private event participation constraints
pred showPrivateEventParticipationConstraints() {
        #Event = 1
        #PrivateEvent = 1
        #User = 4
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2
        #Invitation = 2

        all e: Event | #e.participants = 2 and some u: User | u in e.invited and u not in
        e.participants
}

run showPrivateEventParticipationConstraints for 5
```



The diagram shows that:

1. The participation is obviously not compulsory: user 3 has been invited but he's not participating to the event. On the other hand, user 2, invited to the private event, is participating.
2. The event is private, so user 1, can't participate to the event since he's not invited.

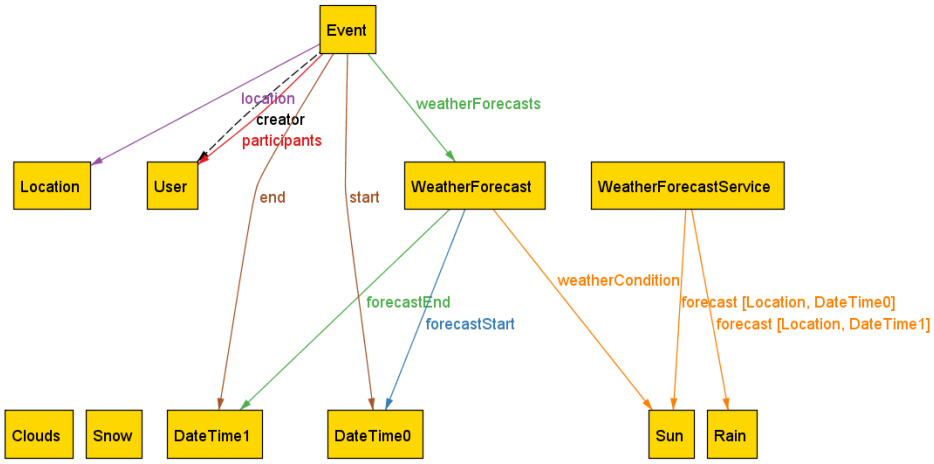
9.2.3. Weather forecast consistency

The diagram below has been produced by the following code:

```
// Show weather forecasts consistency
pred showWeatherForecastConsistency() {
        #Event = 1
        #PrivateEvent = 0
        #User = 1
        #Notification = 0
        #WeatherForecast = 1
        #Location = 1
        #DateTime = 2

        all e: Event | #e.badWeatherConditions = 0
}

run showWeatherForecastConsistency for 5
```



The diagram shows that:

1. The forecast associated to the event starts at the event's start time and ends at the event's end time.
2. The weather condition associated to the forecast is coherent with the one produced by the forecast service. The weather forecast using its start time and the location of the event to show the user the “Sun” weather condition. Sun is also the output of the function “forecast” offered by the forecast service, when arguments “Location” and “DateTime0” are used.

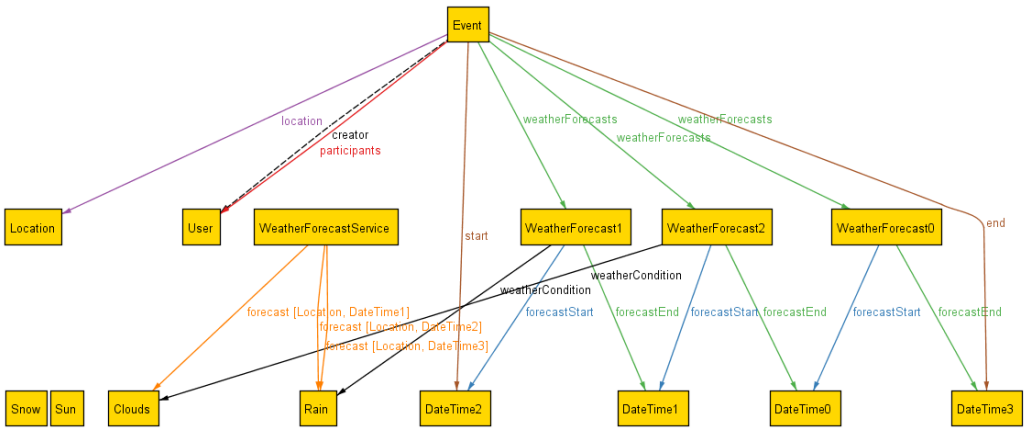
9.2.4. Weather forecast sequence properties

The diagram below has been produced by the following code:

```
// Shows weather forecasts sequence properties
pred showWeatherForecastSequenceProperties() {
        #Event = 1
        #PrivateEvent = 0
        #User = 1
        #Notification = 0
        #WeatherForecast = 3
        #Location = 1

        all e: Event | #e.badWeatherConditions = 0
}

run showWeatherForecastSequenceProperties for 5
```



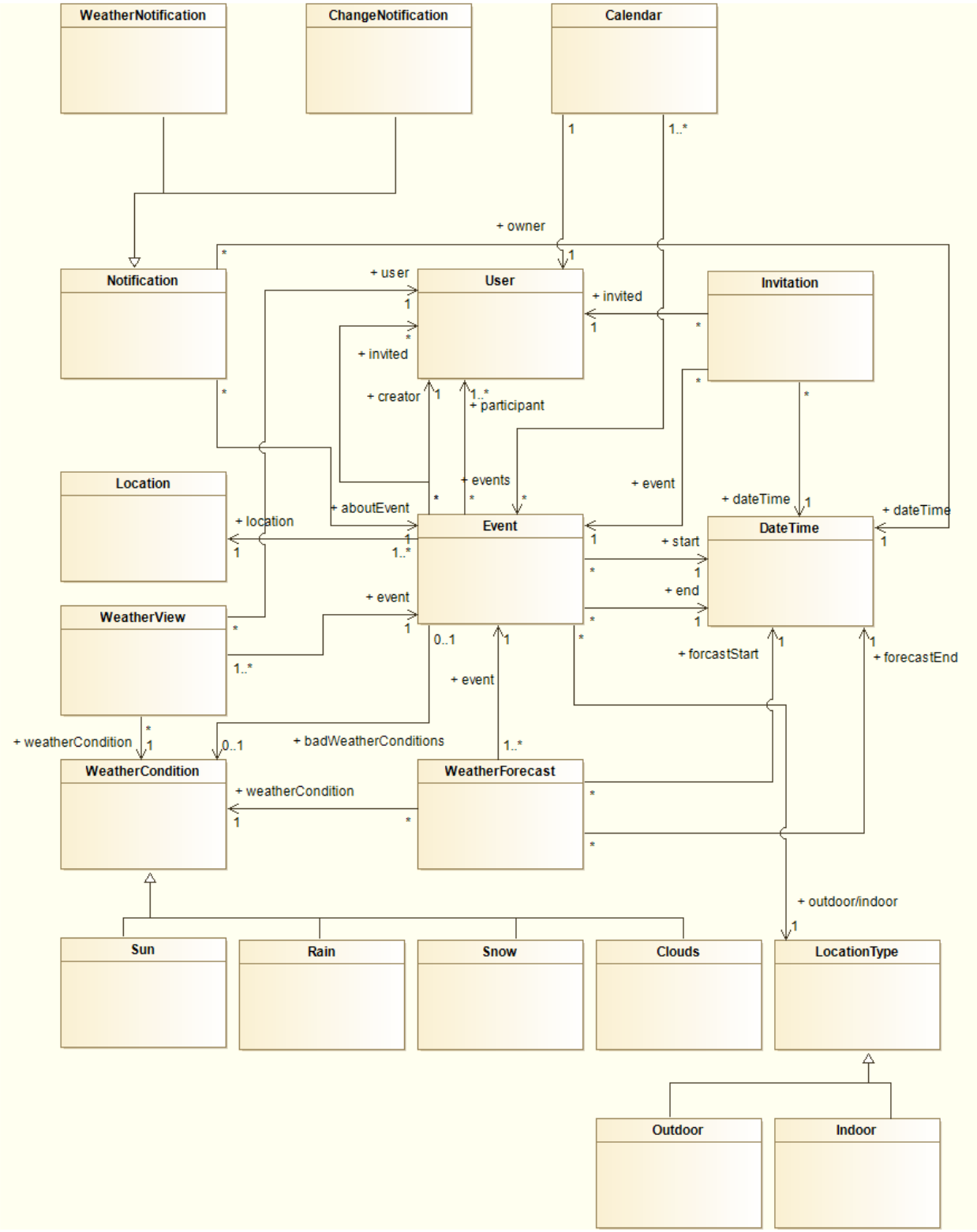
An event can have different weather forecasts over its duration, as should be expected in a real world application. Those different forecasts form a linked chronological sequence starting at the event's start time and ending at the event's end time.

The diagram shows that:

1. The forecast sequence generated is 1,2, 0.
2. Forecast 1 starts at the event's start time, and end's where forecast 2 starts. Forecast 2 ends where forecast 0 begins, and forecast 0 lasts till the end of the event, closing the sequence.
3. Each weather forecast uses the data provided by the weather forecast service as described in the previous diagram. If no forecast is given by the weather forecast service, no weather condition will be assigned to that forecast. This is the case for forecast 0.

9.3. Class Diagram

The following class diagram is a visual representation of the structure described also in the alloy code. It's important to notice that the purpose is just to clarify in a general way how the system should work. The actual implementation will instead be described in the Design Document.



Overall the diagram shows that:

1. Each user owns a personal calendar.
2. The calendar collects the events the user decided to participate in.
3. A user can receive invitations and notifications from the events.
4. The system keeps in memory the last information the user saw about the weather conditions of an event. This is useful to choose how weather notifications should be produced, to avoid redundancy.
5. Each user can participate to events, that can also be private.
6. Each event has one and only one creator.
7. Each event has start date and time and end date and time.
8. A sequence of weather forecasts associated to the event covers its whole duration, granting the user access to forecasts taken by an external weather forecast service.
9. *"Sun"*, *"Rain"*, *"Snow"* and *"Clouds"* are the four weather conditions represented in the system at the current state. More could be added in the future.
10. Each event has associated a set of bad weather conditions, allowing the user to choose the weather conditions he should be warned about for the event.
11. Each event has a location, used for the weather forecast, and can be indoor or outdoor.
12. All the notifications are associated to one and only one event.
13. The notifications can be of two types:
    - Weather notification: used to warn participants about bad weather conditions. This will also suggest to the creator a date he can decide to re-schedule the event to have better weather conditions, if available.
    - Change notification: to warn participants about changes the creator makes to the event. This is especially important for schedule changes.

10. System Models

10.1. Sceneries

Here there are some sceneries of use of the MeteoCal service.

1. Andrea is an important manager and he has a lot of appointments. So far he used to write down his appointments on his diary. During a raining day, he was in a hurry and the diary falls down in a puddle. So he decided to start using an electronic solution for his plans. He looked up “event planner” in a search engine on the web and found the MeteoCal website quickly. In the website's homepage there was a brief description of the proposal software behaviour, and the possibilities of sign up and login. He wasn't already registered and so he clicked on the sign up button. Then a new page was loaded; it contained a light form with only two text fields, one for the username and one for the password, and the standard confirmation box for the password. Andrea tried with the couple ('Andrea','password') but an alert message appears with the following voice: “This username is already been used”. He retried with ('Andrea1','password') and a green box with a confirm message appears. Andrea figured that was too late that day to actually schedule his next events on the new account.

2. The next morning at work Andrea spoke with his boss Stefano about the incident happened to his diary, and suggested him to adopt the same solution. Stefano with Andrea's guide, immediately signed up and logged in the website. After this operation is brought on a page with a calendar, that was represented throw a table. The same day, one of the biggest customers of his company Massimo, called him to establish an appointment. After the call end, Stefano decided to test the program and marks on his calendar an appointment. As the advices suggest him, he had just to click on the cell of the appointment day to add the new event with the desired details.

3. Stefano that was fascinated from the benefits carried from the MeteoCal site, decided to extend the use of this site to all the workers of his company. Initially too many workers didn't feel enough safe about the use of this site regarding on his privacy. But their doubts where quickly solved by the possibility of setting the personal calendar as private.

4. Massimo, the most important customer of Stefano's company recall to anticipate the already fixed appointment. Stefano that day was the fantastic idea to extend the discovered service to him too. They arrange to create a shared event in which Massimo was the creator, and for this reason he could edit the details of the event. If Massimo had edited something that concerns the event, Stefano would have received a notification and he could decide to retire his participation to the event. That was a comfortable way to manage Massimo's contingencies.

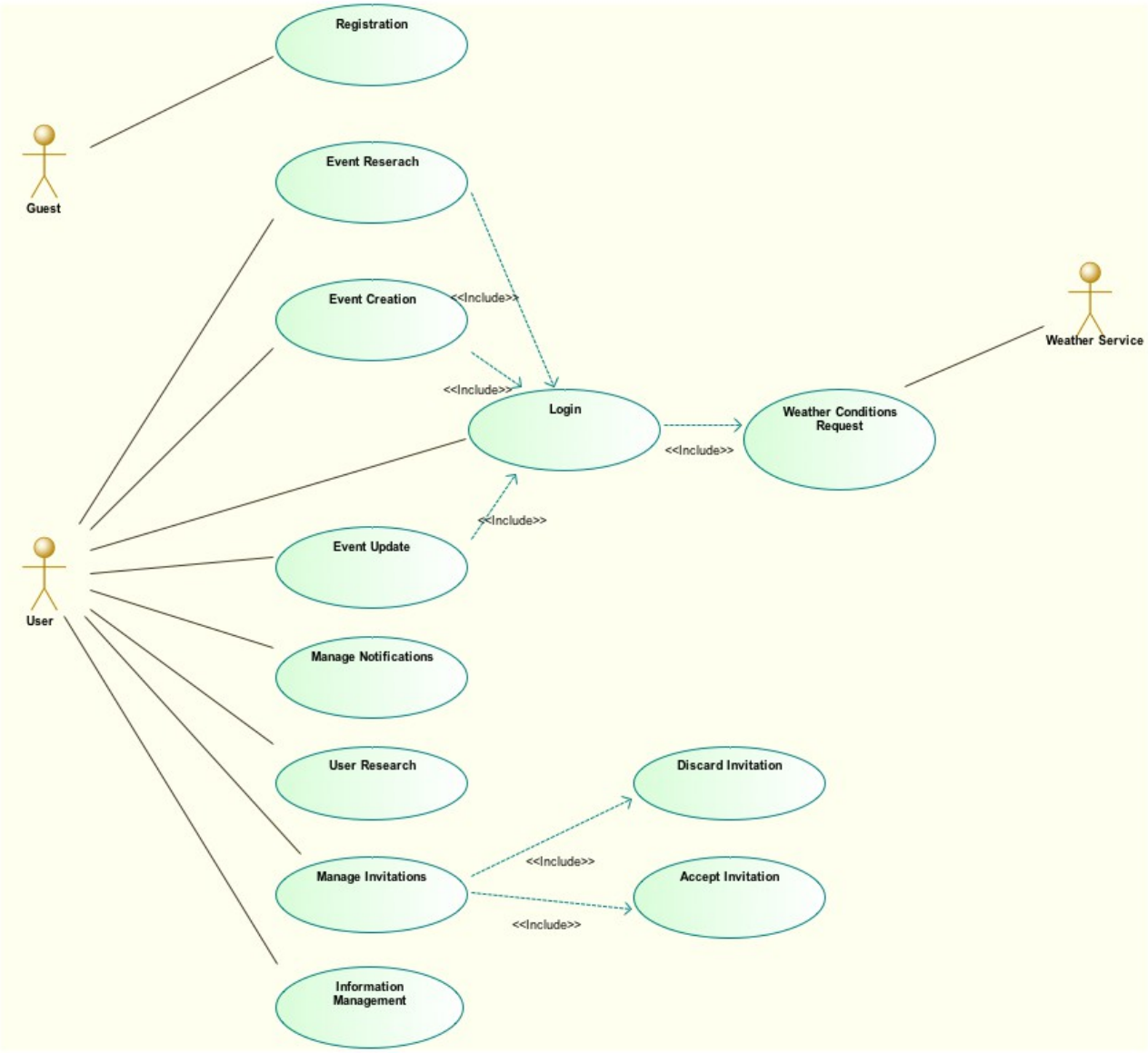
5. Massimo edited many times the shared event features. Stefano only one day before the fixed appointment logged on site and a lot of notifications alerts appear on his personal page. He understood that something changed, so he to opened his notifications page. Luckily Massimo postponed the event and Stefano was able to meet him without problems.

10.2. Use Case Analysis

We can derive some use cases starting from the sceneries identified in the previous paragraph:

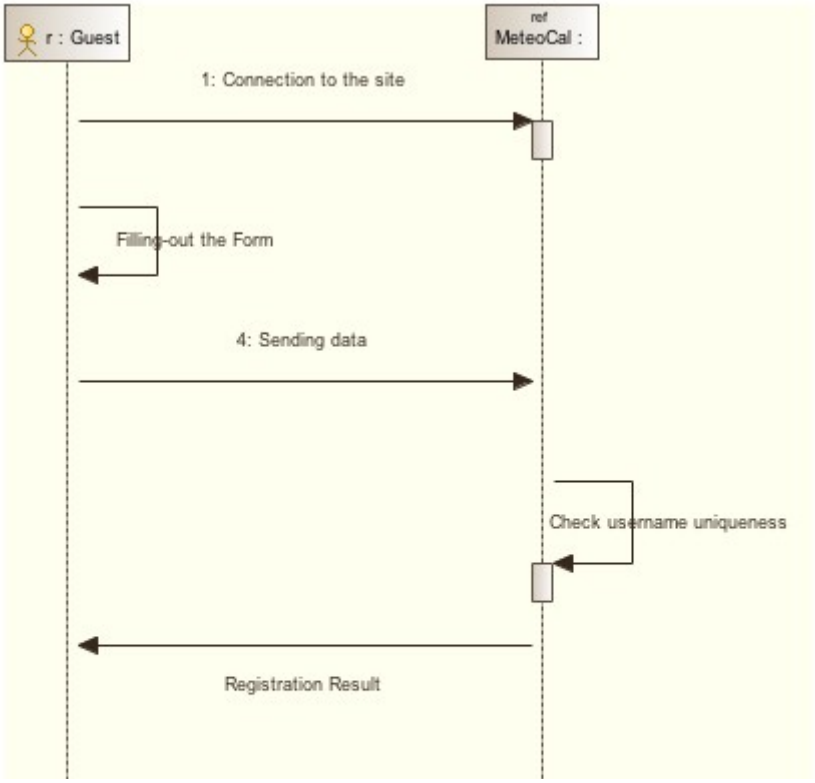
4. Registration request
5. Login operation
6. Information management
7. Event creation
8. Event update
9. Manage notifications
10. Manage invitations
11. User research
12. Event research

The following is a general use case diagram showing the user interacting with the functionalities of the system:



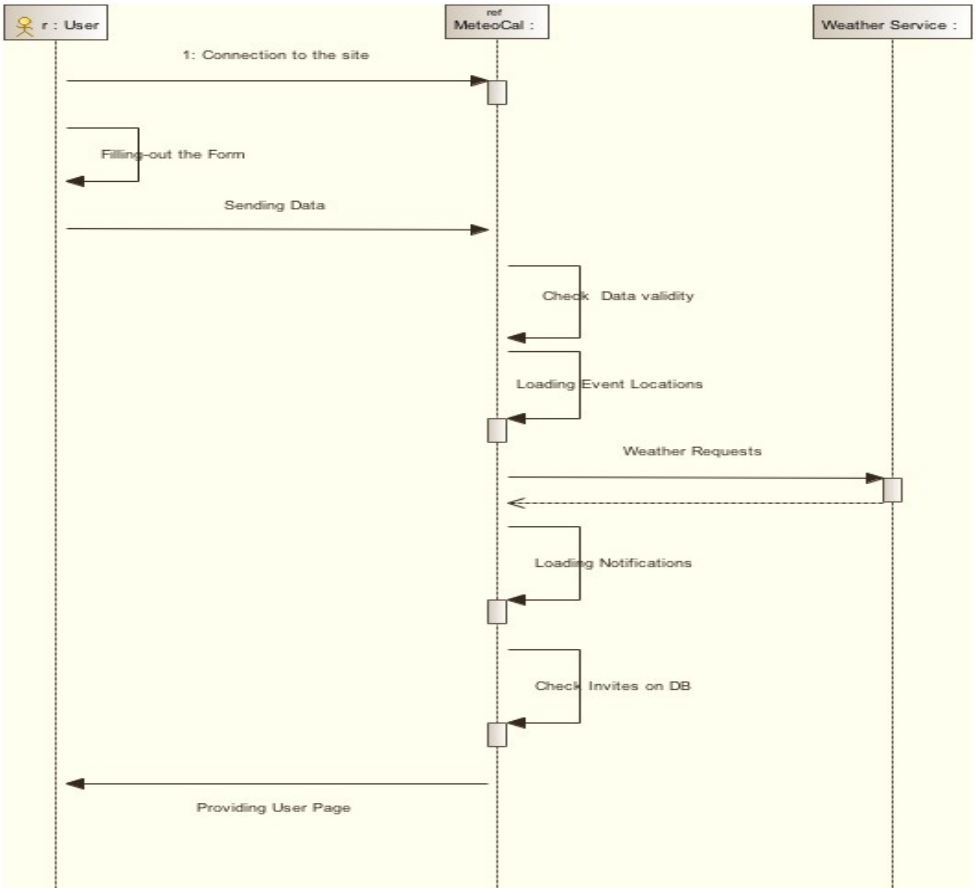
10.2.1. Registration request

Registration request	
Actors	Guest
Input conditions	A guest chooses to sign up.
Output conditions	The Guest is granted access to a User account.
Flow of events	1. The Guest is brought to a page that contains a registration form. 2. Then the Guest has to fill in the registration fields with a username and a password. 3. The system updates its database with these new data. 4. Guest (who is logged in as a User) is brought to his home page.
Exceptions	If a username is already used in the system, the guest G has to choose a different one. If the password doesn't satisfy the regular expressions controls, G has to choose a different one.
Sequence diagram	



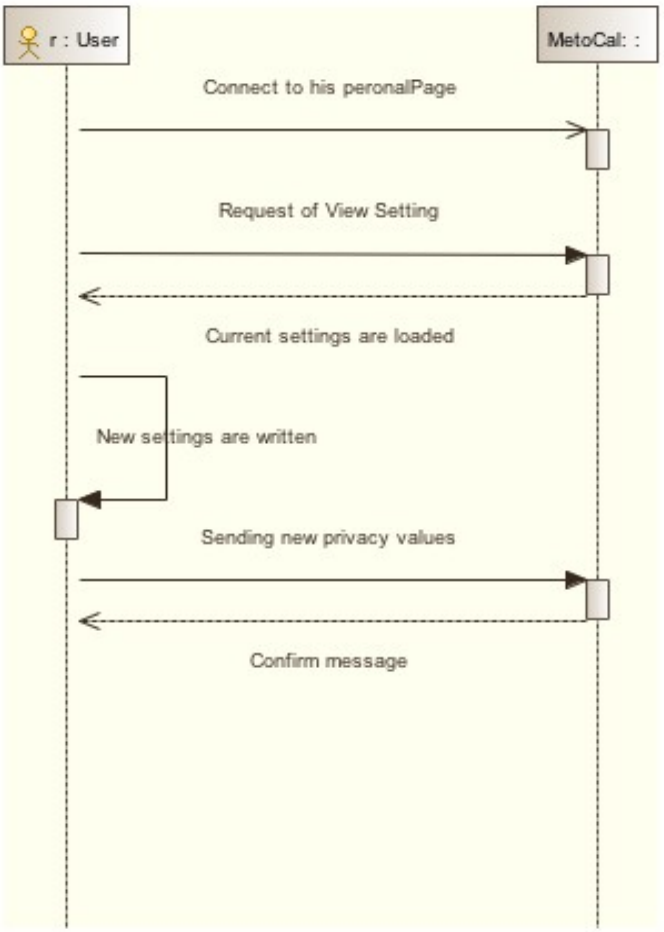
10.2.2 . Login Operation:

Login Operation	
Actors	User
Input conditions	A user (already signed-up) is connected to the website and would like to access his account.
Output conditions	A user is connected to his own calendar personal page.
Flow of events	1. The Guest is brought to a page that contains a registration form. 2. Then the Guest has to fill in the login fields with his credentials. 3. The system checks the confirmed values insert by the guest. 4. The system loads the event's locations. 5. The system asks to the Weather Service, the locations' weather conditions. 6. The system computes the notifications and invites for the user, and provides him his personal page.
Exceptions	If the user U introduces erroneous credentials, the system doesn't lead U to his personal page and instead displays an error message.
Sequence diagram	



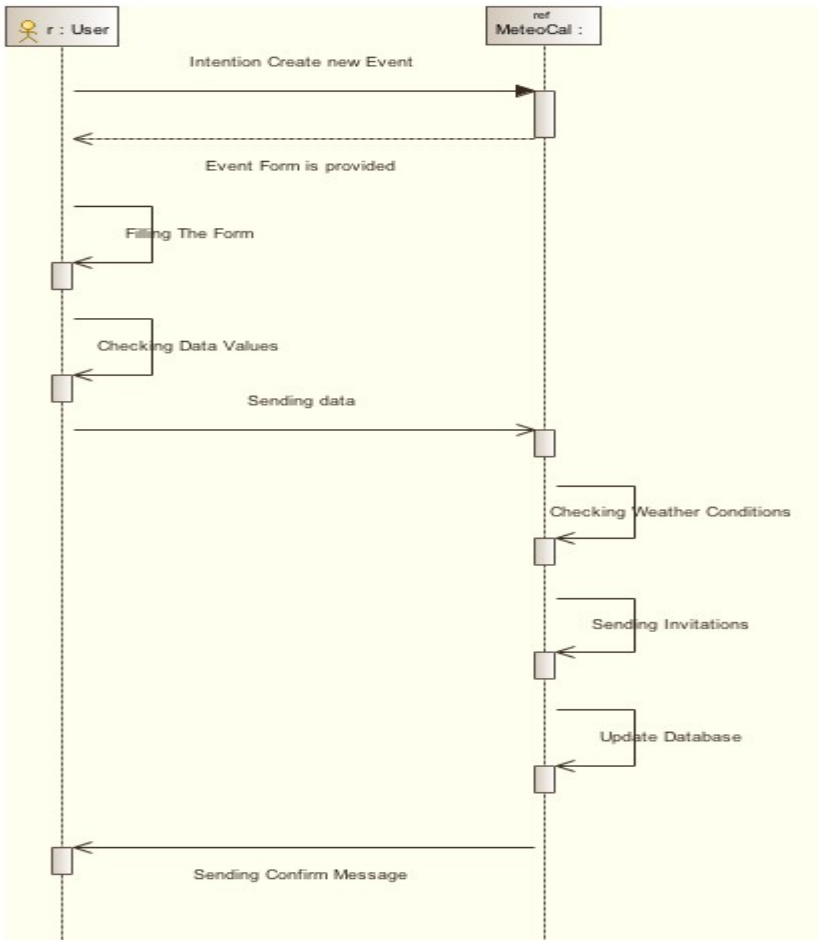
10.2.3. Settings Management:

Settings Management	
Actors	User
Input conditions	A user U, already logged in the system.
Output conditions	Personal settings should be different from the initial ones.
Flow of events	1. The user is on his personal page. 2. The user decides to edit his privacy settings. 3. He modifies the visibility settings of his calendar, or modifies data of his account. 4. He is save the modifications and is brought back to his personal page.
Exceptions	The same password's constraints mentioned earlier have still to be verified.
Sequence diagram	



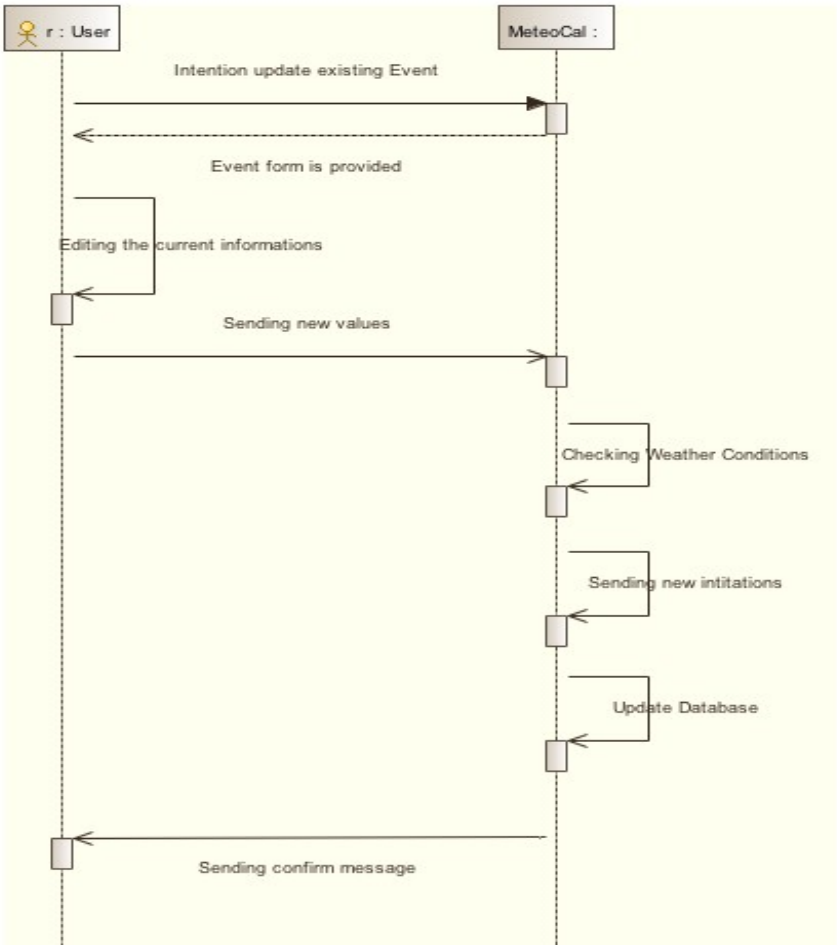
10.2.4. Event Creation:

Event Creation	
Actors	User
Input conditions	The user is already logged in the system, with a set of events in his calendar S.
Output conditions	His calendar has a set of events S' including the events of S and the new event.
Flow of events	1. The user clicks on a cell that corresponds to a day D on his own calendar and choose to create a new event E. 2. The system provides a new form, and U has to fill out the fields with the informations about E. 3. Once that the user confirms the event creation, the system does the required checks and operations for the event. 4. A confirm message his given to the user.
Exceptions	The usernames of the participants must be registered in the system.
Sequence diagram	



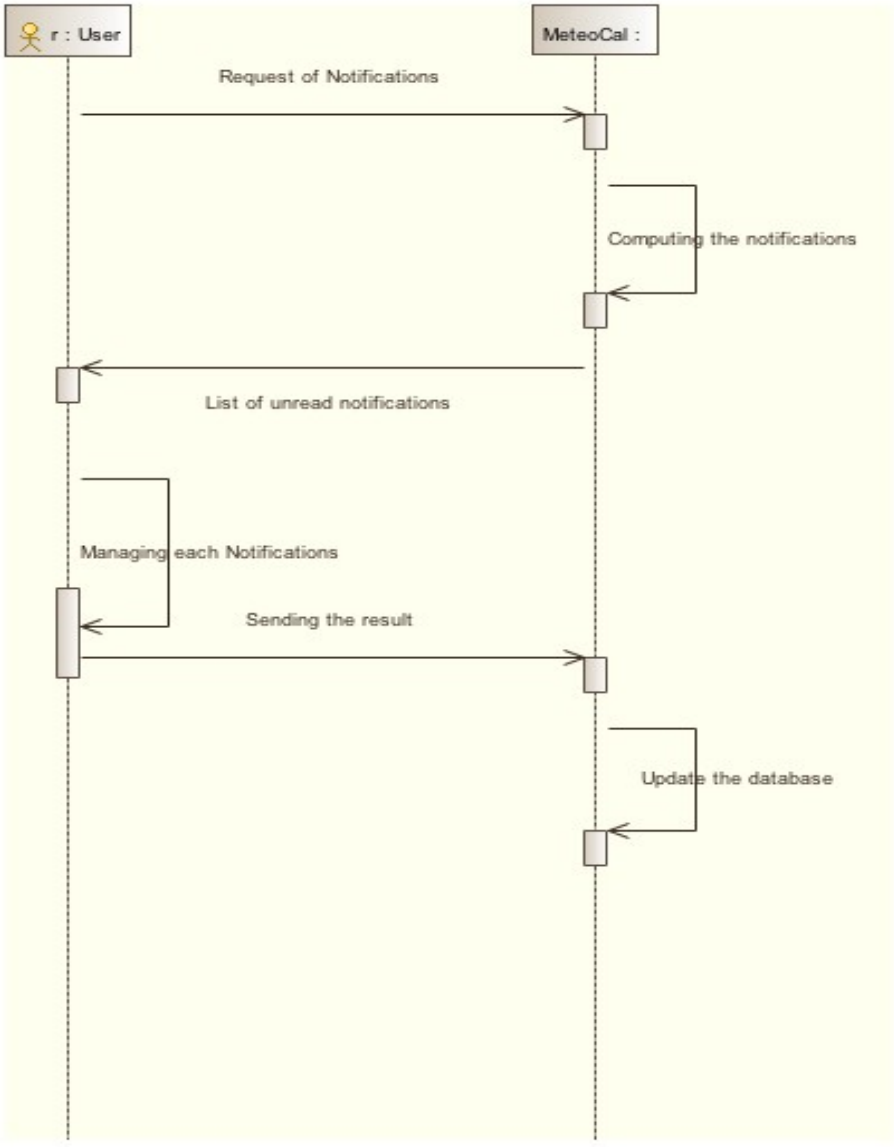
10.2.5. Event Update:

Event Update	
Actors	User
Input conditions	The user is already logged into the system, and is the creator of an event E.
Output conditions	The edited event E, has a different configuration.
Flow of events	1. The user clicks on a cell of a day D with at least one event E such that the user is the event creator, and decide to edit E settings. 2. The user edits the features of E regarding its scheduling, participants or the description. 3. The user has to confirm the new E's features. 4. A notification is sent to all the participants except to the creator.
Exceptions	The same exceptions that could occurs in the event creation phase.
Sequence diagram	



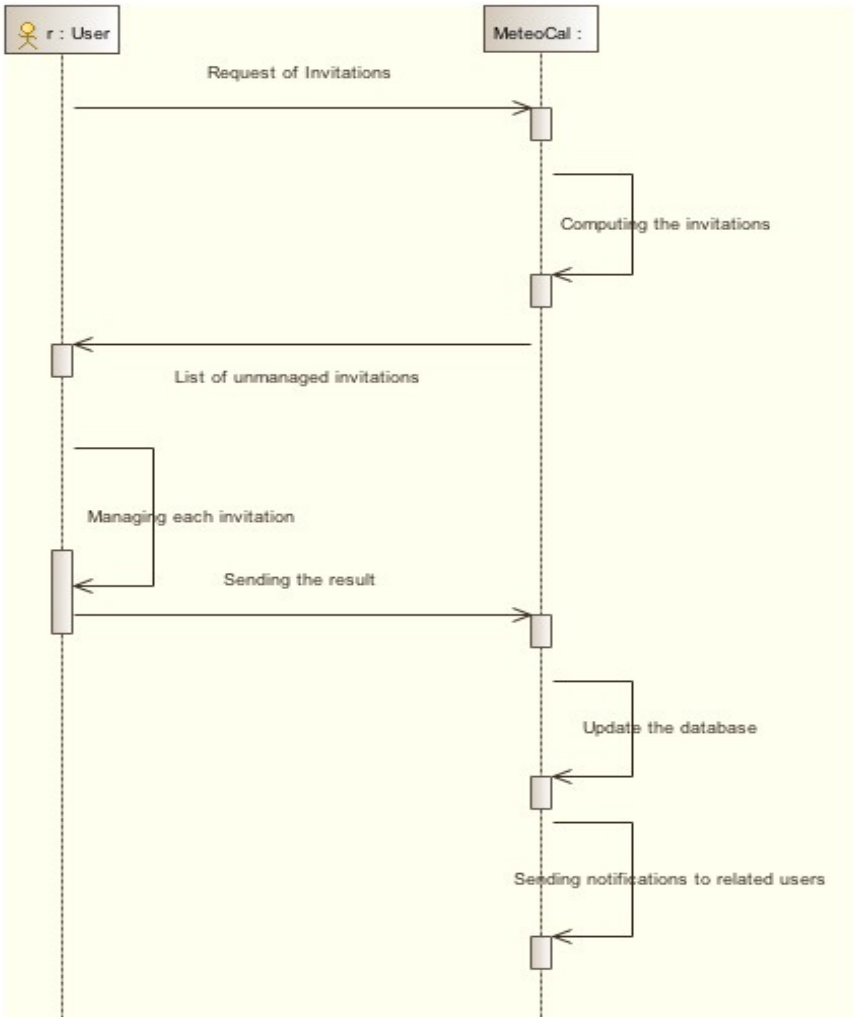
10.2.6. Manage Notifications:

Manage Notifications	
Actors	User
Input conditions	The user is already logged in the system, and wants to see the latest received notifications.
Output conditions	The list of the notifications not managed yet.
Flow of events	1. The user clicks on the notification page's link. 2. The systems provides all the notifications not read yet. 3. The user may manage his notifications just by reading them and signed as already read.
Exceptions	No exceptions here.
Sequence diagram	



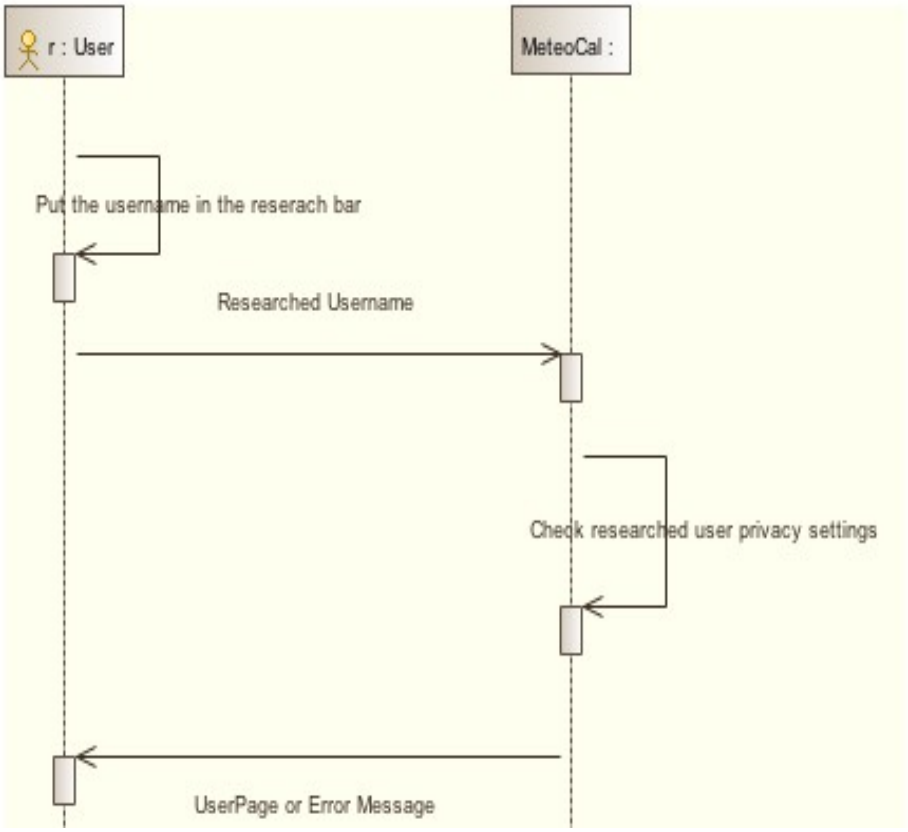
10.2.7. Manage Invitations:

Manage Invitations	
Actors	User
Input conditions	The user is already logged in the system, and wants to see his latest invitations.
Output conditions	A list of the invitations not managed yet.
Flow of events	1. The user clicks on the invitations page's link. 2. The systems provides all the invitations not managed yet. 3. He may accept or discard each invitation.
Exceptions	No exceptions here.
Sequence diagram	



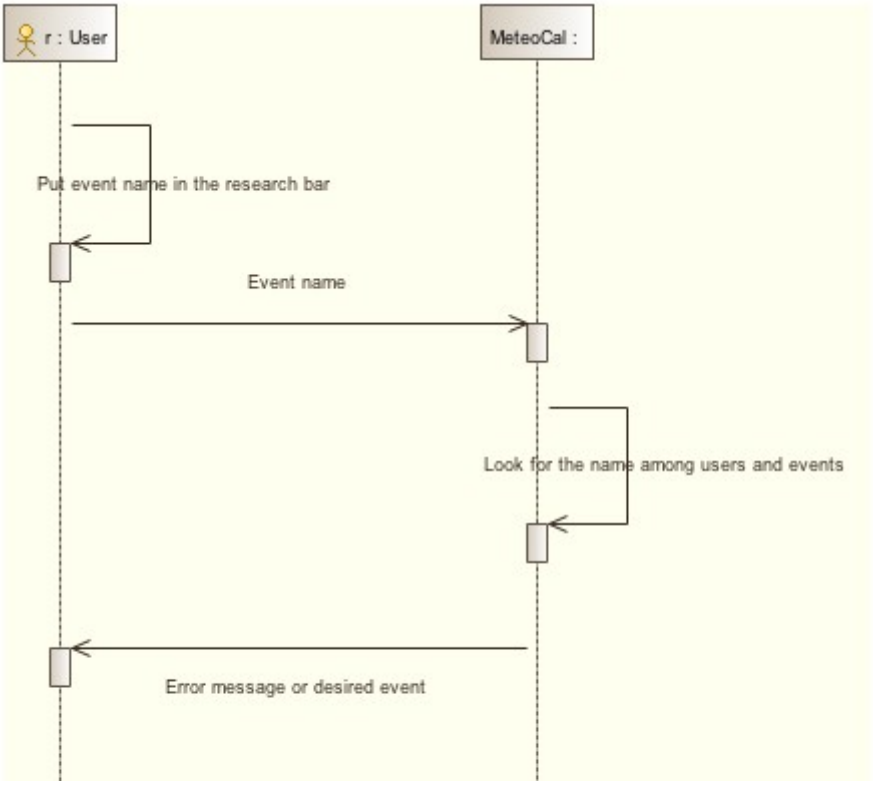
10.2.8. User Research:

User Research	
Actors	User
Input conditions	The user is already logged in the system.
Output conditions	Depending on the researched user's settings, the user will be brought to the researched page or will received an error message.
Flow of events	1. The user writes the username of a second user in the research bar. 2. If the desired user has a public calendar, the user will be brought to his page, otherwise the user will receive an error message.
Exceptions	If there is an event with the same name, a chosen page is provided.
Sequence diagram	



10.2.9. Event Research:

Event Research	
Actors	User
Input conditions	A user U already logged in the system, who is looking for an event E uses the set of keywords K in the search bar.
Output conditions	A set of events according to the keywords.
Flow of events	1. U selects the event research functionality. 2. U fills in the text field with the desired event keywords Ks. 3. The system computes a set of compatible events Es. 4. The system shows the possible empty list of events Es.
Exceptions	If there is an user with the same name, a chosen page is provided.
Sequence diagram	



11. Further additions

11.1 Software and Tools

The tools used in the production of this document are listed below:

- *OpenOffice Writer:*
  to redact this document.
- *Alloy analyzer 4.2:*
  production, execution and visualization of the Alloy model.
- *Modelio:*
  UML and sequence diagrams.
- *Cacoo:*
  designing of the UX sketches presented.

11.2 Production Report

The following report of project hours is provided to give an indication of the efficiency of this collaboration.

- *Andrea Bignoli: 40-45 hours*
- *Leonardo Cella: 30-35 hours*

It should be noted that part of the time spent on this production was shared. We estimate the amount of shared work being around 4-5 hours.