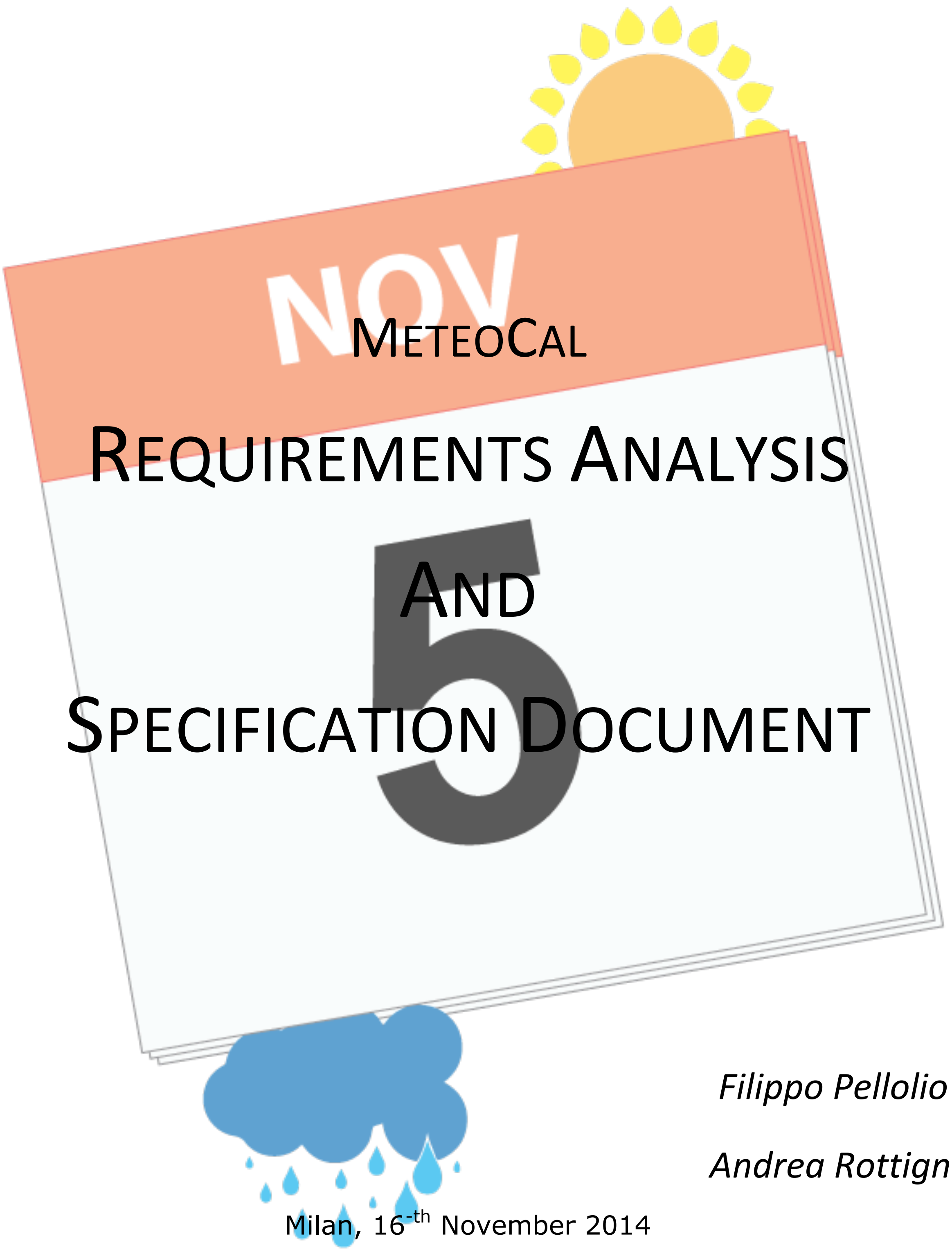
Software Engineering Project



MeteoCal

Requirements Analysis And Specification Document

Filippo Pellolio

Andrea Rottigni

Milan, 16-th November 2014

1 INTRODUCTION

In these document we have discussed some features regarding our project called MeteoCal, which is a weather-based calendar.

After this short introduction, we have analyzed the problem, providing a rather detailed description of it, followed by two sections containing the list of goals and domain properties. Then we have inserted a glossary to make the document more readable and understandable. After that, we have explained what assumptions we have made about the specification and some considerations about the system that we are going to realize.

In the third part of this document we have identified the actors that will interact with our system and we have listed them providing a short description.

In the fourth part we have discussed about functional and non-functional requirements. For what concerns functional requirements we have made a list, for each actor, of all the functionalities that the system should provide. For non-functional ones we have mostly shown some mock-up of the main pages of our web application. We have then briefly touched on architectural consideration.

The fifth part is a rather detailed description of the main functionalities that our system will provide at the end of the development.

In the sixth part we have provided a list of the most important scenarios that can occur during the usage of our system.

The seventh part contains UML diagrams. Firstly, using the scenarios of the previous chapter, we have obtained some use cases. For each use case we have provide a diagram and an itemize description. Secondly, we have realized some sequence diagrams for the most important use case. Then we have realized some state chart diagrams and activity diagrams.

In the eighth part we have proved the consistency of our model through Alloy. We have shown some examples of world generated by the Alloy analyzer in order to demonstrate that our model is consistent and we have reported some assertions to prove the respect of the constraints.

The last part is a list of the tool used to realize this document.

2. ANALYSIS OF THE PROBLEM

We will project and implement for the X software house MeteoCal, an online weather based calendar, that will allow people to schedule their events and receive constant notifications about the forecasted weather conditions on the time of the event.

The user, once registered and logged in, will be able to create or join events and calendars. When an event is created the type and location of such event must be specified, based on this information the system will provide the right weather notifications (for instance an indoor event doesn't need a rain notification). An event can be public or private, if it is public other people will be able to see and join it. A user can also invite people to his private events making them restricted. A calendar can be public too, allowing other users to see when the owner of the calendar is busy, but they won't see what event he is attending unless they are attending it too.

If the forecasted weather conditions aren't suitable for an event the user has in his calendar the system will notify him by mails and on-site notifications and, if asked, will provide a new date for it in which the weather will be more adapt.

An initial set of events types will be created by the administrator of the system, the users can also create some personalized types and use them.

2.1 GOALS

- At the end of the implementation the site will allow the visitor to:
- Sign up or log in
- Create a calendar
- Make a calendar public
- Share a calendar with another user
- Create an event
- Make an event public
- Invite people to his events
- Join a public event
- Accept an invite to a restricted event

- Receive notifications about the weather conditions before his events
- Modify information about an event he has created

2.2 Domain Properties

We take these properties for granted in the analyzed environment:

- The forecasted weather conditions are accurate
- The user really attends every event he declares
- Every event takes place in the specified location, and that location exists

2.3 Glossary

Event Type: A brief description of the event and, most important, it is associated with the weather conditions in which it can be attended.

Guest: A visitor who's either not logged in as a user or not registered; he can access a limited number of functions, he basically can only sign up or log in.

User: A visitor registered in the system who's already logged in, he can access all the functions described in this document, except for the ones specified for the administrator.

Administrator: A special type of user who can add new default events types.

Event: A planned occasion or activity (such as a social gathering).

Public Event: An event that can be attended and added to the calendar by anyone, as long as they know its name.

Restricted Event: An event that can be attended only by the user who created it and the users he decides to invite to it. The invited users are also able to see it but not to modify it.

Private Event: An event that can be seen and attended only by the user who created it. If the creator decides to invite someone else it becomes restricted.

2.4 Assumptions

- Bad weather conditions means different things based on the event type: for a soccer game snow is a bad weather condition, meanwhile for a skiing trip it may not be a problem.
- Reading the specifications it wasn’t clear if a user could have more than one calendar. We assumed that a user can have more than one calendar, we thought this could be useful for separating events related to different aspects of the user’s life (such as a work-related and a family-related calendar)
- The definition of public event wasn’t clear enough in the specifications. For this reason we decide to distinguish between public and private events. In particular, in order to participate to a public one, the owner’s invitation is not mandatory. On the contrary, the only way in which a user can add a private event to his calendar is by accepting the invitation sent by the event’s creator.

2.5 Proposed System

We propose a web platform that will provide users the following features.

Users can easily create and manage events and calendars. They will also be able to see notifications, sent by the system, about weather conditions. Our platform will allow every user to see invitations sent by other user to him; they can accept or decline them.

The page of the administrator will be different in order to allow him to add new type of events suggested by metocal’s users.

2.6 Other Considerations

We think the final result will have a look and feel similar to the market solutions already available (such as Google Calendar or Apple Calendar), because it seems the best solution to make the site really easy to use and understandable for a novice user.

3 ACTORS IDENTIFYING

The relevant actors in the system that we are going to develop are three:

- **Guest:** a guest is a visitor who is not registered or not already logged in. He can only exploit the functionalities provided by the system to sign in, if not register yet, or to log in.
- **User:** a user is a visitor who is not only registered, but also already logged in. He can exploits all the functionalities develop.
- **Administrator:** the administrator is the supervisor of the system and he is the only person who can add new type of events into the system, making them available to all the users.

4. REQUIREMENTS

Combining the goals and the assumptions we made about the system we can derive this requirements the final system is going to provide:

1. **Registration of a new user in the system**
2. **Authentication of a visitor as a registered user**
3. **Managing of personal information**
4. **Creation of a new calendar**
    - Choice between public and private
5. **Creation of a new event**
    - Choice between public and private
    - Choice among the default and personalized types
    - Creation of a new personalized event type
    - Choice of the optimal weather conditions
    - Choice of the location of the event
6. **Managing of an owned event**
    - Option to invite other users
    - Change the event type
    - Change the location of the event
    - Change the date of the event
7. **Managing invitations**
    - Decline an invitation
    - Add the event to a calendar
8. **Consultation of notifications**
    - Postpone an event to a suggested date

4.1 Functional Requirements

Now that we know what the system will allow to do, we are going to state which of these functionalities every actor will be able to access:

Guest:

- Sign up
- Log in

User:

- Create a calendar
- Change a calendar privacy settings
- Delete a calendar
- Create an event
- Create an event type
- Change the information about an event he created
- Delete an event he created
- Invite another user to an event
- Accept an invitation
- Decline an invitation
- Search for public events
- Add a public event to one of his calendars
- Remove an event from one of his calendars
- See his notifications
- Update profile information

Administrator:

- Create a new default event type

4.2 NON FUNCTIONAL REQUIREMENTS

4.2.1 USER INTERFACE

We will develop a web application and for this reason the user interface will be well suited for this kind of application. Our purpose is to realize a nice and user-friendly interface in order to make as easy as possible to understand how to use MeteoCal and to attract the largest number of people. In this section, through some mockup, we will show a first sketch of our application.

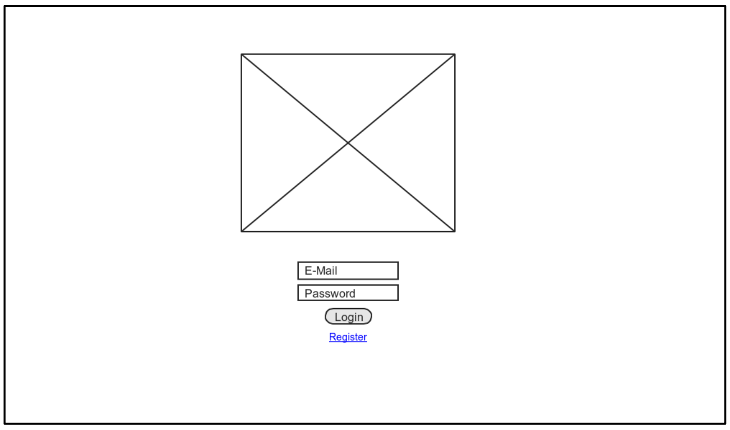


Fig. 1: Login and registration page

In Fig. 1 is represented the page where a visitor can either login or sign in through the apposite link. In the center of the page there is the logo of MeteoCal. It's very simple and intuitive to understand how to use the functions provided by this page.

We have decided to provide to MeteoCal's users the possibility to decide the style of their calendar. In particular they will be able to see the calendar organized per month, per week or focus on a single a day. In Fig. 2,3,4 are shown these three layouts. They have some common elements such as the upper bar where there is the MetoCal's logo, a field that allows user to look for events or other users, the icon of notifications with a badge present only in case of new notification, the profile picture of the user and his name. On the left part there is section that allows selecting one ore more calendars and a button to create a new calendar. Under the main bar there are some buttons that allows navigating among months, days or weeks and a button to create a new event. There is also the possibility to change the layout of the calendar.

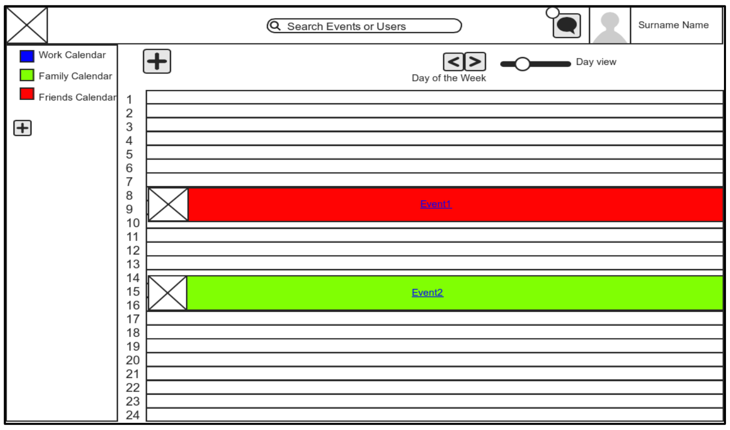


Fig. 2: Day Layout

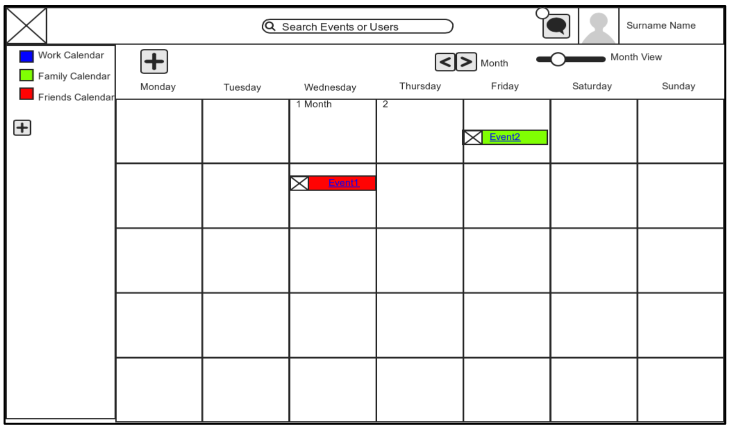


Fig. 3: Month Layout

The event colors are not casual in fact it correspond to the color of the calendar in which the event is contained. Every event has a small image that represents the forecasted weather, if available. Clicking on the event it is possible to see some details regarding the event itself.

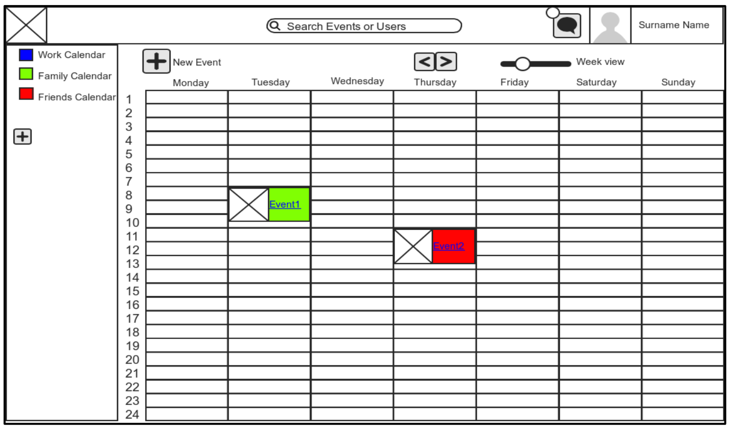
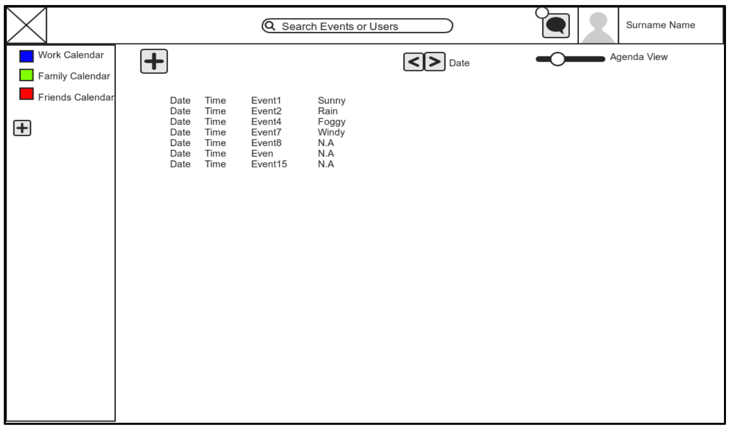


Fig. 3: Week Layout

To give our users a simply way to manage their events we will include an agenda layout that is a sort of agenda.



Date	Time	Event1	Sunny
Date	Time	Event2	Rain
Date	Time	Event4	Foggy
Date	Time	Event7	Windy
Date	Time	Event8	N.A
Date	Time	Even	N.A
Date	Time	Event15	N.A

Fig. 4: Agenda Layout

In Fig. 5 is illustrated how a user will see his notifications. In the notifications fields there some link useful to perform action related to the notification received. Also for this menu our purpose is to make it easily understandable and usable by users.

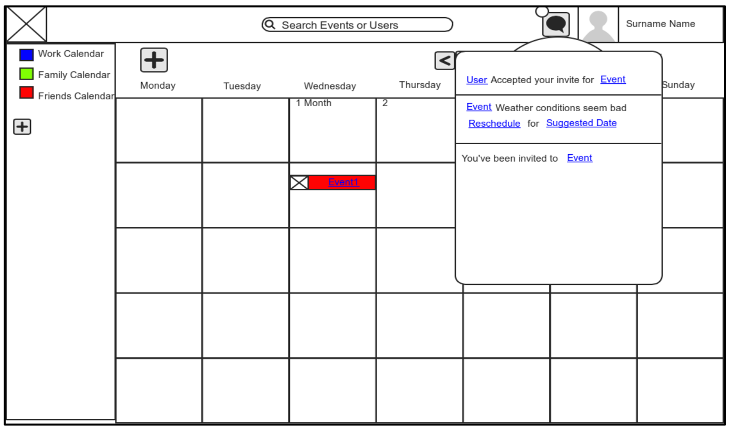
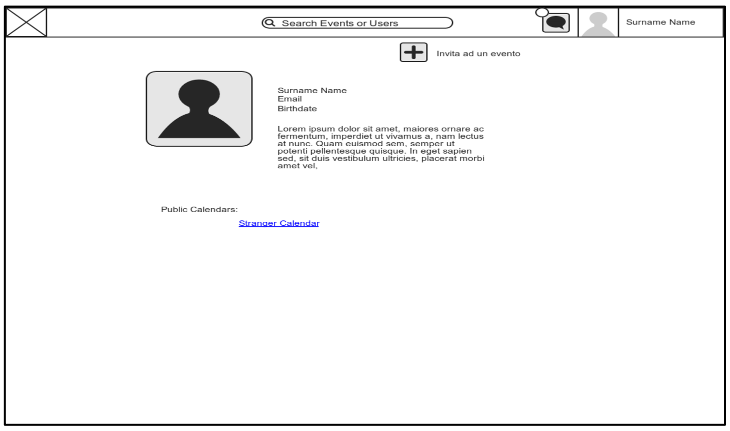


Fig. 5: Notification Menu

The user can also access his profile page or a stranger’s profile page, where he will find all the information about the selected user.



Lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque. In eget sapien sed, sit duis vestibulum ultricies, placerat morbi amet vel,

Fig. 6: Profile Page

If the user selects his own profile this page will be slightly different: no invite button will be present, the information will be modifiable and a “Save Changes” button will appear at the bottom of the page.

4.2.2 Documentation

We will draft the following documents that will help us to organize our work and to make our project as good possible.

- **Project plan:** to well organize the project phases in order to achieve the goals, respecting all the deadlines.
- **Requirement Analysis and Specification Document (RASD):** to analyze the requirements and explain the domain, the goals and the system to be develop.
- **Design Document (DD):** to provide a written description of our project, especially describing the structure of our web application and its division in tiers.
- **JavaDoc:** to make our code understandable in order to make easier future modification or maintenance.
- **Installation Manual:** a guide to install MeteoCal.
- **User Manual:** a guide to use MeteoCal.
- **Testing Document:** report regarding tests of another MeteoCal project.

4.2.3 Architectural considerations

We will use Java Platform Enterprise Edition.

Of course, some information will be stored in a database. We will provide more details about information in the class diagram because it may be consider as a first sketch of our database, but the precise structure of our database (ER diagram), will be specified in the design document.

In order to use MeteoCal an Internet connection and a recent web browser are needed.

5 SPECIFICATIONS

In this chapter we will provide a detailed explanation about the functionalities of our platform:

- The administrator will log in the same way the normal users do, but will be redirected to a reserved page.
- From the administrative page he will have the possibility to add new event types that will be visible to all the users.
- A not registered user won't be able to see anything apart from the welcome page, where he will be able to register or log in.
- While registering the visitor will be asked for some mandatory personal information such as his name.
- A user, once logged in, will be redirected to his own personal calendar page, where he will be able to see his planned events and his notifications.
- A user can search for another user using his name or his e-mail address.
- Every user can access a page showing the personal information of another user.
- Every user can access his personal information page and update it.
- The system will periodically check the forecasted weather conditions before an event and notify the user if they drastically changed either in a better or worse way.
- If the forecasted conditions for an event are not compatible with it, the system will suggest, within the notification, some new dates for the given event with a better weather prediction.
- Every user can find and add a public event to his calendar just by searching it by its name.
- Every user can see the list of participant of a public event.
- A user can invite every other user to one of his restricted events.
- An invite will show up in the invited user's notifications section when another user invites him to an event.
- When creating an event with active weather notifications the user will be forced to enter a location for it.

- If the event a user wants to create is an indoor one, he will just need to check the indoor checkbox while creating it and the weather notifications will be disabled.
- If the user can’t find a suitable event type for his event among the default ones, he can create one. He will be asked which weather conditions are compatible with the event type and, after he saves it, it can be reused every time he needs it.
- If an event is postponed by its owner a notification stating the new date will be sent to all the participants, and they will be asked to accept or decline the invite as like as it was a new event.

6 POSSIBLE SCENARIOS

Here we are going to explain some possible scenarios that can occur.

- Bob is a very busy man, but when it is possible he loves organizing excursion. He needs something that helps him to organize all his activities and, above all, something that can suggest him the best day to schedule his excursions in order to avoid rainy and foggy days. For all these reasons he decides to try MeteoCal: he goes on the MeteoCal home page and he uses the dedicated form to perform the registration to the platform.
- A week ago, Ann planned a bike ride for the following Sunday. But, three day before her trip, the forecasted weather conditions are not good: on Sunday it is likely to rain. Due to this fact the system notifies Ann proposing her another day, the closest with better weather conditions, when she can perform her activity. She decides to postpone her events.
- David decides to organize a soccer game with his friends at the local pitch for the following Friday, so he create the event in his MeteoCal page specifying Sport Event as event type and proceeds to invite 9 of his friends. During the following days David receives notifications for every accepted invite, and by Tuesday everyone replied positively. On Wednesday the system checks the forecasted weather conditions, it’s predicted to be snowing on Friday, and, since snow is listed as not compatible with the type Sport Event, the system sends a notification to David. David reads the notification on Thursday and decides to postpone the game to Saturday as the system is suggesting, since Saturday is predicted to be a sunny day. A notification regarding the date change is sent to all the participants, who need to confirm their presence again. Unfortunately George, one of David’s friends, has another game scheduled for the same date and must decline the invitation, so the event is deleted from his calendar. David accesses his MeteoCal again on Friday and checks his notifications, 8 accepted the change of date but George declined it, so he needs to find another player. Luckily he knows a lot of soccer players and sends a last minute invite to Eric, whom he knows to be free on Saturday. Eric promptly replies affirmatively and a notification is sent to David. David reads his notification and is glad to know that finally he has a full roster.

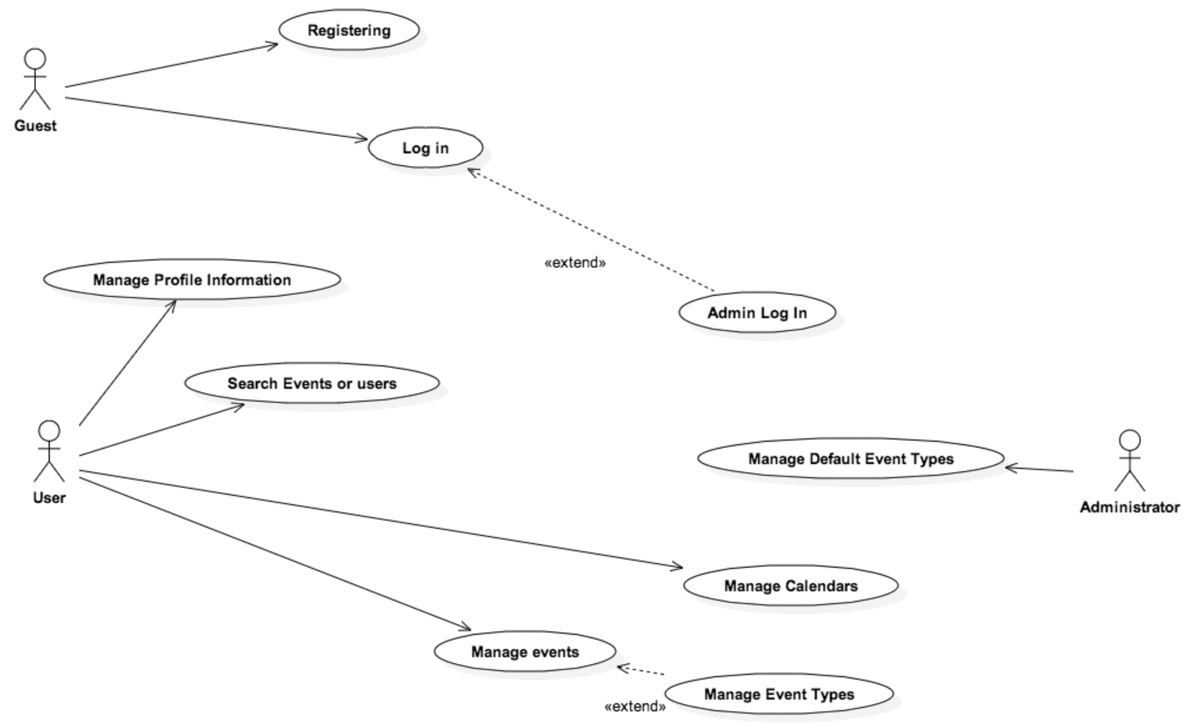
- A famous singer is going to play at the Giuseppe Meazza Stadium in two months, so his manager decides to publicize it creating an event accessible to all the users of MeteoCal. He proceeds to log in with his account and creates a new event of type Outdoor Music Concert located in Milan and he makes it public. Bruce really likes this singer and has already bought a ticket for his concert, now he can also add the official event to his calendar and he will be notified promptly if the concert will be postponed due to bad weather conditions.
- Henry needs to go to the dentist, so he search for his dentist's name on MeteoCal and he lands on his profile page, here he finds out that his dentist has a public calendar that shows his appointments. He can now choose a date to schedule his appointment where he knows his dentist to be free, the privacy of the other clients is also respected since the appointments are private events and he can only see that the dentist is not free in that lapse of time. He chooses to schedule an appointment for the next Tuesday at 10, and as soon as he finishes the phone call with the dentist's secretary he receives an invite on MeteoCal for his appointment.
- Peter is a doctor. Very often, his patients, call Jane, peter's secretary, asking for an appointment and they proposed a day and a time slot when Peter is already busy with other patients, so it is hard, for Jane, to find the right time for both: patient and doctor. Peter is already a user of MeteoCal, but, unfortunately, his calendar is private. In order to simplify the appointment procedure he decides to make his calendar public. In this way, when a patient wants to take an appointment he can go on MeteoCal and check the doctor's calendar and he can call Jane proposing a feasible time slot without losing time. All the appointment are private events, so every patient can only see when the doctor is busy, but not with whom in order to respect the privacy of Peter's patients.
- Robert is a user of MeteoCal and he decides to create an event. But, filling out the requested information, he doesn't find an appropriate event type. He notices that MeteoCal offers the opportunity to create personalized event type. So he creates the appropriate event type, specifying the suitable weather conditions to perform this event. Then he saves the event and the system adds it to Robert's calendar and sends the invite notifications to all the people invited by Robert.

- The administrator decides to delete tug of war, a default event type, because is used very rarely. Unfortunately Mike has organized a tug of war event. The system, before deleting this event type, checks if someone has selected it and it notices that Mike is the owner of a tug of war event. So the system changes this event into an indoor event and it sends to Mike a notification about the deletion, making him aware about this change.

7 UML MODELS

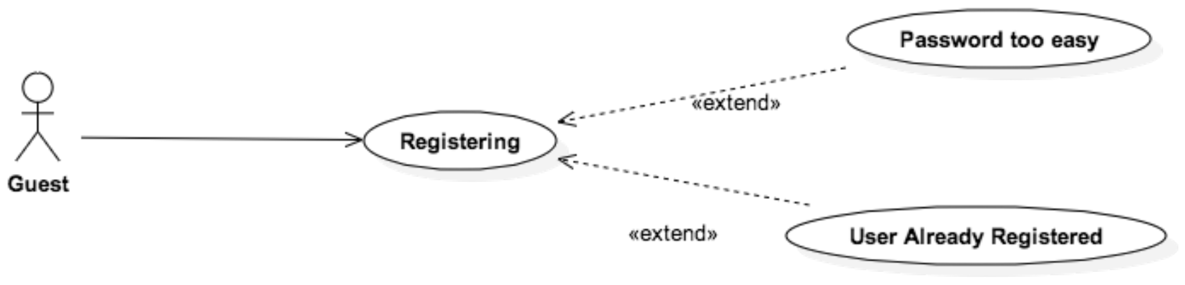
7.1 USE CASE DIAGRAMS

This is a general image that describes everything that can be done in MeteoCal, it is not a use case and it's here only to give an overall view to the reader.



In the following sections we will give detailed information about every use case.

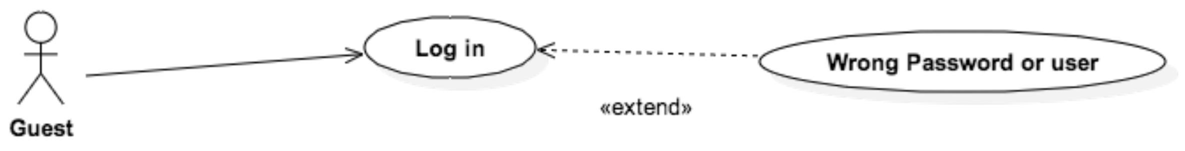
7.1.1 Registering



Name	**Registering**
Actors	Guest
Entry Conditions	The visitor must be in the MeteoCal Home Page.
Flow of Events	• The Guest clicks on the "Register" link in the Home Page. • The Guest is redirected to a form where he'll be able to enter all the information needed by the system. • The Guest correctly compiles the form. • The Guest pushes the button "Register".

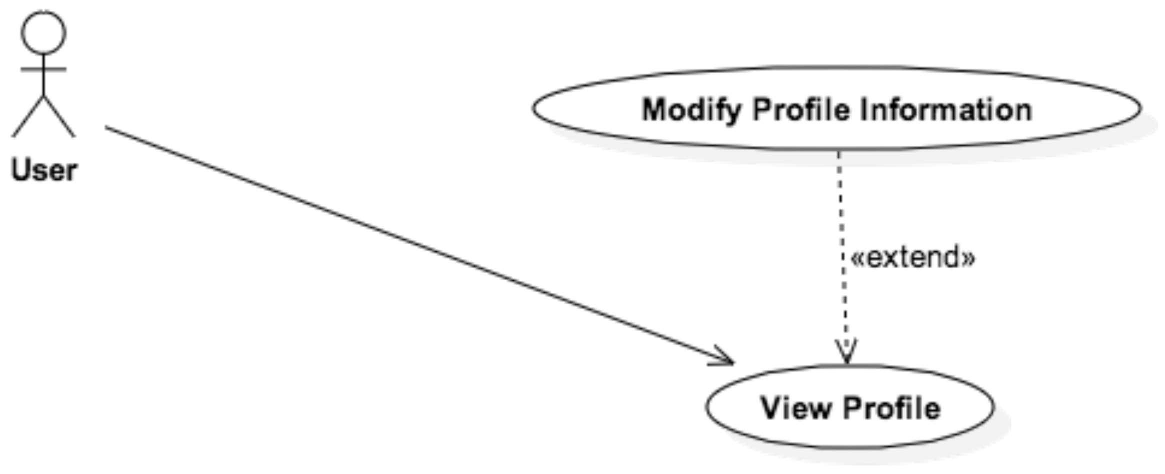
Exit Conditions	The Guest is redirected to MeteoCal’s Home Page where he will now be able to login. A new user is registered in the database
Exceptions	• If one of the information entered by the Guest is unacceptable an error message is shown. • If the password chosen by the guest doesn’t respect the System standards an error message is shown.

7.1.2 Log In



Name	**Log In**
Actors	Guest
Entry Conditions	The guest must be in the MeteoCal Home Page.
Flow of Events	• The guest inserts his e-mail and password in the appropriate fields and clicks submit. • The system checks the correctness of the data inserted.
Exit Conditions	None.
Exceptions	If the guest inserts incorrect e-mail or password an error page is loaded.

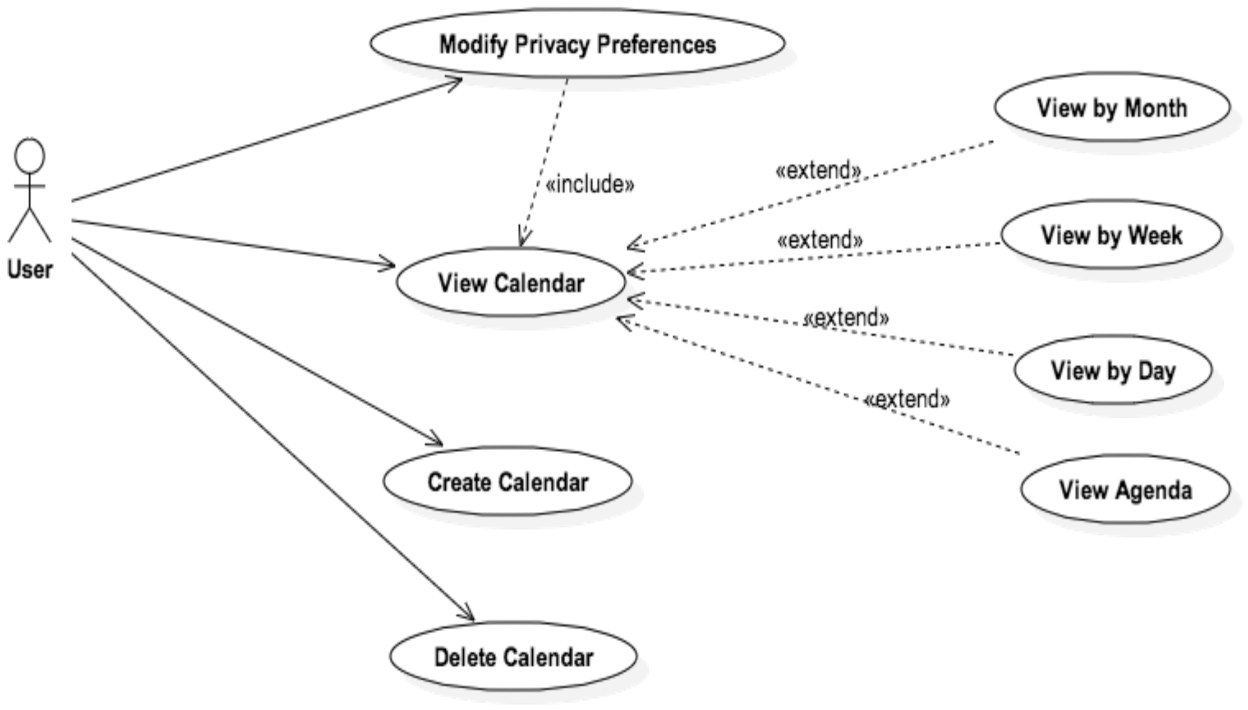
7.1.3 Profile Managing



Name	**View Profile**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The user clicks on his name in his home page. • The system loads the profile page of the user.
Exit Conditions	None.
Exceptions	None.

Name	**Modify Profile Information**
Actors	User
Entry Conditions	The user must be in his profile page.
Flow of Events	• The user clicks on the button in charge of allowing the modification of profile information. • The system reloads the profile page of the user allowing the modification of user's information. • The user changes his personal information such as name, surname, e-mail, password etc. • The user clicks save. • The system uploads the user's information.
Exit Conditions	The profile page of the user is reloaded with the new information.
Exceptions	If the user inserts some wrong information or forgets to complete some fields the system an error message is loaded.

7.1.4 Calendar Managing



Name	**Change the calendar view**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The User chooses the view he wants to see through the slider upon the calendar. • The System reloads the calendar with the chosen view.
Exit Conditions	None.
Exceptions	None.

Name	**Select the visible calendars**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The User chooses the calendars he wants to be

	shown in the calendar main page through the checkboxes in the left bar of the page. • The System reloads the calendar hiding and showing the proper events.
Exit Conditions	None.
Exceptions	None.

Name	**Change a calendar privacy settings**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The User clicks on the name of one of his calendar in the checkboxes list in the left bar. • The System redirects the user to a page that shows the information about that calendar. • The user changes the privacy settings through a radio button in the page (choosing public or

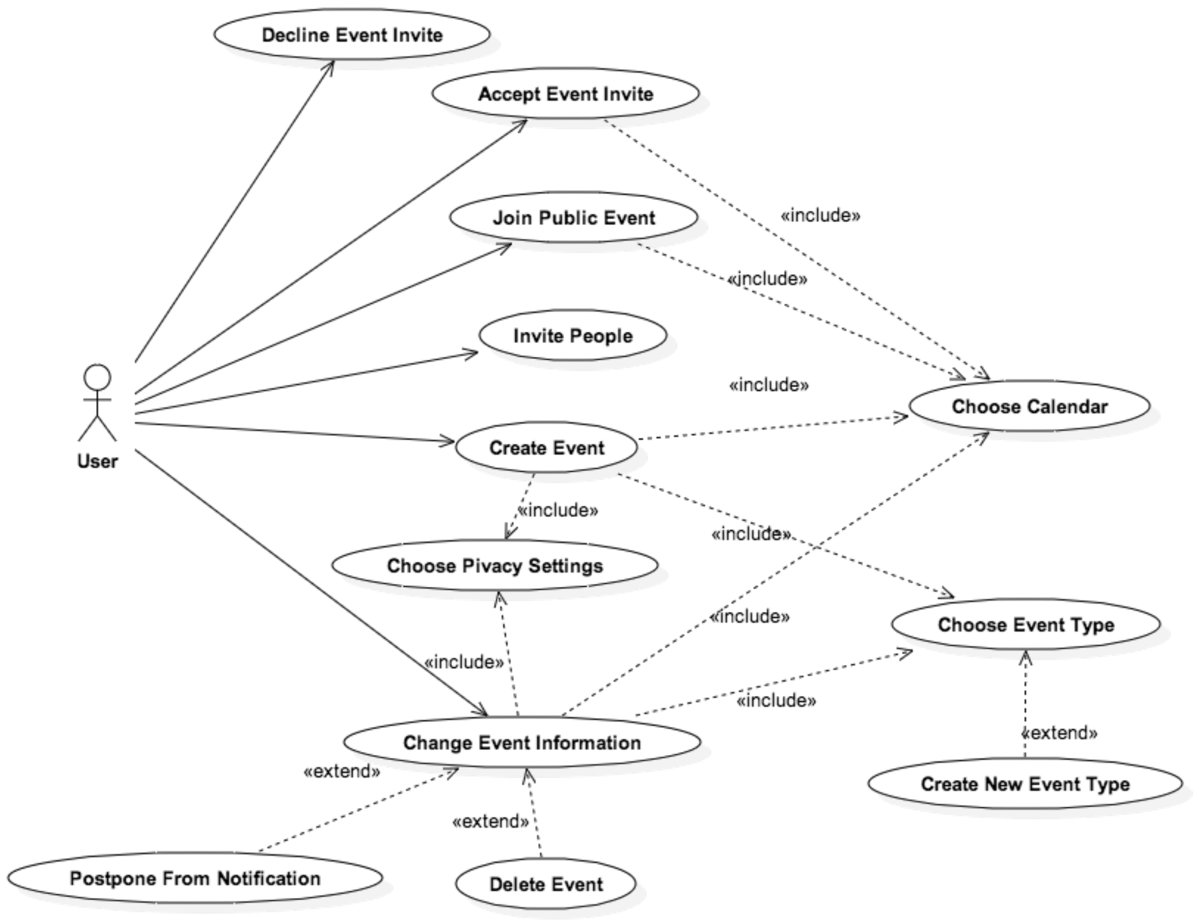
	private). • The user pushes the “Save” button.
Exit Conditions	• The System redirects the user to his Home Page and the new information is stored.
Exceptions	• The user exits the page without pushing the “Save” button. In this case the information is discarded.

Name	**Delete a Calendar**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The User clicks on the name of one of his calendar in the checkboxes list in the left bar. • The System redirects the user to a page that shows the information about that calendar. • The User pushes the “Delete Calendar” button.

	• The System asks the user, using a popup, if he’s sure he wants to delete that calendar and drop all the events on it. • The user selects “Yes”.
Exit Conditions	• The System redirects the user to his home page, the calendar is deleted from the database and all the events attached to it are removed, notifying the other participant users if necessary.
Exceptions	• The user exits the page without pushing the “Delete Calendar” button. In this case nothing changes in the system. • The user selects “No” when asked if he’s sure. In this case he remains in the calendar information page and the calendar is not deleted.

Name	**Create a Calendar**
Actors	User
Entry Conditions	The user must be in his Home Page.
Flow of Events	• The User clicks on the plus sign under the last calendar checkbox in the left bar. • The User is redirected to a form where he will insert the information about the new calendar. • The User inserts all the information needed. • The User clicks on the “Create” button.
Exit Conditions	The user is redirected to his Home Page, the new calendar is created and it will be visible in the left bar.
Exceptions	• The user exits the page without pushing the “Create” button. In this case the information is discarded.

7.1.5 Events Managing



Name	**Create Event**
Actors	User
Entry Conditions	The user must be in his home page.
Flow of Events	• The user clicks on the button that allows creating event. • The system shows him the page in charge of creating events. • The user must choose the event type, invoking the Choose Event Type use case, the privacy settings, invoking the Choose Privacy Settings use case, the calendar, which the event belongs to, invoking the Choose Calendar use case and invite people, invoking the Invite People use case. • The user clicks create. • The system adds the event to the user's calendar.

Exit Conditions	The user is redirected to his home page.
Exceptions	In case the user inserts some incorrect information, such as not select an event type, the privacy preferences or the calendar, an error message is displayed.

Name	**Change Event Information**
Actors	User
Entry Conditions	The User must be in his home page or in the notification menu.
Flow of Events	• If the user is in his home page he must select one of the events from one of his calendar. If he is the notification menu he must clicks on postpone, invoking the Postpone From Notification use case. • The system shows a page that allows the user to modifying event information.

	• The user can change the event information. In particular, he can change the privacy preferences, invoking Choose Privacy Setting use case, the event type, invoking the Choose Event Type use case, and the calendar, which the event belongs to, invoking the Choose Calendar use case. He can also delete an event invoking the Delete Event use case. • The user clicks save or delete. • The system modifies the event information in the database. • The system notifies all the participants of the event about the modification or the deletion.
Exit Conditions	The user's home page is loaded.
Exceptions	In case the user commits something wrong, such as not select an event type, the privacy preferences or the calendar, an error message is displayed.

Name	**Invite People**
Actors	User
Entry Conditions	The User must be in the event page or on the profile page of another user.
Flow of Events	• The user click on the button that allows inviting people. • If the user is in the event page the system shows a list of all people that are user of MeteoCal. The user selects all the people that he wants to invite. • If the user is on the profile page of another he can invite him clicking on the apposite button. • The system sends an invite notification to the selected people.
Exit Conditions	The user’s home page is loaded.
Exceptions	If the user closes the application before the clicking invite no invite notification is sent.

Name	**Choose Privacy Settings**
Actors	User
Entry Conditions	The User must be in the page that allows creating or modifying event.
Flow of Events	• The user chooses the privacy preferences of the event he is creating. • The user clicks save. • The system sets the privacy settings according to the user's preferences.
Exit Conditions	The home page of the user is loaded.
Exceptions	None.

Name	**Choose Calendar**
Actors	User
Entry Conditions	The User must be in the page that allows creating or modifying event. Otherwise, he must have received an

	invitation and click on accept in the notification menu. Or he must be in the searching event page, clicking on a public event.
Flow of Events	• The user chooses the calendar, which will contains the event he is creating, adding or modifying. • The user clicks save. • The system adds the event to the selected calendar.
Exit Conditions	The user's home page is loaded.
Exceptions	If no calendar is selected an error message is shown.

Name	**Choose Event Type**
Actors	User
Entry Conditions	The User must be in the page that allows creating or modifying event.

Flow of Events	• The user chooses the event type among the default ones or he creates a new event type, invoking the create event type use case. • The user clicks save. • The system associates the selected event type to the event.
Exit Conditions	The user's home page is loaded.
Exceptions	If no event type is selected an error message is shown.

Name	**Create New Event Type**
Actors	User
Entry Conditions	The User must be in the page that allows creating or modifying event and he is adding an event type to the event.
Flow of Events	• The user chooses to insert a personalized event type. • The system loads a list of event type already

	created by the user and a form that allows creating new event type. • The user must complete the following fields: event type name and best weather conditions. • The user clicks create • The system adds the new event type to the event type available when the user is creating an event.
Exit Conditions	The form is closed and the user is in the page where he was before: creation event page or modifying event page.
Exceptions	If the user put some incorrect information in one of the fields such as an already existent event name, a not valid name or he forgets to select the best weather conditions the system shows an error message.

Name	**Postpone From Notification**
Actors	User
Entry Conditions	The User must be in the notification menu.
Flow of Events	• The user clicks on the link postpone or on the link select another, which is present in a notification about bad weather conditions. • If the user has clicked select another date the system loads a form where the user can change the event date. • The user clicks postpone. • The system modifies the event sending a notification to all the participants.
Exit Conditions	The user's notification menu is reloaded.
Exceptions	If the user selects an invalid date an error message is shown.

Name	**Delete Event**
Actors	User
Entry Conditions	User home page.
Flow of Events	• The user must select one of his events. • The system loads the page that allows modifying events. • The user clicks on delete. • The system notifies all the participants about the deletion. • The system deletes the event from the user's calendar.
Exit Conditions	The user's home page is loaded.
Exceptions	None

Name	**Decline Event Invite**
Actors	User
Entry Conditions	The user must be in the notification menu.
Flow of Events	• The user clicks on the decline option in the notification menu. • The system notifies the one who has invited the user about the decision.
Exit Conditions	The notification menu is reloaded.
Exceptions	None.

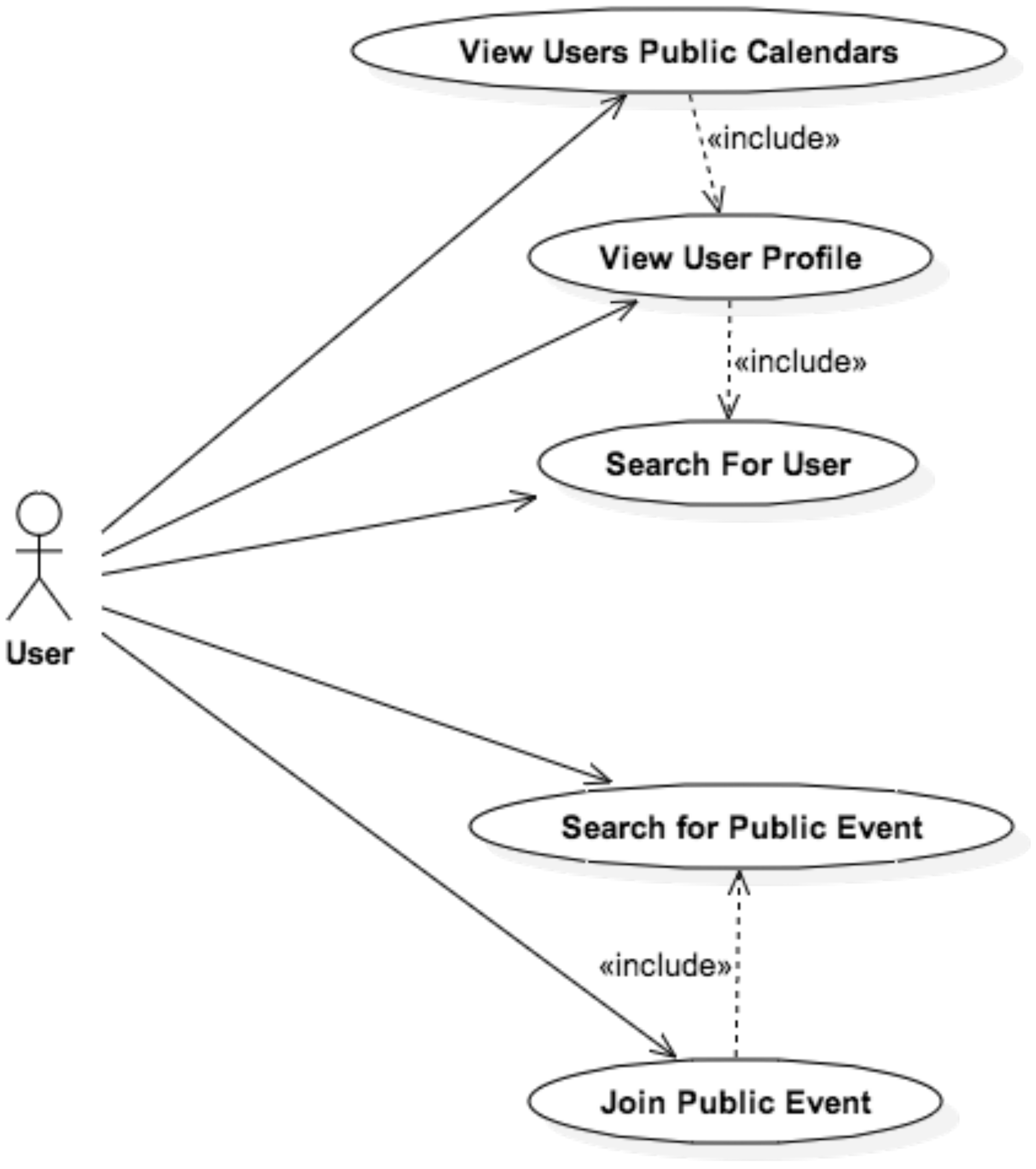
Name	**Accept Event Invite**
Actors	User
Entry Conditions	The user must be in the notification menu.
Flow of Events	• The user clicks on the accept option in the

	notification menu. • The system loads a menu that allows choosing the calendar where the user wants to add the event. • The user choose a calendar, invoking the choose calendar use case. • The system notifies the one who has invited the user about the decision.
Exit Conditions	The notification menu is reloaded.
Exceptions	If the user does not select a calendar an error message is shown.

Name	**Join Public Event**
Actors	User
Entry Conditions	The user must be in the search page.
Flow of Events	• The user clicks on one of event, which he wants

	to add to one of his calendars. • The system loads a page that allows selecting the calendar where the user wants the event to be added. • The users must select one of his calendars. • The system adds the selected event to the selected calendar.
Exit Conditions	The user’s home page is loaded.
Exceptions	If the user doesn’t select the calendar an error message is shown.

7.1.6 Search of a Public Event or another User



Name	**Searching for another User’s Profile Page**
Actors	User
Entry Conditions	The User must be in a page where the search bar is visible.
Flow of Events	• The User types the name or the email of the user he wants to find. • As the User types the System provides him possible matches in a box under the search bar.
Exit Conditions	The user is redirected to the selected user’s profile page.
Exceptions	None.

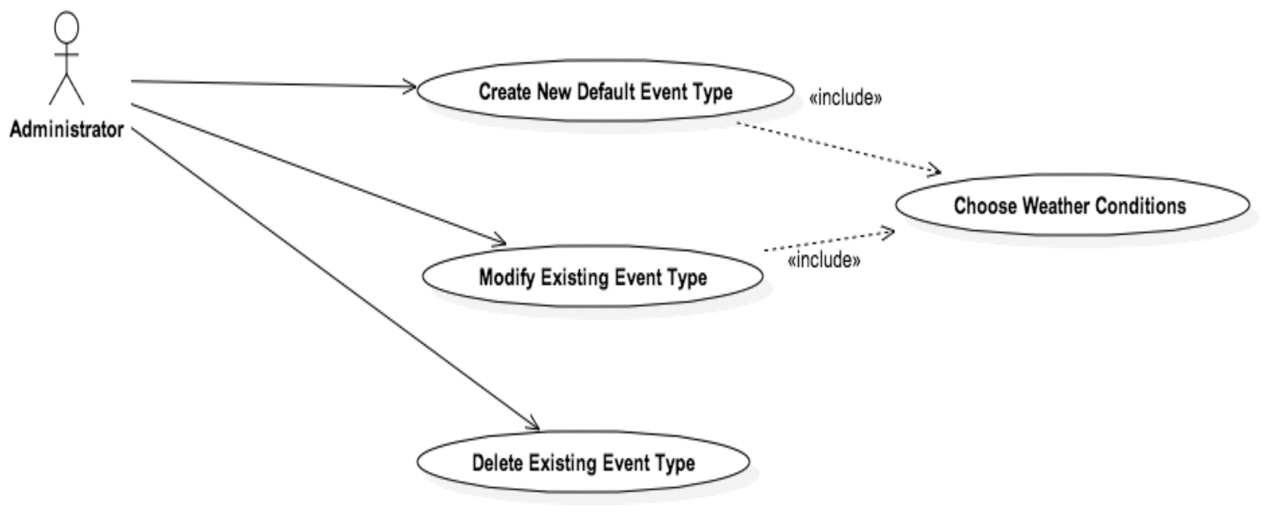
Name	**Viewing another User’s Public Calendars**
Actors	User
Entry Conditions	The User must be in the selected user’s profile page.

Flow of Events	• In the selected user's profile page the User selects, if available, the public calendar he wants to see by clicking on it in the list provided.
Exit Conditions	The user is redirected to a page that shows the public calendar in the way previously descripted.
Exceptions	None.

Name	**Searching for a public event**
Actors	User
Entry Conditions	The User must be in a page where the search bar is visible.
Flow of Events	• The User types the name of the Event he wants to find. • As the User types the System provides him possible matches in a box under the search bar. • When the wanted Event shows up in the box the user clicks on its name.

Exit Conditions	The Event will eventually show up in the designated box.
Exceptions	None.

7.1.7 Managing Default Event Types


Administrator
Create New Default Event Type
«include»
Choose Weather Conditions
«include»
Modify Existing Event Type
Delete Existing Event Type

Name	**Creating a new Default Event Type**
Actors	Administrator
Entry Conditions	The Administrator just logged in and he's in his home page.
Flow of Events	• The administrator select the option "Create new Default Event Type" • The System redirects him to a form he can fill with all the information needed. • The Administrator fill in the form, choosing the

	good and bad weather conditions for the new type of event. • The administrator pushes the “Create” button.
Exit Conditions	The Administrator is brought back to his default page; the new Default Event Type is saved and is now visible to all the users.
Exceptions	• The Administrator exits the page without pushing the “Create” button. In this case the information is discarded.

Name	**Modifying an existing Default Event Type**
Actors	Administrator
Entry Conditions	The Administrator just logged in and he’s in his home page.
Flow of Events	• The administrator selects the option “Modify existing Default Event Type”. • The System shows him a list of the existing

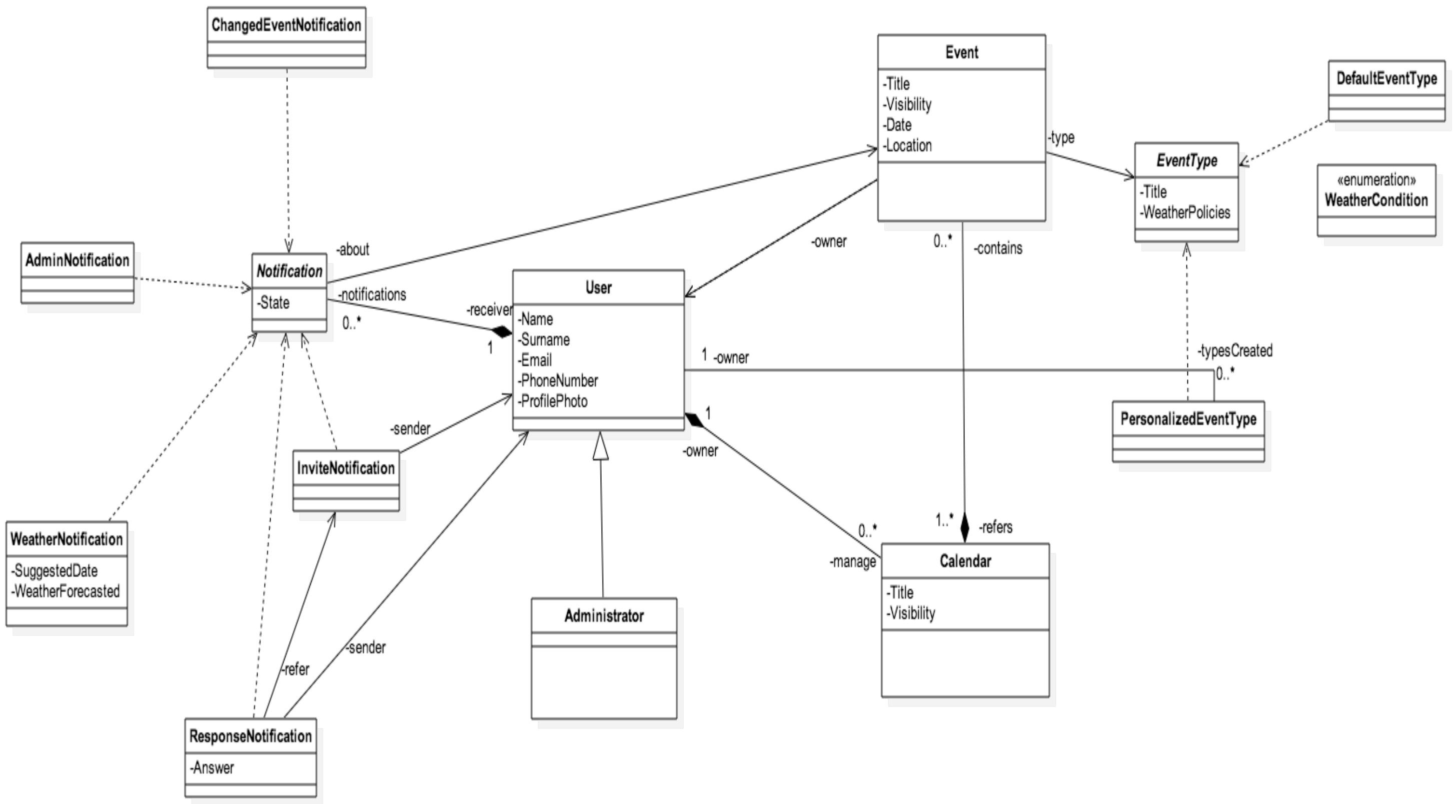
	Default Event Types. • The Administrator chooses the one he wants to modify by clicking on it. • The System redirects him to a page showing the information of the selected Event Type. • The Administrator makes the changes he wants. • The Administrator pushes the “Save” button.
Exit Conditions	The Administrator is brought back to his default page; the new information about the Default Event Type is stored.
Exceptions	• The Administrator exits the page without pushing the “Save” button. In this case the information is discarded.

Name	**Removing an existing Default Event Type**
Actors	Administrator
Entry	The Administrator just logged in and he’s in his home

Conditions	page.
Flow of Events	• The administrator selects the option "Remove Default Event Type". • The System shows him a list of the existing Default Event Types. • The Administrator chooses the one he wants to remove by clicking on it. • The System asks him a confirmation using a popup. • The Administrator selects "Yes".
Exit Conditions	The Administrator is brought back to his default page, the Default Event Type is removed from the Database, all the events using it are changed to an indoor Event Type and the owners are notified.
Exceptions	• The Administrator clicks "No" when asked if he's sure. In this case nothing changes in the System.

7.2 CLASS DIAGRAM

This is the class diagram, which represents our project:



7.3 Sequence Diagram

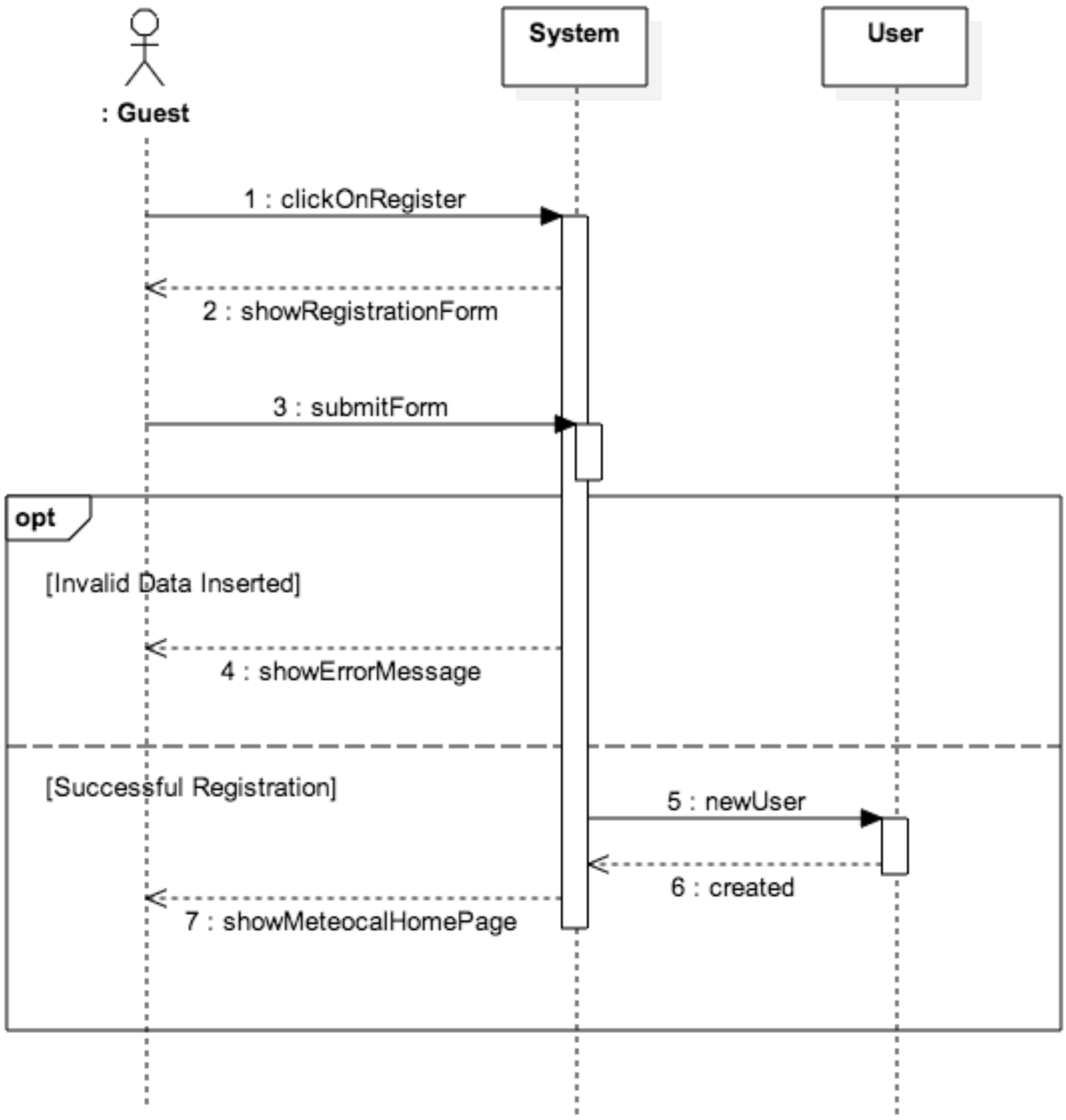
In this section we will represent some sequence diagram. They are not extremely detailed, but this is a choice to make them more readable.

We present sequence diagram for the following functions:

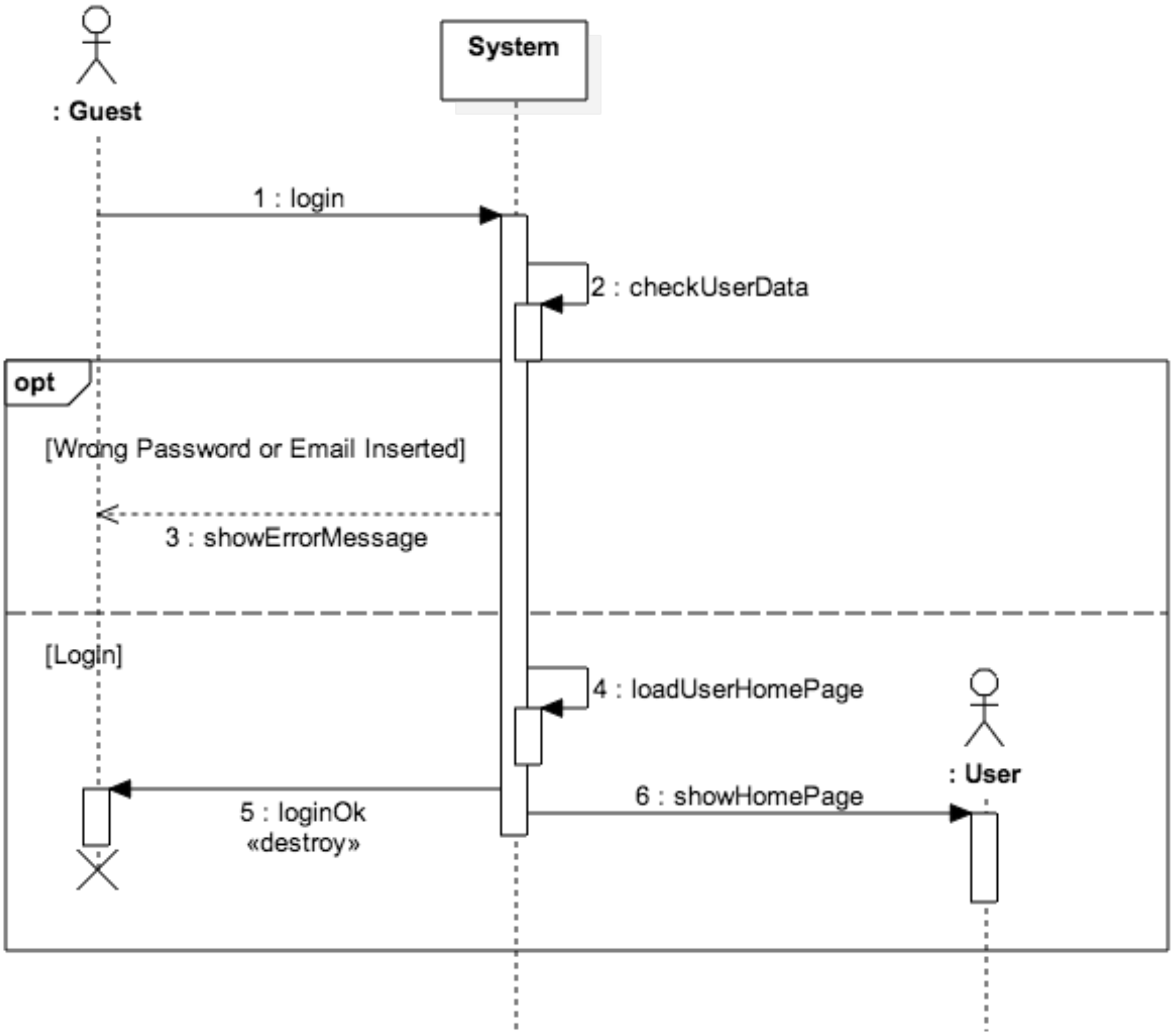
- Registration
- Log in[*]
- Create calendar
- Select the visible calendars
- Search and add public event
- Create event
- Change event information
- Create event type
- Postpone event because of a bad weather notification
- Accept or decline event invite
- View and modify profile information
- Create default event type
- Delete default event type

[*] It isn't a real sequence diagram, but it explains in a simply way what are a user and a guest in our model.

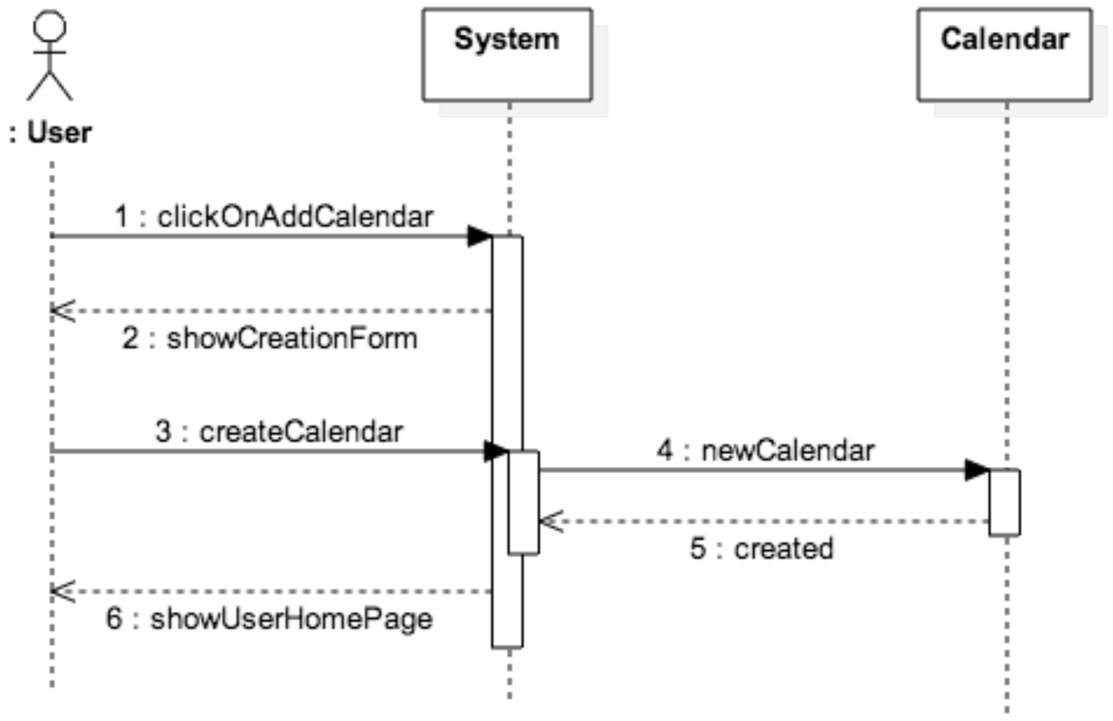
7.3.1 Registration



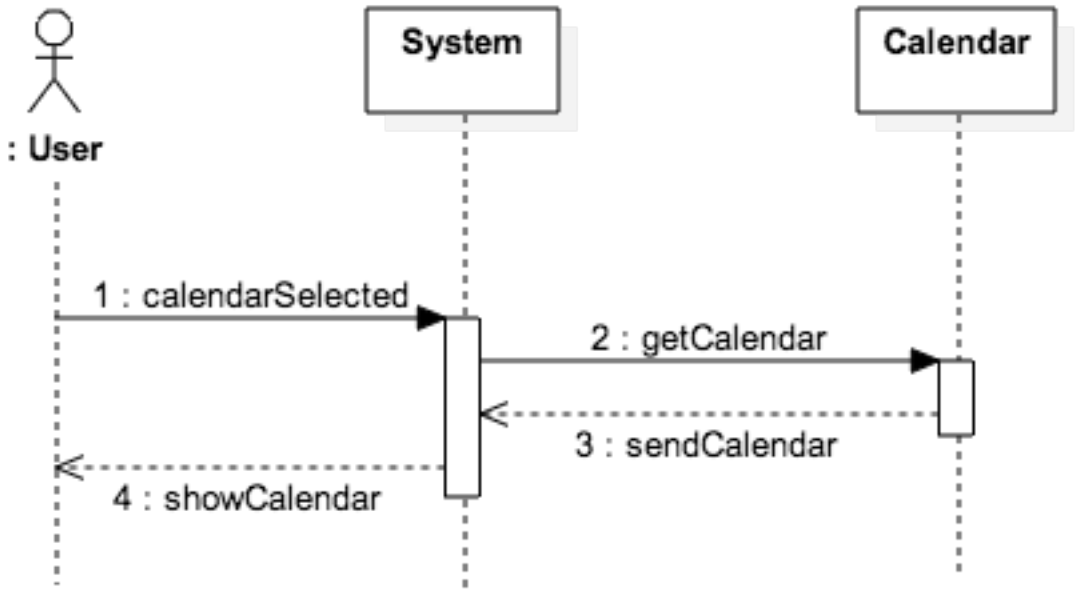
7.3.2 Log In


: Guest
System
1 : login
2 : checkUserData
opt
[Wrong Password or Email Inserted]
3 : showErrorMessage
[Login]
4 : loadUserHomePage
: User
5 : loginOk
«destroy»
6 : showHomePage

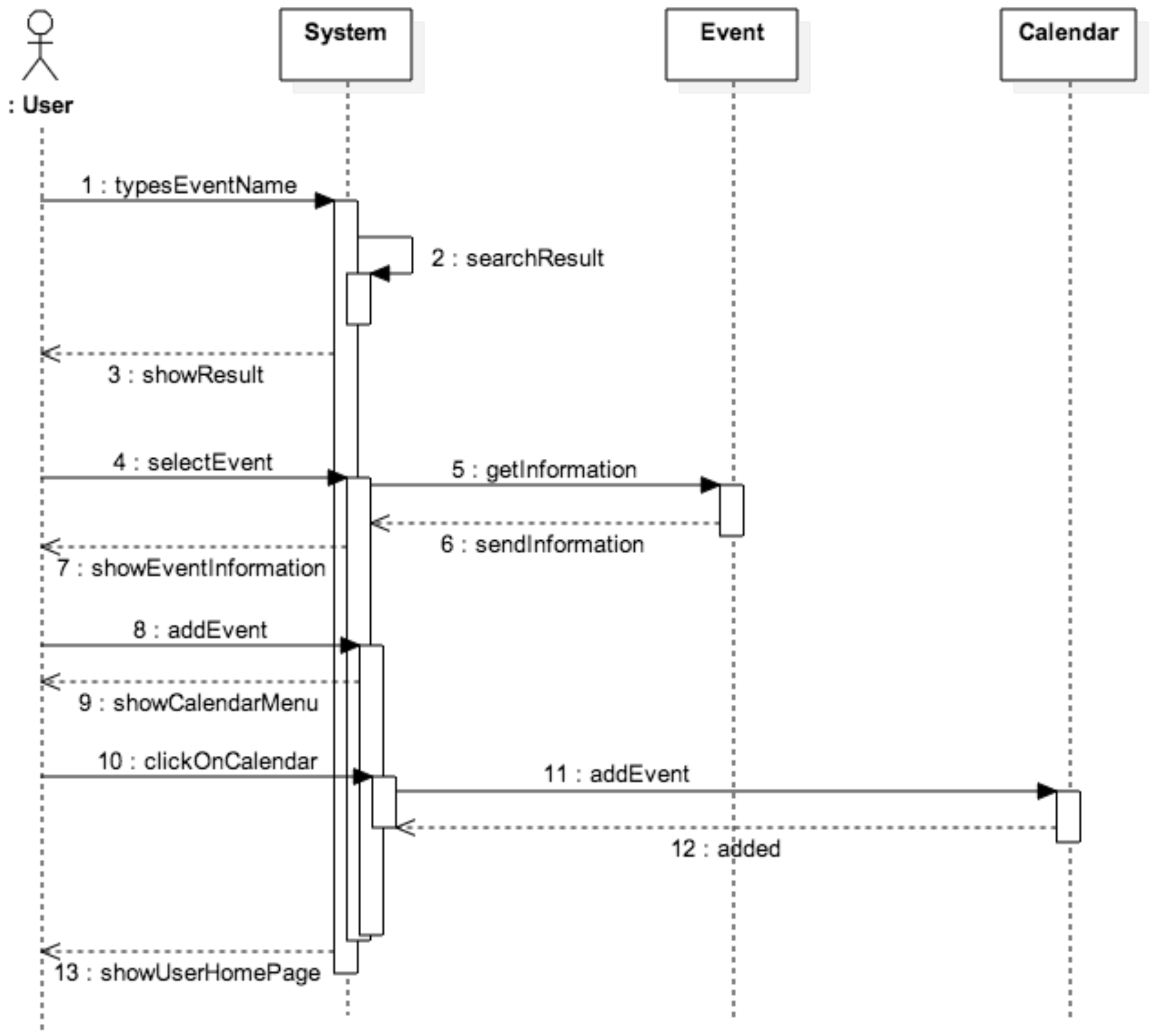
7.3.3 Create Calendar



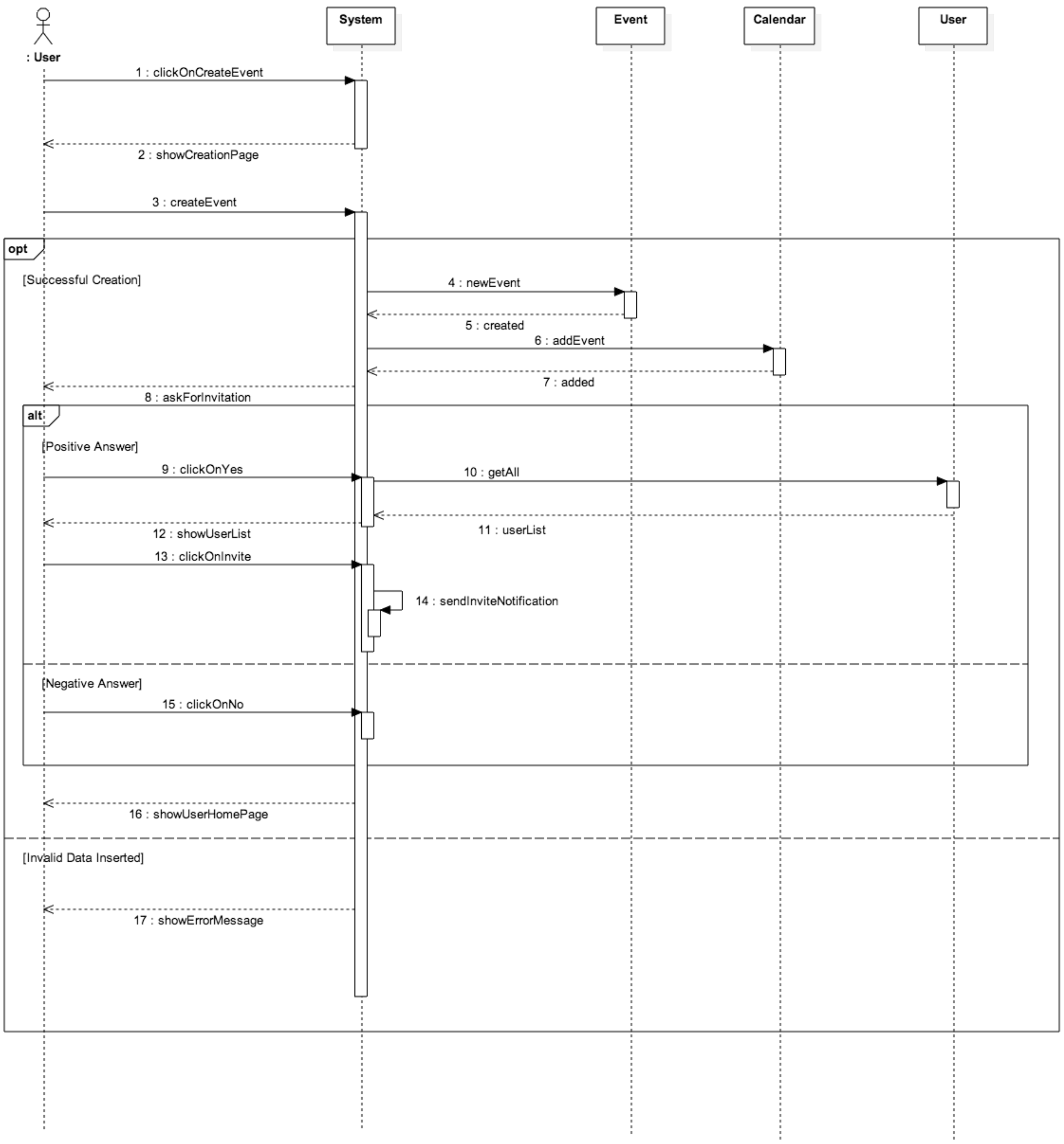
7.3.4 Select The Visible Calendars



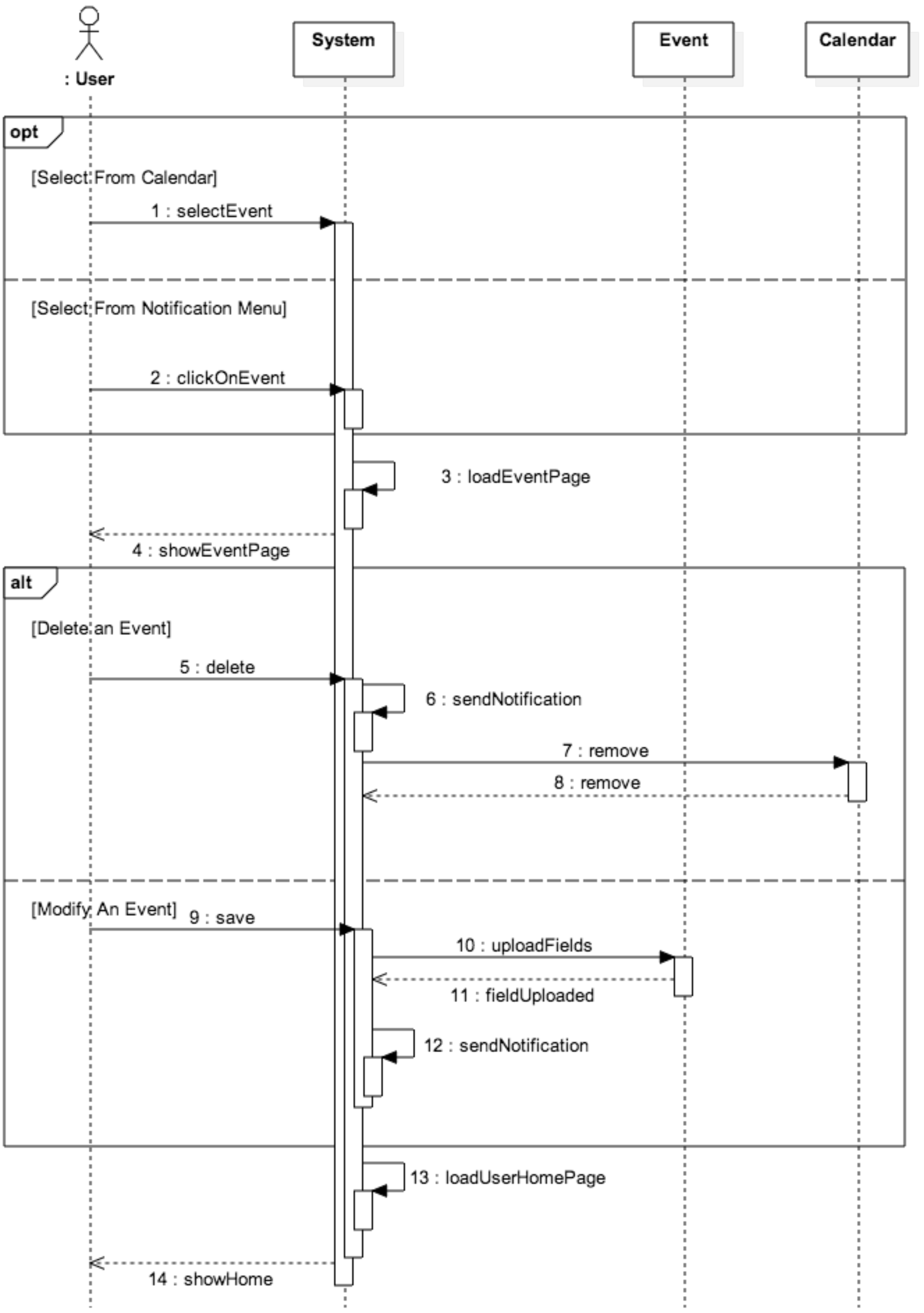
7.3.5 Search And Add Public Event



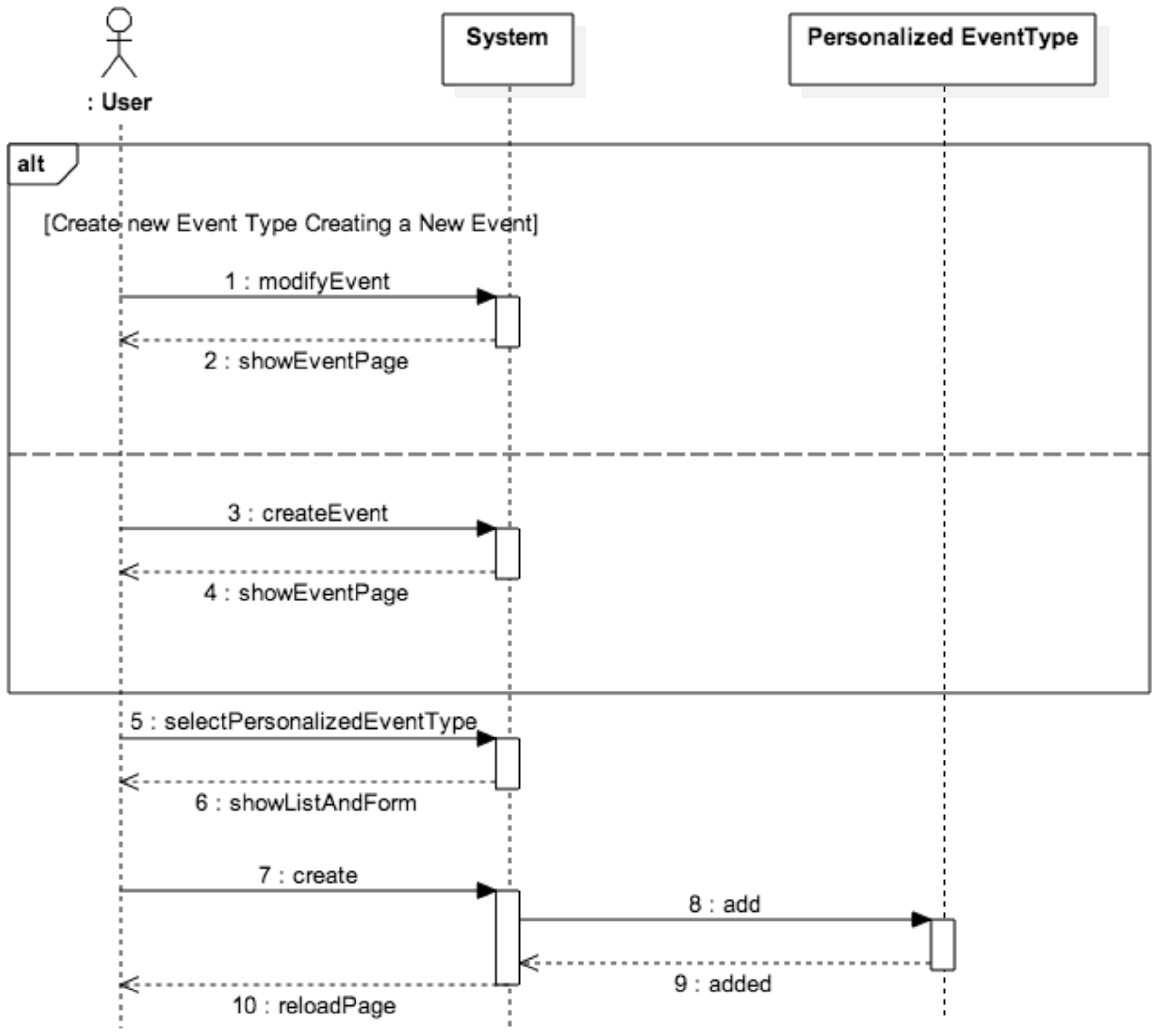
7.3.6 Create Event



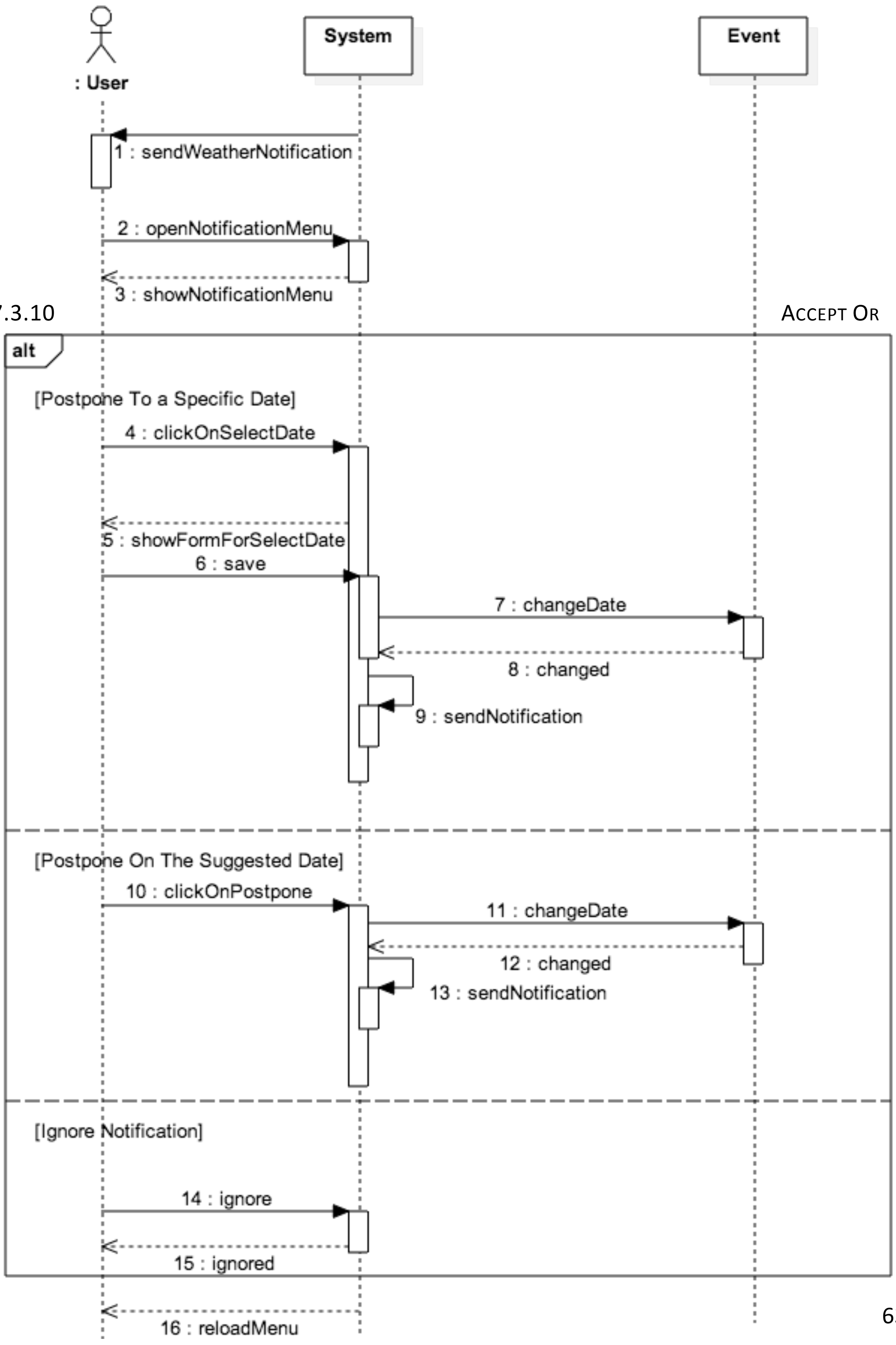
7.3.7 Change Event Information



7.3.8 Create Event Type

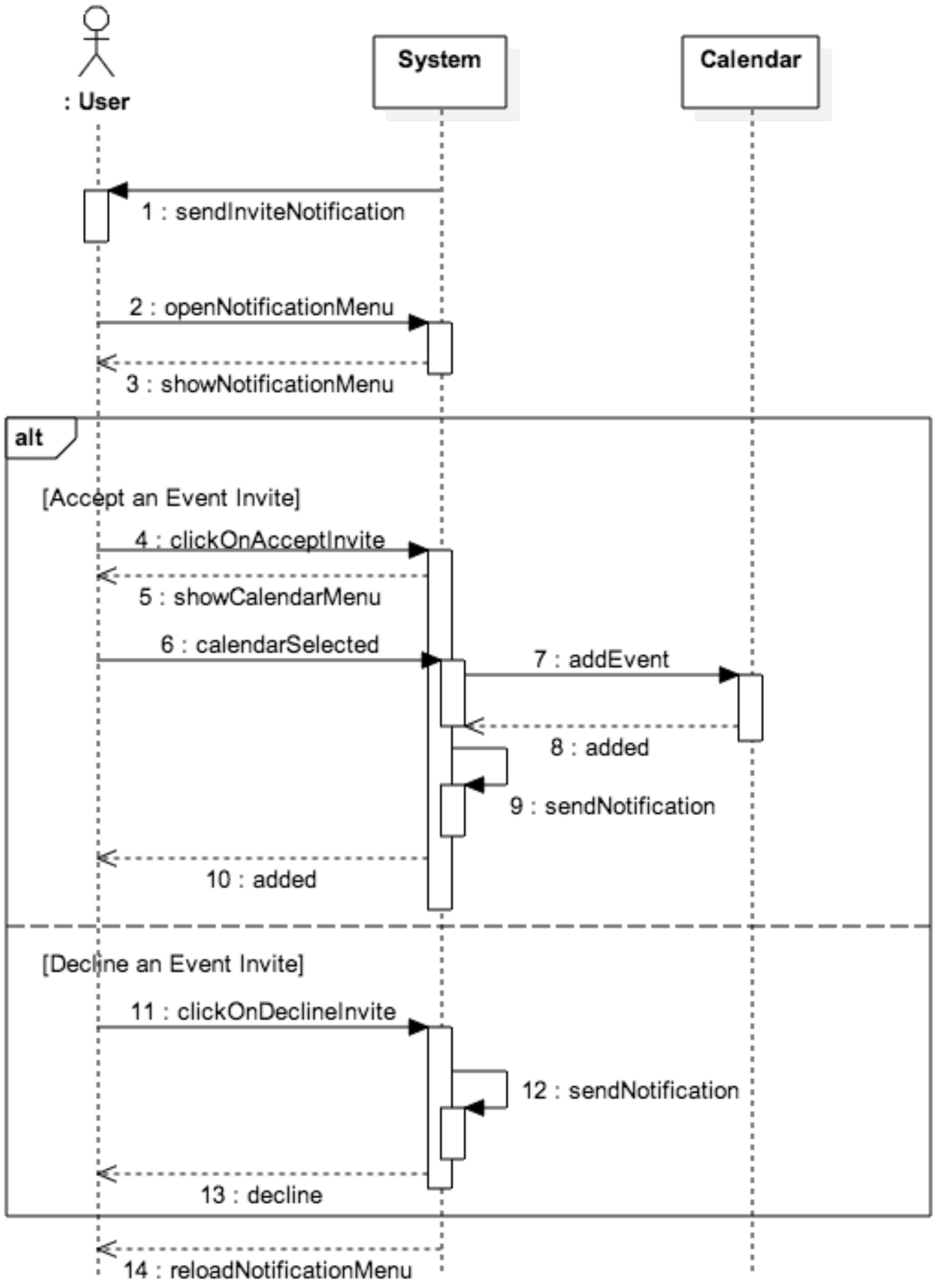


7.3.9 Postpone Event Because Of Bad Weather Notification

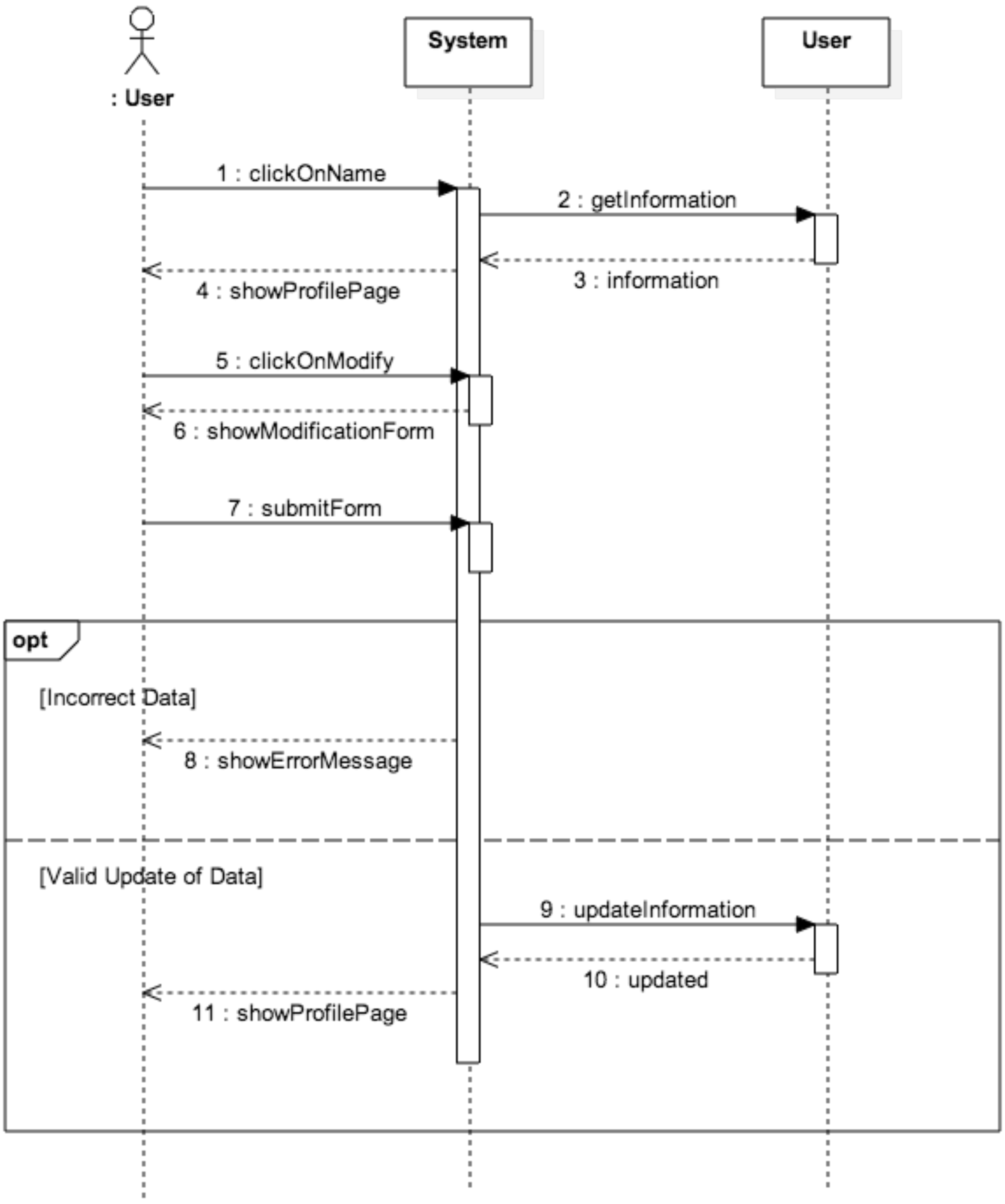


7.3.10 Accept Or

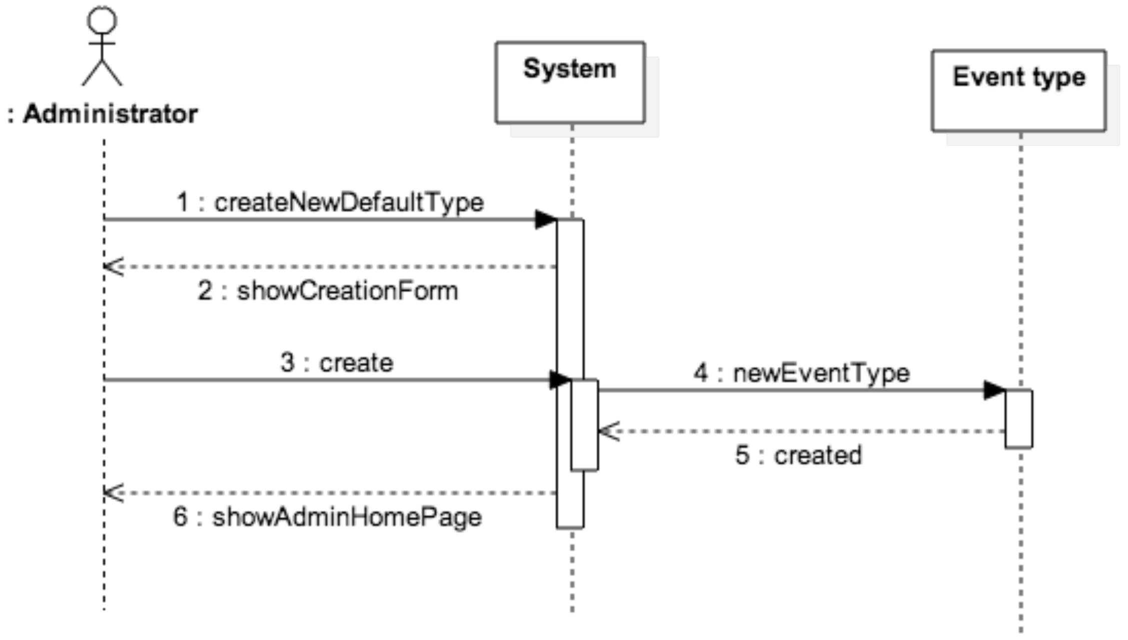
Decline Event Invite



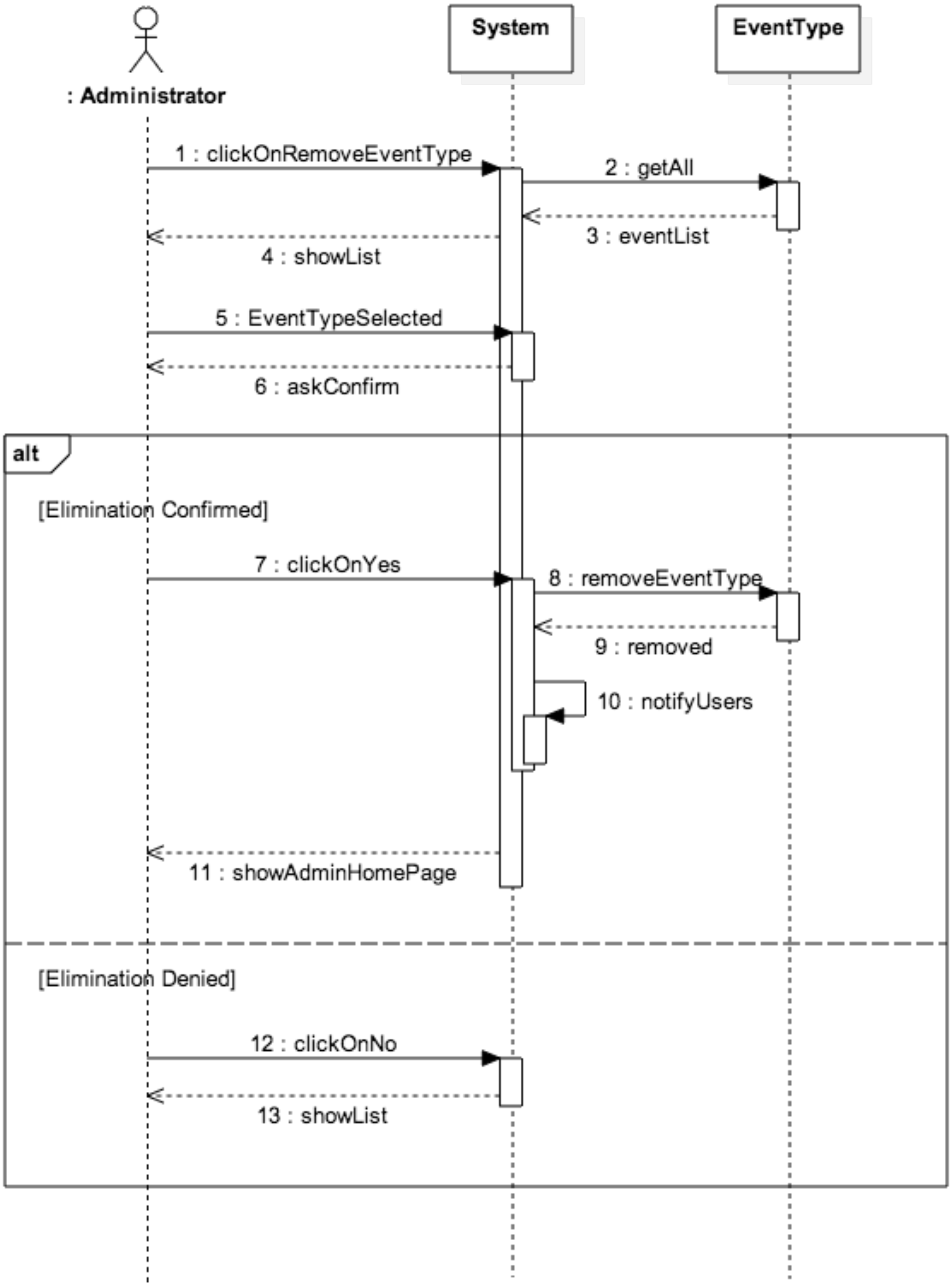
7.3.11 View And Modify Profile Information



7.3.12 Create Default Event Type

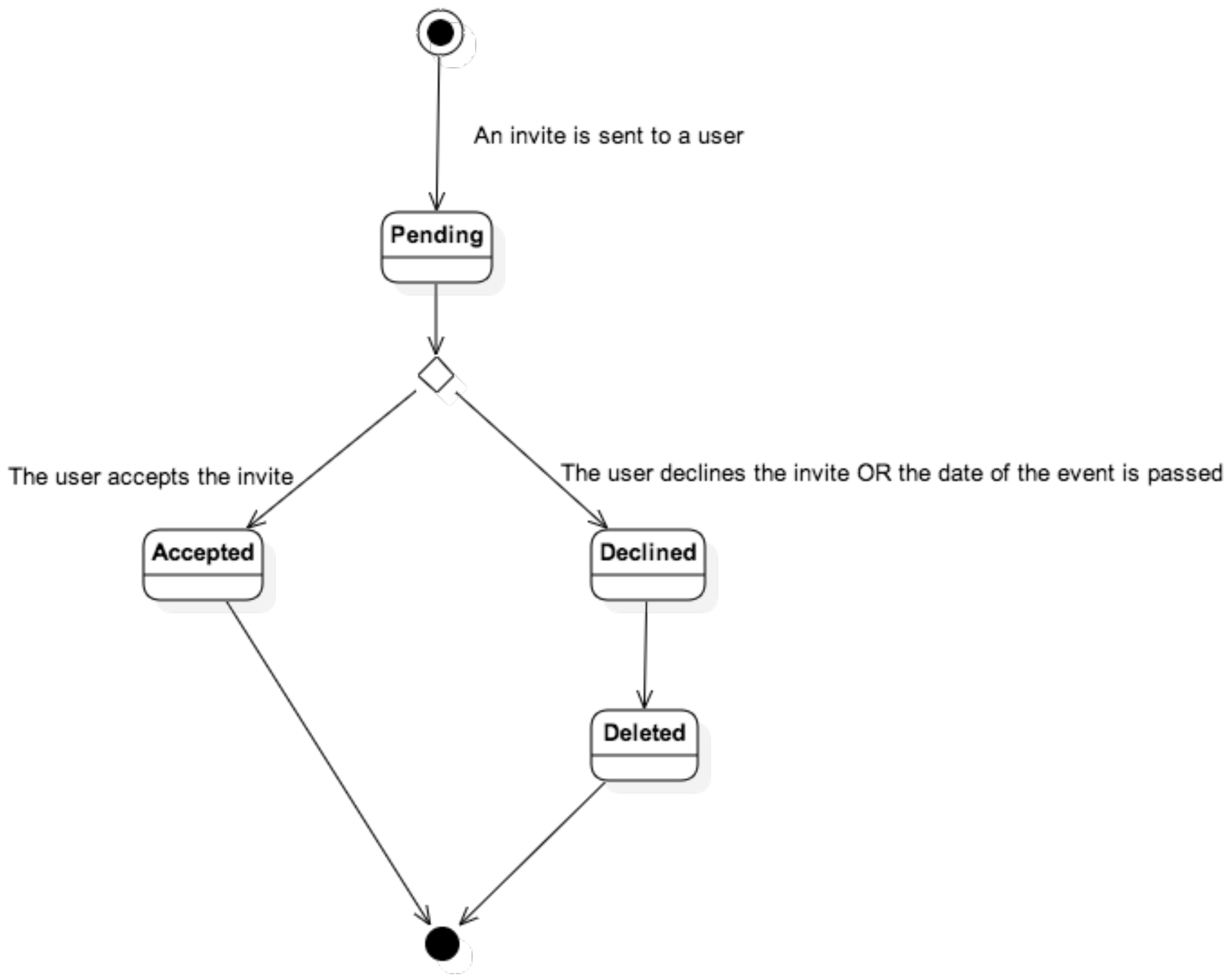


7.3.13 Delete Default Event Type



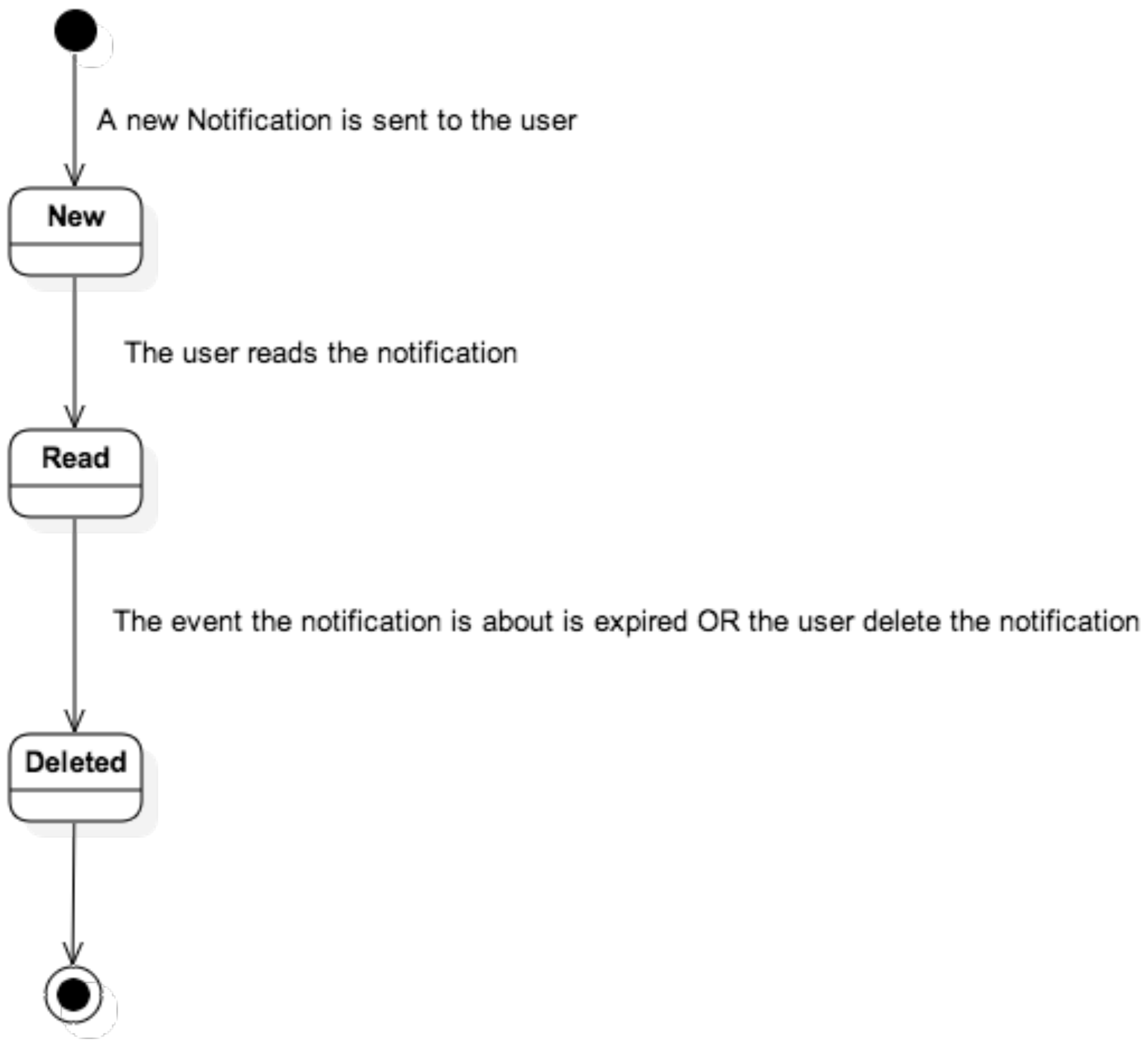
7.4 State Charts

7.4.1 Invite State Chart



An invite is sent to a user
Pending
The user accepts the invite
The user declines the invite OR the date of the event is passed
Accepted
Declined
Deleted

7.4.2 Notification State Chart

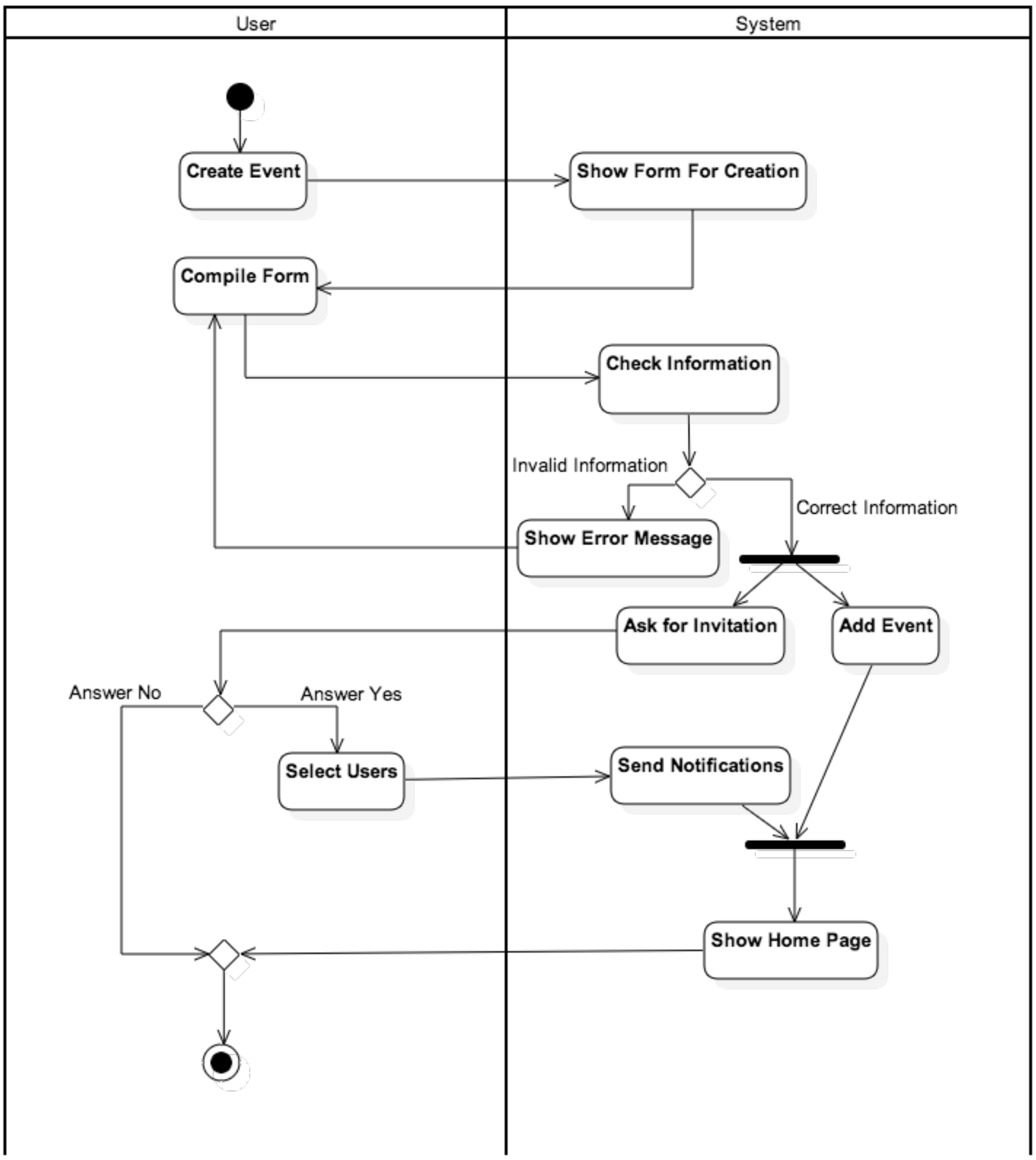

A new Notification is sent to the user
New
The user reads the notification
Read
The event the notification is about is expired OR the user delete the notification
Deleted

7.5 Activity Diagrams

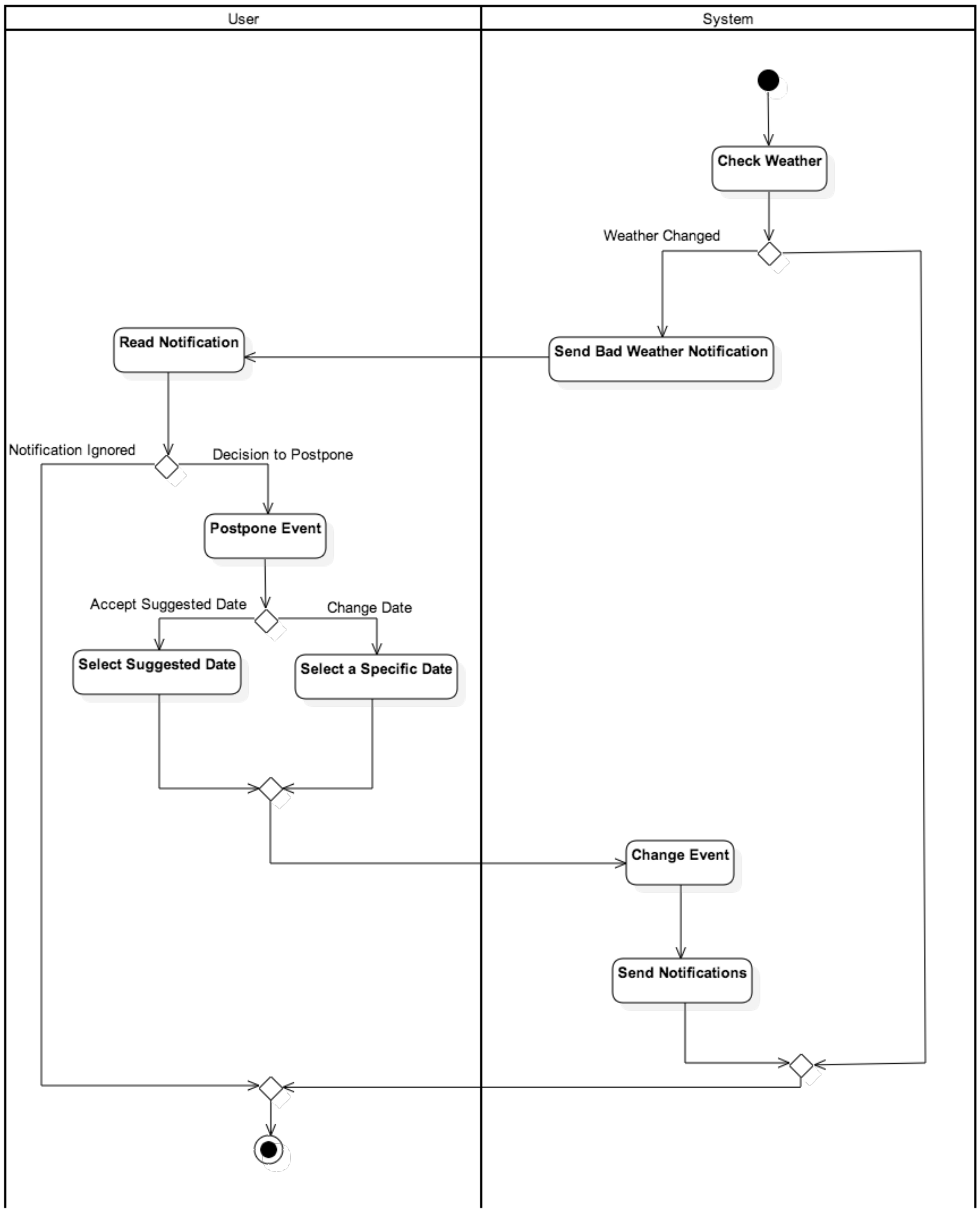
In this section we will realize two activity diagrams that represent two of the main activities:

- Event creation
- Bad weather notification received

7.5.1 Event Creation

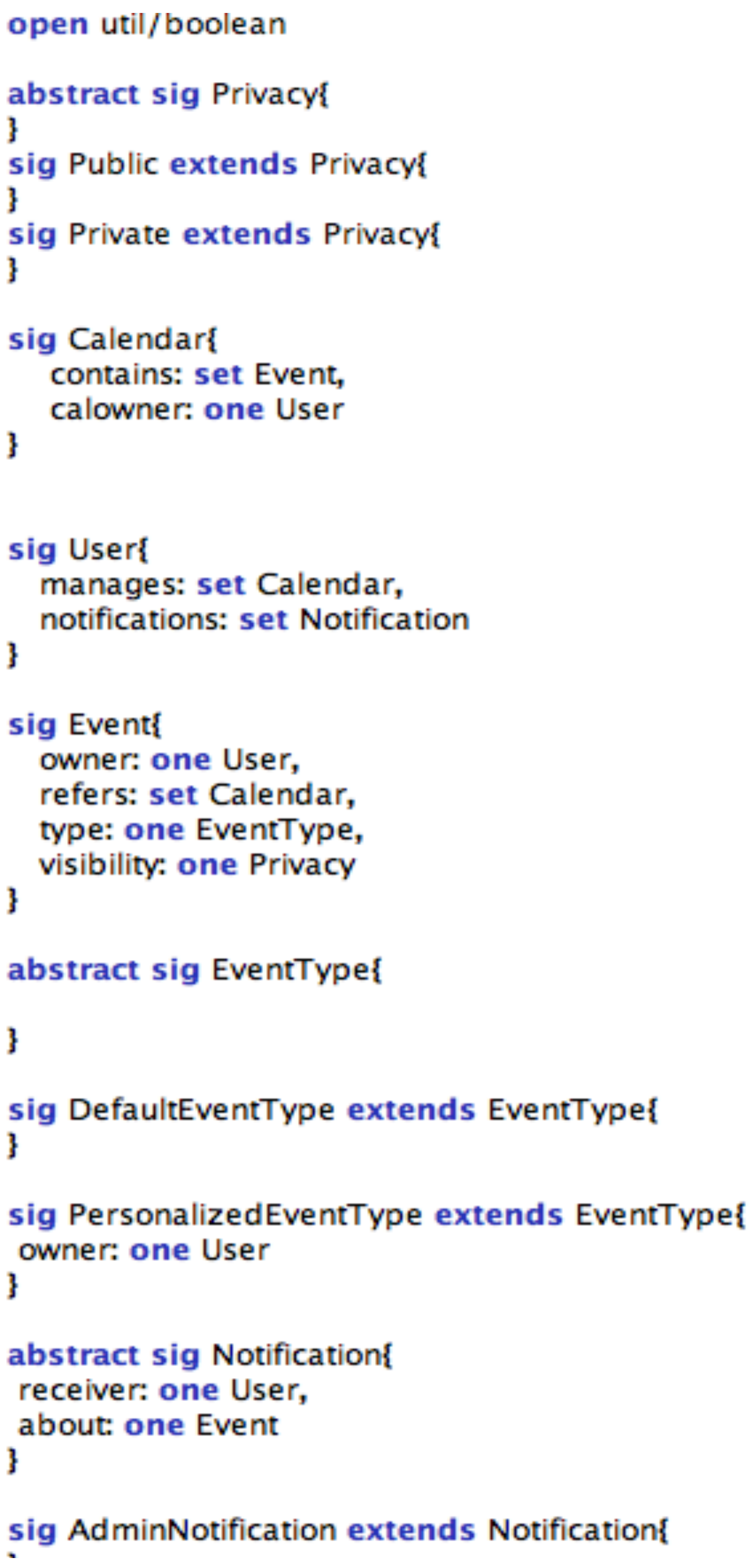


7.5.2 Bad Weather Notification Received



8 ALLOY MODEL

In this chapter we will analyze our model with the Alloy Analyzer to be sure that it is consistent. Here follows the code used to describe our model in Alloy:



```
open util/boolean

abstract sig Privacy{
}
sig Public extends Privacy{
}
sig Private extends Privacy{
}

sig Calendar{
    contains: set Event,
    calowner: one User
}


sig User{
   manages: set Calendar,
   notifications: set Notification
}

sig Event{
   owner: one User,
   refers: set Calendar,
   type: one EventType,
   visibility: one Privacy
}

abstract sig EventType{

}

sig DefaultEventType extends EventType{
}

sig PersonalizedEventType extends EventType{
 owner: one User
}

abstract sig Notification{
 receiver: one User,
 about: one Event
}

sig AdminNotification extends Notification{
```

```
sig InviteNotification extends Notification{
 sender: one User
}

sig ChangedEventNotification extends Notification{


}

sig ResponseNotification extends Notification{
 answer: Bool,
 sender: one User,
 refer: one InviteNotification
}

sig WeatherNotification extends Notification{
}


fact CalendarUserConsistency{
    // if a user manages a calendar then he's its owner
    all c: Calendar,u:User| u =c.calowner implies c in u.manages
    // maintain the relationship between the variables manages and owner
    all u: User, c: Calendar| (c in u.manages) implies c.calowner=u
}

fact calendarEventConsistency{
   //  if an event refers to a calendar then the calendar contains it
   all e:Event,c:Calendar | c in e.refers implies e in c.contains
    // and viceversa
   all e:Event,c:Calendar| (e in c.contains) implies (c in e.refers)
}

fact eventUserConsistency{
   // if a user owns an event he must also have it in one of his calendars
   all e:Event, u : User| e.owner=u implies (e in u.manages.contains)
}

fact notificationsConsistency{
   // if a notification is sent to a user it must be in his notifications
   all n:Notification, u:User { (n in u.notifications) iff (n.receiver=u ) }
}

fact CantInviteYourself{
    //The receiver of an invite can't be the sender of it
    all n: InviteNotification| not(n.receiver=n.sender)
}
```

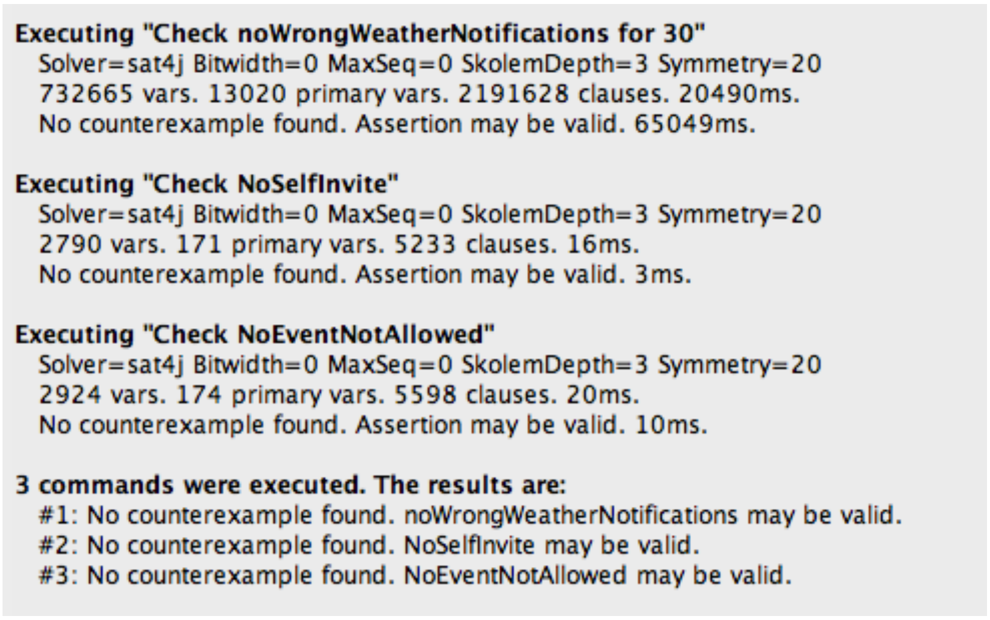
```
  // If an Event has a personalized type, that type must have been created by the Event's owner
  all e: Event, p: PersonalizedEventType | e.type=p implies p.owner=e.owner
}

fact invitationRights{

  // you can only invite people to your events
  all i:InviteNotification | i.sender=i.about.owner

  no i,i2:InviteNotification | i.sender=i2.sender && i.receiver=i2.receiver && i.about=i2.about && i!=i2

}

fact responseToInvite {
   // the response must be sended to the inviter
   all r: ResponseNotification | r.sender=r.refer.receiver && r.receiver=r.refer.sender
   // the response must refer to the same event of the invite
   all r: ResponseNotification | r.about=r.refer.about

   // there is at most 1 response to an invitation
   all i: InviteNotification | lone r:ResponseNotification | r.refer=i
}

fact acceptedInvite {
  // if you accept an invite you must attend the event
  all r:ResponseNotification | (r.answer=True) implies (r. about in r.sender.manages.contains )
  //if you decline an invite you must not attend the event
 all r:ResponseNotification | (r.answer=False) implies not (r. about in r.sender.manages.contains )
}

fact participationRights{
   // You can join an event only if it is public, if you are its owner or if you have been invited and responded positively
   all u:User, e:Event { (e in u.manages.contains) implies (( e.visibility= Public )|| (e.owner=u ) || (one n:ResponseNotification { n.sender=u&& n.about=e&&n.answer=True }
}

fact AllParticipantsWeatherNotification{

   // the weather notification is sent only to the event participant
   all w:WeatherNotification| w.receiver in w.about.refers.calowner

    // The weather notification is sent to all the event participant
    all w:WeatherNotification{ all u:User{ (u in w.about.refers.calowner) implies  (some w2:WeatherNotification{ w2.receiver=u && w2.about=w.about } ) } }
}

fact AllParticipantsChangedNotification{

   // the weather notification is sent only to the event participant
   all w:ChangedEventNotification| w.receiver in w.about.refers.calowner

    // The weather notification is sent to all the event participant
    all w:ChangedEventNotification{ all u:User{ (u in w.about.refers.calowner) implies  (some w2:ChangedEventNotification{ w2.receiver=u && w2.about=w.about } ) } }
}
```

8.1 Assertions

Here are reported some Assertions we made to verify that our model respects its constraints.

```
//ASSERTIONS

assert noWrongWeatherNotifications{

   // You can't receive a notification about an event you're not attending
   no u:User,w: WeatherNotification { (u not in w.about.refers.calowner ) && (w.receiver =u )  }
   no u:User,w: ChangedEventNotification { (u not in w.about.refers.calowner ) && (w.receiver =u )  }

}

check noWrongWeatherNotifications for 30

assert NoSelfInvite{
  //You can't send an invite to yourself
  no i:InviteNotification{ i.sender=i.receiver  }
}

check NoSelfInvite

assert NoEventNotAllowed{
  // You can only attend another user's private event if you responded positively to an invite for it from him
  all u:User,e:Event{ (e in u.manages.contains && e.owner!=u&& e.visibility=Private) implies
 ( one r:ResponseNotification{ r.sender=u && r.about=e && r.receiver=e.owner&& r.answer=True})  }
}

check NoEventNotAllowed
```

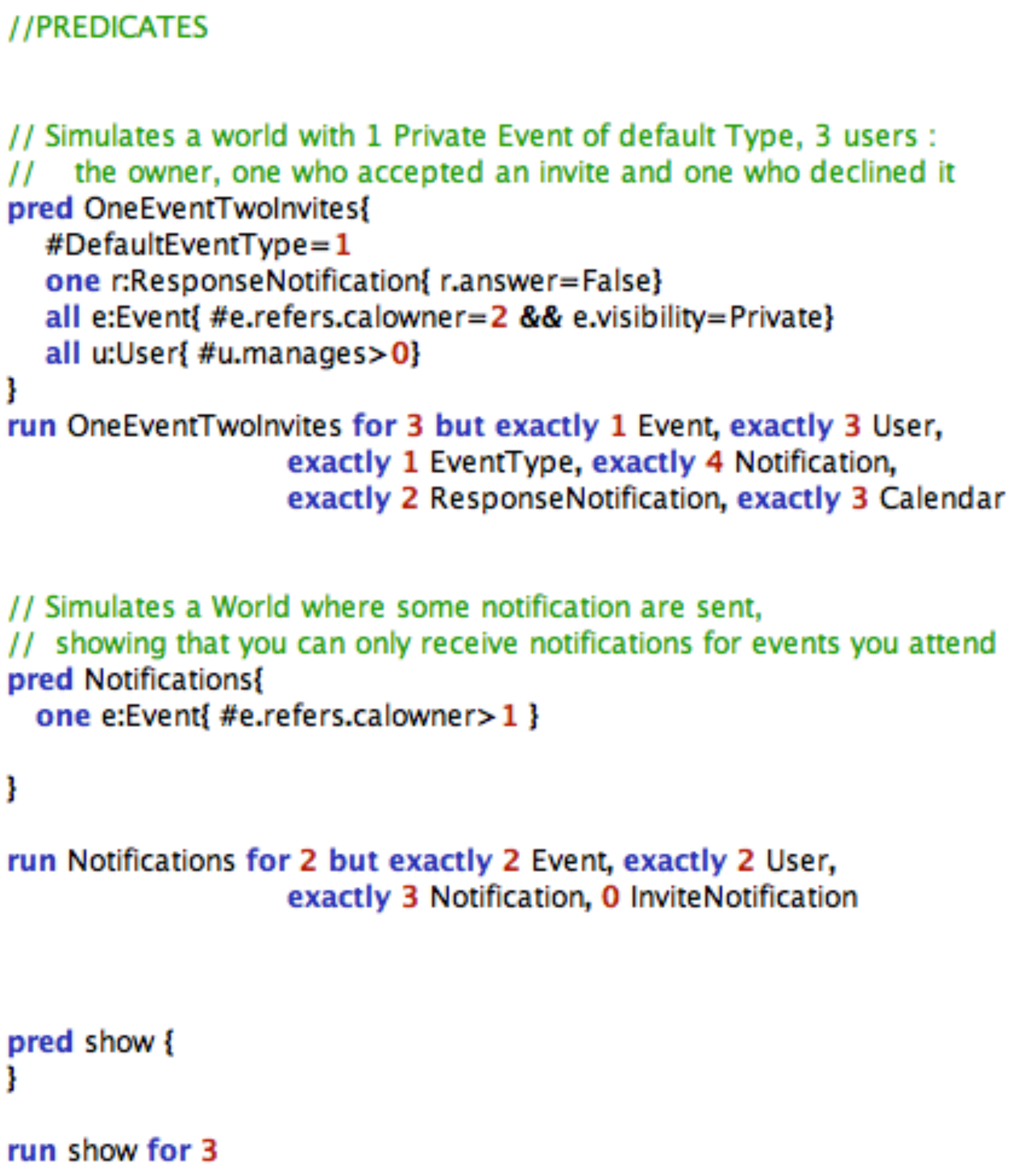
The Analyzer as shown verified all of these Assertions:



```
Executing "Check noWrongWeatherNotifications for 30"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   732665 vars. 13020 primary vars. 2191628 clauses. 20490ms.
   No counterexample found. Assertion may be valid. 65049ms.

Executing "Check NoSelfInvite"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   2790 vars. 171 primary vars. 5233 clauses. 16ms.
   No counterexample found. Assertion may be valid. 3ms.

Executing "Check NoEventNotAllowed"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   2924 vars. 174 primary vars. 5598 clauses. 20ms.
   No counterexample found. Assertion may be valid. 10ms.

3 commands were executed. The results are:
   #1: No counterexample found. noWrongWeatherNotifications may be valid.
   #2: No counterexample found. NoSelfInvite may be valid.
   #3: No counterexample found. NoEventNotAllowed may be valid.
```

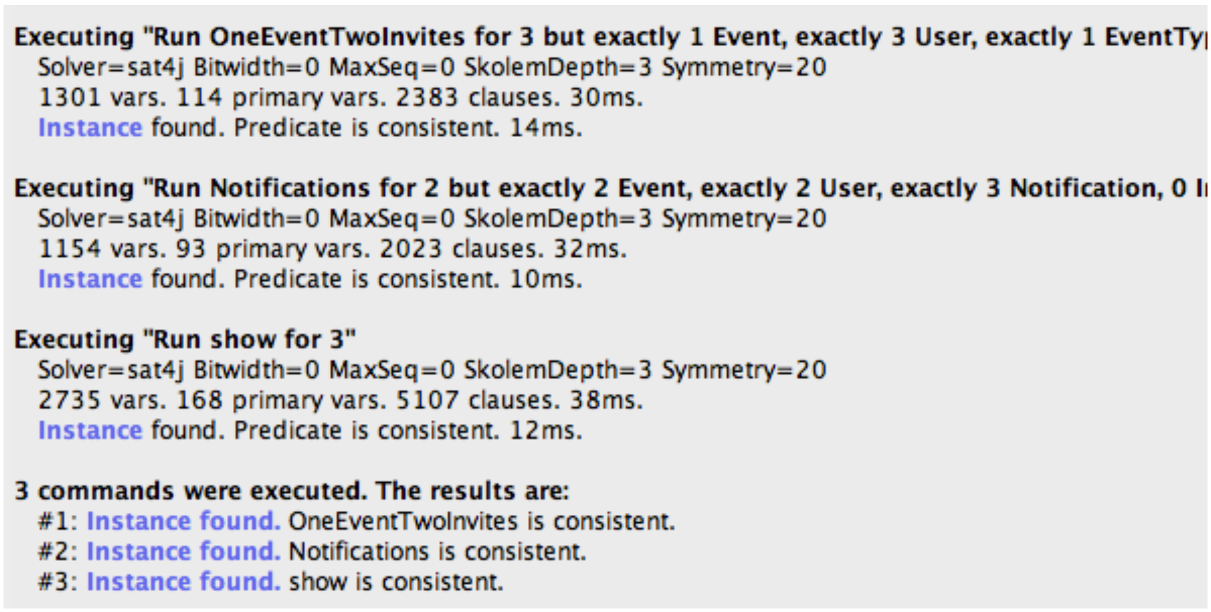
8.2 PREDICATES AND WORLDS GENERATED

Here are stated the few predicates we tried to run in the Analyzer to verify that our model is consistent.



```
//PREDICATES

// Simulates a world with 1 Private Event of default Type, 3 users :
//    the owner, one who accepted an invite and one who declined it
pred OneEventTwoInvites{
   #DefaultEventType=1
   one r:ResponseNotification{ r.answer=False}
   all e:Event{ #e.refers.calowner=2 && e.visibility=Private}
   all u:User{ #u.manages>0}
}
run OneEventTwoInvites for 3 but exactly 1 Event, exactly 3 User,
                        exactly 1 EventType, exactly 4 Notification,
                        exactly 2 ResponseNotification, exactly 3 Calendar


// Simulates a World where some notification are sent,
//  showing that you can only receive notifications for events you attend
pred Notifications{
  one e:Event{ #e.refers.calowner>1 }

}

run Notifications for 2 but exactly 2 Event, exactly 2 User,
                        exactly 3 Notification, 0 InviteNotification



pred show {
}

run show for 3
```

And these are the results:

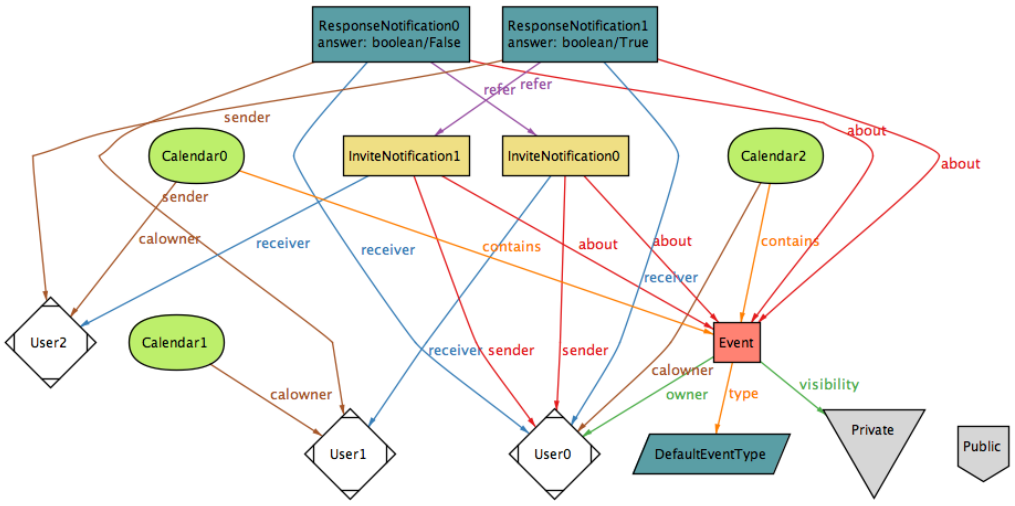


```
Executing "Run OneEventTwoInvites for 3 but exactly 1 Event, exactly 3 User, exactly 1 EventTy
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   1301 vars. 114 primary vars. 2383 clauses. 30ms.
   Instance found. Predicate is consistent. 14ms.

Executing "Run Notifications for 2 but exactly 2 Event, exactly 2 User, exactly 3 Notification, 0 I
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   1154 vars. 93 primary vars. 2023 clauses. 32ms.
   Instance found. Predicate is consistent. 10ms.

Executing "Run show for 3"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=3 Symmetry=20
   2735 vars. 168 primary vars. 5107 clauses. 38ms.
   Instance found. Predicate is consistent. 12ms.

3 commands were executed. The results are:
   #1: Instance found. OneEventTwoInvites is consistent.
   #2: Instance found. Notifications is consistent.
   #3: Instance found. show is consistent.
```

Now the world generated by this predicate will be shown, in these images some of the arrows that indicates the associations have been hidden to make them more understandable.

8.2.1 World Generated By OneEventTwoInvites

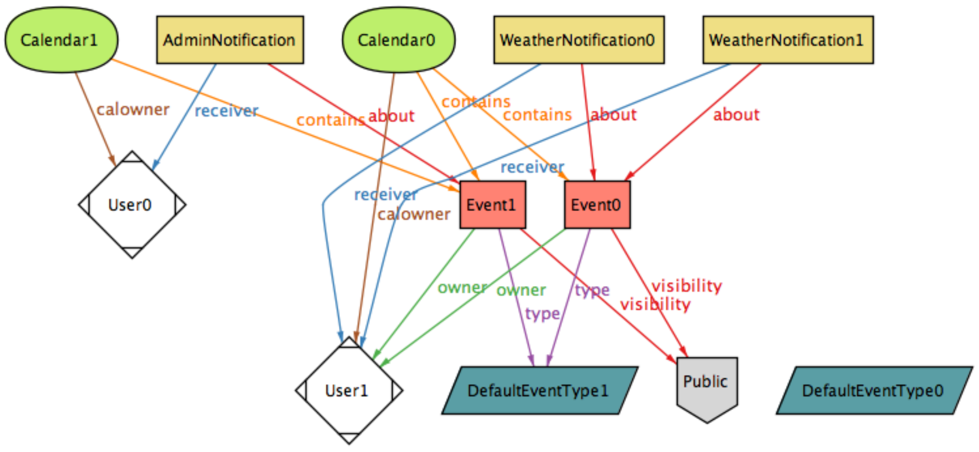
This is the Result of the simulation ran on the predicate OneEventTwoInvites:



Here is shown how a user can be invited to a private event and how only the owner and the user who accepted the invite can add the event to their calendar.

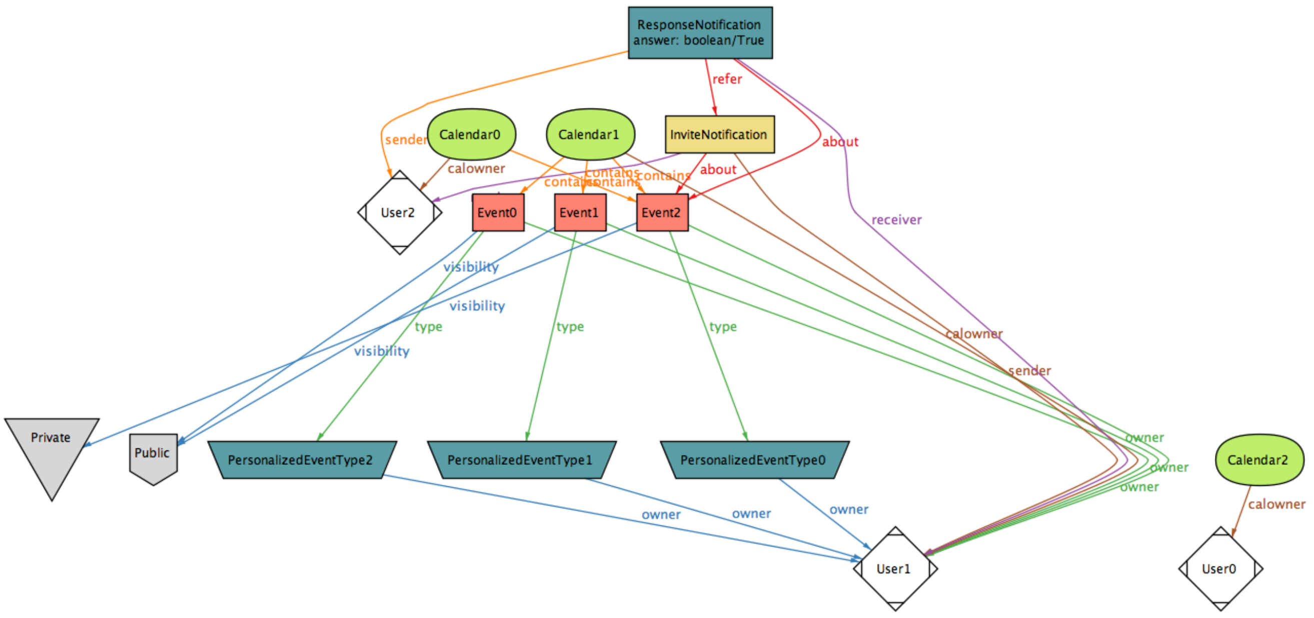
8.2.2 World Generated by Notifications

In this world we can see how a user can receive only notifications regarding events he is attending:

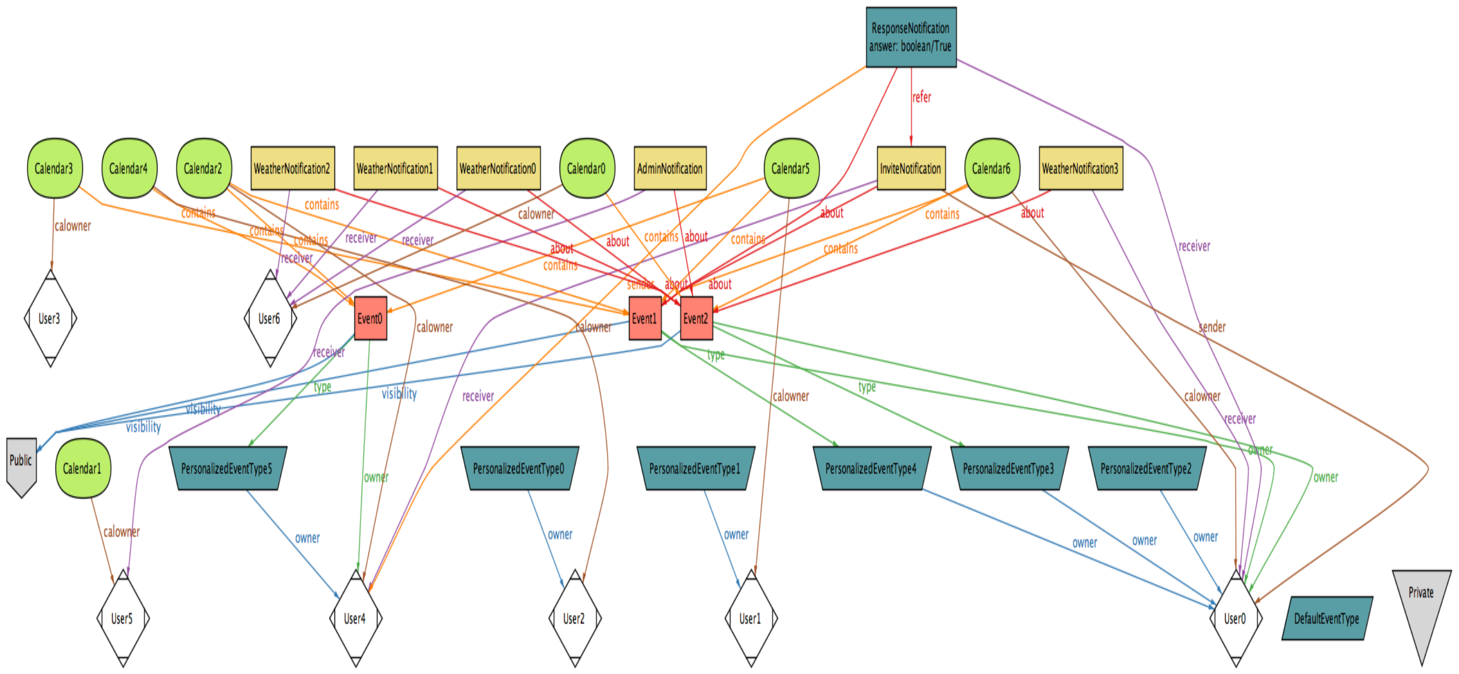


8.2.3 World Simulation

And at last this is a small simulation of a possible complete world, the number of instances simulated is quite small because the whole world grows exponentially with respect to them:



Just for completeness the last page is a world generated with a maximum of only 7 instances per object.


ResponseNotification
answer: boolean/True
refer
Calendar3
Calendar4
Calendar2
WeatherNotification2
WeatherNotification1
WeatherNotification0
Calendar0
AdminNotification
Calendar5
InviteNotification
Calendar6
WeatherNotification3
User3
User6
Event0
Event1
Event2
Public
Calendar1
PersonalizedEventType5
PersonalizedEventType0
PersonalizedEventType1
PersonalizedEventType4
PersonalizedEventType3
PersonalizedEventType2
User5
User4
User2
User1
User0
DefaultEventType
Private
calowner
contains
receiver
about
sender
visibility
type
owner

9 USED TOOLS

This is the list of the tools that we have exploited to realize this document:

- Microsoft Office Word 2011: to redact this document
- StarUML: to create use cases diagrams, class diagram, sequence diagrams, activity diagrams and state chart diagrams.
- Alloy Analyzer 4.2: to create alloy model in order to prove the consistency of our world
- GoMockingBird platform: to realize UI mockups.