MeteoCal
Design Document

Simone Graziussi

7th December 2014

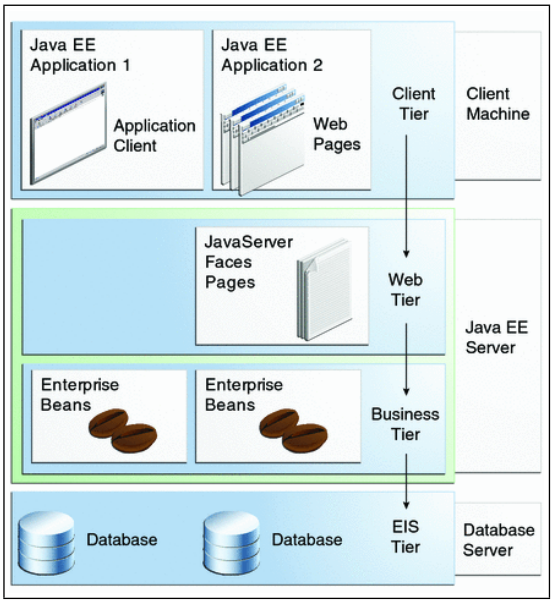
Contents

Part I
Architecture

1 Java Enterprise Edition

MeteoCal software will be based on the Java Enterprise Edition architecture. The platform is divided into:

- Client-tier components
- Web-tier components
- Business-tier components
- Enterprise Information System (EIS)-tier software



In *MeteoCal* client tiers will be web clients, and as such they consist of two parts:

- Dynamic web pages containing various types of markup language (mainly HTML), which are generated by web components running in the web tier
- A web browser, which renders the pages received from the server

The system will also include components based on the JavaBeans component architecture to manage the data flow between:

- the application client and components running on the Java EE server
- server components and the database

Business code, which is the logic that will manage all *MeteoCal* functions (for example saving and retrieving events, managing notifications, etc.), will be handled by enterprise beans running in either the business tier or the web tier.

The EIS-tier will take care of storing the persistent data in a MySQL database.

Part II
Persistent Data

2 Conceptual Design

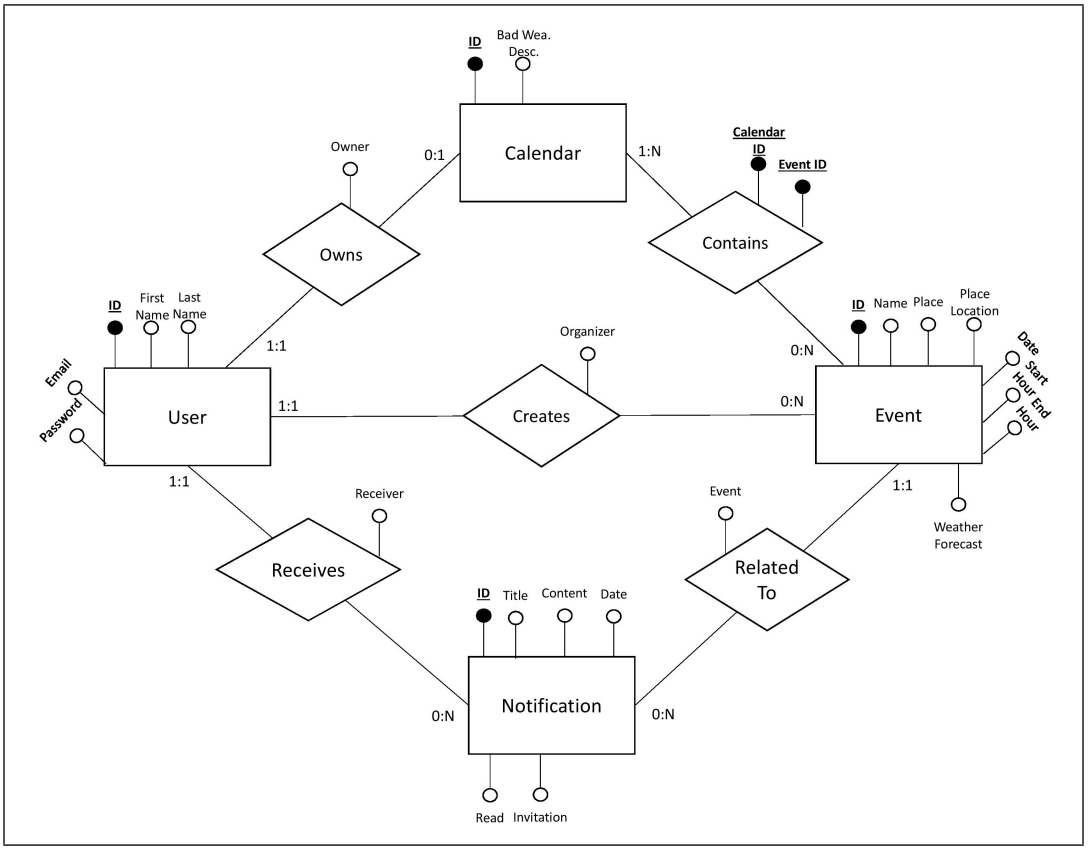
MeteoCal system needs to store in the database information about users, calendars, events and notifications. In this section the general structure of the database will be explained.

The User data is characterized mainly by first and last name, an email address (which works as the log-in user-name) and a password.

The Calendar table contains only some information about the user's bad weather concept and is linked to the User table by the "owner" attribute. With the assumption of one calendar per user, made in the RASD document, this table wasn't really needed but could be useful for possible future implementations of more than one calendar per user.

A N:N relationship links a calendar and its events. The Event data is composed by information about name, time, space and, possibly, by a weather forecast. This table is also linked to the User one by the "organizer" attribute.

The notifications are stored in the Notification table, containing information about title, content, date and a boolean value telling if the notification is an invitation. This table is linked to Event and User by the "event" and "receiver" attributes.

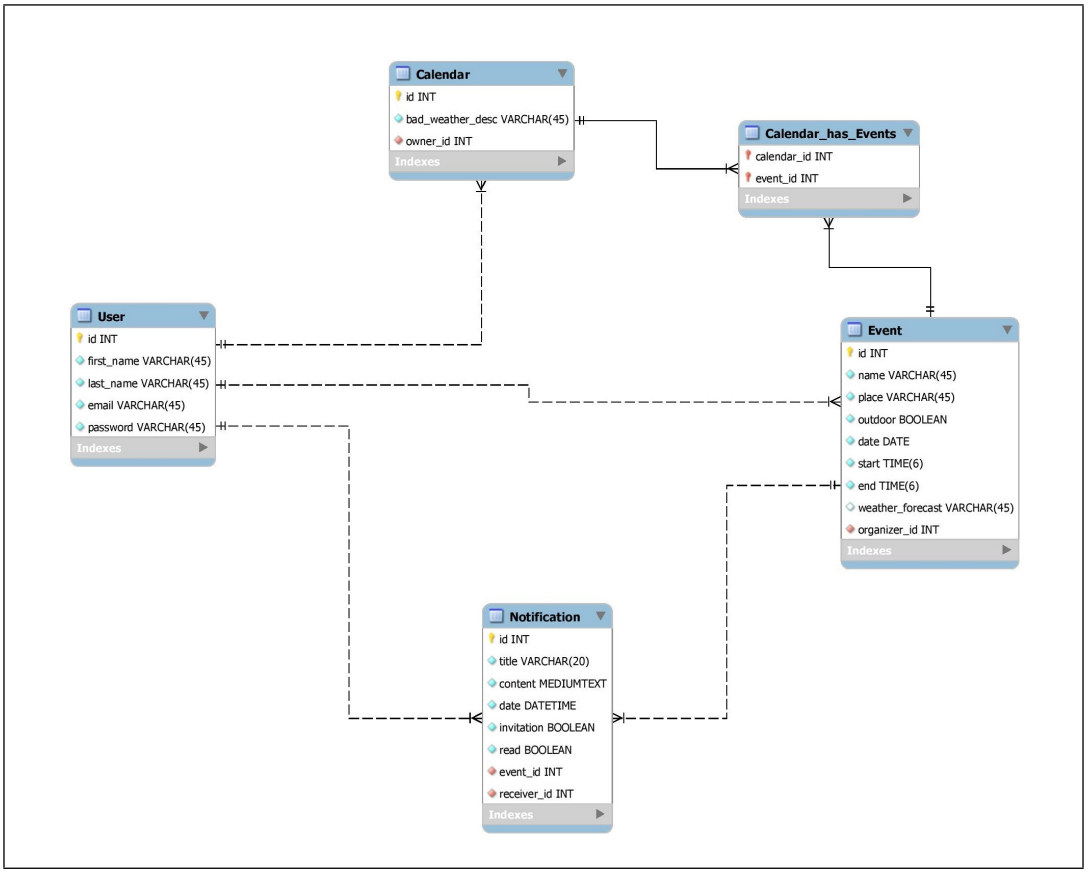


3 Logical Design

The logical design of the database gives some more details about the structure of the stored data.

First of all, we can notice that all tables use IDs as primary keys to have an easy way to link concepts. For example, in the Calendar table the owner is saved as a user ID, as well as the receiver of a Notification.

All relationships between tables are 1:N, except one. For the former kind, it was decided to store the foreign key(s) directly in one of the two tables (for example, the Notification table has "receiver_id" and "event_id" as attributes), while for the latter, Calendar-Event, an external table is required, containing the event_id-calendar_id associations.



Part III
User Experience Diagrams

4 Introduction

User Experience Diagrams show the interactions between the user and *MeteoCal*, with little consideration of what the system actually does to provide this features.

These diagrams will be useful to understand the main actions the user will be able to perform and how many different screens the system will provide. As a matter of fact, the diagrams just represent the screens (with optional sub-screens called screen compartments) that the user will be able to navigate and the available actions for each of them. Associations with input forms describe the presence of fields the user will have to input to perform a specific action, while associations with entities mean that the user will be able to get or set information about that specific object.

The system functions have been split into three main blocks to provide a clearer view on the concepts.

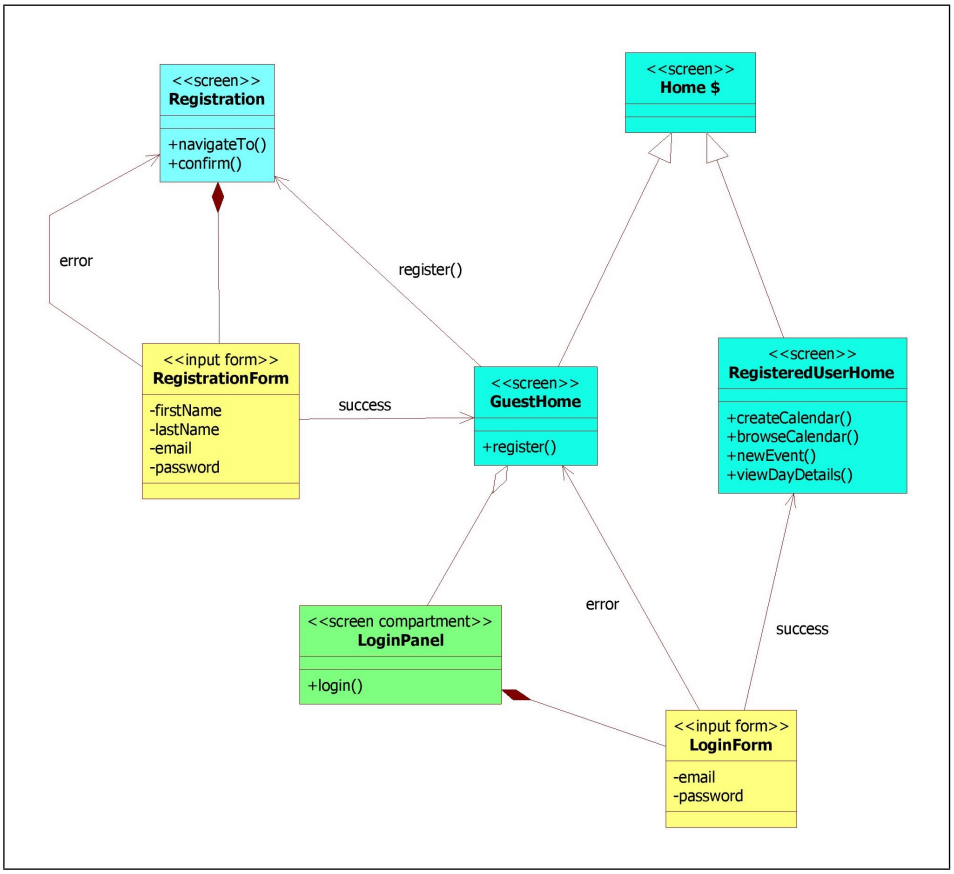
5 Registration and Log-in

In this section it will be shown how the user is able to start the system as a guest and then register or log into the system.

Two different types of Home Page are available: the Guest Home and the Registered User Home. The first one is shown at the system start-up and will offer two possible actions: register or log-in.

If the user needs to register, the system opens a new screen where the user will be able to input his/her personal information and then submit the form. If the registration is successful, the user will be redirected once again to the Guest Home, while he/she will remain in the Registration page in case of errors.

Directly from the Guest Home, the user will be able to log-in providing user-name and password and, if they are correct, he/she will be redirected to the Registered User Home, which will provide all *MeteoCal* functions.

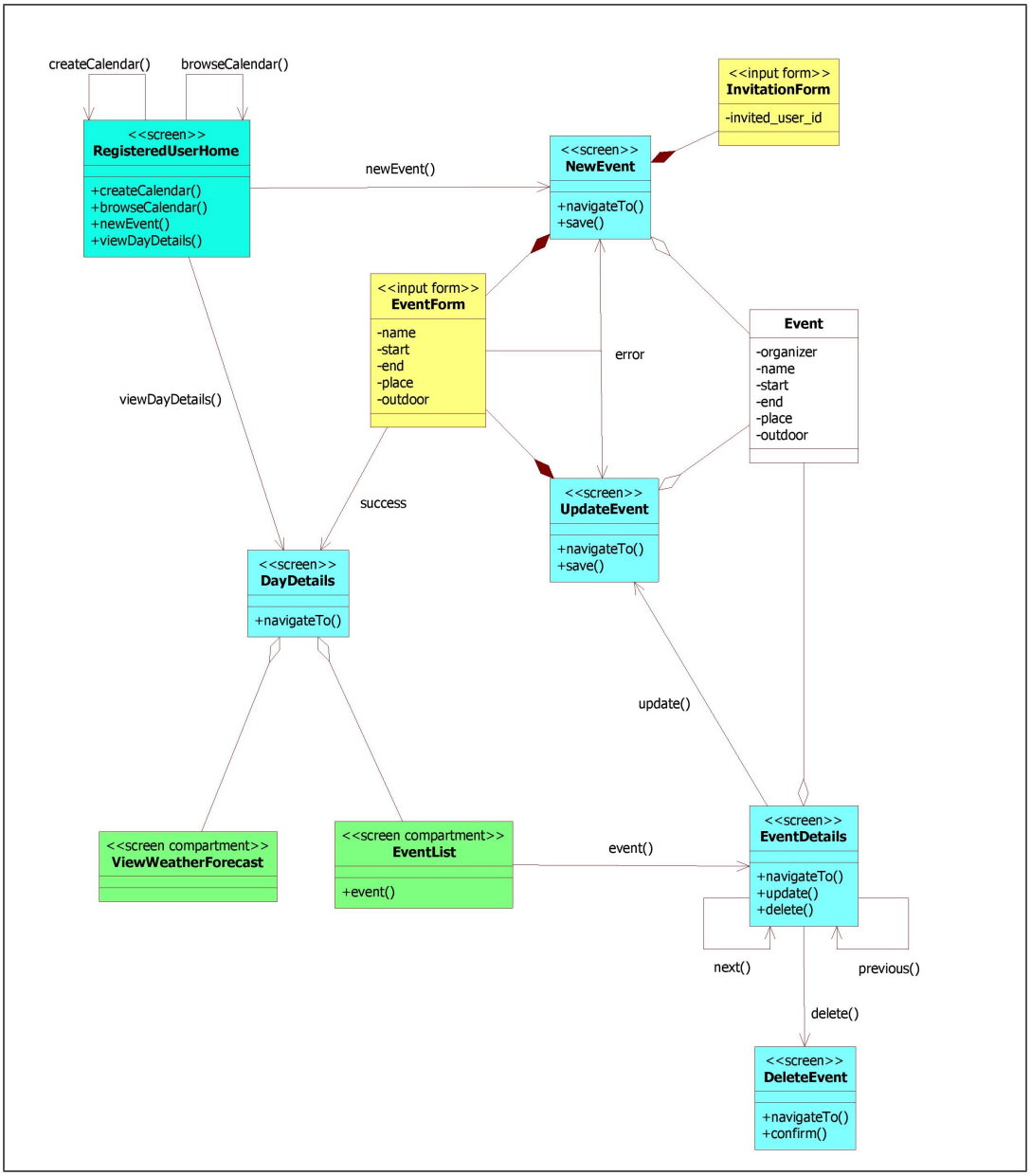


6 Calendar Browsing and Event Management

Starting from the Registered User Home screen in the previous section, the user will be able to create and browse his/her calendar.

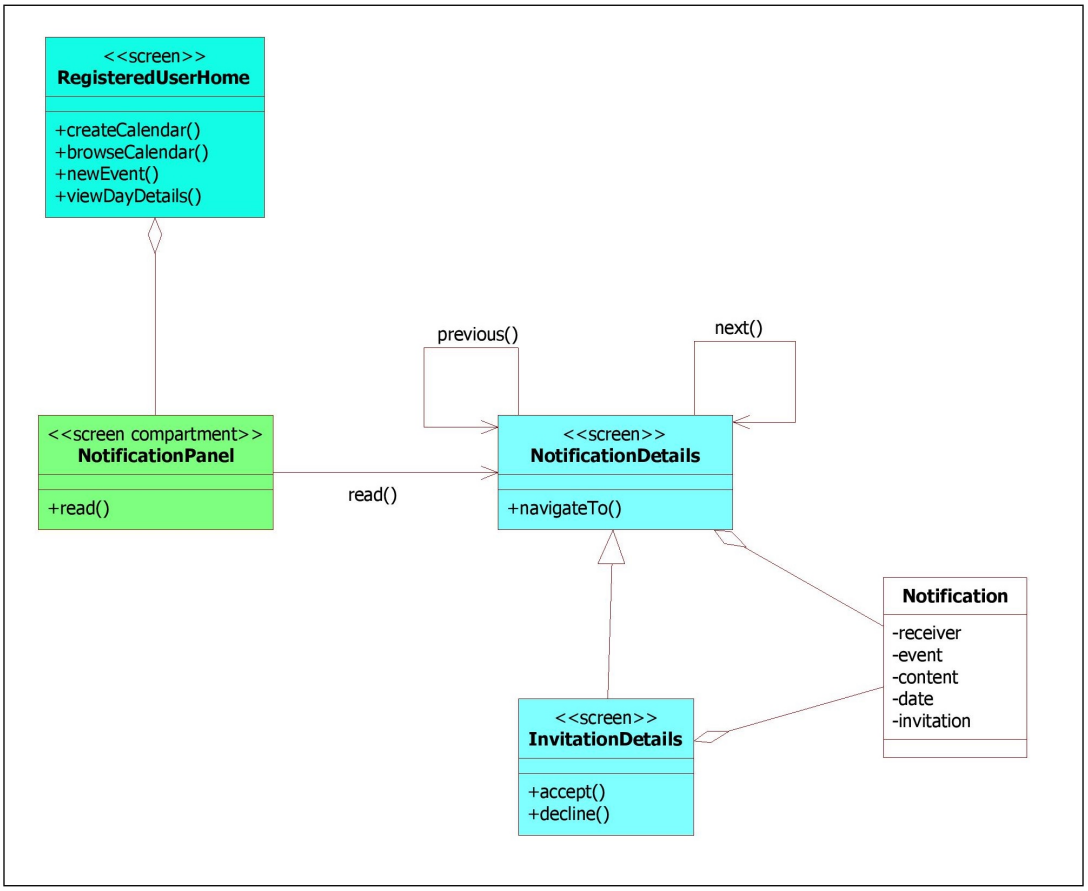
In particular, the most interesting actions related to the calendar itself are:

- **add an event**: the user will be redirected to a new screen where he/she will be able to provide all data about the event and, optionally, a list of invited users. If an error is detected, the user will stay on the same screen to correct them, while if all data is consistent he/she will be redirected to the Day Details screen.
- **view the details for a selected day**: from the Day Details screen, the user will be able to read the event list and the weather forecast, if available. From the event list the user will be able to click on one of them and reach the event details screen, where he/she will read all event data and, possibly, update or delete it.


createCalendar()
browseCalendar()
<<screen>>
RegisteredUserHome
+createCalendar()
+browseCalendar()
+newEvent()
+viewDayDetails()
newEvent()
<<input form>>
InvitationForm
-invited_user_id
<<screen>>
NewEvent
+navigateTo()
+save()
<<input form>>
EventForm
-name
-start
-end
-place
-outdoor
error
Event
-organizer
-name
-start
-end
-place
-outdoor
viewDayDetails()
success
<<screen>>
UpdateEvent
+navigateTo()
+save()
<<screen>>
DayDetails
+navigateTo()
update()
<<screen compartment>>
ViewWeatherForecast
<<screen compartment>>
EventList
+event()
event()
<<screen>>
EventDetails
+navigateTo()
+update()
+delete()
next()
previous()
delete()
<<screen>>
DeleteEvent
+navigateTo()
+confirm()

7 Notification Management

In the Registered User Home screen the user will be able to read the list of unread notifications. By clicking on one of them, he/she will reach the Notification Details screen and read its content. If the notification is an invitation, the user will be able to press a button to accept or decline it.



Part IV
Boundary-Control-Entity Diagrams

8 Introduction

Boundary-Control-Entity Diagrams extend the User Experience Diagrams proving some clues on how the system will actually work.

The boundary classes represent the objects that interface with the system actors. In our case, the only actor is the user (guest or registered) and so boundary classes are the same as the screens seen in the UX Diagrams.

The control classes implement the business logic of the application and, for this reason, provide some methods to perform all *MeteoCal* functions.

Entities represent the system data as seen in RASD's Class Diagram.

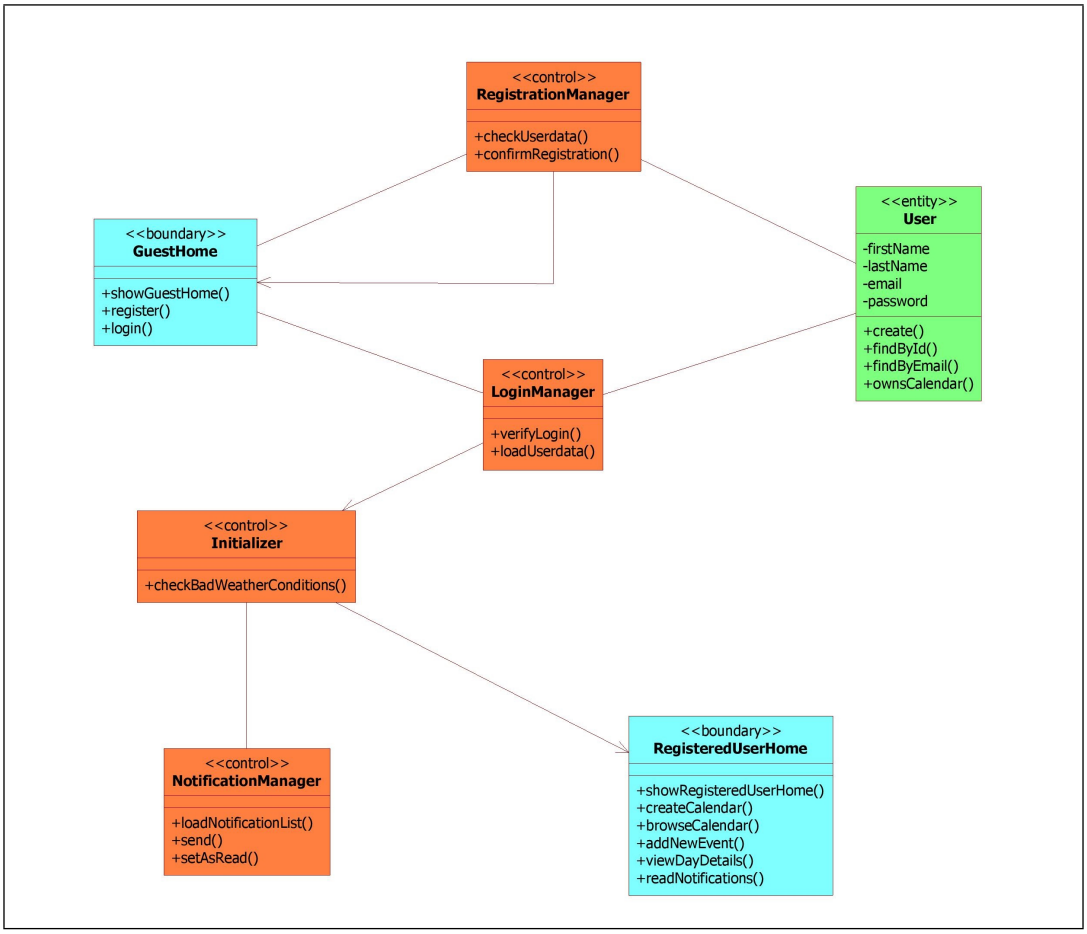
As in the UX Diagrams, the system has been split in three parts.

9 Registration and Log-in

As seen in the User Experience Diagrams, from the Guest Home the user will be able to register or log-in. For the former, the control unit Registration Manager will take care of validation (it will, for example, check if the provided e-mail is valid and if it has already been taken) and, if all data is consistent, of confirming the action by creating a User instance.

Log-in Manager will provide similar functions to the previous control unit, it will validate e-mail and password and, if correct, will log the user in.

Right after the log-in, the Initializer will take control to perform some preliminary actions. In the current implementation, it will just check if in the next day there are outdoor events and bad weather conditions to send an alert to the user through the Notification Manager. After this, the Registered User Home will be loaded and the user will be able to perform the actions described in the next sections.



10 Calendar Browsing and Event Management

From the Registered User Home, the user will be able to create a calendar and then browse it. This functions are implemented by the Calendar Manager, which will take care of the creation of a new Calendar instance and then load the days table for each selected month.

The creation of a new event will be provided by the Event Manager, which will validate the user input, provide the weather forecast if available and finally create a new Event instance. It will also call the Notification Manager if some invitations need to be sent.

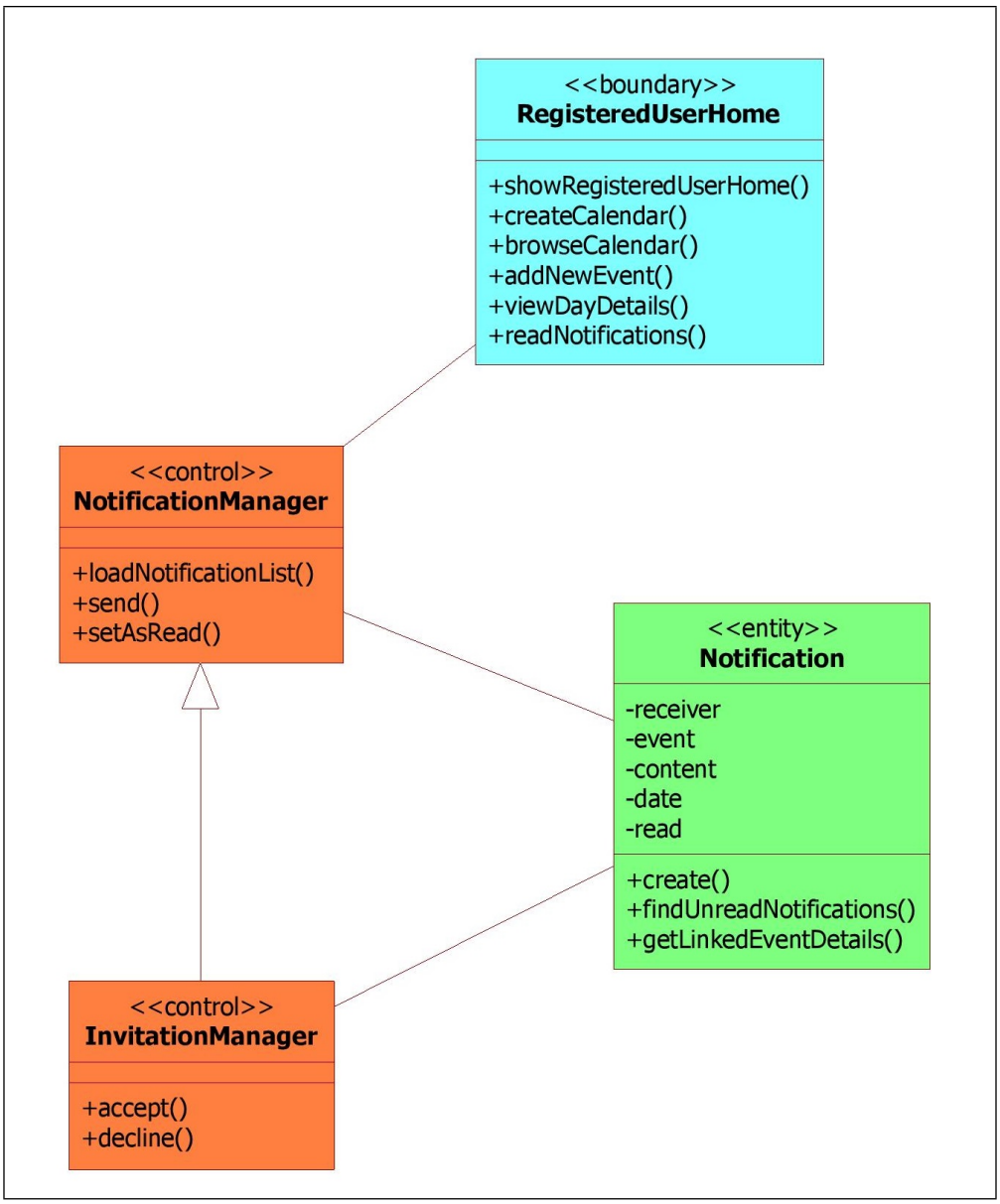
From the home page, when the user clicks on a day, the Calendar Manager will load the Day Details screen. The Event Manager will load the event list for the day and, in case of event update or delete, it will take care of all necessary procedures.


<<entity>>
Calendar
-owner
-badWeatherDesc
+create()
+findByOwner()
<<boundary>>
RegisteredUserHome
+showRegisteredUserHome()
+createCalendar()
+browseCalendar()
+addNewEvent()
+viewDayDetails()
+readNotifications()
<<control>>
CalendarManager
+addCalendar()
+loadCalendarData()
+loadMonthTable()
+loadDayDetails()
<<boundary>>
DayDetails
+showDayDetails()
+readWeatherForecast()
+viewEventList()
+viewEventDetails()
+deleteEvent()
+updateEvent()
<<control>>
EventManager
+loadEventList()
+loadEventDetails()
+checkEventData()
+addEvent()
+updateEvent()
+deleteEvent()
+sendInvitations()
+loadWeatherForecast()
<<control>>
NotificationManager
+loadNotificationList()
+send()
+setAsRead()
<<entity>>
Event
-organizer
-name
-start
-end
-place
-outdoor
+create()
+delete()
+getEventsByDay()
+getEventById()
+isOrganizer()
+getInvitedUsers()

11 Notification Management

As previously described, the Registered User Home will contain a list of unread notifications. This list will be provided by the Notification Manager, which will also take care of setting them as read and sending new notifications.

In case of an invitation, the Invitation Manager will provide the implementation of the accept/decline buttons, with all related actions that we will explore in detail in the Sequence Diagrams.



Part V
Sequence Diagrams

12 Introduction

Sequence Diagrams will provide a more detailed explanation of what has been described in the BCE Diagrams. They will show the sequence of methods called in the boundary, control and entity classes to better describe how the system behaves after a user input.

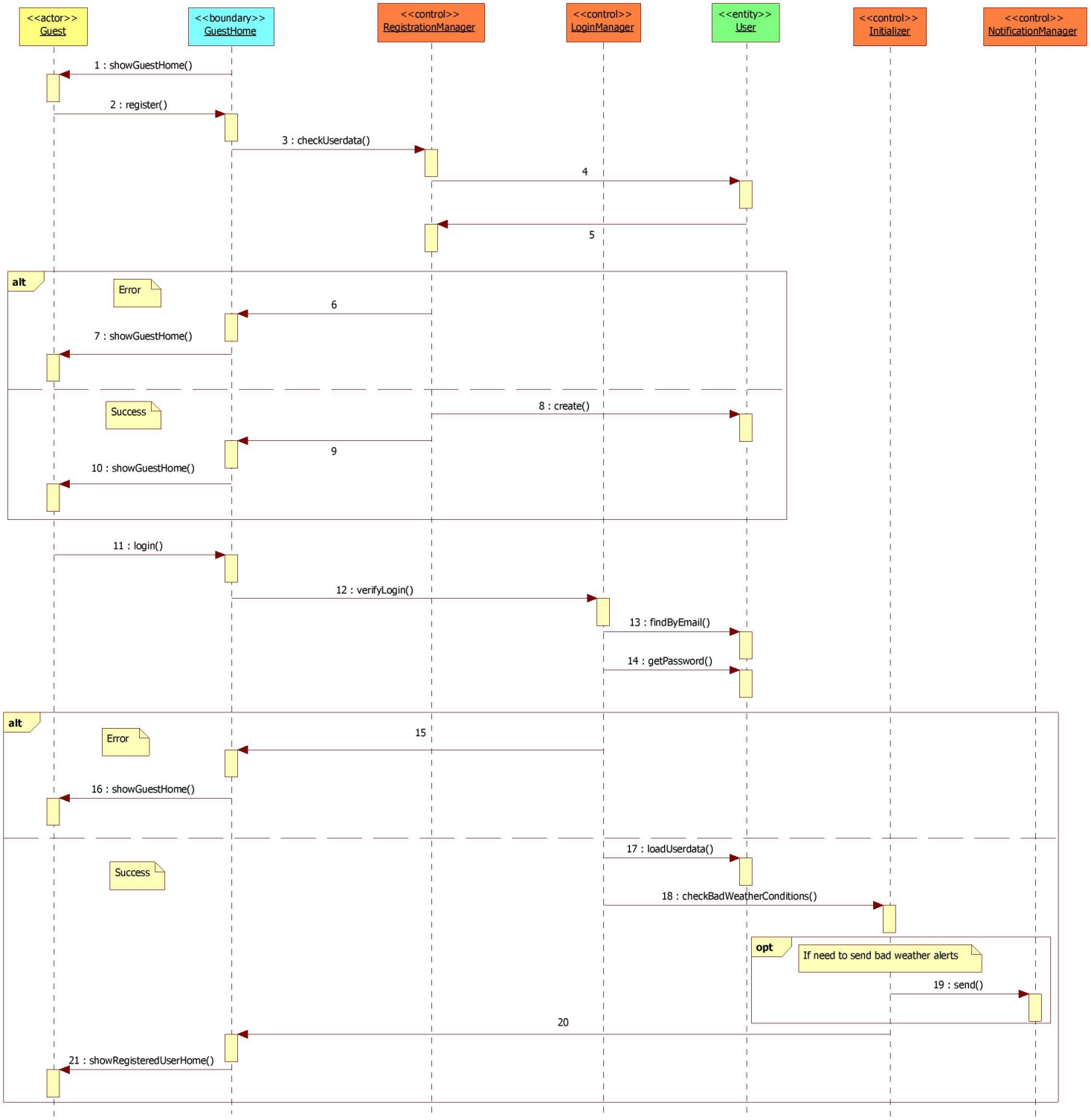
The system has been split in the usual three parts.

13 Registration and Log-in

From the Guest Home, the user can choose to register. After filling the required data, the Registration Manager checks it internally (e.g. all required fields must be non-empty) and by performing some queries in the User entity (e.g. e-mail address already in use). If an error is detected the user will remain in the registration page, while in the other case the Registration Manager will create a new User instance and then redirect the user to the Guest Home.

From there, the guest will be able to provide his/her log-in credentials, which will be validated by the Log-in Manager (that will query the User entity for an instance with provided e-mail and password). In case of error, the user will be notified and try again, while if everything is correct the system will load the user data and call the Initializer.

The Initializer will check if in the next day the user has outdoor events with a bad weather forecast and, if some are found, it calls the Notification Manager to send an alert. After this, the Registered User Home is shown.

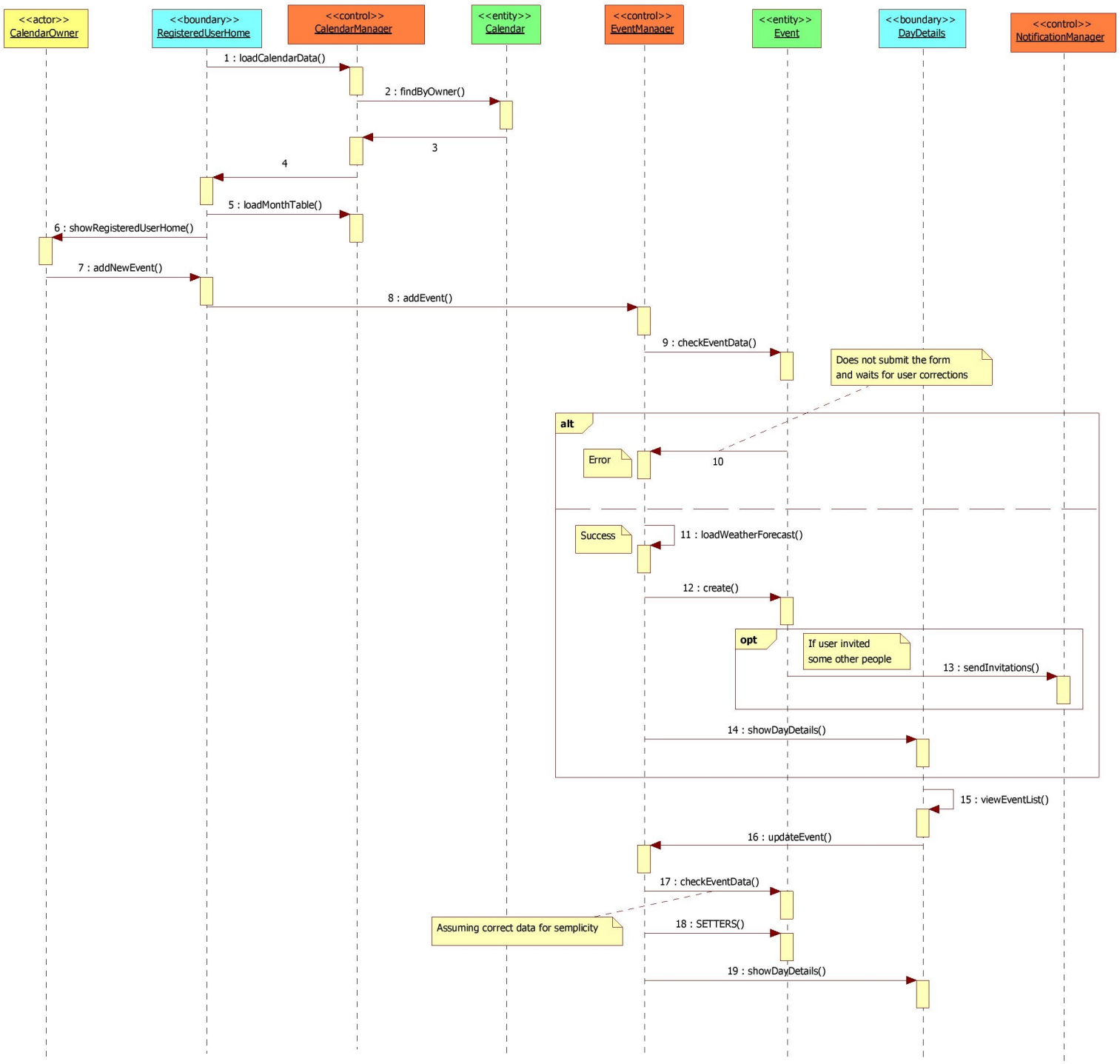

<<actor>>
Guest
<<boundary>>
GuestHome
<<control>>
RegistrationManager
<<control>>
LoginManager
<<entity>>
User
<<control>>
Initializer
<<control>>
NotificationManager
1 : showGuestHome()
2 : register()
3 : checkUserdata()
4
5
alt
Error
6
7 : showGuestHome()
Success
8 : create()
9
10 : showGuestHome()
11 : login()
12 : verifyLogin()
13 : findByEmail()
14 : getPassword()
alt
Error
15
16 : showGuestHome()
Success
17 : loadUserdata()
18 : checkBadWeatherConditions()
opt
If need to send bad weather alerts
19 : send()
20
21 : showRegisteredUserHome()

14 Calendar Browsing and Event Management

In this section we assume that the user already owns a calendar. From the Registered User Home the Calendar Manager will load the user's calendar information by querying the Calendar entity and then load the current month table.

If the user decides to create a new event, the Event Manager will receive the input and then will check if the data is consistent (e.g. the start time is before the end time). If the input has some errors, the system will notify the user and wait for another submit, while in the other case the Event Manager will create a new Event instance (with a weather forecast, if available), optionally send some invitations and then redirect the user to the Day Details page.

From the Day Details page the user can select, from the event list, to update or delete an event. The Event Manager will check the data consistency and take care of all required steps. At the end, it will redirect the user to the Day Details page.

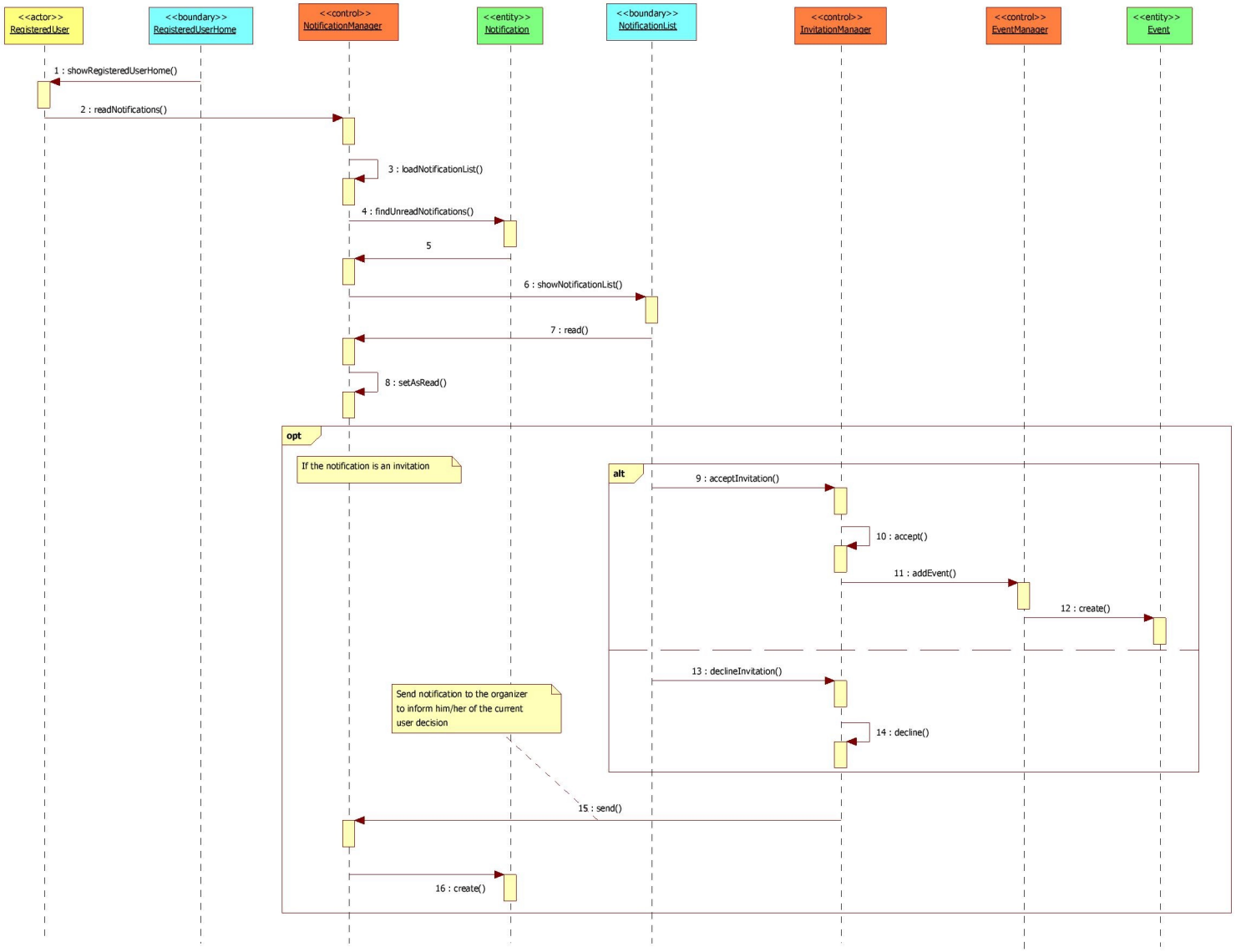

<<actor>>
CalendarOwner
<<boundary>>
RegisteredUserHome
<<control>>
CalendarManager
<<entity>>
Calendar
<<control>>
EventManager
<<entity>>
Event
<<boundary>>
DayDetails
<<control>>
NotificationManager
1 : loadCalendarData()
2 : findByOwner()
3
4
5 : loadMonthTable()
6 : showRegisteredUserHome()
7 : addNewEvent()
8 : addEvent()
9 : checkEventData()
Does not submit the form
and waits for user corrections
alt
Error
10
Success
11 : loadWeatherForecast()
12 : create()
opt
If user invited
some other people
13 : sendInvitations()
14 : showDayDetails()
15 : viewEventList()
16 : updateEvent()
17 : checkEventData()
Assuming correct data for semplicity
18 : SETTERS()
19 : showDayDetails()

15 Notification Management

In this section we will explore the notification management from the receiver point of view. After loading the Registered User Home, the user will be able to read the notification list, provided by the Notification Manager by querying the Notification entity for unread instances.

When the user clicks on one notification, the system will show its content and set it as read.

If the notification is an invitation, the user will be able to either accept or decline it. In the former case, the Invitation Manager will add the event to the user's calendar by creating an Event instance. In both situations, the Invitation Manager sends to the event organizer a notification to inform him/her of the current user decision.


<<actor>>
RegisteredUser
<<boundary>>
RegisteredUserHome
<<control>>
NotificationManager
<<entity>>
Notification
<<boundary>>
NotificationList
<<control>>
InvitationManager
<<control>>
EventManager
<<entity>>
Event
1 : showRegisteredUserHome()
2 : readNotifications()
3 : loadNotificationList()
4 : findUnreadNotifications()
5
6 : showNotificationList()
7 : read()
8 : setAsRead()
opt
If the notification is an invitation
alt
9 : acceptInvitation()
10 : accept()
11 : addEvent()
12 : create()
13 : declineInvitation()
14 : decline()
Send notification to the organizer
to inform him/her of the current
user decision
15 : send()
16 : create()

Part VI
Conclusion

16 Used Tools

To redact this document, the following tools have been used:

- **Texmaker**: to build this document using LaTeX
- **WhiteStarUML**: for all UML diagrams (User Experience, Boundary-Control-Entity, Sequence Diagrams)
- **MySQL Workbench**: for the persistent data representation

17 Changelog

Versions of this document:

- **7th December 2014**: first release