Politecnico di Milano

MeteoCal

Design Document

Authors:
DAVERIO Matteo
DE MARIA Valerio

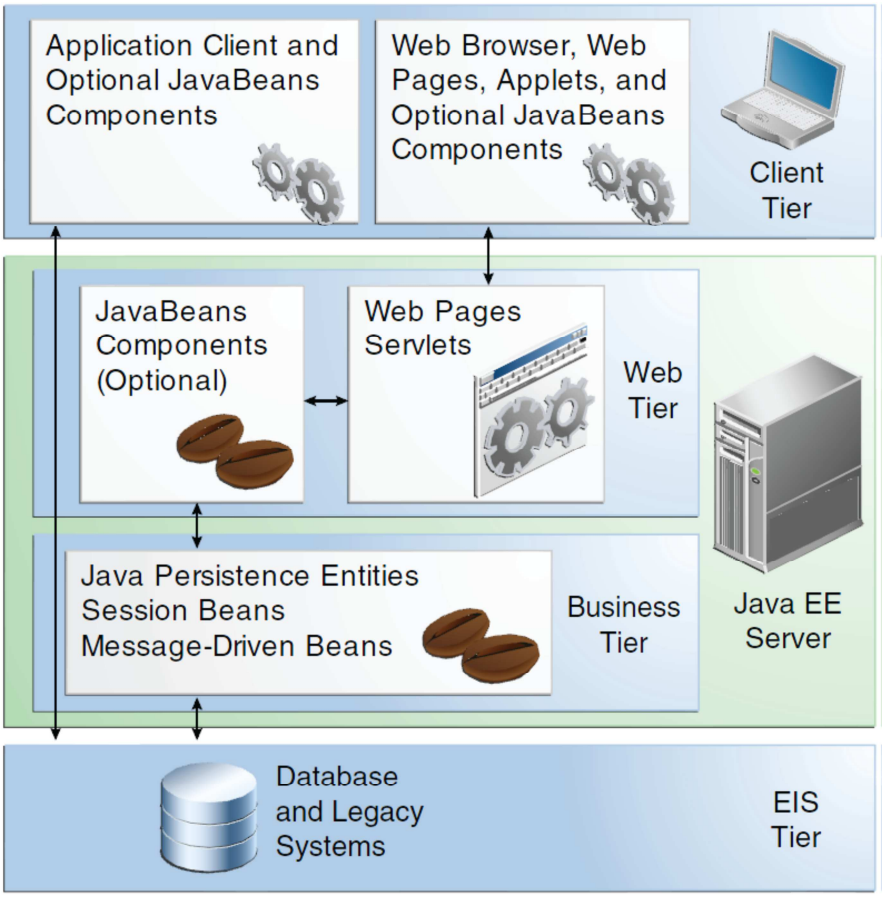
December 6th 2014

TABLE OF CONTENT

1. ARCHITECTURE DESCRIPTION

1.1. JEE Architecture Overview

Before starting to explain our application's architecture we want to focus on JEE Architecture.



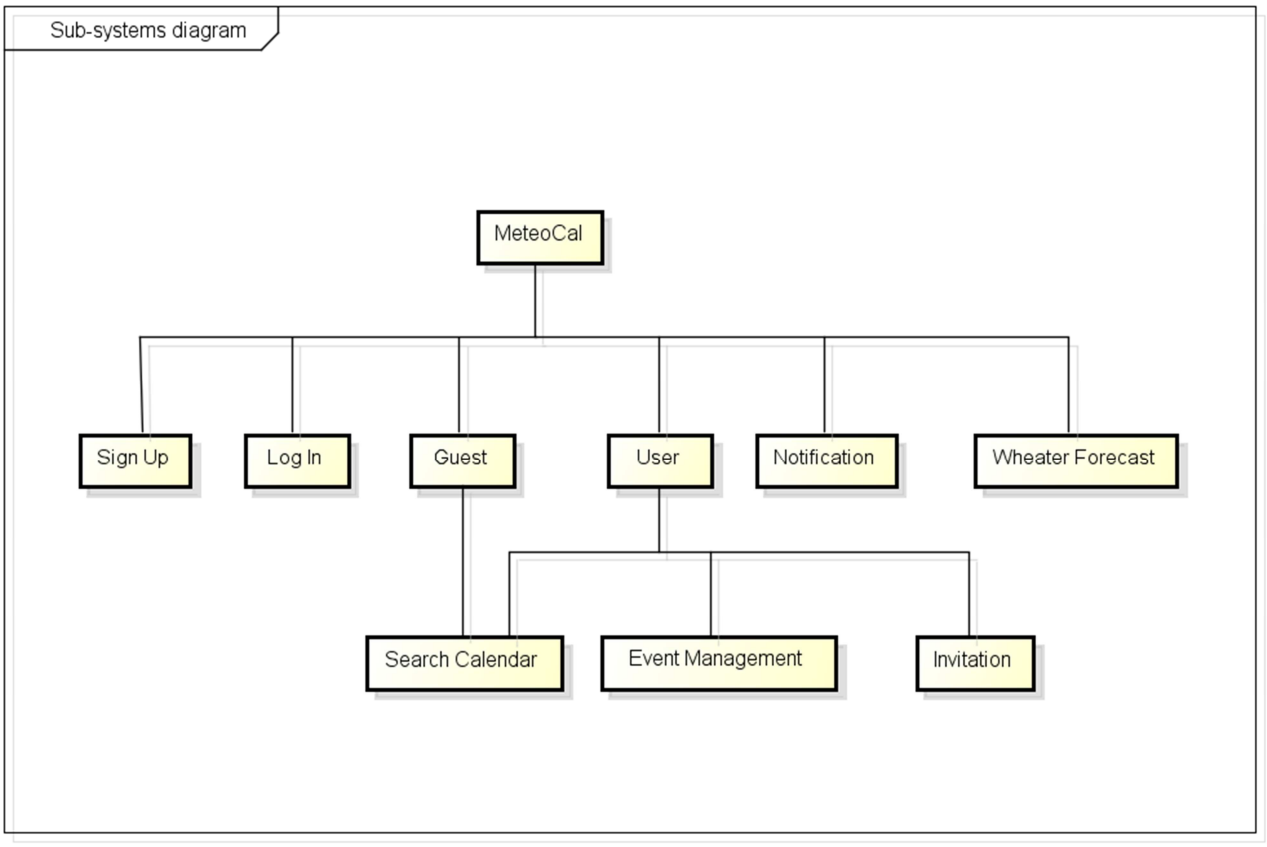
JEE has a four tiered architecture divided as:

- **Client Tier**: it contains Application Clients and Web Browsers and it is the layer that interacts directly with the actors. As our project will be a web application the client will use a web browser to access pages;
- **Web Tier**: it contains the *Servlets* and Dynamic Web Pages that need to be elaborated. This tier receives the requests from the client tier and forwards the pieces of data collected to the business tier waiting for processed data to be sent to the client tier, eventually formatted;
- **Business Tier**: it contains the Java Beans, that contain the business logic of the application, and Java Persistence Entities.

- **EIS Tier**: it contains the data source. In our case it is the database allowed to store all the relevant data and to retrieve them.

1.2. Identifying Sub-System

We decide to adopt a top-down approach at least at this point of the project. Maybe, once defined the sub-systems, we will adopt a bottom-up approach in order to create more reusable components.
So we think it is now necessary to decompose our system into other sub-systems, in order to make it easy to understand the issues that we will find in implementing functionalities and to separate, logically, groups of functionalities and state clearer their interaction.
We separate our system into the sub-systems show in the scheme hereunder.



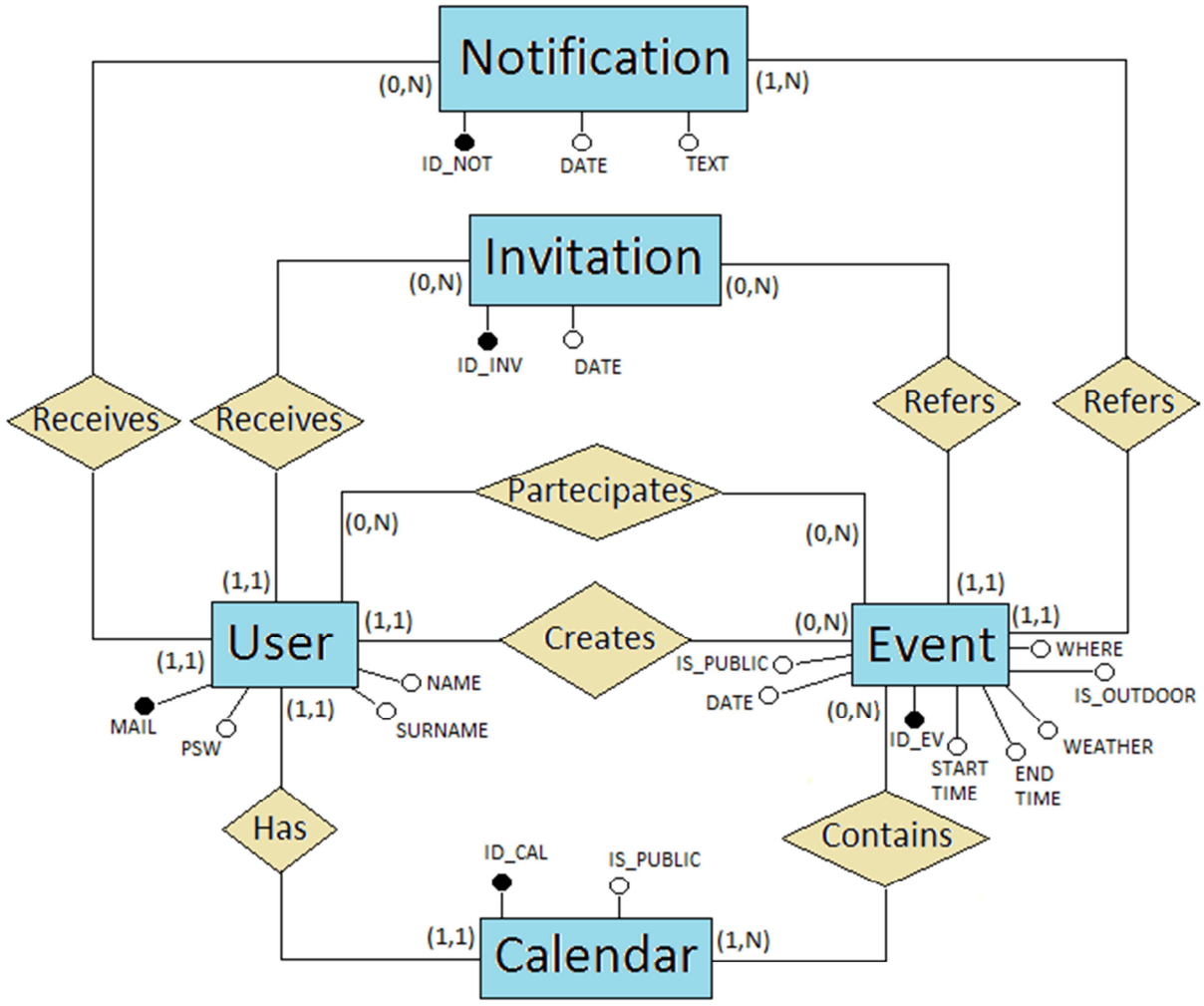
2. PERSISTENT DATA MANAGEMENT

Our data is stored into a relational database. You can find below the design of our database in terms of Entity-Relationship Diagram. Moreover we will explain in a detailed way entities, relations and, in general, the motivations of our design.

2.1. Conceptual Design

Conceptual design allows to start thinking about the data we want to store and about the relations between them.

Here it is the E/R design diagram:



Our system can be used both by Users and Guests, the last ones are not signed-up so we have no data about them in our database.

Notification and *Invitation* are two entities similar each other in terms of relations and attributes but we decided to distinguish them because they are characterized by different features :

Notification: is generated by the system to notify the user about creation (that's the reason for 1 in the MIN cardinality between *Notification* and *Refers*), update, deletion and proposed rescheduling of a specific event. The MIN cardinality between *Notification* and *Refers* is set to 1 because for every *Event* created, there is always a corresponding created *Notification*. The record of a *Notification* is deleted from the DB once user confirms having seen it or decides to accept or decline the changings proposed by the system.

Invitation: is sent to the user when he is invited to a specific event. The record of an Invitation is deleted from the DB once the invited user decides either to accept or decline it.

Each user has exactly one calendar. The entity *Calendar* contains all the events in which the calendar's user is involved (either as creator or as participant). The MIN cardinality between *Calendar* and *Contains* is set to 1 because for every *Event* created there is at least one corresponding *Calendar* which contains it.

Concerning the attributes, the following comments apply :

In *Notification* the attribute "text" represents the message that will be associated to that notification like "the date of event X has been changed into date Y".

In *User* we take as primary key the attribute "mail" because it holds the necessary properties to be it. The attributes "name" and "surname" are the only ones referring to personal data of the *User*; additional personal

information such as age or gender are here out of scope, given that purpose of our system is event management and not social networking.

In *Calendar* and *Event* we put the Boolean attribute “IsPublic” to represent the privacy settings (public or private).

In *Event* we save the weather information in the string attribute “weather”, this field will be fulfilled by the system after the event creation in the case weather forecast are available. Otherwise, it remains blank. “IsOutdoor” is a Boolean attribute to represent the fact that the event will take place outdoor or indoor.

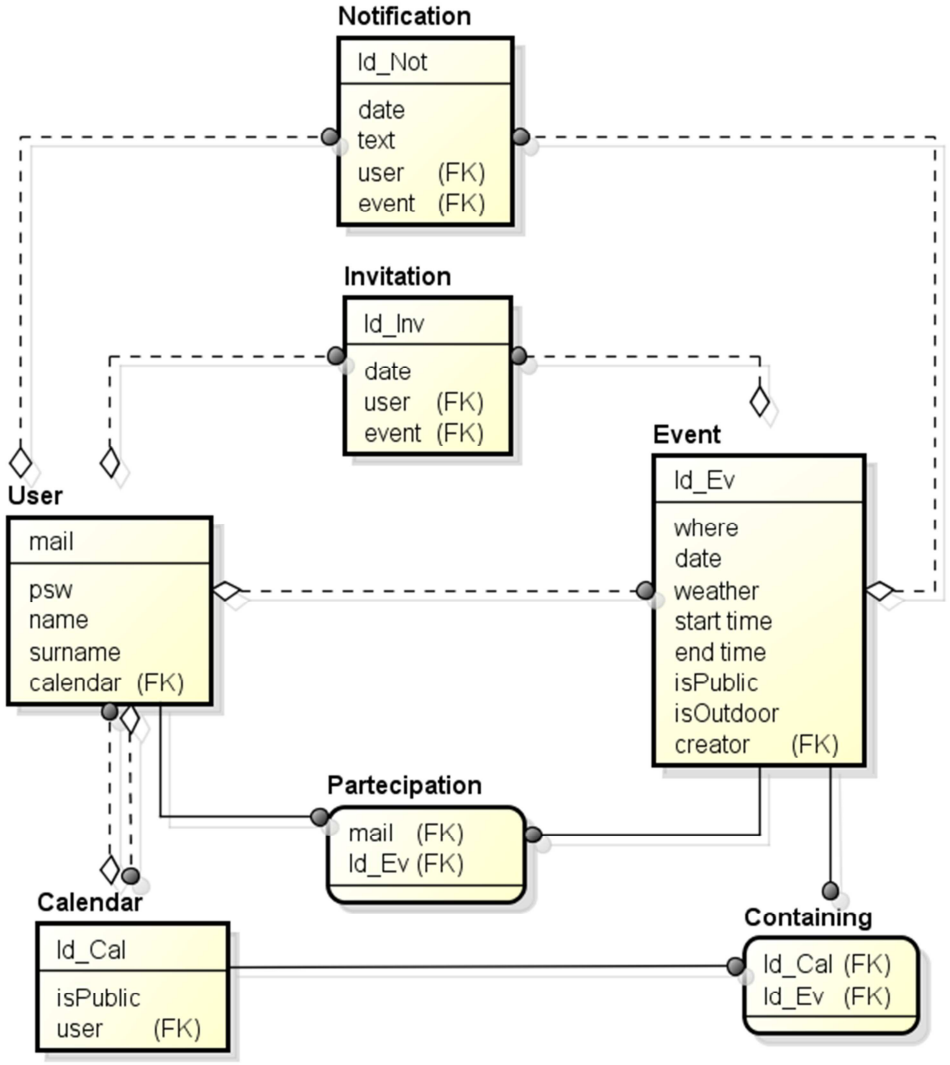
2.2. Logical Design

Logical Design has the aim to better represent the database structure of our system

2.2.1. Translation to Logical Model

- Relation “Creates” between User and Event is translated inserting the foreign key “creator” into the table Event.
- Relation “Receives” between User and Invitation is translated inserting the foreign key “user” into the table Invitation. The same for the relation between User and Notification.
- Relation “Refers” between Invitation and Event is translated inserting the foreign key “event” into the table Invitation. The same for the relation between Notification and Event.
- Relation “Has” between User and Calendar is translated inserting the foreign key “calendar” into the table User and the foreign key “user” into the table Calendar. It would be possible also insert only one of these foreign keys but we decided to insert both make easier future queries in our DB (we can reach the user of a calendar directly from the table Calendar and vice versa).

- Relation “Participates” between User and Event has 0:N cardinalities on both edges so we create a new table “Participation” with the two foreign keys “user” and “event” as primary key.
- Relation “Contains” between Calendar and Event has 0:N /1:N cardinalities on both edges so we create a new table “Containing” with the two foreign keys “calendar” and “event” as primary key.



The final model has the following physical structure:

- **User**(mail, psw, name, surname, *calendar*);
- **Event**(Id_Ev, where, date, weather, start time, end time, isPublic, isOutdoor, *creator*);
- **Notification**(Id_Not, date, text, *user*, *event*);
- **Invitation**(Id_inv, date, *user*, *event*);
- **Calendar**(Id_Cal, isPublic, *user*);
- **Participation**(*mail*, *Id_Ev*);
- **Containing**(*Id_Cal*, *Id_Ev*);

We have underlined the primary keys of each table and we put in Italic the foreign keys that each table contains (maybe it is not so understandable which table foreign keys refer to, but it is easily understandable watching the schema drawn above).

3. USER EXPERIENCE

In this part, we want to describe the experience given by our system to its user with an User Experience(UX) Diagram. To create that, we use a Class Diagram, using some stereotypes (*<<screen>>*, *<<screen compartment>>* and *<<input form>>*) to make it more understandable. In this Diagram, *<<screen>>* represents pages, *<<screen compartment>>* represents parts of the page that can be shared with others. *<<input form>>*, eventually, represents some input fields that can be fulfilled by a user (this information will be submitted to the system clicking on a button).

We also decided to divide the UX Diagram in functionalities in order to better understand all the Diagram.

Each of the paragraphs below is titled with the name of the functionality represented by the UX Diagram drawn.

3.1. Homes

We can see below the Diagram that represents the different home pages showing how a generic user can interact with all the system. All the two home pages inherit from an Home screen that is also marked with $, so it is reachable from every page of the system. For an easier interpretation of the schema we decided to draw some components that are contained in all the screens, but we decided to draw them only in this UX Diagram, if not, it would have been very difficult to understand other diagrams.

Here is a list of these elements:

- **Log In Panel** and **Guest Menu**: They are in all guest's pages;
- **Notification Panel, User Menu, Calendar** and **Log Out Panel**: They are in all user's pages;

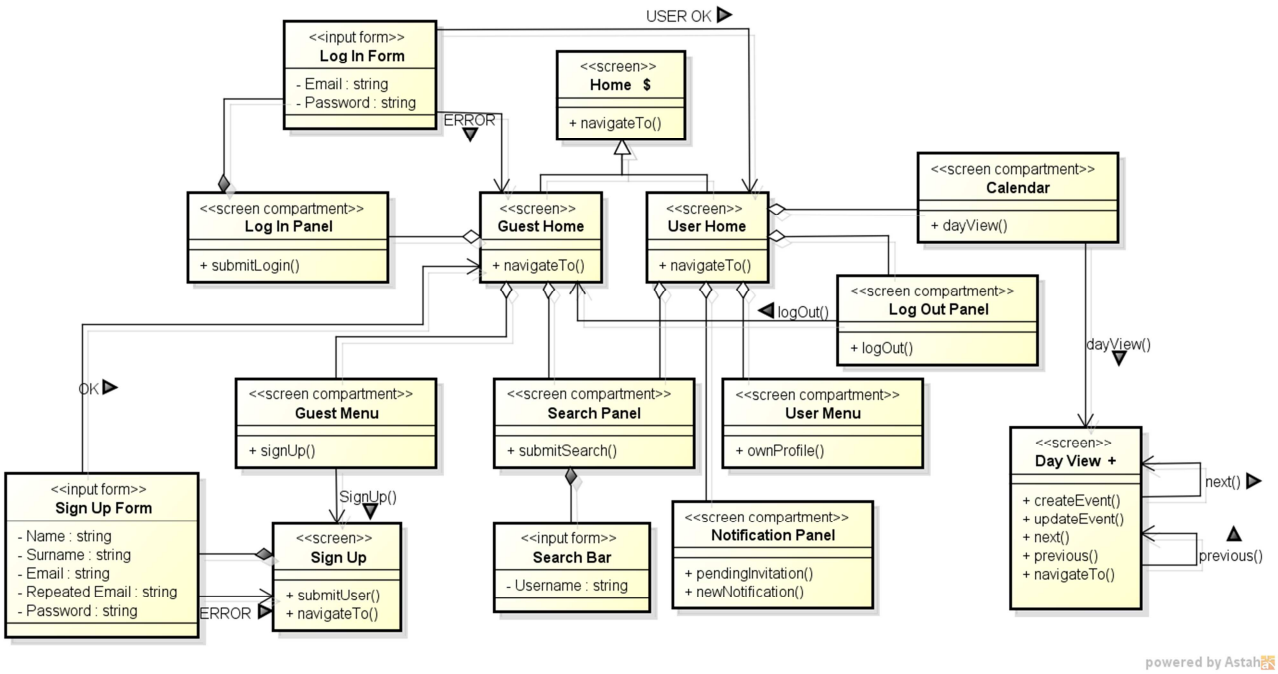
- **Search Panel:** It is in all pages of guest and user;

Moreover, from a *Guest Home* we can log in through the *Log In Panel* and, if there is no error, we will be redirected to our user's home page. If there is an error we will be redirected to the *Guest Home* again. If we've already logged in we can perform a logout through the *Log Out Panel*.

Menus contain the principal functionalities exploited by each type of generic user.

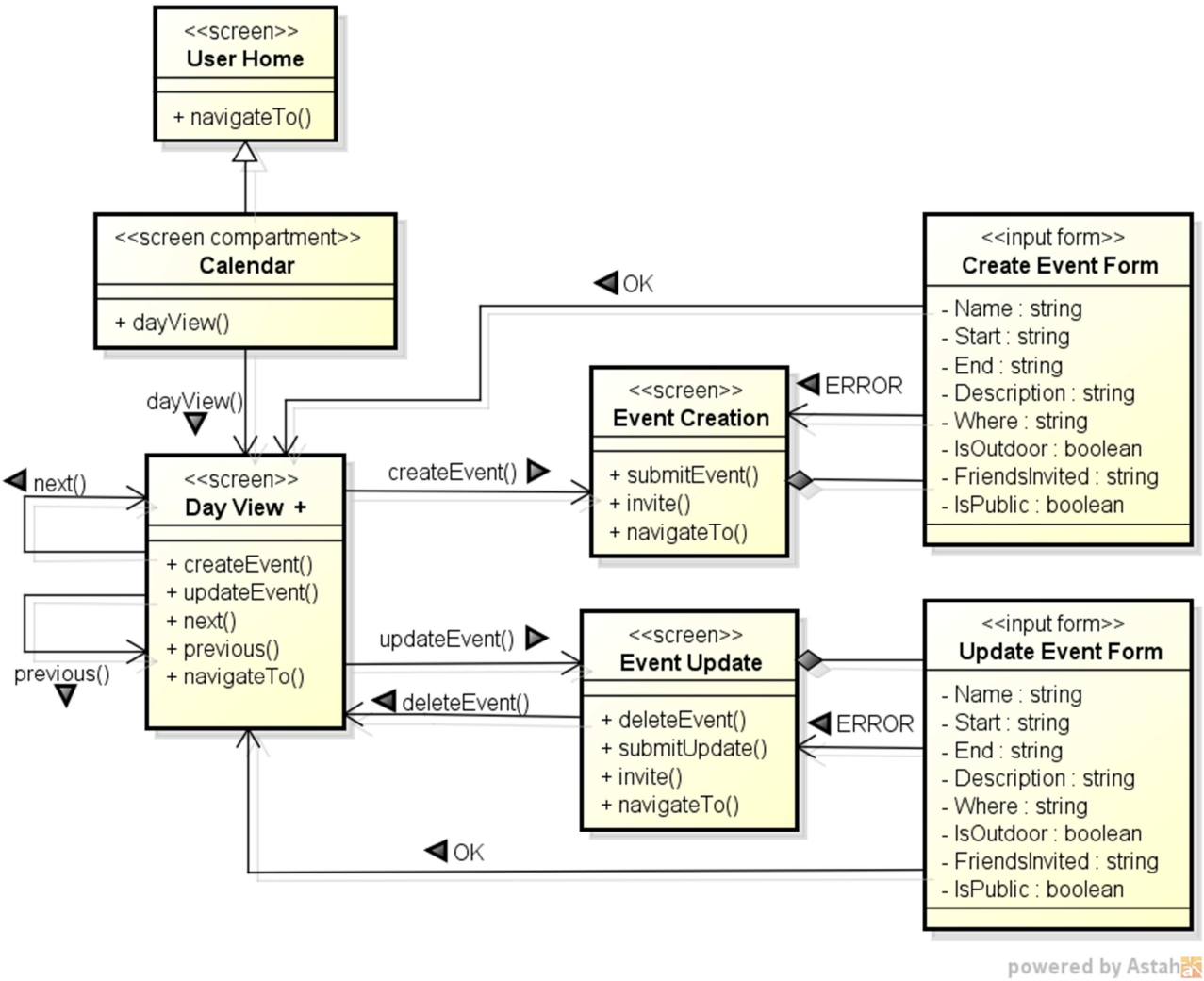
The *Notification Panel* shows if there are new notifications(like delete of an event, change on detail of an event, or propose to change detail of an event by the system) or pending invitations to an event and allows to reach the associated pages.

The last functionality expressed by this UX Diagram is the possibility of signing up reachable from the *Guest Menu*. If we click on the right link the *Sign Up* screen is shown. This screen contains a form to be fulfilled with all the data of the new user. Eventually, we can submit the data to the system (that will have different clear behaviors in case of error or not).



3.2. Event Managing

This part of the UX Diagram allows to understand how the event managing functionalities are offered through the user experience.



From the *Calendar,* a screen compartment of the *User Home*, we can access our daily view(*Day View* screen), which represent a view of the

event present in that day and then, if we want to, create, modify or delete it.

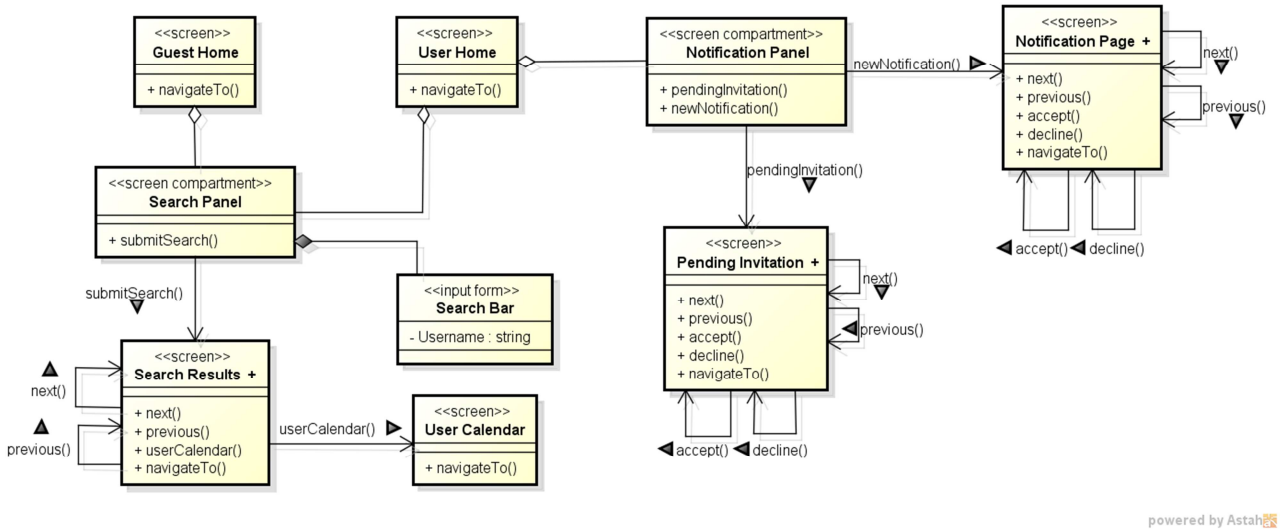
If we want to insert a new event which is not yet in the system we can do it through an appropriate screen (*Event Creation* screen).

If we want to update or delete an event, we can do through an appropriate screen (*Event Update* screen), simply modifying data about the event, or pressing delete button.

Every time that a user create, modify or delete an event the *Day View* page is reloaded with the result of the operation.

3.4. User Search, Notification and Invitation Management

This part of the UX Diagram allows to understand how the user search, notification and invitation management functionalities are offered through the user experience.



Search functionalities can be performed through the *Search Bar,* and can be used by a user or by a guest. From the input form associated to the compartment we can submit a username for the research and send them to the system that will answer with the *Search Results* page, showing a

set of *User Preview*s. Starting from here we can also click on the user's name and see its related calendar. If someone makes his calendar private, he does not compare in the *Search Results* page. Viewing other user calendar's we can see all his public events.

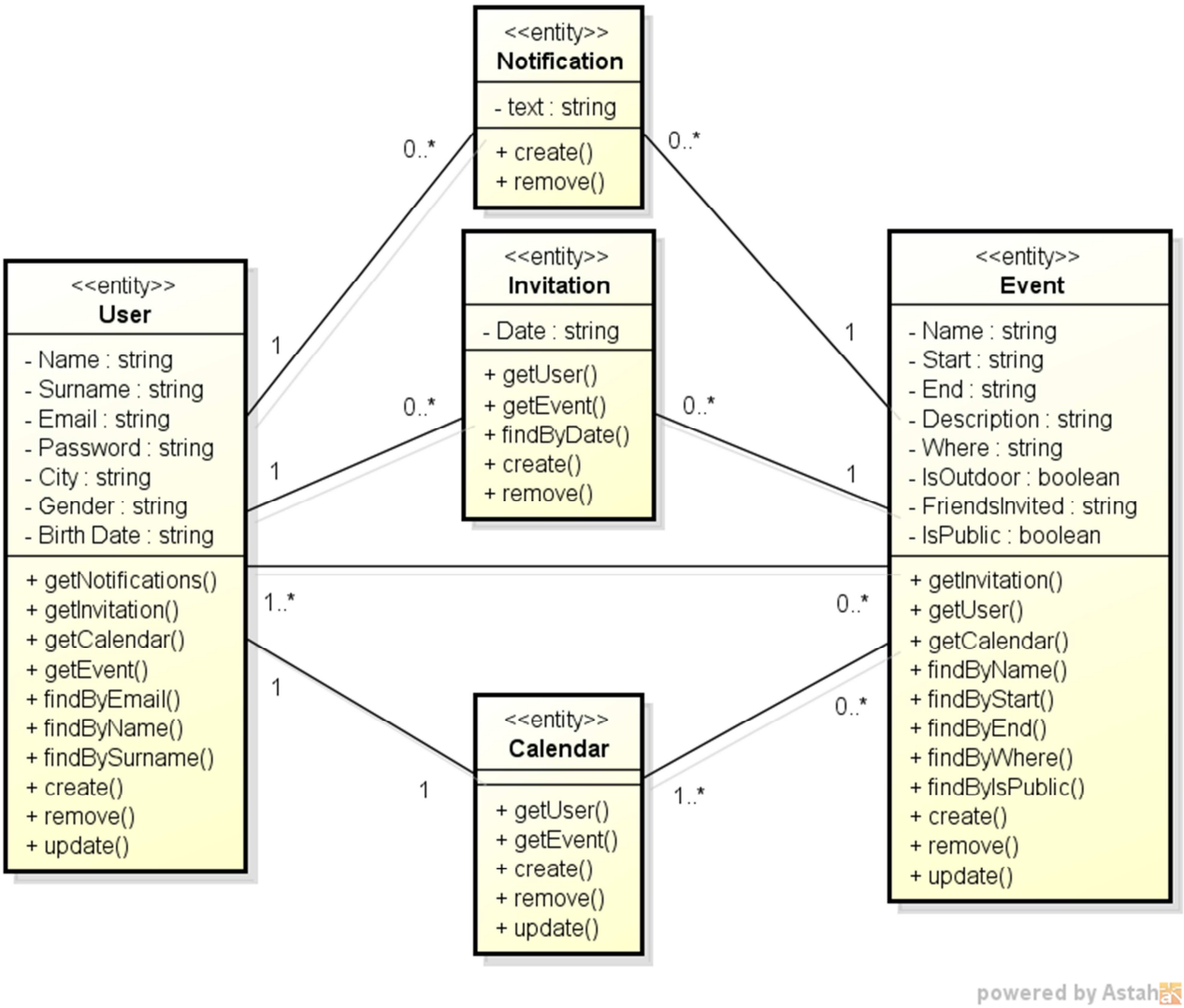
We can also receive notification or invitation requests. Invitations are sent from a user, which is proposing you to participate to his event. Notification are messages sent from the system, like to propose a change of date in one of your event, to let you know that an event in your calendar received some modification from his creator, and so on. These event will be notified by the *Notification Panel*. Through the appropriate link we can access to the *Pending Invitation* screen that shows us the set of pending invitations and, for each of them, it allows to accept it or decline it. Same things for the notifications, where we can see all the new notification in the *Notification Page* screen where also here, for each of them, we can accept it, decline it, or put the notification as read.

4. BCE DIAGRAMS

We decided to give a further design schema of MeteoCal using the Boundary-Control-Entity pattern, because it is very close to the Model-View-Controller pattern (in fact we can say that boundaries maps to the view, controls map to the controller and entities map to the model) and UML defines a standard to represent it.
It is important to know that boundaries are partially derived from the Use Cases Diagram provided in the RASD (so they don't need further explanations) and they gather some screen in the UX Diagram. All the methods of the screens are written into the appropriate boundaries (maybe some names changed a bit only to make them more understandable in this context). There is a method *showXXX()* for every screen in the UX Diagram. Screen compartment doesn't have a "show method" because we considered them as pieces of screens and not as standalone screens.
Finally it is important to say that entities do not represent the ER Diagram, but only a conceptual view of the entities used in the BCE.
We decided to separate the BCE diagram into sub-diagram, to make it clearer and understandable.

4.1. Entity overview

All entities of the BCE model will be presented in a very fragmented way, to prevent the diagrams to be too chaotic. For this reason we provide an entity overview, in the way that the reader can always refer to this diagram.

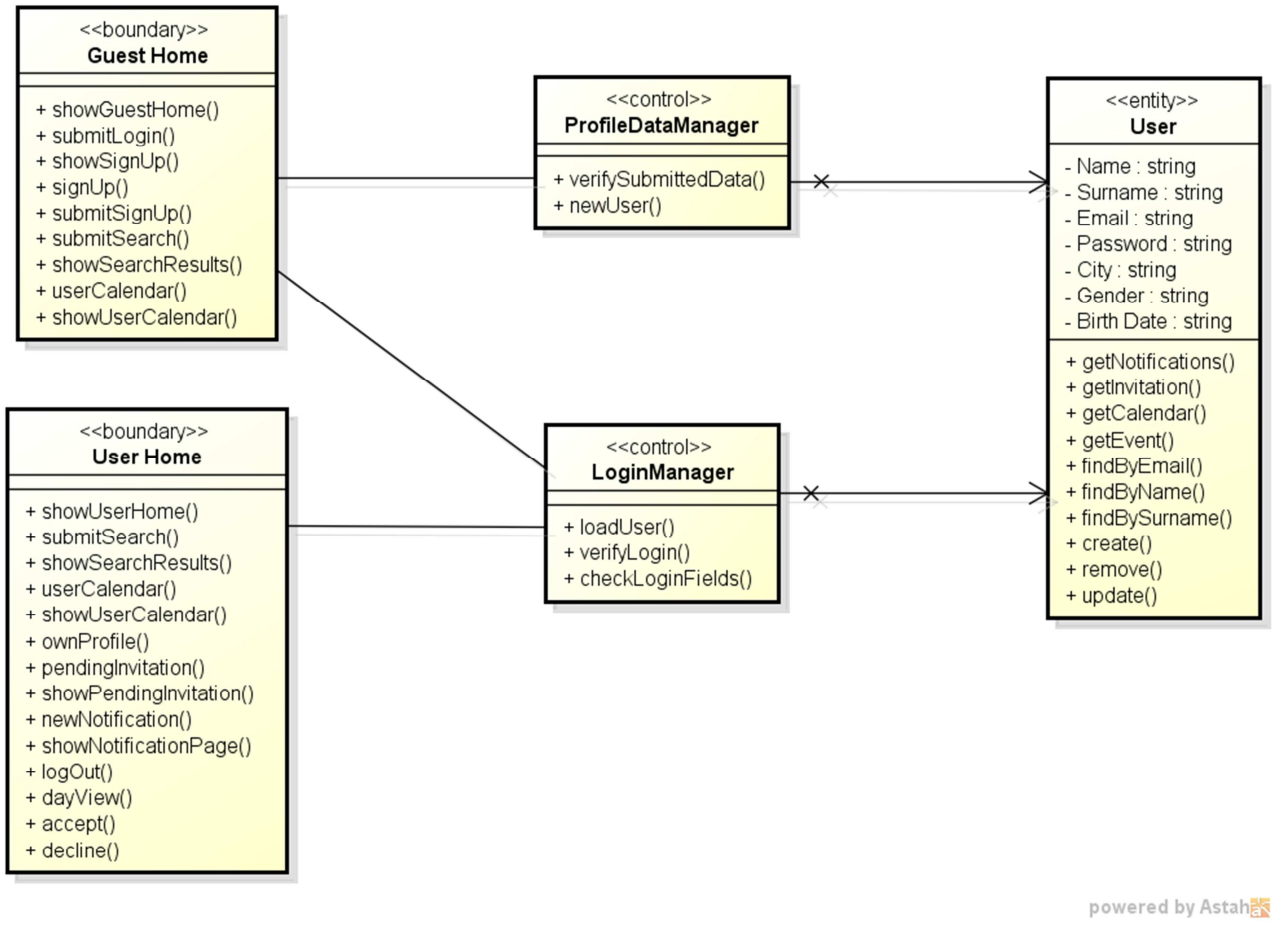


4.2. Sign up and Log in

The two boundaries *GuestHome* and *UserHome* represent the interface with users and the basic functionalities that they can exploit in their home page.
We write down now a synthetic description of the controllers used:

- ***ProfileDataManager:*** this controller manages the verification of profile data in case of sign up or modification of profile information. It also creates a new user in case of signing up correctly.

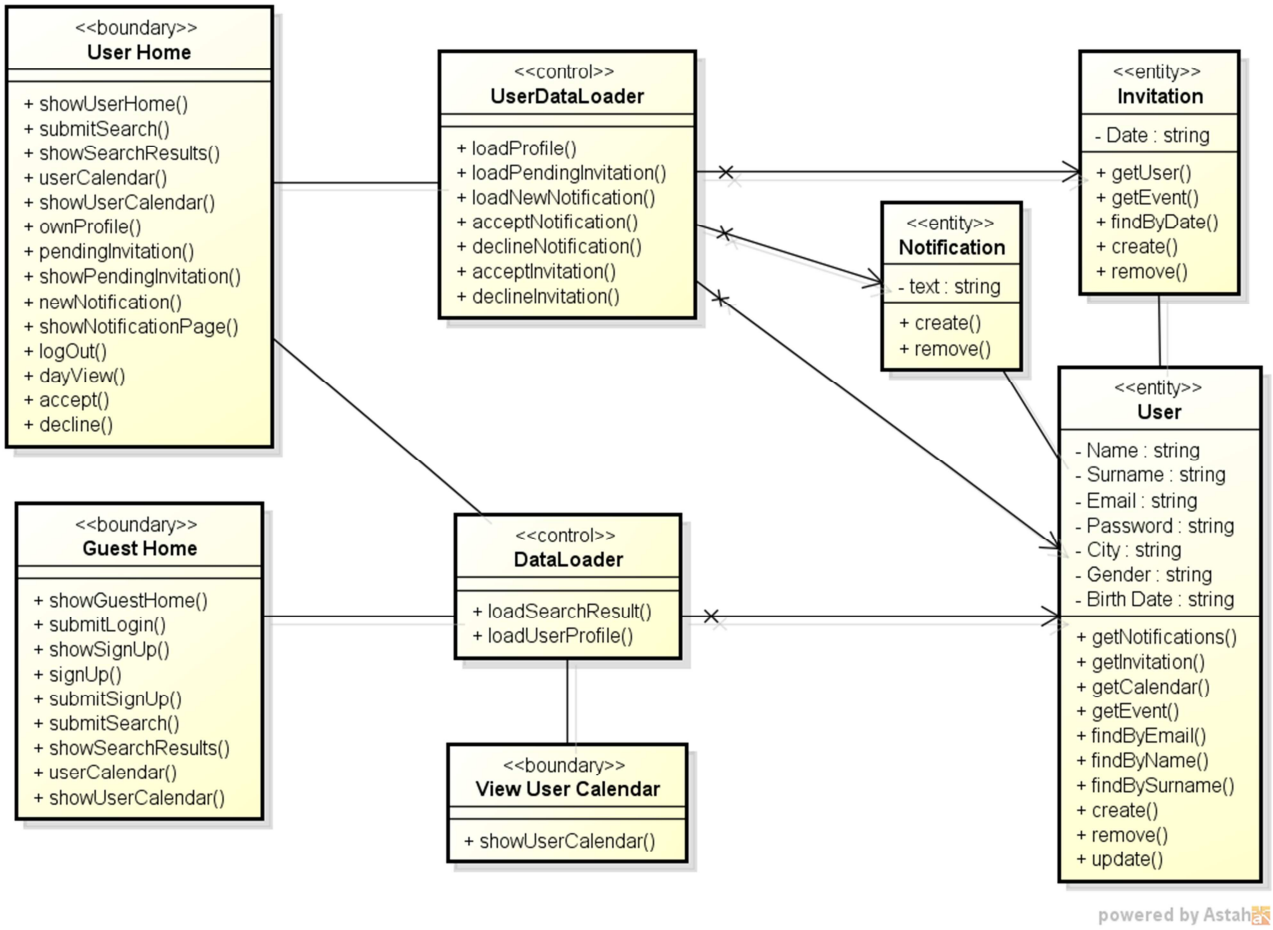
- ***LogInManager:*** this controller manages the verification of log in fields. The logic is that firstly, it examines if the fields are filled correctly (we need *checkLogInFields()* because it might happen that a user fills the password field with only three characters, and we know that, for instance, our password has to be at least eight characters long. In this case there is no need to look for the user in the database, because we know that the log in will be incorrect in any case). Then the controller can load the correspondent user finding it by e-mail address and then check the password.



4.2. User Search, Notification and Invitation

Searching functionalities are shared between the user and the guest.
Notification and invitation can be received only by users.
The controllers are:

- ***UserDataLoader:*** the loader of personal data about the user. This controller is linked to the user, the notifications and the invitations. It is used to manage the notification and invitation panel of the user. In fact it loads personal user profiles, pending invitations and new notifications. It also allow to accept or to decline invitations and notifications.
- ***DataLoader:*** this loader loads information that are not dependent from the user. In fact it loads other users' profile and search results (with "search results" we mean the results of a search performed to find users).



4.4.4. Events management

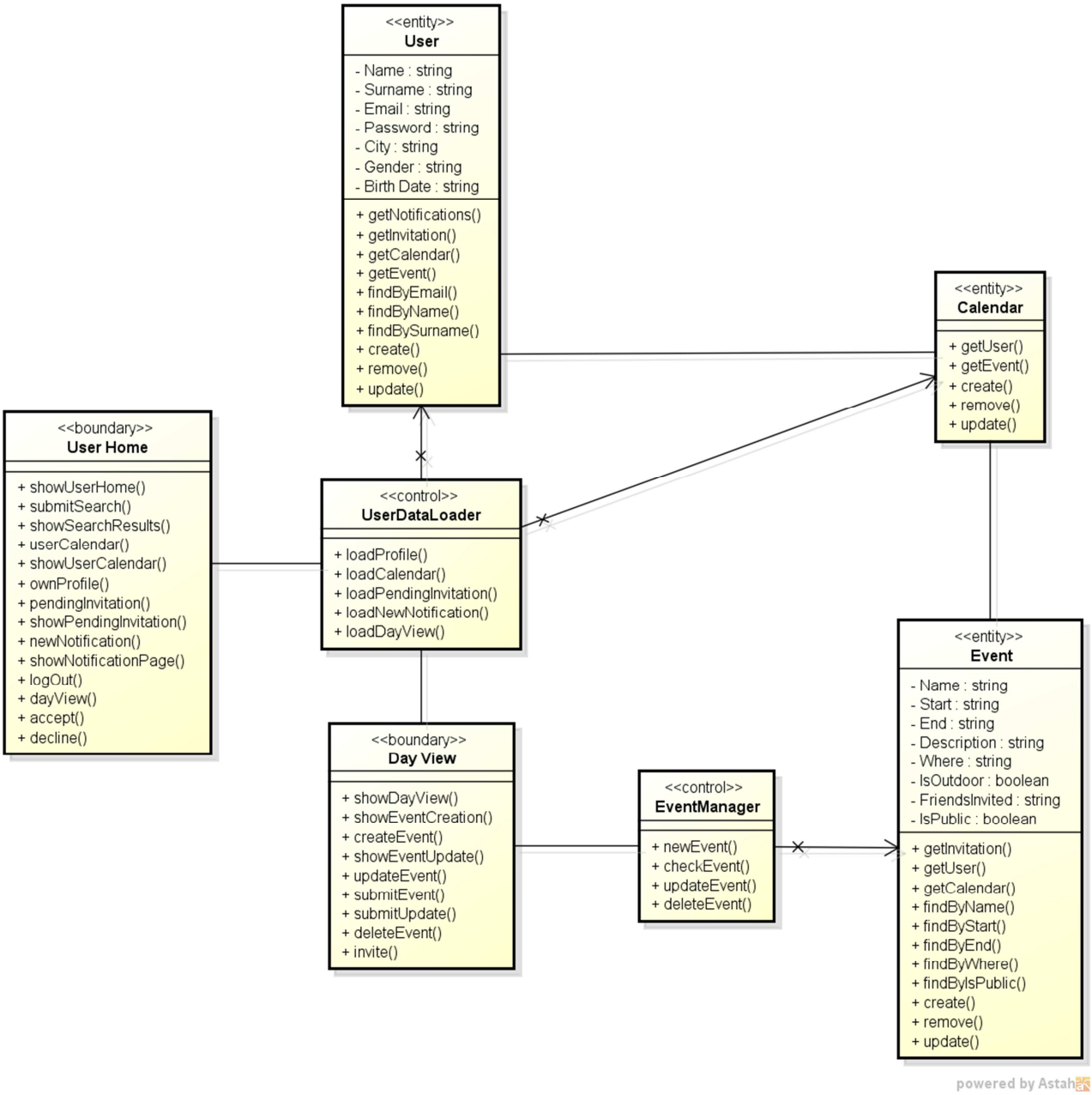
The *UserDataLoader* is used to:

- Show User Profile;
- Show User Calendar;
- Show Calendar Day View.

The *EventManager*:

- Creates a new event;
- Update an existing event;
- Check if the event that user is going to create is fulfilled with valid values;
- Delete an existing event.

All these features are put in relation with the Event entity using the entity's methods (see Sequence Diagram section).


<<entity>>
User
- Name : string
- Surname : string
- Email : string
- Password : string
- City : string
- Gender : string
- Birth Date : string
+ getNotifications()
+ getInvitation()
+ getCalendar()
+ getEvent()
+ findByEmail()
+ findByName()
+ findBySurname()
+ create()
+ remove()
+ update()
<<entity>>
Calendar
+ getUser()
+ getEvent()
+ create()
+ remove()
+ update()
<<boundary>>
User Home
+ showUserHome()
+ submitSearch()
+ showSearchResults()
+ userCalendar()
+ showUserCalendar()
+ ownProfile()
+ pendingInvitation()
+ showPendingInvitation()
+ newNotification()
+ showNotificationPage()
+ logOut()
+ dayView()
+ accept()
+ decline()
<<control>>
UserDataLoader
+ loadProfile()
+ loadCalendar()
+ loadPendingInvitation()
+ loadNewNotification()
+ loadDayView()
<<entity>>
Event
- Name : string
- Start : string
- End : string
- Description : string
- Where : string
- IsOutdoor : boolean
- FriendsInvited : string
- IsPublic : boolean
+ getInvitation()
+ getUser()
+ getCalendar()
+ findByName()
+ findByStart()
+ findByEnd()
+ findByWhere()
+ findByIsPublic()
+ create()
+ remove()
+ update()
<<boundary>>
Day View
+ showDayView()
+ showEventCreation()
+ createEvent()
+ showEventUpdate()
+ updateEvent()
+ submitEvent()
+ submitUpdate()
+ deleteEvent()
+ invite()
<<control>>
EventManager
+ newEvent()
+ checkEvent()
+ updateEvent()
+ deleteEvent()
powered by Astah

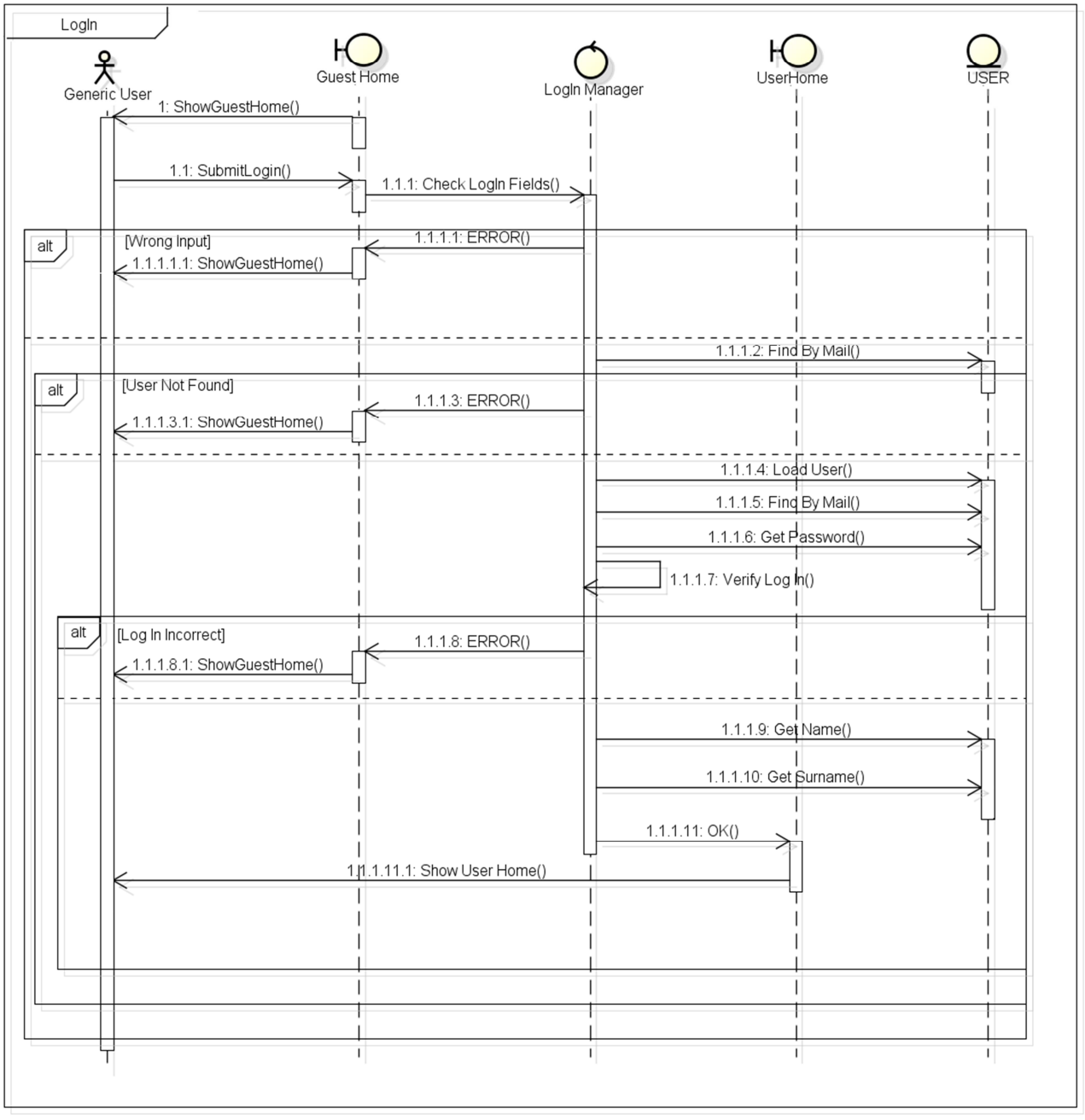
5. SEQUENCE DIAGRAMS

We provide some sequence diagram to let the reader better understand BCE diagrams described above. All the methods used are the methods listed into the BCE in boundaries, controls and entities.

5.1. Log In

A generic user:

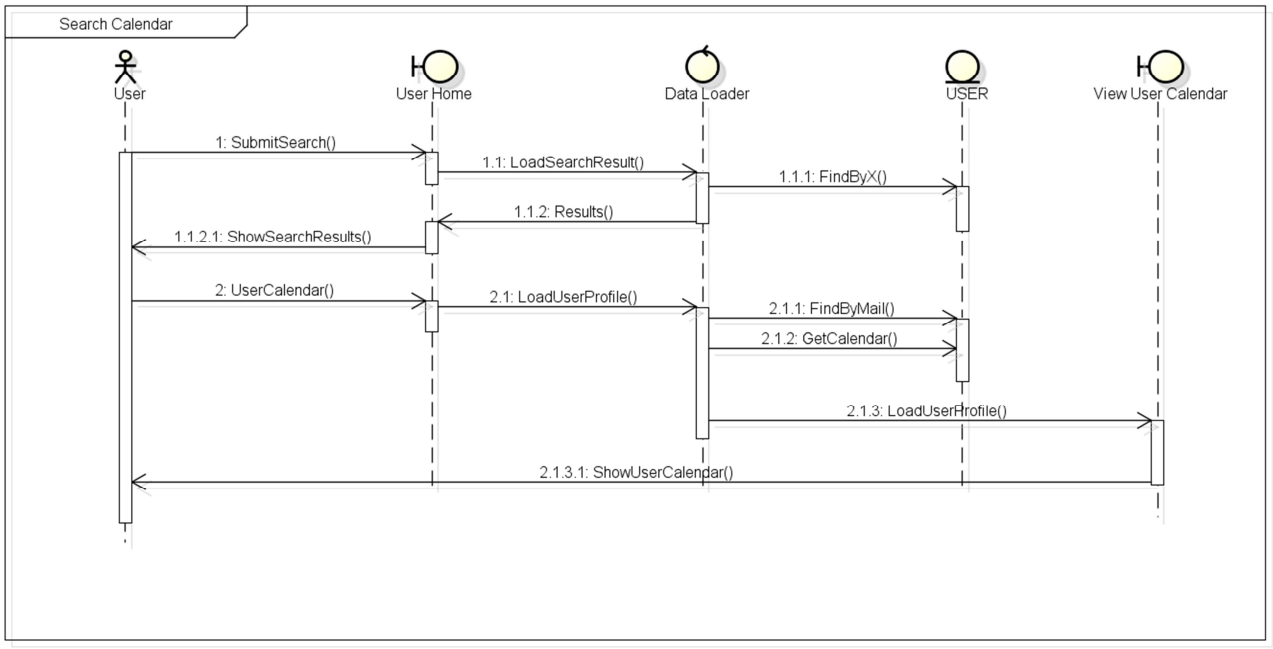
- Logs in.



5.2. Search Calendar

An user:

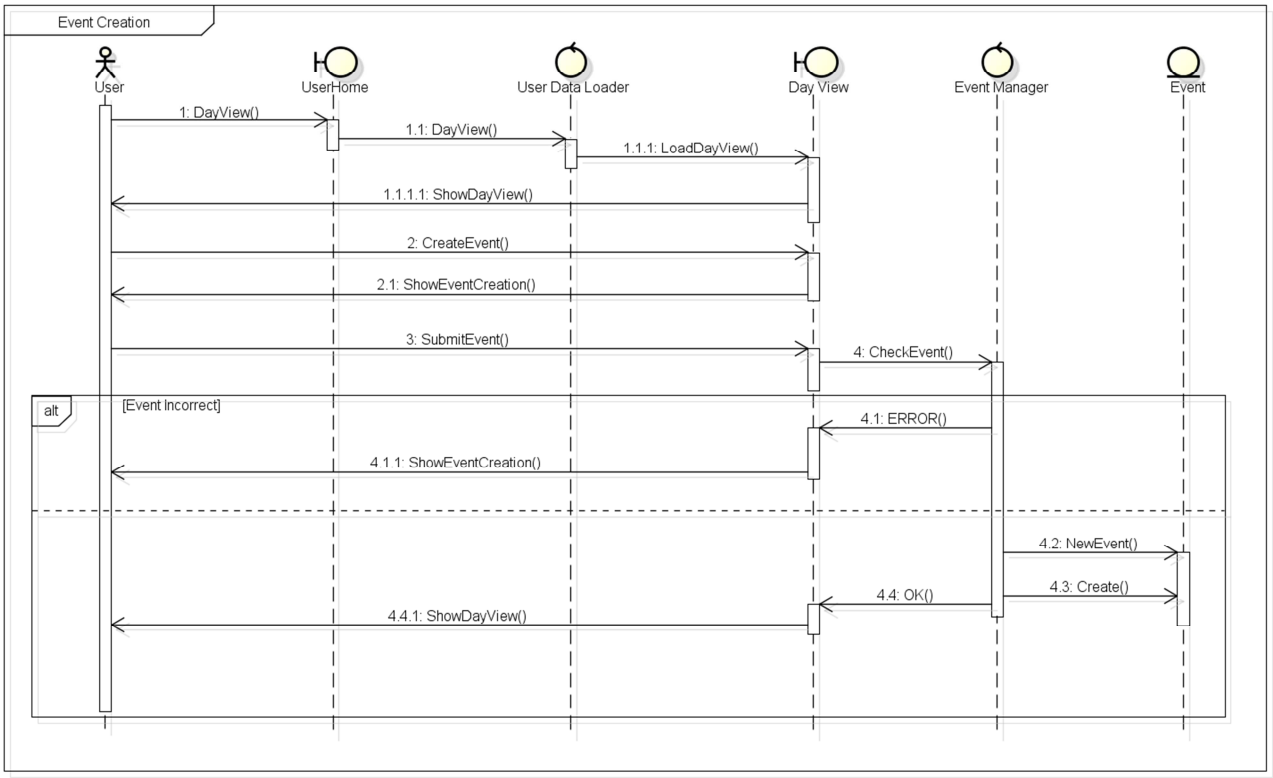
- Searches another user;
- Clicks on a specific user to see his calendar.



5.3. Event Creation

An user:

- Clicks on a specific day;
- Create an event on that day;

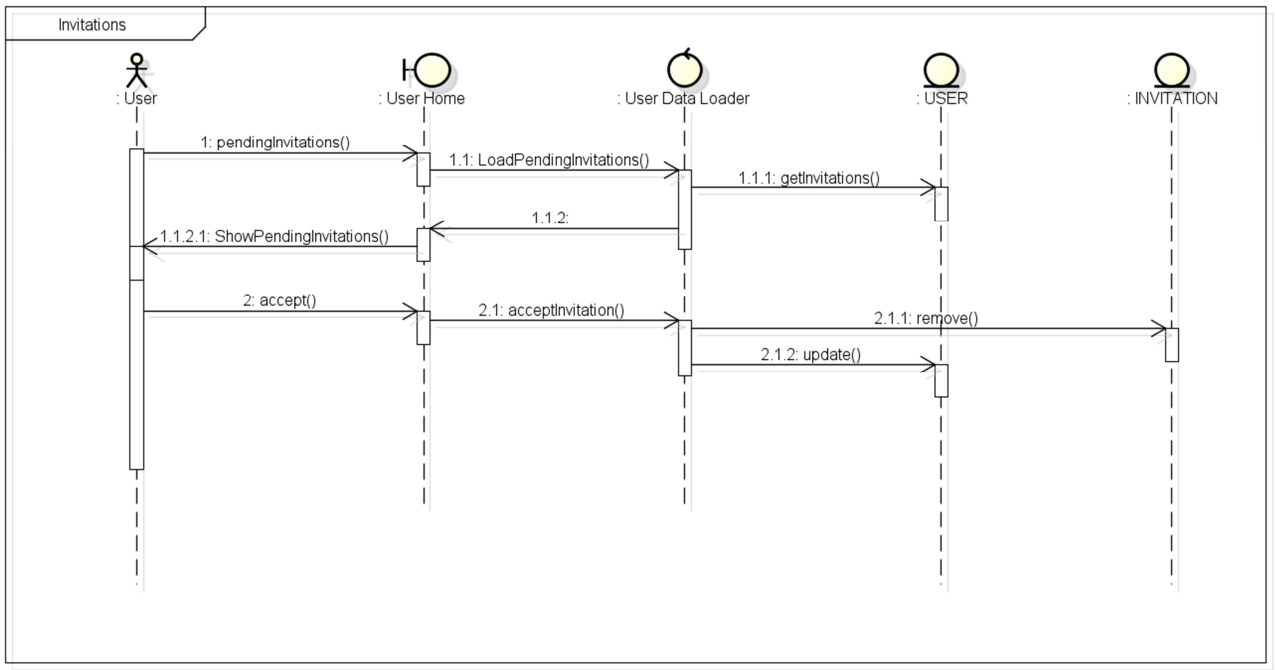


5.4. Invitations view

An user:

- Clicks on the invitation's icon;
- Accept/Decline invitations;

In the diagram hereunder the user accepts the invitation while the case of not acceptance is similar, but Decline() is called instead of Accept() on the boundary User Home.



We decided not to show the sequence diagram of notifications view because it works like the diagram of invitations.

6. USED TOOLS

- Paint: to create the E/R schema;
- Astah Professional : to create all the other diagrams;
- Microsoft Office Word 2010: to redact and to format this document;