Design Document

Edoardo Scibona
Mat. 836884

December 7, 2014

Contents

1 Introduction

1.1 Purpose

This document will illustrate the design choices made for the development of the MeteoCal system.
These design choices are influenced by the requirements and choices described in the *Requirement Analysis and Specification Document* and will touch upon subsystems, data management, user experience and system functionalities.
This document will be primarily used as a reference by the project's developers but may also be useful to the customer who commissioned the project, who may want to understand better and verify some design choices.
Researchers and future developers may also use this document to gain a better understanding of the project's design phase.

1.2 Definitions, acronyms and abbreviations

JEE Java Enterprise Edition

ER Model Entity Relationship Model

UX diagram User Experience diagram

BCE diagram Boundary Control Entity diagram

For more definitions, acronyms and abbreviations regarding this project refer to the *Requirement Analysis and Specification Document*.

2 Architecture

2.1 General architecture

As we have already specified in the constraints section of the *Requirement Analysis and Specification Document* the system must be developed on the JEE platform.
The JEE platform has a four-tiered architecture and our design choices, with the client-server structure defined for the project, should facilitate the mapping of logical aspects of the system onto the tiers provided by JEE.

2.2 Subsystems

In this section we will identify, through a top-down approach, the major subsystems that compose the MeteoCal system.
Through the description of the subsystems it will be clearer how the functionalities, required by the MeteoCal system and described in the *Requirement Analysis and Specification Document*, will map to the software modules to be realized.
A good understanding of the subsystems will also aid in realizing software modules that present high cohesion and low coupling, thus making them easier to reuse and maintain.

We have identified the following subsystems.

Sign up manager deals with the registration of new users to the service.

Login manager deals with the authentication of returning users to the service.

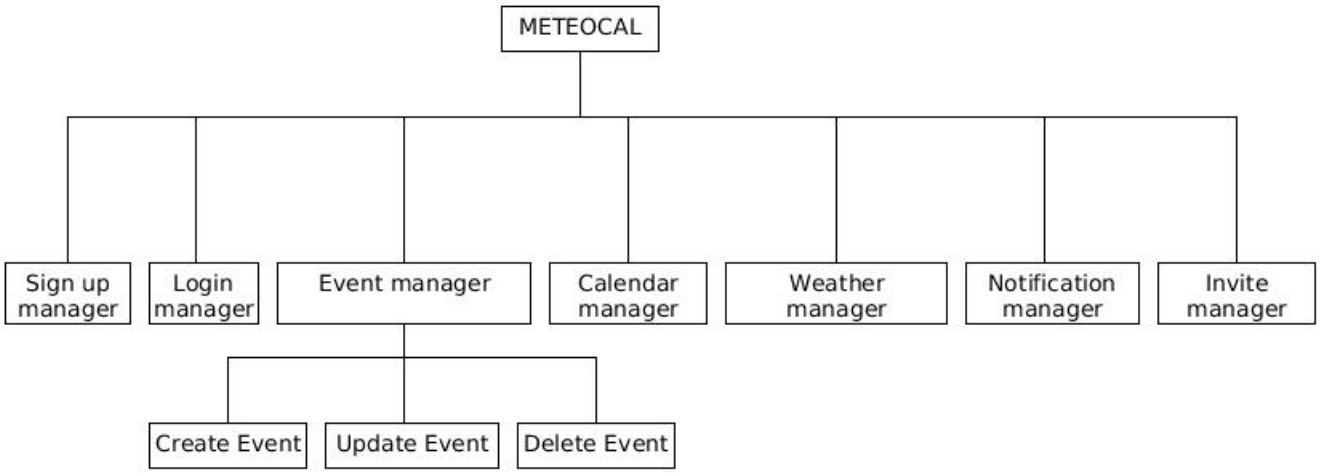
Calendar manager manages the calendars of the users, updating them and keeping them consistent.

Event manager manages the events and the operations that users can perform on events.
It is composed by some smaller subsystems such as **Create event**, **Update event** and **Delete event** that deal directly with the event's operations.

Notification manager manages the notifications shown to the users.

Invite manager manages the creation of invites and tracks users responses.

Weather manager interacts with the weather API and enriches events with weather information.

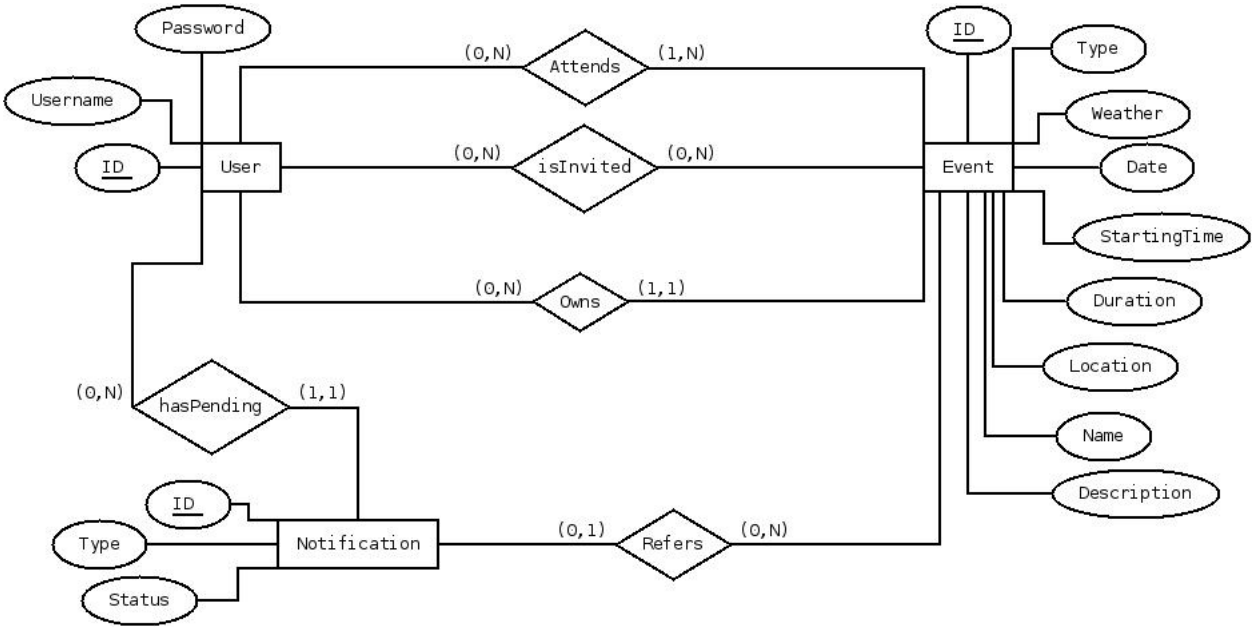


3 Persistent data management

In this section we derive from the ER model the structure that we will use in the database.

3.1 Conceptual data model

In this section we will present and analyse the ER model from which the logical model for the database will be derived.
We have chosen to simplify the ER model keeping only the meaningful and necessary entities and relationships.



We have three main entities: User, Event and Notification. These entities are at the core of our system and are linked with each others by relationships. Every entity has its own *ID* as the primary key.
The *User* entity represents users of the service with their attributes that are *Username* and *Password*.
The *Event* entity represents the events generated by users; the *Type* attribute denotes the type of the event, that can be either outdoor or indoor; the *Name*,

Description, *Location*, *Date*, *StartingTime* and *Duration* attributes are essential to define the event in a meaningful way for the users; finally the *Weather* attribute is used to store the weather for events under the outdoor type, it should be noted that the *Weather* attribute is optional as weather forecast may not be available or needed; the *Description* attribute is also optional.
The *Notification* entity represents the possible notifications generated by the system and displayed to the users; the *Type* attribute differentiates the notifications in the following categories: general, weather alerts and invites; the *Status* of the notification may be read or unread for general and weather notifications or may be accepted or declined for invites.
The *Attends*, *isInvited* and *Owns* relationships link the *User* to the *Event* and they will become tables in the logical data model.
Similarly the *Notification* entity is linked to the *User* entity by the *hasPending* relationship and to the *Event* entity with the *Refers* relationship.

3.2 Logical data model

Having described the ER model we can now distil the following database structure, where we denote tables in bold, primary keys with underlining, foreign keys with the form *TableNameID* and optional attributes with *Attribute**.
The three main entities are transformed in their own tables maintaining their original attributes, extra tables are built starting from the relationships and adding as keys the primary keys of the entities participating in the relationship.

User (ID, Username, Password)

Event (ID, Name, Type, Location, Date, StartingTime, Duration, Description*, Weather*)

Notification (ID, Type, Status)

AttendingUsers (UserID, EventID)

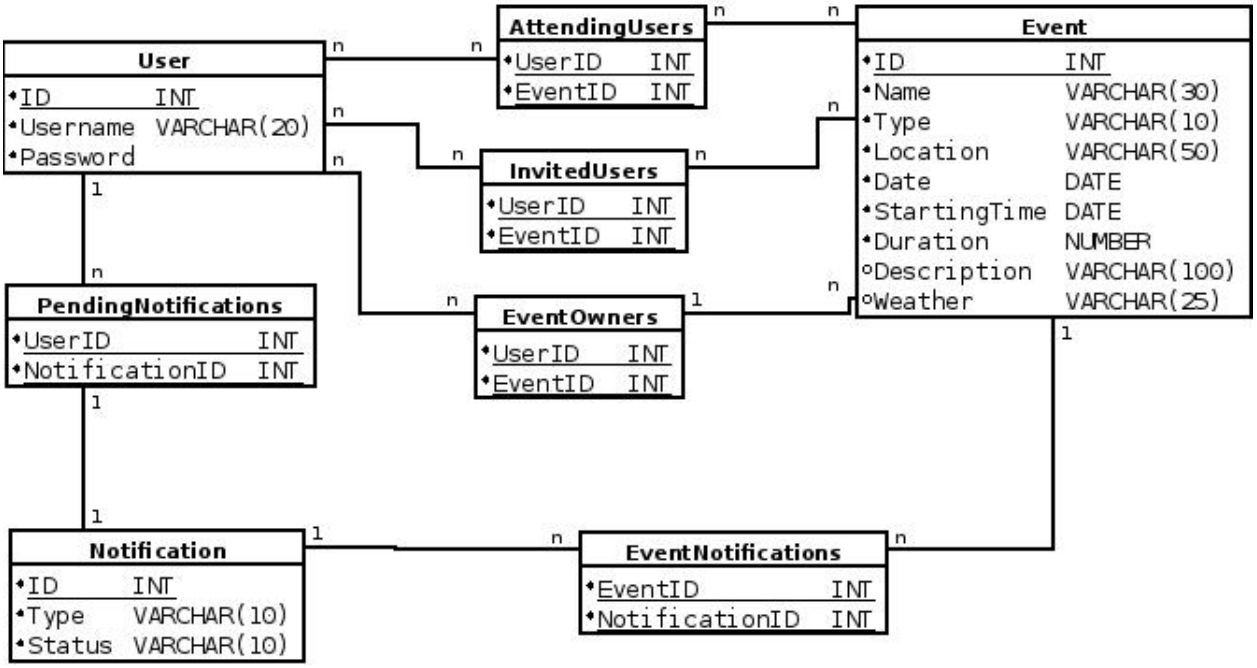
InvitedUsers (UserID, EventID)

EventOwners (UserID, EventID)

PendingNotifications (UserID, NotificationID)

EventNotifications (EventID, NotificationID)

We show here the database diagram.



4 User experience

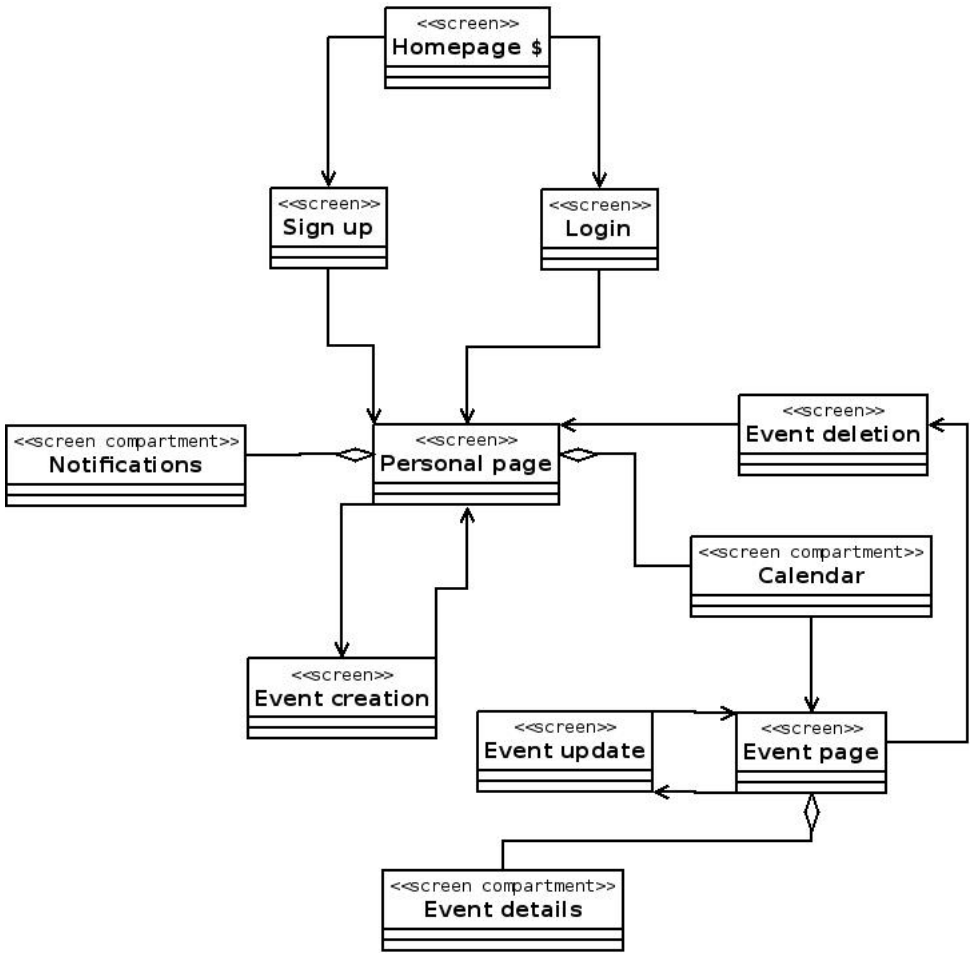
In this section we will show the UX diagrams depicting the interaction of the user with the service. There will be a UX diagram for every major functionality to improve the readability and reduce the unnecessary clutter. We will use the common notation for UX diagrams such as:

- *«screen»* stereotype to denote pages seen by the user.
- *«screen compartment»* stereotype to denote shareable parts of pages.
- *«input form»* stereotype to denote forms that the user can compile.
- *$* symbol to denote pages reachable from every other page.
- *+* symbol to denote pages, or part of pages, with multiple sub-pages.

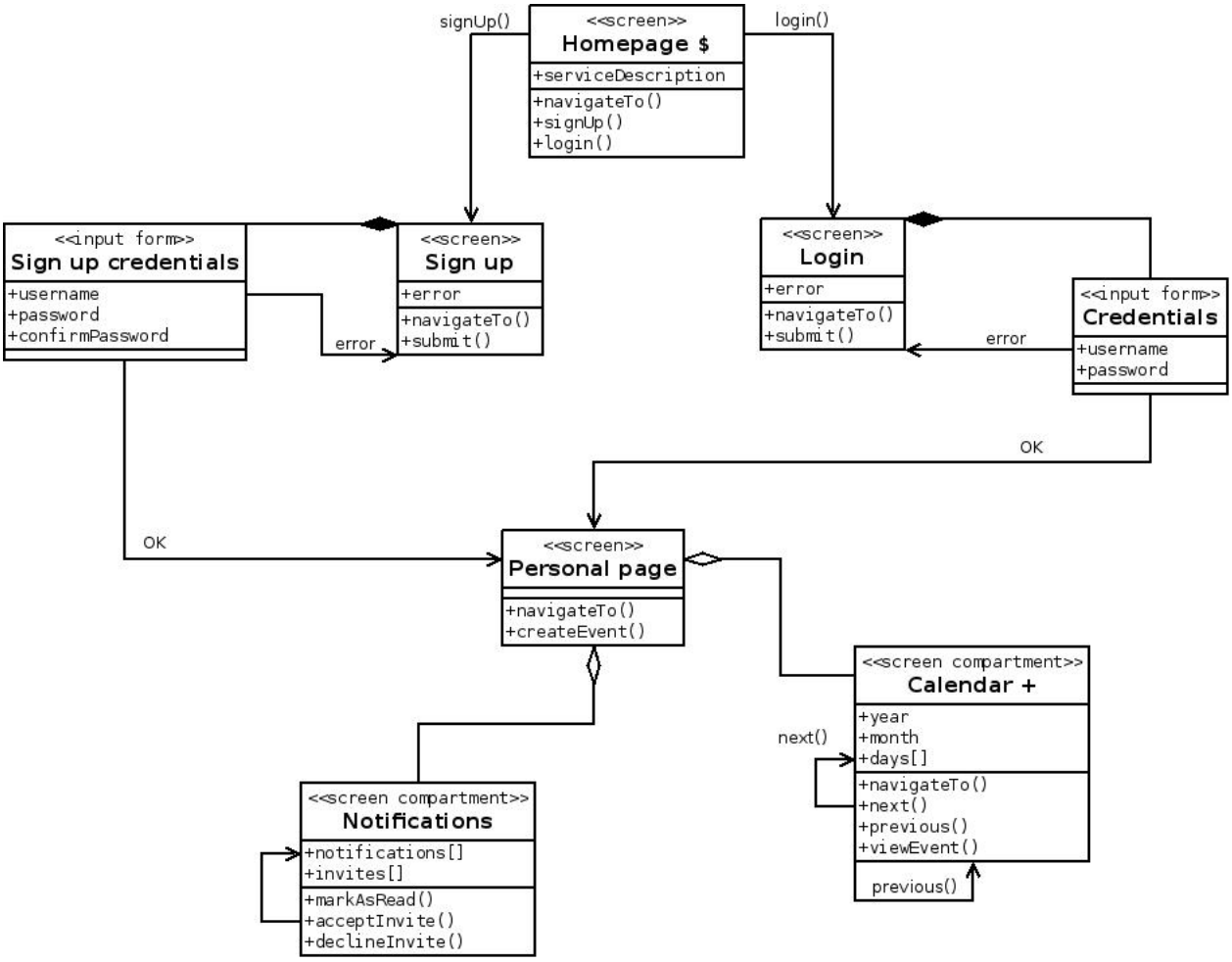
Actions, either performed manually by the user or automatically by the system, will be displayed on arcs.

4.1 Overview

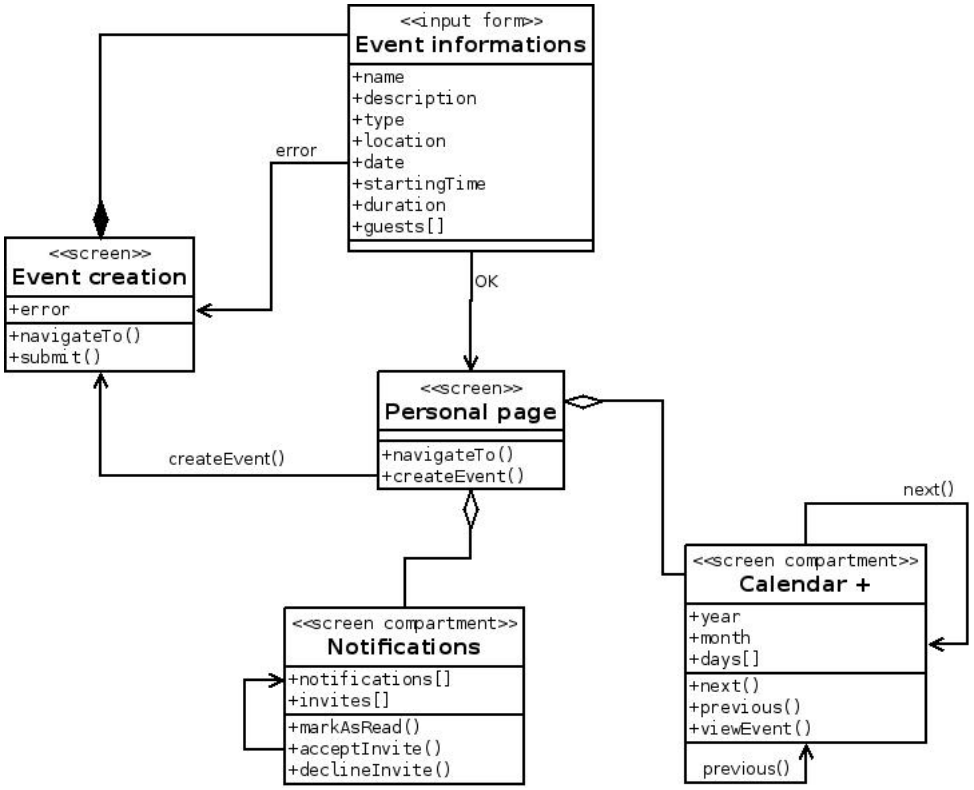
We show here a rough but useful overview of the UX diagram. All the arcs correspond to successful actions.



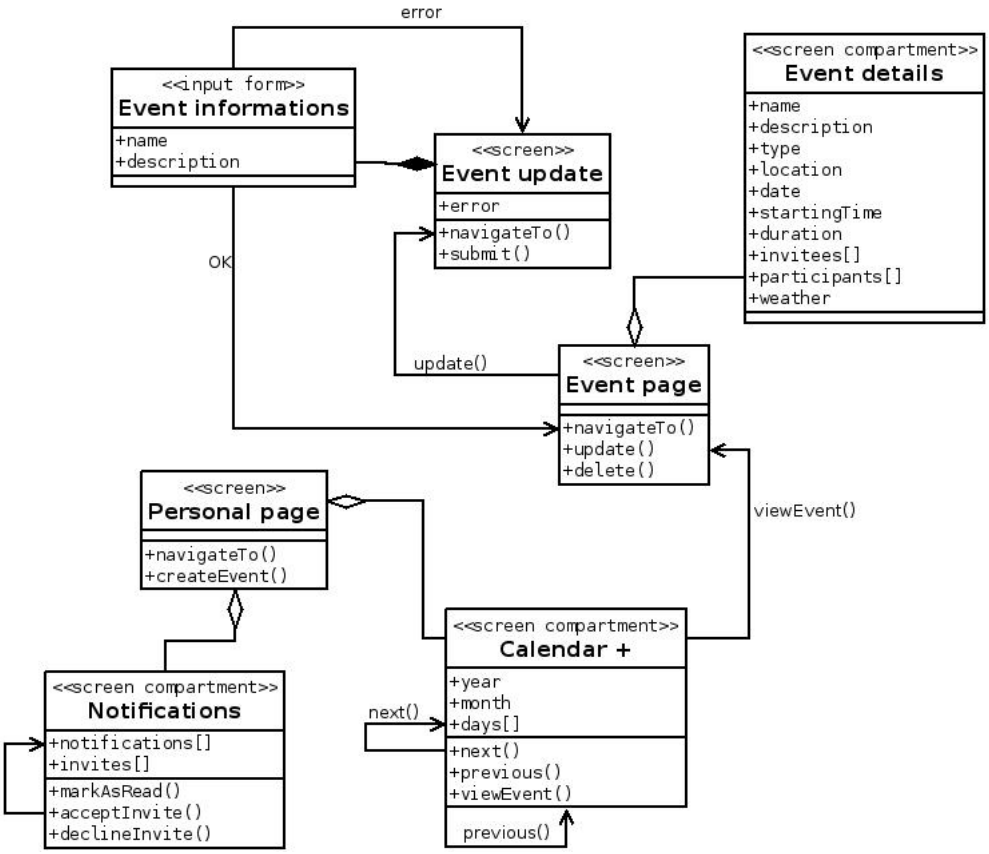
4.2 Sign up and login



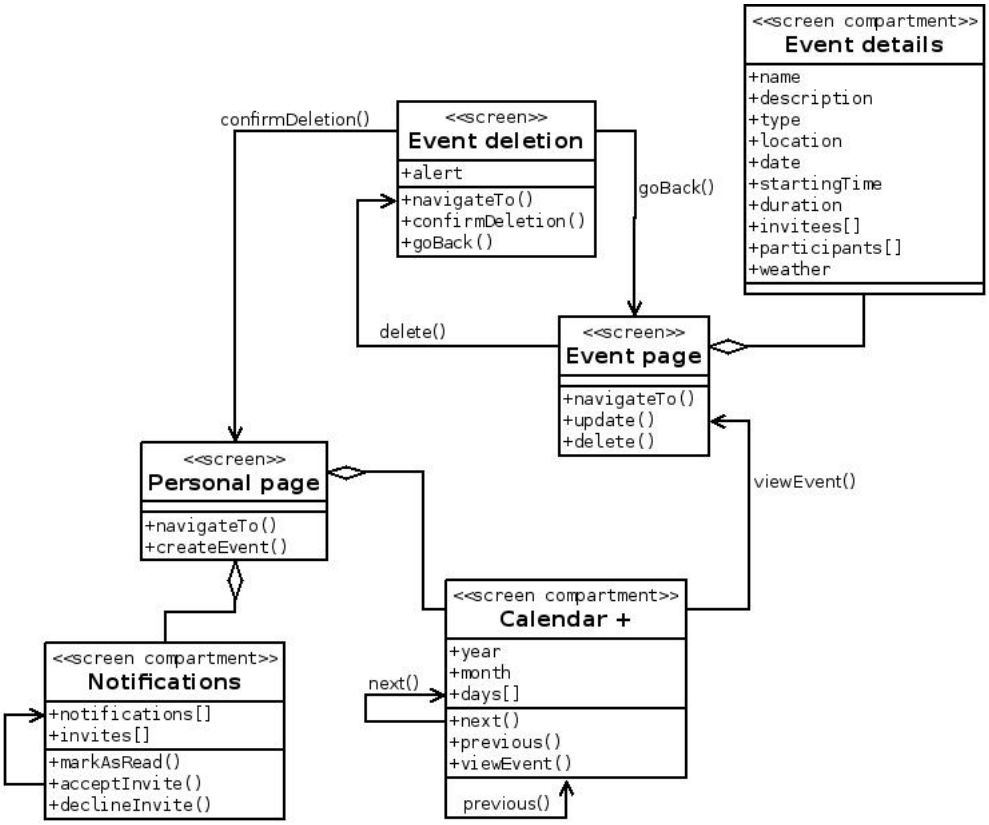
4.3 Event creation



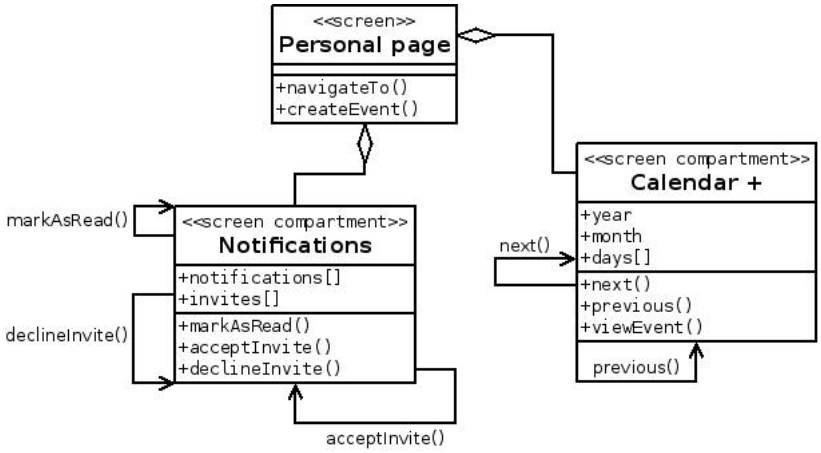
4.4 Event update



4.5 Event deletion



4.6 Notifications and invites



5 UML diagrams

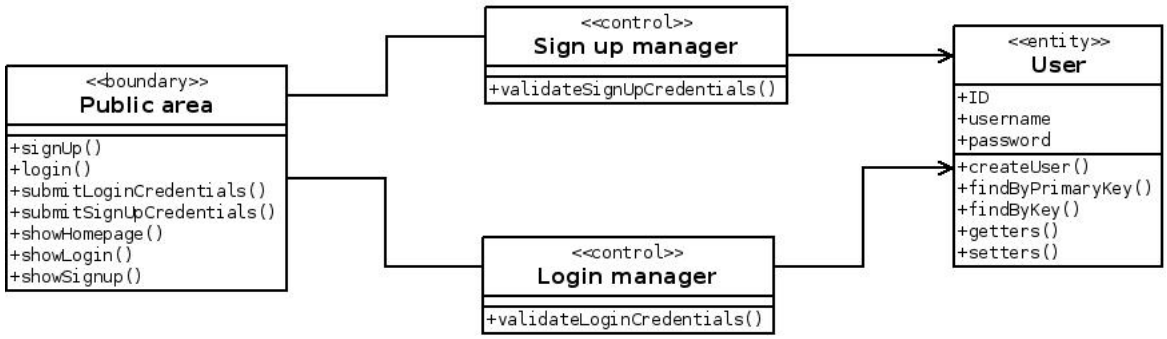
In this section we will show some additional UML diagrams which will explain better the systems interactions and architecture.

5.1 BCE diagrams

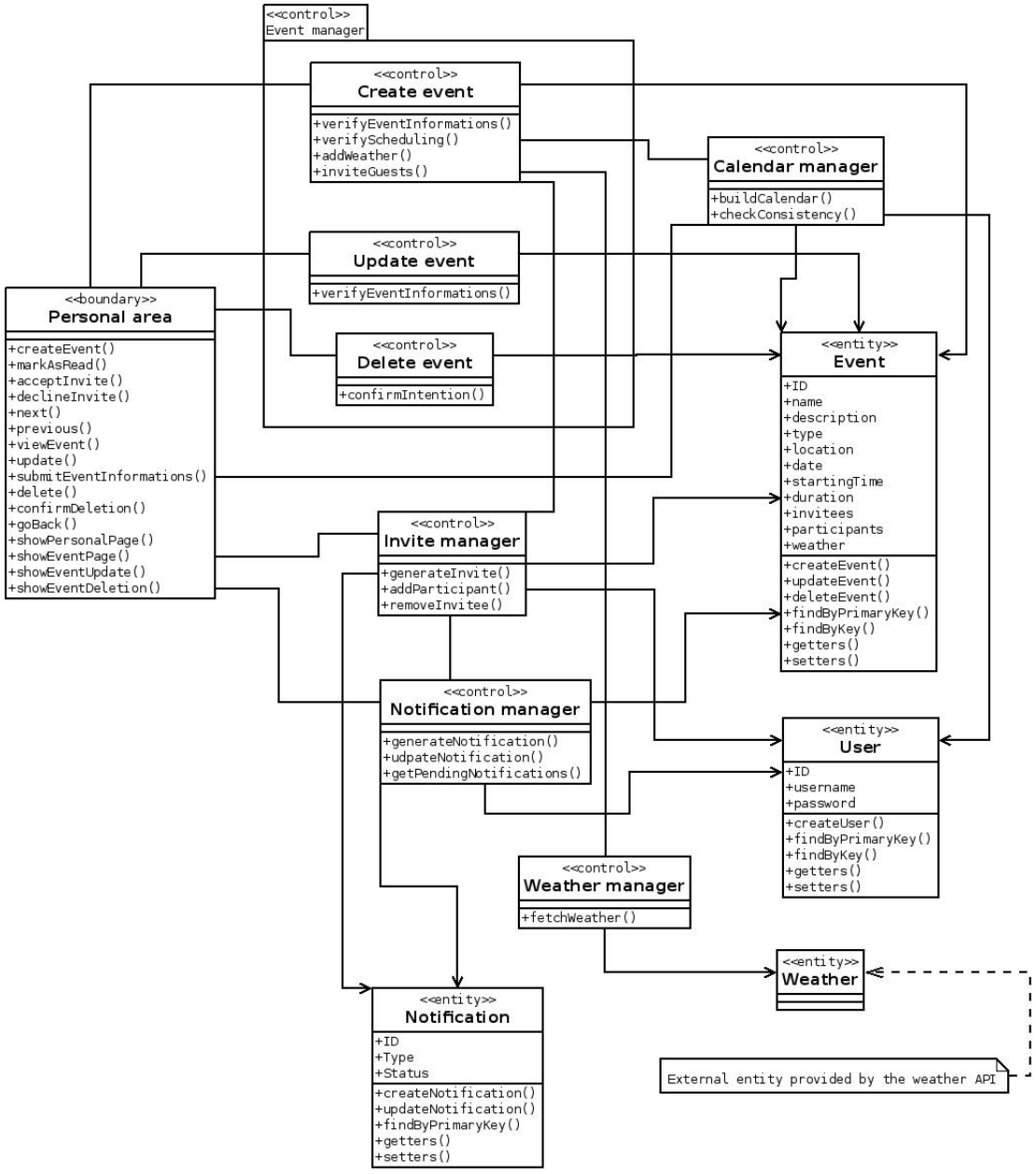
In this section we will show the BCE diagrams illustrating the major boundaries of the system to be realized and also how they interact with the data side of the system, represented by entities, through the control structures, that are mapped closely to the subsystems identified previously.
We will use the common notation for BCE diagrams such as:

- *«boundary»* stereotype to denote a boundary area containing a set of pages, screens in the UX diagrams, with common functionalities.
- *«control»* stereotype to denote the logical parts of the system in charge of accepting boundaries' inputs and operating on entities.
- *«entity»* stereotype to denote the data side of the system.

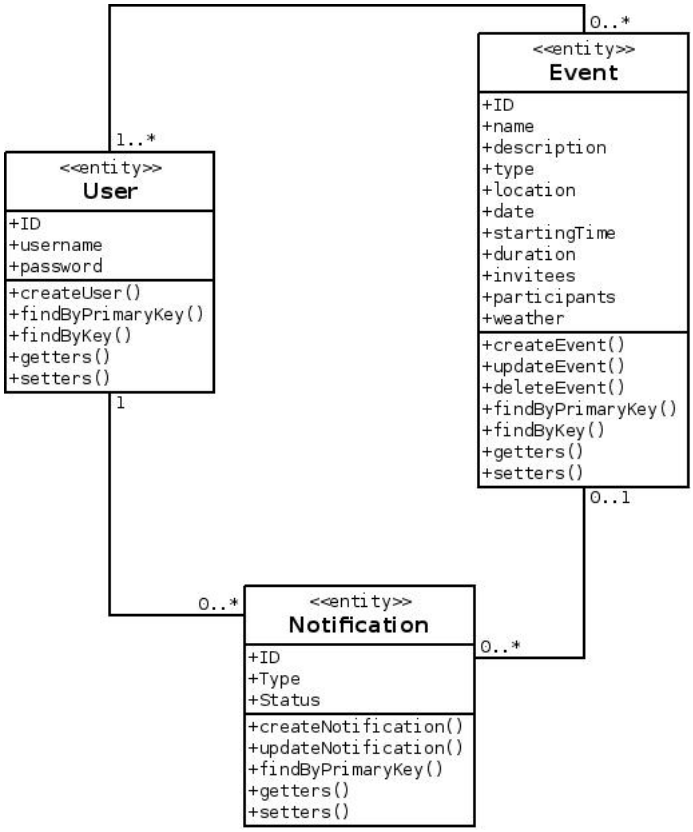
5.1.1 Public area



5.1.2 Personal area



5.1.3 Entities relationships



6 Document informations

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

6.1 Effort

As of December 7, 2014 approximately **18 hours** have been spent creating this document and refining it.

6.2 Changelog

The last time that the document was updated was December 7, 2014.

6.3 Tools used

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

UMLet to design the subsystem diagram.
http://www.umlet.com/

Dia Diagram Editor to design the ER, DB, UX and BCE diagrams.
http://dia-installer.de/