A generic Web Submission System

Bachelorarbeit
der philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Chantal Peeters
Juli 2011

Leiter der Arbeit
Prof. Dr. Oscar Nierstrasz
Jorge Ressia

Institut für Informatik und angewandte Mathematik

Abstract

Nowadays there is a myriad of submission systems for conferences, workshops, journals, academic positions, etc. Each of these systems is targeted to solve a specific problem and they are not reusable in other contexts.

We provide a unifying solution as a web system which enables submissions of files and offers a structured platform for the discussion of these submissions. The submission process can be configured through the definition of a workflow.

We developed different submission web application to validate our approach. We show how to configure our web system to use it for these different applications.

Acknowledgements

I would like to express my gratitude to everyone who supported me during the time I was working on this thesis. It would not have been possible to complete this work without them.

First of all I would like to thank my supervisor Jorge Ressia who always supported me when I had problems. He always took his time to discuss the problems and questions I had. In times when I do not progressed well, he encouraged me.

I also thank Prof. Dr. Oscar Nierstrasz for giving me the possibility to write my Bachelor thesis at the Software Composition Group and for his support.

Finally I thank my friends and especially my boy-friend for motivating and supporting me during the work on this thesis.

July 2011, Chantal Peeters

Contents

List of Figures

List of Tables

1 Introduction

Let us consider the current professor hiring process at the Institute of Computer Science and Applied Mathematics at the University of Bern. The process is as follows:

The dean, as the head of the Faculty of Natural Sciences, chooses a person to form a committee and to manage the application process for the specific job. A job advertisement is published at the website of the University and interested persons have now the option to apply by post or by email. The manager has to make all applications available to all committee members, so that they can prepare for the first meeting. At this meeting they decide who will be invited for an interview. This decision will be communicated to the applicants accordingly. After the interviews, a candidate will be selected.

Now let us consider another process, namely the process of searching papers for the online journal JOT[1]. The paper reviewing process starts as the Editor-in-Chief receives a submission by email and continues with the following steps:

- *"Papers that are clearly unacceptable will be returned by the Editor-in-Chief (EiC) without being reviewed.*
- *Papers for review will be assigned to an Associate Editor (AE). The AE may decide to reject the paper with a short review.*
- *Otherwise the AE will select three expert reviewers to carry out a detailed assessment.*
- *The AE will recommend acceptance or rejection of the paper based on the returned reviews and on the AE's own assessment. In rare cases, additional reviews may be solicited.*
- *A paper may be accepted, rejected, or provisionally accepted pending a major revision. (…)*
- *A revised paper must be accompanied by a cover letter detailing how the revised manuscript addresses the concerns raised by the reviewers.*
- *Accepted papers will be published without delay upon receipt of the final camera-ready copy"*[2]

The disadvantage of these approaches is that almost everything is done by email or post. The papers or applications are not stored in a common space and thus a general overview is missing. Consequently the involved persons have to check manually that all desired persons receive the desired papers / applications and this takes an unnecessary amount of effort.

There are already submissions system to manage paper submissions like EasyChair and CyberChair, but they are oriented towards paper conferences and the workflow is kind of fixed. Thus they are not applicable to either of the problems we just presented.

[1] *"The Journal of Object Technology (JOT) is a peer-reviewed, open-access journal dedicated to the timely publication of previously unpublished research articles, surveys, tutorials, and technical notes on all aspects of object technology." [*The Journal of Object Technology. About Jot. Mission: http://www.jot.fm/index.html (accessed on 20.07.2011)]

[2] The Journal of Object Technology. Review Process: http://www.jot.fm/authors.html (accessed on 20.07.2011).

Introduction

The aim of this bachelor project is to provide a unified solution to the different submission requirements conferences, journals, job applications, etc have. So this solution should primarily simplify the submissions and the management of files of any kind.

2 Basic Idea of the System

Our solution is a framework for a web system. The system manages submission events (hereinafter referred to as “events”). An event determines a topic to which a user can submit files, for example a job advertisement, by its name. Furthermore it contains a description of the topic and a grading scale for the submissions. An event represents the whole submission process using a workflow and several roles. Such events are created by the system administrator who also defines the manager. The manager is responsible to provide the event with the necessary information like the description, the grading, the roles and a workflow.

A role is a collection of authorizations. It determines what actions a user can perform in his/her current role and which not. In the system itself, there are two roles, the administrator role and a default role. The administrator role is assigned only to the administrator; all other users will receive the default role. The only noticeable difference is that an administrator is entitled to create new events and the other users not.

Each event has by default a visitor, a manager and an author role. More roles can be defined. The defined manager obtains the manager role. The visitor role is allocated to all users. Each visitor who submits a document gains the author role automatically. Additionally the roles can be assigned to users individually by the administrator and the manager. Note that a user can have more roles but always only one current role.

The workflow models the time flow of the submission process of an event. It consists of workflow phases that in each case define a time period. For any period is defined how many files a user may submit and which roles allow a user to create related submissions.

Files, for example an application, will be submitted separately as submissions that include a title and a short description of the file. These submissions can be viewed by authorized users and provide a discussions basis. Authorized users can submit files which relate to one of these initial submissions as related submissions. Note that submissions can only refer to these initial submissions. Suppose the system is used for paper submission and a paper is submitted as initial submission by an author. If another user submits a review of the paper, the author can answer by submitting his/her answer as a related submission of his/her paper.

3 Use Cases and implemented Functionalities

The implemented functionality of the system can be seen from the following list of use cases.

1. Register
2. Log in
3. Forgotten password
4. Log out
5. Change password
6. Change email address
7. Change profile
8. Change role
9. Delete account
10. User overview
11. See overview of all events
12. See overview of events where the user has a role except the visitor role
13. See event description
14. Create a new event with a registered user as the manager
15. Create new event with a not registered person as the manager
16. Describe an event
17. Define grading scale for event
18. Editing workflow: add new workflow item
19. Editing workflow: prepare existing workflow item
20. Delete event
21. Create new role for an event
22. Add permission to role
23. Delete permission from role
24. Add role to a registered user
25. Add role to a not registered person
26. See overview of event members
27. Send emails
28. Create a submission
29. See own submissions of active role
30. Prepare own submission
31. Delete own submission
32. See overview of all unrelated submissions
33. See foreign profile
34. Get PDF from foreign profile
35. See foreign submission
36. See overview of all submissions related to a specific submission
37. Create related submission

The use cases are explained in more detail below.

Note:

In the preconditions is sometimes mentioned “active event is set”, “active submission is set” or “active user for profile is set”. The “active event” will be set by selecting an event on the website events.php or default_mainpage.php. The “active submission” is set, when a link of one submission from one of the sites my_submissions.php, submissions.php or related_submissions.php is followed. And finally, the “active user for profile” is set by clicking on a link to a profile on one of the websites event_members.php, submissions.php or user_overview.php.

All the initial websites of the use cases 1 – 29 and 32 can be accessed through the navigation of the web page. Note that the preconditions of a use case must be fulfilled and only these pages are displayed in the navigation whose main action may be performed by the user in its active role. For the other use cases look at the explanation at their detailed description.

If a web page is accessed directly from the URL, it may be that the user is redirected to another page or when the user attempts to perform an action, he/she receives an error message. The system directs the user if not all the preconditions are met. If the user has no authorization in its active role for the desired action, an error message is displayed. This case is not treated in the alternative flow, because it presupposes an abuse of the system.

1. Register

Actors:	User
Preconditions:	-
Postconditions:	User is registered.
Initial page:	register.php
Normal flow:	1. User enters a user name. 2. User enters his/her email address. 3. User enters his/her password. 4. User confirms his/her password. 5. User clicks on the “create account” button. 6. System sends a confirmation with a link by email to the User and displays the website registration_success.php.
Alternative flow:	5a. Registration fails. 1. System displays an error message. 2. User can fill in the form again.

2. Log in

Actors:	User
Preconditions:	User is registered.
Postconditions:	User is logged in. Active role of user is set to the default role or to the administrator role if the user is the administrator.
Initial page:	login.php
Normal flow:	1. User enters his/her user name into the form. 2. User enters his/her password. 3. User clicks the “log in” button. 4. System displays the website default_mainpage.php.

Alternative flow:	1a. Login fails. 1. System displays an error message. 2. User can fill in the form again.

3. Forgotten password

Actors:	User
Preconditions:	-
Postconditions:	User has a new password.
Initial page:	forgotten_password.php
Normal flow:	1. User enters his/her user name in the appropriate form. 2. User clicks the “send email” button. 3. System sends an email with a new password to the user and displays a confirmation.
Alternative flow:	2a. Typed user name does not exist. 1. System displays an error message. 2. User can enter his/her user name again.

4. Log out

Actors:	User
Preconditions:	-
Postconditions:	User is logged out.
Initial page:	logout.php
Normal flow:	1. System displays a login page.
Alternative flow:	

5. Change password

Actors:	User
Preconditions:	User is logged in.
Postconditions:	User has a new password.
Initial page:	change_user_properties.php
Normal flow:	1. User enters a new password. 2. User confirms his/her new password. 3. User clicks the button “change password”. 4. System displays a confirmation.
Alternative flow:	3a. Changing password fails. 1. System displays an error message. 2. User can fill in the form again.

6. Change email address

Actors:	User
Preconditions:	User is logged in.
Postconditions:	User has a new email address.
Initial page:	change_user_properties.php
Normal flow:	1. User enters a new email address. 2. User clicks the button “change email”. 3. System sends a code by email to the new email address and displays the website confirm_email.php.

	4. User enters the code. 5. User clicks the button “confirm”. 6. System displays the website change_user_properties.php with the new email address.
Alternative flow:	2a. Changing email address fails. 1. System displays an error message. 2. User can enter a new email address again.

7. Change profile

Actors:	User
Preconditions:	User is logged in.
Postconditions:	Profile is changed.
Initial page:	profile.php
Normal flow:	1. User fills in the form. 2. User clicks the “save changes” button. 3. System loads the profile page again.
Alternative flow:	2a. Changing fails. 1. System displays an error message. 2. User can fill in the form again.

8. Change role

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Active role of user is changed.
Initial page:	change_role.php
Normal flow:	1. User selects a role. 2. User clicks the “change role” button. 3. System displays actual version of page.
Alternative flow:	

9. Delete account

Actors:	User
Preconditions:	User is logged in.
Postconditions:	User’s account with all his/her submissions and related submissions are deleted.
Initial page:	delete_account.php
Normal flow:	1. User clicks the “yes” button. 2. System displays website index.php.
Alternative flow:	3a. Deleting fails. 1. System displays an error message.

10. User overview

Actors:	User
Preconditions:	User is logged in.
Postconditions:	-
Initial page:	user_overview.php

Normal flow:	1. System shows a list with all users. For each user it displays his/her user name, first name, name and a link to his/her profile.
Alternative flow:	

11. See overview of all events

Actors:	User
Preconditions:	
Postconditions:	
Initial page:	events.php
Normal flow:	1. System shows a list with all events. For each event there exists a link.
Alternative flow:	

12. See overview of events where the user has a role except the visitor role

Actors:	User
Preconditions:	User is logged in.
Postconditions:	
Initial page:	default_mainpage.php
Normal flow:	1. System displays a list with all events where the user has a role except the visitor role. For each event exists a link and the roles of the uses are shown.
Alternative flow:	1a. No such events exist. 1. System displays a message.

13. See event description

Actors:	-
Preconditions:	Active event is set.
Postconditions:	
Initial page:	event_info.php
Normal flow:	1. System displays the description of the active event.
Alternative flow:	

14. Create a new event with a registered user as the manager

Actors:	User
Preconditions:	User is logged in.
Postconditions:	New event is created.
Initial page:	create_event.php
Normal flow:	1. User enters a name for the event. 2. User selects a registered user to be the manager. 3. User clicks the “create event” button. 4. System displays a confirmation.
Alternative flow:	3a. Creating fails. 1. System displays an error message. 2. User can repeat the procedure.

15. Create new event with a not registered person as the manager

Actors:	User
Preconditions:	User is logged in.
Postconditions:	New event is created. New user is registered.
Initial page:	create_event.php
Normal flow:	1. User enters a name for the event. 2. User selects a not-registered person to be the manager and enters a user name and the email address of this person. 3. User clicks the “create event” button. 4. System sends an invitation email to the entered email address, registers the user name and email address. 5. System displays a confirmation.
Alternative flow:	3a. Creating fails. 1. System displays an error message. 2. User can repeat the procedure.

16. Describe an event

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Event description is updated.
Initial page:	event_description.php
Normal flow:	1. User enters a description. 2. User clicks the “set description” button. 3. System displays a confirmation.
Alternative flow:	2a. Updating fails. 1. System displays an error message. 2. User can repeat the procedure.

17. Define grading scale for event

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Grading for event is updated.
Initial page:	event_grading.php
Normal flow:	1. User enters grades and their acceptations. 2. User clicks the “define grading” button. 3. System displays a confirmation.
Alternative flow:	2a. Updating fails. 1. System displays an error message. 2. User can repeat the procedure.

18. Editing workflow: add new workflow item

Actors:	User
Preconditions:	User is logged in. Active event is set.

Postconditions:	New workflow item is added to event.
Initial page:	event_workflow.php
Normal flow:	1. User enters a name for the new workflow item. 2. User enters a start date. 3. User enters an end date. 4. User clicks the “add as new item” button. 5. System displays a confirmation.
Alternative flow:	4a. Adding fails. 1. System displays an error message. 2. User can repeat the procedure.

19. Editing workflow: prepare existing workflow item

Actors:	User
Preconditions:	User is logged in. Active event is set. At least one workflow item exist.
Postconditions:	Workflow item is updated.
Initial page:	event_workflow.php
Normal flow:	1. User selects a workflow item. 2. User clicks the “choose” button. 3. User edits the settings of the selected workflow item. 4. User clicks the “change item” button.
Alternative flow:	4a. Updating fails. 1. System displays an error message. 2. User can repeat the procedure.

20. Delete event

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Active event and all submission related to this event are deleted.
Initial page:	event_delete.php
Normal flow:	1. User clicks the “yes, delete event” button.
Alternative flow:	

21. Create new role for an event

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	New role is added to active event.
Initial page:	new_role.php
Normal flow:	1. User enters a role name. 2. User clicks the “add new role” button. 3. System displays a confirmation.
Alternative flow:	2a. Role name already exists. 1. System displays an error message. 2. User can repeat the procedure with another name.

22. Add permission to role

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Permission is added to selected role.
Initial page:	permissions.php
Normal flow:	1. User selects a role. 2. User selects a permission. 3. User clicks the “add permission” button. 4. System reloads the website and displays the actual permissions.
Alternative flow:	3a. Adding fails. 1. System displays an error message. 2. User can repeat the procedure.

23. Delete permission from role

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Permission of selected role is deleted.
Initial page:	permissions.php
Normal flow:	1. User selects a role. 2. User selects a permission. 3. User clicks the “delete permission” button. 4. System reloads the website and displays the actual permissions.
Alternative flow:	3a. Deleting fails. 1. System displays an error message. 2. User can repeat the procedure.

24. Add role to a registered user

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Role is added to the selected user for the active event.
Initial page:	define_event_role_distributions.php
Normal flow:	1. User selects a role. 2. User selects a registered user. 3. User clicks the “add role to user” button. 4. System displays a confirmation.
Alternative flow:	3a. Adding fails. 1. System displays an error message. 2. User can repeat the procedure.

25. Add role to a not registered person

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	New user is registered. Role is added to the new user for the active event.
Initial page:	define_event_role_distributions.php
Normal flow:	1. User selects a role. 2. User selects a not registered person and enters a user name and the email address of this person. 3. User clicks the “add role to user” button. 4. System sends an invitation email to the entered email address and registers the user name and email address. 5. System displays a confirmation.
Alternative flow:	3a. Adding fails. 1. System displays an error message. 2. User can repeat the procedure. 4a. Registration fails. 1. System displays an error message. 2. User can repeat the procedure.

26. See overview of event members

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	
Initial page:	event_members.php
Normal flow:	1. User selects a role. 2. User clicks the “show users” button. 3. System shows a list with the user name, first name, names and email address of each registered user. Furthermore it displays a link to the user’s profile.
Alternative flow:	

27. Send emails

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	Email is send.
Initial page:	event_mail.php
Normal flow:	1. User selects a role 2. User enters a subject 3. User types a message. 4. User clicks the “send email” button. 5. System displays a confirmation message.
Alternative flow:	4a. Sending email fails. 1. System displays an error message. 2. User can repeat the procedure.

28. Create a submission

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	New submission is created. Active submission is set.
Initial page:	new_submission.php
Normal flow:	1. User enters a title. 2. User enters other authors. 3. User enters an abstract. 4. User browses a file. 5. User clicks the “submit” button. 6. System displays the website my_submission.php.
Alternative flow:	5a. “submit” button does not exist because the submission is closed. 5b. Submitting fails. 1. System displays an error message. 2. User can repeat the procedure.

29. See own submissions of active role

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	
Initial page:	my_submissions.php
Normal flow:	1. System shows a list with all unrelated submissions the user has made. For each submission it displays the title, creation date and a link to the submission, to the file and, if the user is entitled, to the related submissions. System shows a second list with all submissions the user has made a related submission to. For each submission it displays the title, the name of the submitter, the creation date and a link to the related submission and the file.
Alternative flow:	1a. No such submissions exist. 1. System displays a message.

30. Prepare own submission

Actors:	User
Preconditions:	User is logged in. Active event is set. Active submission is set.
Postconditions:	Submission is updated.
Initial page:	my_submission.php *

Normal flow:	1. User prepares the authors. 2. User prepares the abstract. 3. User browses another file (perhaps). 4. User changes the grade. 5. User clicks the “submit” button. 6. System displays a confirmation.
Alternative flow:	4a. Grades are not shown because the submission is not related to another submission or it is related to a submission of the user. 5a. “Submit” button does not exist because the submission is closed. 5b. Updating fails. 1. System displays an error message. 2. User can repeat the procedure.

* The web page my_submission.php can be reached via the pages my_submissions.php and submissions.php.

31. Delete own submission

Actors:	User
Preconditions:	User is logged in. Active event is set. Active submission is set.
Postconditions:	Submission is deleted.
Initial page:	my_submission.php *
Normal flow:	1. User clicks the “delete submission” button. 2. System displays the web page “my_submissions.php”.
Alternative flow:	1a. “Delete submission” button does not exist because the submission is closed. 1b. Deleting fails. 1. System displays an error message.

* The web page my_submission.php can be reached via the pages my_submissions.php and submissions.php.

32. See overview of all unrelated submissions

Actors:	User
Preconditions:	User is logged in. Active event is set.
Postconditions:	
Initial page:	submissions.php
Normal flow:	1. System shows a list with all submissions. For each submission, it displays the title, user name, first name, last name, creation date and links to the file, submission and profile of the submitter and a link to an overview of related submissions.
Alternative flow:	

33. See foreign profile

Actors:	User
Preconditions:	User is logged in. Active user for profile is set.
Postconditions:	
Initial page:	profile_view.php *
Normal flow:	1. System displays the profile details of the active profile.
Alternative flow:	

* The web page profile_view.php can be reached via the pages event_members.php, submissions.php and user_overview.php.

34. Get PDF from foreign profile

Actors:	User
Preconditions:	User is logged in. Active user for profile is set.
Postconditions:	
Initial page:	profile_pdf.php *
Normal flow:	1. System creates and opens a PDF file with the profile details of the active profile.
Alternative flow:	

* The web page profile_pdf.php can be reached via the pages event_members.php, submissions.php and user_overview.php.

35. See foreign submission

Actors:	User
Preconditions:	User is logged in. Active event is set. Active submission is set.
Postconditions:	
Initial page:	submission_view.php *
Normal flow:	1. System displays information about the submission and a link to the file.
Alternative flow:	

* The web page submission_view.php can be reached via the pages related_submissions.php and submissions.php.

36. See overview of all submissions related to a specific submission

Actors:	User
Preconditions:	User is logged in. Active event is set. Active submission title is set.
Postconditions:	Submission is updated.
Initial page:	related_submissions.php *

Normal flow:	1. System shows a list with all submissions which are related to the active submission. For each submission it displays the name of the submitter, the grade, the creation date, the name of the workflow item it is created in and a link to it.
Alternative flow:	

* The web page related_submissions.php can be reached via the pages my_submissions.php and submissions.php.

37. Create related submission

Actors:	User
Preconditions:	User is logged in. Active event is set. Active submission is set.
Postconditions:	New related submission is created.
Initial page:	new_related_submission.php *
Normal flow:	1. User enters a title. 2. User enters other authors. 3. User enters an abstract. 4. User browses a file. 5. User grades the related submission. 6. User clicks the “submit” button. 7. System displays the website my_submission.php.
Alternative flow:	5a. Grades are not shown because the submission is related to a submission from the user himself/herself. 6a. “submit” button does not exist because the submission is closed. 6b. Submitting fails. 1. System displays an error message. 2. User can repeat the procedure.

* The web page new_related_submissions.php can be reached via the pages related_submission_view.php, related_submissions.php and submission_view.php.

4 Programming language

The most part of the submission managing system is written in PHP. For example, all classes contain entirely PHP code. In addition to PHP, the web pages consist of HTML and JavaScript code. However, JavaScript is only used for generating dialog boxes with little messages.

4.1 Reasons for PHP

PHP (originally called “Personal Home Page”) is primarily designed for web development, especially to create dynamic and interactive web pages. So PHP can be used to handle user input, for example from forms.

PHP can be easily embedded in HTML. This allows us to design our websites just as usual.

Because PHP is object oriented, we can take full advantage of this concept in order to implement the model.

Another good feature form PHP is the convenient database support. The language offers commands, which allow access to a MySQL database server. You have to know only a few of these functions and you can use the SQL queries directly.

Last but not least, PHP is distributed worldwide; it is used for about 75% of all websites[3] as a server-side script language. So most programmers know PHP and can directly go through the code, to understand the structure and make changes. For programmers who do not know PHP, it should not be difficult to learn the language, because the syntax of PHP is like in Java, C or PERL. And for people who have never programmed before, there exist a lot of good tutorials in the worldwide web.

[3] Wikipedia. PHP: http://de.wikipedia.org/wiki/PHP (accessed on 04.07.2011).

5 Structure of the System

The system is divided into three parts: The view part as the user interface, which contains all web pages, the model and the database, which stores the current state of the system.

This approach has the advantage that we can exchange the view part, without altering the model. Moreover, we may replace the database with any desired and therefore, we need only to implement a few interfaces. For details, look at paragraph 5.4 Interaction between Model and Database.

Additionally, there is a test part, which contains tests about the model.

5.1 Model: Domain Concepts

Basically the model (see Figure 2: UML of Mode) consists of Users, Events and Submissions, which must be managed.

A User is identified by its user name and holds an email address, a password, a profile and an active role. Furthermore, it has an activation code, which is used to activate the account, and knows if its account is active or not.

A Profile is an object that contains detailed information such as the gender, the full name and details of the address about a user. Look at the UML of Figure 1 for the other information a Profile holds. Providing this information is voluntary, therefore, missing data is filled with an empty string.

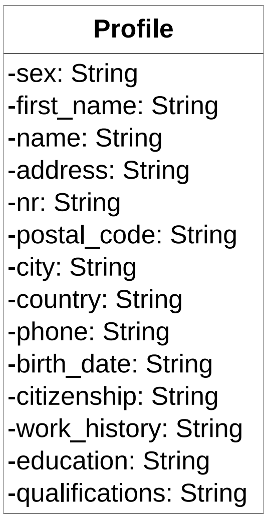

Figure 1: UML of class Profile

Updating the Profile is the responsibility of the ProfileUpdater.

A Role contains only a name and an array of permissions. That means the actions a user is allowed to execute are defined by its active role. Initially, a user has no role. A user's role is set only by the RoleGateKeeper, more on the RoleGateKeeper will follow later.

The object called UserRegistrar is responsible for the registration of a user, for changing user data and keeps track of all registered users. However, the UserRegistrar itself stores no list of all Users objects, but it can get a list from the database. To register a user a user name, an email address and a password are needed. The UserRegistrar must ensure that this user name is not already in use, generate an activation code and create a new User object. Note that the secret password of the user will not be stored in this

User object but the encrypted password. The UserRegistrar encodes the password using the MD5 one-way process to increase the security of the system and to ensure the personal data privacy. Finally, the UserRegistrar orders to send an email to the user that includes an activation link containing the user name and the generated activation code. This email is sent by an object called Mailer, which is responsible for sending emails from a system mail address. The UserRegistrar also offers the option to register another mail address and invite the owner. For this newly created user a random password is generated. The UserRegistrar is also able to delete a user.

Immediately after registering, a user cannot log into the system; he/she has to activate his/her account first. The object called UserActivator is responsible for activating user accounts. As said, after the registration, a user receives an email with an activation link. When the user follows this link, the UserActivator examines the link and depending on the result, it activates the account or not.

The object called UserLogin is responsible for the login, which means that it ensures that only users with a correct user name and password and an active account can log in to the system and it logs the legitimate one.

An Event models a topic to which a user can submit files and the whole submission process for this topic. An Event is identified by its name, has a description, defines the grading for the submission and knows the user name of its manager. A manager is a normal user, but has a specific role for the event, the manager role. Each Event has its own roles, at least three; a manager role, owned at least by the manager, a visitor role that everyone has and an author role, which no one has immediately after the initialization. The role distribution is stored in the Event.

The object which is responsible for adding and deleting permissions to roles is called RoleGateKeeper. It also keeps the name of the system administrator, an administrator role, a default role and default roles to initialize an event. The administrator, in the administrator role, owns all rights except the right to delete his/her own account. If a user, except the administrator, has selected no event, the default role is set to its active role. A user can change its active role for an event using the RoleGateKeeper. This object also verifies that a user has really the right to its active role.

In addition, an Event has a workflow, an array, which contains the names of the WorkflowItem objects in chronological order. A WorkflowItem specifies a time period with a start and an end date and indicates how many submissions a user can upload in this period and which roles may submit.

Regulations and modifications of the workflow are managed by the WorkflowManager. For example, it ensures that the WorkflowItems are arranged chronologically in the workflow and they do not overlap in time. It keeps track of all WorkflowItems, but it has no stored list of all these Items. Like the UserRegistrar, the WorkflowManager may generate all existing workflow items using the database.

The events are managed by the EventManager, which is responsible for creating a new event. A new event should not have an already existing title and it should be initialized with the default manager, author and visitor role from the RoleGateKeeper. The Event-Manager handles changes in the description and in the role distribution for an Event and the creation of roles for an Event. The EventManager has no list of all Events, but knows how to set them up using the database.

The principal constituents of the system are the Submissions. A Submission belongs to a user, an event and a workflow item of the event. Therefore it holds the user name of the submitter, the names of possible other authors, the name of the event and the name of the workflow item. Additionally it contains a title, an abstract, the path of the submitted file, the date of the submission and the name of the role in which the user has made this submission. A Submission can refer to another Submission, but it does not have to. If it does, the user name of the submitter of the related submission, its title and optionally a grade are stored; otherwise these three variables are filled with an empty string. Note that a Submission does not contain the submitted file, only the path of the file. Moreover note that a user can use a specific title only once for a specific event.

The SubmissionUpdater is responsible for creating and updating the submissions. For example, it ensures that a user does not create two submissions with the same title for one event, that a user has not reached the maximum number of submissions yet and that a user only edits a submission of the current workflow item. Furthermore, it manages the file upload. And once again, the SubmissionUpdater must be able to generate existing Submissions without having a list of them.

In addition, there exists an object called PdfFileCreator, which is responsible for creating PDF files. In this system, the PdfFileCreator is only used to produce PDF files from profiles.

Last but not least there is an object called Deleter. This object guarantees that no inconsistencies occur while deleting an event or a user account. An account may only be deleted, if the user is not also the administrator of the system or a manager for an event. When the account is deleted, the Deleter also deletes all submissions of the user and all related ones. An event can always be deleted. The Deleter orders the EventManager to delete the event, the WorkflowManager, to delete the WorkflowItems of the event and the SubmissionUpdater to delete all submissions of the event.

The remaining objects; the SessionAdministrator, the PostValidator and the Request-Handler are discussed in the paragraph 5.2.

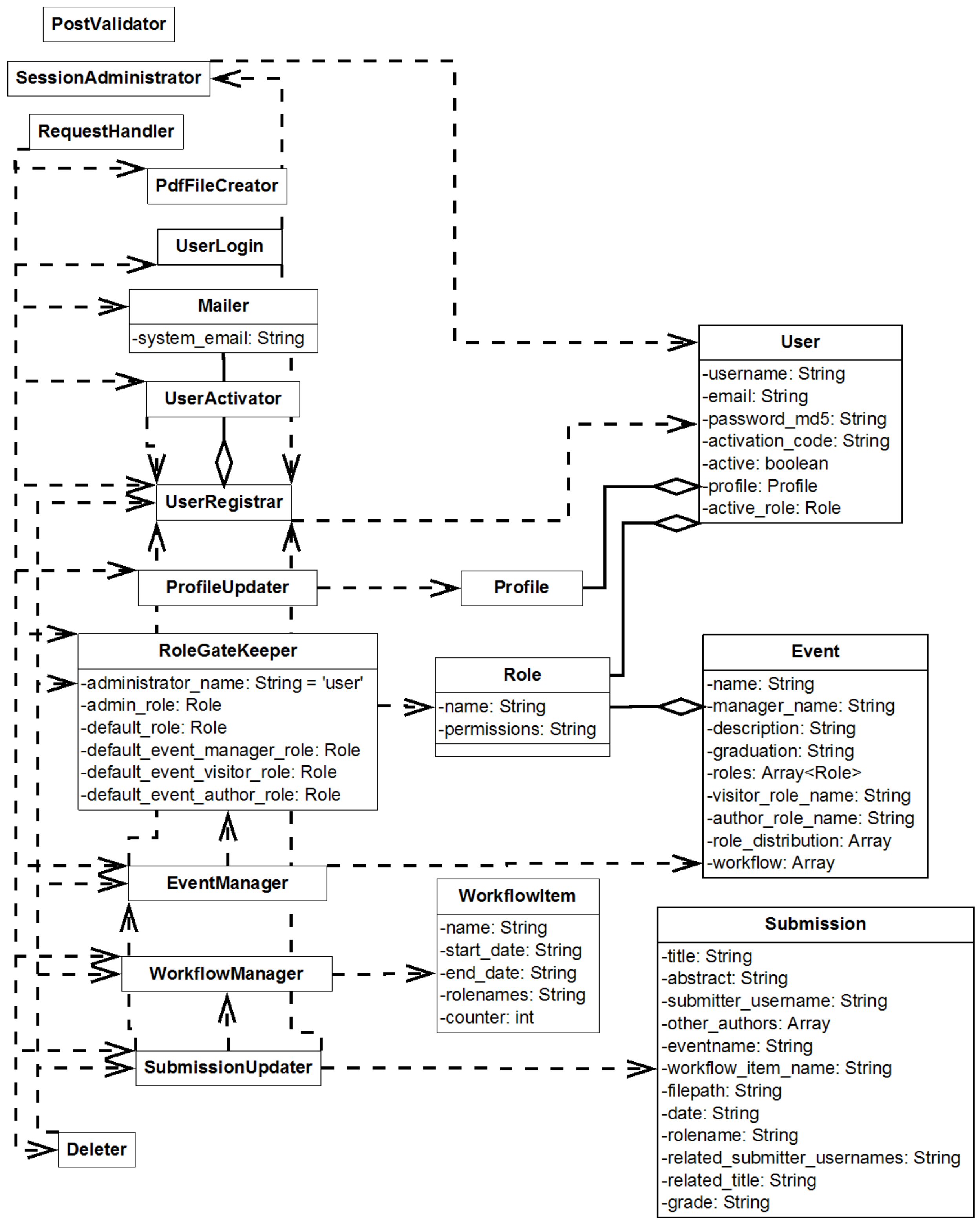

Figure 2: UML of Model

5.2 Interaction between View Part and Model

The view part has access to three classes from the model:

- SystemAdministrator
- PostValidator
- RequestHandler

The session support in PHP offers the possibility to preserve certain data across subsequent accesses of the website. These data can be stored in an array of strings, the so-called session array ($_SESSION[]). This array is stored in a cookie on the user side.

In our system, when a visitor logs in, the related user object is saved in a serialized way in a session variable. Furthermore, the session variables are used to save the name of the current event, the title and the user name of the current submission and so on. The object called SessionAdministrator handles these session variables. So the UserLogin, the “access” and “set” pages do not set or get these variables themselves, they use the SessionAdministrator.

To ensure that the user inputs are correct, they are validated in the PostValidator before being used. All rules for user input are defined in the PostValidator. For example, it is defined whether an input may be empty, how many characters the password should have at least, if an entered date is in the correct format and whether the entry is free of html code or not. Allowing html code in user input can cause an error when it should be displayed.

All other requests to the model are performed by the RequestHandler. It checks whether a user has the permission to execute the request in his/her active role. For certain actions concerning an event it is also checked whether the authorization is even valid for the current time period. If the user is entitled to the inquired action, it forwards the request to the appropriate object.

5.3 View Part

The view part contains all web pages. There are five types of pages: special files, whose names start with “access”, files for the navigation and header, normal web pages, which will be displayed in the web, special files, whose names start with “set” and a style sheet.

These “access pages” contain no html, only PHP code. They check if the user is logged in, on success, the user object is assigned to a specific variable, which can be used in the web pages, if not; the user is linked to the login.php page. Depending on the page, more session variables were checked and allocated to specific variables.

The navigation of the website is divided in two parts. One part of the navigation is at the left side and one is on the top right of the web page. The links in the navigation depend on whether a user is logged in, whether an active event is set and on the permissions the user has at the moment. In the file header_navigation.php, the header and the places of the two navigation parts are defined and it is checked whether the navigation for a logged user or an unlogged user should be used. The division into these five files has the advantage that the design can be changed quickly and a change of the contents of the navigation needs to be made in only one place.

The normal web pages contain the following html skeleton:

```
<?php
   //a first php part
?>
<!DOCTYPE  html  PUBLIC  "-//W3C//DTD  XHTML  1.0  Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
         <meta http-equiv="Content-Type" content="text/html;
         charset=UTF-8">
         <link href="style.css" rel="stylesheet" type="text/css" />
         <title></title>
   </head>
   <body>
         <?php require_once("header_navigation.php") ?>
         <div id="content">
               //content
         </div><!-- div end content-->
   </body>
</html>
```

Furthermore, they comprise mainly tables and forms and PHP and JavaScript code. Every web page for which a user has to be logged in includes an access page. Each variable set in the access page, can be used in the webpage.

Files called “set pages” whose names start with “set”, take the information sent from a form with the GET method and set it to session variables.

The style sheet is a CSS file to define the design of the web pages. This allows us to adapt the design in a very simple way.

5.4 Interaction between Model and Database

A very important task of the UserActivator, UserRegistrar, ProfileUpdater, RoleGate-Keeper, EventManager, WorkflowManager, SubmissionUpdater and Deleter is to inform the database about changes, in order that the database is always up-to-date. These objects do not send the information directly to the database, but they send messages to objects that implement one of the database interfaces called IUserDB, IPermissionDB, IEventDB, IWorkflowDB and ISubmissionDB. They ask the SubmissionSystem which object they have to talk to. The SubmissionSystem class is implemented using the Singleton Design Pattern. Thus there is always only one instance of the Submission-System.

Except in test cases, the classes UserDB, PermissionDB, EventDB, WorkflowDB and SubmissionDB were used. Let us call them DB-Classes. They all extend the DB class. This class contains all information about our database to be able to connect with it. The connect method is also the only task of a DB object. So the task of the DB-Classes is to run the correct MySQL queries to the database.

Look at the UML-diagram at Figure 3 which DB-Class appeals to which tables of the database.

If the database should be structured differently, or another storage device, such as a MS Acesses database, should be used, only the five interfaces IUserDB, IPermissionDB, IEventDB, IWorkflowDB and ISubmissionDB have to be implemented and a database connection hast to be established. Then the database can easily be replaced by changing the settings in the object SubmissionSystem.

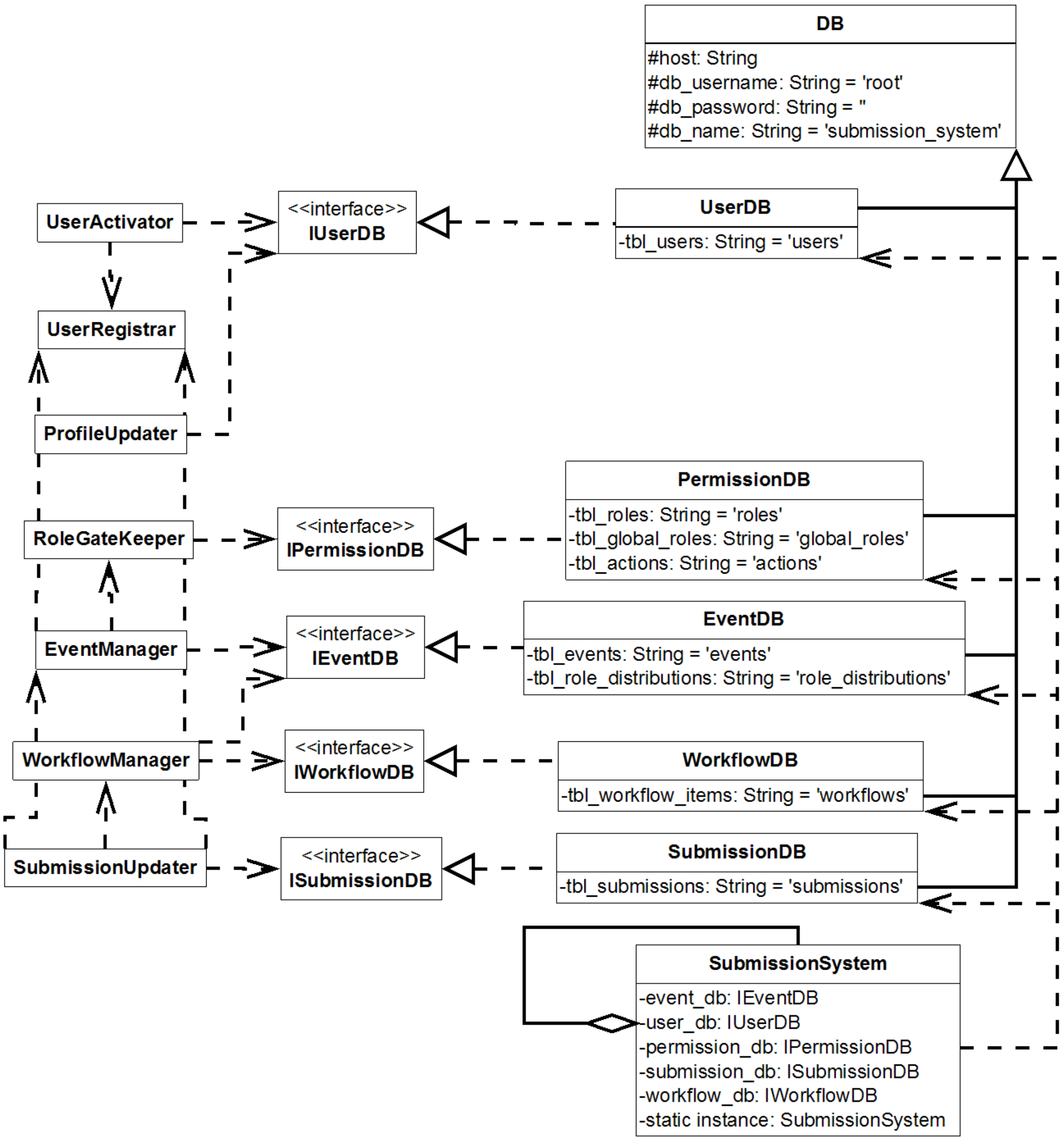
DB
#host: String
#db_username: String = 'root'
#db_password: String = ''
#db_name: String = 'submission_system'
UserActivator
<<interface>>
IUserDB
UserDB
-tbl_users: String = 'users'
UserRegistrar
ProfileUpdater
PermissionDB
-tbl_roles: String = 'roles'
-tbl_global_roles: String = 'global_roles'
-tbl_actions: String = 'actions'
RoleGateKeeper
<<interface>>
IPermissionDB
EventDB
-tbl_events: String = 'events'
-tbl_role_distributions: String = 'role_distributions'
EventManager
<<interface>>
IEventDB
WorkflowManager
<<interface>>
IWorkflowDB
WorkflowDB
-tbl_workflow_items: String = 'workflows'
SubmissionUpdater
<<interface>>
ISubmissionDB
SubmissionDB
-tbl_submissions: String = 'submissions'
SubmissionSystem
-event_db: IEventDB
-user_db: IUserDB
-permission_db: IPermissionDB
-submission_db: ISubmissionDB
-workflow_db: IWorkflowDB
-static instance: SubmissionSystem

Figure 3: UML of Database part

5.5 Database

The database holds the current state of the system. Each visitor of the web site generates his/her own instances of the objects, which however can all access the same database. That is why this database is so important.

The database structure (see Figure 4) reflects almost the pattern of the model:

The table called *"user"* stores all attributes of a user and all attributes of his/her profile. Since each user name is unique, it is used in this table as the primary key.

For storing the roles three tables are needed. The table *"actions"* stores the actions and an associated index number which is generated automatically and is unique. This is also the primary key of the table. The table called *"global_role"* saves the permissions of the five roles of the RoleGateKeeper, by adding the role name and index number of the action for each authorization. The attributes *"rolename"* and *"number"* form the primary key, where *"number"* is also a foreign key. The roles of the individual events are saved in a table called *"role".* An entry in this table corresponds to a permission of a role. A role will be identified by the foreign key "event name" and the attribute *"rolename"*. Again, the foreign key "number" is used to specify the permission. The three attributes *"eventname"*, *"rolename"* and "*number"* represent the primary key. The use of the table *"action"* has the advantage that in case of renaming an action, only the value in this table must be changed. I.e. no values in the tables *"roles"* and *"global_role"* must be changed.

The properties of an event are stored in the tables *"event"* and *"role_distribution"*. The attributes *"eventname"*, *"manager_name"*, *"description"*, *"visitor_rolename"* and *"author_rolename"* correspond to the string objects of an event. The array which specifies the "*grading"* is stored in a serialized way. The workflow of an event is saved directly as a serialized array in the attribute "workflow". Because an event is identified by its name, the attribute *"eventname"* is the primary key.

For each role a specific user has an entry exists in the table *"role_distribution".* In this table, all three attributes form the primary key.

All attributes of the WorkflowItem object are recorded in the table *"workflow_item"*, which has the attributes *"eventname"* and *"itemname"* as primary keys.

The table called *"submission"* uses the three attributes *"eventname"*, *"submitter"* and *"title"* as the primary key, since a Submission is clearly defined on these three properties. All other properties of a Submission are also stored in this table.

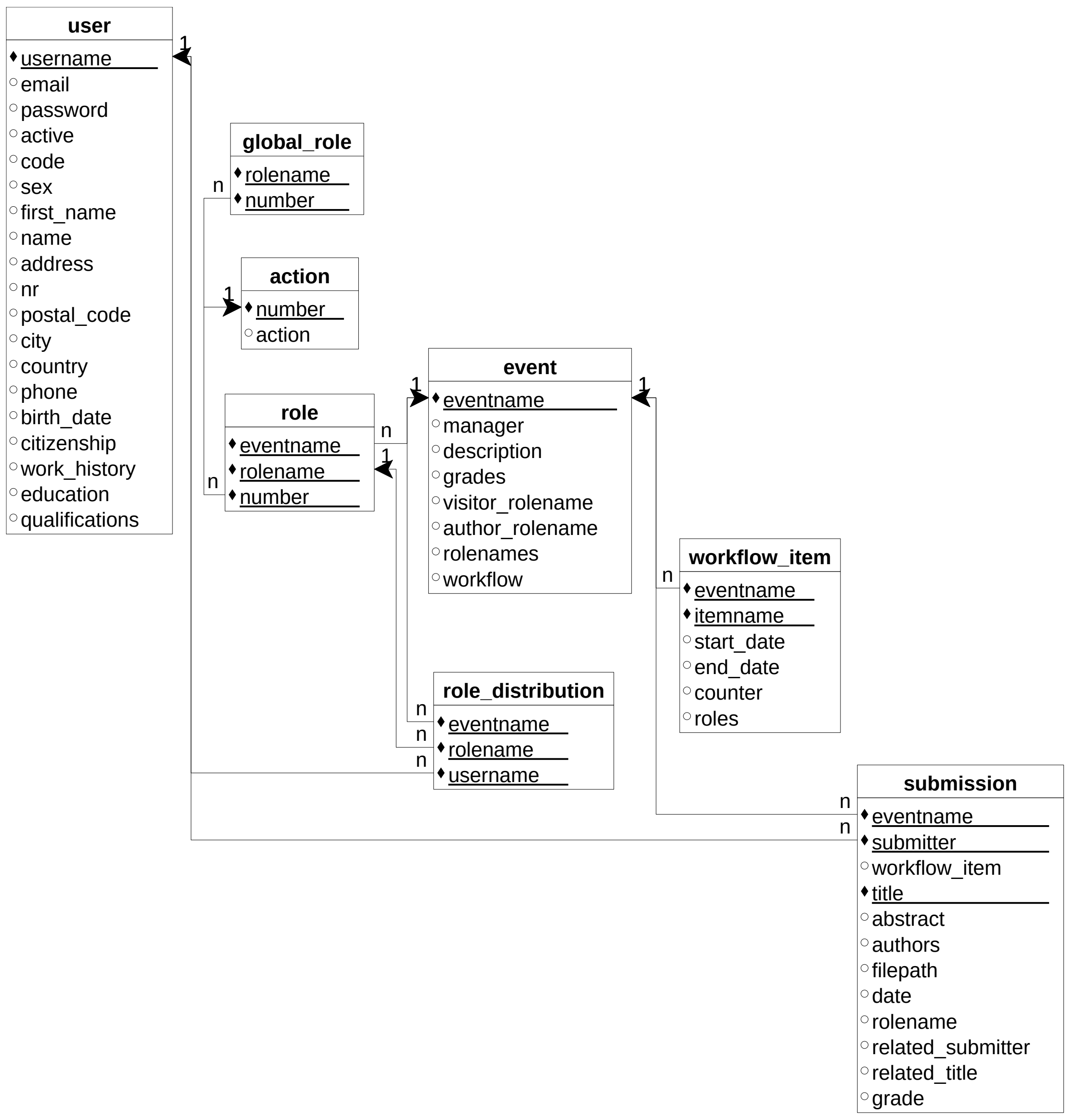

Figure 4: UML of databases

5.6 PHPUnit Tests

This part tests the model of the system. For each class in the model exists a test class that contains unit tests to test all methods.

Exclusively for testing purposes we created the classes UserMemory, Permission-Memory, EventMemory, WorkflowMemory and SubmissionMemory that implement the interfaces IUserDB, IPermissionDB, IEventDB, IWorkflowDB and ISubmissionDB. These classes do not access a database, but simulate a storage unit. Before a test is performed, the required “memory classes” are passed to the object SystemAdministrator. This allows us to test the model isolated and independent from the database. Consequently, the data in the database is not changed and the tests run faster.

5.7 Implemented Rules

In the system, certain rules are implemented, which will be briefly explained here.

5.7.1 Rules concerning the Roles

1. The administrator, the person who owns the *admin role,* is already registered with
   - user name: *administrator*
   - password: *000000*
2. If a visitor creates a submission, he/she receives the author role which also becomes his/her active role. So the user creates the submission as an author.

5.7.2 Rules concerning the User Input

1. Rules about the format of the entries are all implemented in PostValidator class and can be easily changed. It is defined, for example, that a date must be entered in the format “yyyy-mm-dd”, that a password must contain more than 4 characters and that no special html characters are allowed in the input.

5.7.3 Rules concerning the Submissions

1. An unrelated submission can only be created in the first item of the workflow of an event. (This is implemented in the Submission Updater.)
2. Only a submission created in the current item can be updated or deleted. (This is implemented in the Submission Updater.)
3. A submission to which another submission already refers cannot be deleted. (This is implemented in the Submission Updater.)
4. While adding a workflow item to an event, a folder is created. All submissions created during this workflow phase are stored in the newly created folder : generic-system/uploads / [eventname]/[workflow_item_name]
5. Only a submission related to another one with a different author can contain a not empty grade.

5.7.4 Rules concerning the Workflow Item

1. A WorkflowItem is active when the current time is bigger or equal as the start date and smaller than the end date. So it is active from the day of the start date till the day of the end date begins.
2. The Items of a workflow are arranged chronologically and they do not overlap in time.
3. If the end date of a workflow item is changed, all dates of the following items are shifted to the same time period.
4. If the start date of a workflow item is changed to an earlier date and this date is in the interval of the previous item, the end date of the previous item is set to the desired start date of the following item.
5. The roles set in a workflow item as a role with permission to create related submissions are only then really authorized when the permission *createRelatedSubmission* is set so them before.

6 Step-by-step Instantiation of the System

This chapter shows how to set up the Web Submission System to use it as a paper submission system.

Suppose you are the chief of an online magazine like JOT. This magazine has four issues a year, including one in November for which you search papers. Every one is allowed to submit a paper, but only papers which received the grade A from all reviewers at the end of a review process are published. The review process is subdivided in 5 different phases as shown in Table 1:

Table 1: Review process for magazine example

phase	paper submission	review submission	author response	last review submission	formatted paper submission
start date	2011-07-18	2011-08-18	2011-09-18	2011-09-30	2011-10-10
end date	2011-08-18	2011-09-18	2011-09-30	2011-10-10	2011-10-20

In the

- **paper submissions phase**, all visitors (users with the active role visitor) are allowed to submit their paper. Note, one person can only submit one paper.
- **review submission phase**, the reviewers from the magazine read the submitted papers and write reviews and grade the papers. They can grade a paper with A (accepted), B (provisionally accepted pending a major revision) or C (rejected).
- **author response phase,** the authors who receive at least one time the grade B revise their paper based on the reviews and submit it again.
- **last review submission phase,** the reviewers check the new papers again, grade them and write last reviews.
- **formatted paper phase,** all authors who received either in the review submission or in the last review submission phase the grade A from all reviewers submit their paper again but in a formatted way. So for example they have to remove all page numbers to make it easier to put it directly into the magazine.

To avoid problems of different file types, all papers have to be submitted as a PDF file.

First, install the system as explained in the attached “Step-by step Installation Guide” and, for local usage, ensure that Apache, MySQL and local mail server are running.

1. Open the webpage *login.php* in your browser and login as the administrator with
   - user name: *administrator*
   - password: *000000*
2. Go to *Create a new event.* To be able to do all additional settings for the event, you have to take yourself as the manager. So
   - enter a name, for example *Papers for issue November 2011*.
   - select to *take a registered user* as the manager
   - select the *administrator*

   Click the *create event* button.

3. You were now on page *event_description.php*. Enter a description of the event and click the *set description* button. For example:
   We are looking for interesting papers for the November issue of our magazine.
   Please submit your papers before the 18th of August as a PDF file. Your submissions will be corrected by 2-3 people and you get their reviews till the 18th of September.
   If you received the grade B, you should revise your paper based on the reviews and submit it again till the 30th of September. The reviewers will look at it again and send you a new review and a grade.
   If you receive either for your first or your second submission the grade A from all reviewers, please submit your paper in the asked format. You can have a look at the specific rules at our webpage.
   If you did not receive only grades A, we feel sorry, but your paper was not good enough.
4. Go to *Event > Set Grading* and define the following grades:

   Table 2: Grading scale for magazine example

Grade	Description
A	*accepted*
B	*provisionally accepted pending a major revision*
C	*rejected*

   Click the *define grading* button.
5. Go to *Roles > Create new Role*, enter the role name *pc member* and click the *add new role* button.
6. Once again, enter the role name *special author* and click the *add new role* button.
7. Go to *Roles > Set Permissions for Roles*. Select the
   - Role: *visitor*
   - Permission: *createSubmission*

   and click the *add permission* button. This setting allows each logged user to submit a paper, because every user owns the visitor role.
8. To allow an author to see all review of the own paper and to write a response, add the following permission to the role *author:*
   - *createRelatedSubmission*
   - *getSubmissionsRelatedTo*

   You can check the correct settings in the table that is displayed at the web page.
9. To allow a pc member to see all submissions, all involved persons and to write reviews add the following permissions to the role *pc member:*
   - *createRelatedSubmission*
   - *updateMySubmission*
   - *getAllUnrelatedSubmissions*
   - *getSubmissionsRelatedTo*
   - getUsernamesWithRole

   Again, you can check the correct settings in the table that is displayed at the web page.

10. To allow a special author to submit the end version of the own paper, add the following permissions to the role *special author:*
   - *createRelatedSubmission*
   - *updateMySubmission*
   - *getSubmissionsRelatedTo*

   Once more, you can check the correct settings in the table that is at the web page. The survey of all roles and their permissions should now look like this table:

Table 3: Survey of all roles and their permissions

	visitor	**author**	**manager**	**pc member**	**special author**
addNewRoleToEvent			x		
addPermission			x		
addRoleToUserForEvent			x		
createAndAddWorkflowItem			x		
createNewEvent					
createRelatedSubmission		x	x	x	x
createSubmission	x	x	x		
deleteEvent			x		
deleteMyAccount	x	x			
deleteMySubmission		x	x		
deletePermission			x		
getAllRegisteredUsers					
getAllUnrelatedSubmissions			x	x	
getSubmissionsRelatedTo		x	x	x	x
getUsernamesWithRole			x	x	
sendMail			x		
updateEventDescription			x		
updateGradingForEvent			x		
updateMySubmission		x	x	x	x
updateWorkflowItem			x		

11. Go to *Event > Set Workflow* and add a new item as follows:
    - Name: *paper submission*
    - Start date: *2011-07-18*
    - End date: *2011-08-18*

    Click the *add as new item* button.
12. Select the just created item to prepare and click the *choose* button.
13. Set the counter to *1 and c*Click the *change item* button.
14. Add a second item with these properties:
    - Name: *review submission*
    - Start date: *2011-08-18*
    - End date: *2011-09-18*
15. Select the item *review submission* to prepare and click the *choose* button.
16. Select the roles *pc member* and *manager* to give them the permission to create related submissions and click the *change item* button.

17. Add a third item with these properties:
    - Name: *author response*
    - Start date: *2011-09-18*
    - End date: *2011-09-30*
18. Select the item *author response* to prepare and click the *choose* button.
19. Change the following:
    - Counter: *1*
    - Roles with permission to create related Submissions: *author*

    Click the *change item* button.
20. Add a fourth item with these properties:
    - Name: *last review submission*
    - Start date: *2011-09-30*
    - End date: *2011-10-10*
21. Select the item *last review submission* to prepare and click the *choose* button.
22. Select the roles *pc member* and *manager* to give them the permission to create related Submissions and click the *change item* button.
23. Add a third item with these properties:
    - Name: *formatted paper submission*
    - Start date: *2011-10-10*
    - End date: *2011-10-20*
24. Select the item *formatted paper submission* to prepare and click the *choose* button.
25. Change the following:
    - Counter: *1*
    - Roles with permission to create related Submissions: *special author*

    Click the *change item* button.
26. Go to *Roles > Define Role Distribution*. Now we will define our pc members. Select
    - the role: pc member
    - the user: *a not registered person,* with
        - user name: *tom*
        - email: *newuser@localhost*

    and click the *add role to user* button.
27. Do the same for our second person. Select
    - the role: pc member
    - the user: *a not registered person,* with
        - user name: *dick*
        - email: *newuser@localhost*

    and click the *add role to user* button.
28. Repeat step 27 for:
    - the role: pc member
    - the user: *a not registered person,* with
        - user name: *harry*
        - email: *newuser@localhost*

Now, the event *Papers for issue November 2011* is ready for use.

7 Examples for Usage

The Web Submission System can be used in very different situations. We will explain only four of them.

7.1 Application System

As required by the task, the system can be used as an application portal.

The procedure for one job vacancy at the University of Bern explained in the introduction can now be designed as follows:

1. The dean or his secretary creates a new event with the name of the job vacancy and the user name of the hired manager.
2. The manager sets the description of the job. This description should also provide an indication of how the application documents should be submitted. So the best way is to keep the application, the CV and any additional files like qualifications or an important publication in one PDF file.
3. The manager defines the desired grading scale.
4. The manager defines a new role called *committee member.*
5. The manager adds the newly created role to all persons he/she wants to have in the committee.
6. The manager adds the following permissions to the role *committee member:*
   - *createRelatedSubmission*
   - *updateMySubmission*
   - *getSubmissionsRelatedTo*
   - *getAllUnrelatedSubmissions*
   - *getUsernamesWithRole*
7. The manager adds the following permission to the role *visitor:*
   - *createSubmission*
8. The manager sends an email (from the system) to all pc members to inform them about their role and task.
9. The manager adds a new workflow item called *application submission* and defines the time period in which the applications should be submitted.
10. The manager changes the number of allowed submission for the *application submission* item to *1*.
11. The manager adds a second workflow item called *review submission* and defines the time period in which the committee members should submit their reviews.
12. The manager selects *manager* and *pc member* as the roles with permission to create related Submissions.

Now all settings are defined and the applicants may, during the period of the first workflow phase (*application submission*), apply. Then in the second item, the committee members write their review. They can access to all applications and all reviews. The manager can change the dates of the workflow phases at any time and he/she can also send a reminder email to all committee members.

If desired, two other roles could be defined: one role for candidates who are not invited for an interview and one other for candidates who are invited for an interview. Hence, a standard email for the cancellation or an email with an advance notice of the invitation could be sent to them. And in addition, the committee members can see at a glance which candidate has been invited.

7.2 JOT

We will next analyze the definition of the JOT example with our submission system. So in this case, the system is used as a paper submission system. The Editor-in-Chief either creates an event for every year or for different topics. Anyway, such events have only one workflow phase. In this phase, the authors can submit a paper, the reviewers can write there review and so on. So there is no fix timetable. Such an event can be created like this:

1. The Editor-in-Chief (EiC) creates an event with a name, for example the year or the topic of the papers and with an Associate Editor (AE) as the manager.
2. The EiC defines a new role called *expert reviewer.*
3. The EiC adds the following permissions to the role *expert reviewer:*
   - *createRelatedSubmission*
   - *updateMySubmission*
   - *getSubmissionsRelatedTo*
   - *getAllUnrelatedSubmissions*
   - *getUsernamesWithRole*

   The permission *getUsernamesWithRole* allows an expert reviewer to see which user has which role for this event.
4. The EiC adds the following permissions to the role *author:*
   - *createRelatedSubmission*
   - *getSubmissionsRelatedTo*
5. The EiC adds the following permission to the role *visitor:*
   - *createSubmission*
6. The EiC adds a new workflow item called *paper submission* and defines the time period in which this event should be used for paper submission.
7. The EiC selects *author, manager* and *expert reviewer* as the roles with permission to create related Submissions
8. The AE sets the description of the event. This description should provide an indication that the papers should be submitted as a PDF file and the expiration of the process.
9. The AE sets the desired grading as in the following grading scale.

Table 4: Grading scale for JOT

Grade	Description
A	*accepted*
B	*provisionally accepted pending a major revision*
C	*rejected*

10. The AE adds the *expert reviewer* role to three expert reviewers.
11. The AE sends an email (from the system) to all expert reviewers to inform them about their role and task.

When someone submits a paper to this event, the expert reviewers write a review. Then the AE will write a review in which he/she recommends acceptance or rejection of the paper based on the returned reviews and on his/her own assessment. The author waits till the AE has written a review and reads it. Depending on this received grade, the author has to submit a revised version of the paper or already a final camera-ready copy or nothing.

The EiC can create also a new event for accepted papers. Then the AE would add the author role to the authors with an accepted paper and let them submit their final camera-ready copy. To prevent other users to submit anything for this event, the EiC should ensure that the role visitor has not the permission *createSubmission*.

The EiC can create a separate event also with one workflow item for the *special section.* If the EiC adds the *author role* to the guest editors, only they can submit.

7.3 Music Selection

The system can assist in the selection of music of an orchestra.

I play in a wind band, where the music selection is regulated as follows:

We have a music committee, headed by a chairwoman. About nine months before we will have a concert she asks all members to send her an email with their suggestions of compositions they would like to play at this concert. Recordings of these pieces can be attached, but it is not necessary. The chairwoman seeks the missing recordings and sends all recordings and suggestions to the other committee members. After a month, the committee meets, discusses all suggestions and defines our concert program.

With the help of the Web Submission System this procedure would now run like this:

At any time the chairwoman can create an event for the concert. For the time flow of the search, she sets three phases. In the first phase, all members of the orchestra can submit their proposals. In the second one, all members can download the suggested music, can comment the pieces and can upload missing recordings. In the third phase, the committee members write their first comments about the pieces before they meet for the first time.

After the concert program is appointed, a member of the Music Commission creates another event. This is used to upload recordings of the selected pieces, so they are accessible to all members for exercise purposes.

In addition, the chairwoman can create another event in which members can always bring the suggestions of music that they want to play one time.

7.4 Planning Phase of a House

The Web Submission System could be used to share construction plans while planning a house. Nowadays the architect, the civil engineer, the heating- and ventilation planner and the electrical planner send their plans by post or by email. Using the system would simplify the planning phase.

Suppose the architect is requested to plan a house and hires the civil engineer, the heating- and ventilation planner and the electrical planner. Therefore, the architect creates an event for this house and appoints himself/herself to be the manager.

The architect adds three roles for the event: one role for the civil engineer, one role called *planner* for the electrical planner and the heating- and ventilation planner and one role called *manual worker* for the master builder, the electrician, the plumber, the carpenter and the heating engineer. At least, the architect defines the workflow so that the planning phase would run as follows:

1. The architect submits all plans for the house as a PDF and as a DWG[4] file.
2. The civil engineer has two weeks to dimension the house and to correct the architect’s plans. The system is used to share the reviews and proposals.
3. The architect submits the revised plans.
4. The planners have now two weeks to draw in the electrical lines, the sockets, the outlets etc. So each of them submit their own plans.
5. The architect and the civil engineer remediate each plan and submit them again.
6. The planners revise their plans.

Using the system has several advantages:

- During the whole planning phase, the manual worker can have a look at their plans and can begin to prepare their work. No email has to be send to them.
- Now everyone can access all the planner plans, which simplify the coordination of the planners. So they can download the other plans and have a look that for example the electrical lines and the heating ducts do not overlap.
- All involved persons are always up-to-date.
- All plans are stored in one system and can be downloaded several times.
- The annoying email traffic is eliminated.

[4] DWG is a file format for several CAD software, a software for construction plans.

8 Extensions of the System

Of course, as every system, also the Web Submission System can still be optimized. There are some features whose implementation or revision I would highly recommend. I have had to omit the implementation of them mainly due to lack of time. But there are also some features that are simply “nice to have”. So they make the system more comfortable.

8.1 Recommended Features

The recommended features increase the security of the system. For professional use of the system, for example as an application system, a security check is essential. On the one hand, it has to be ensured that user input does not affect the system; on the other hand the system has to be protected against hacker attacks.

8.1.1 Protection against malicious User Input

Malicious user input can be entered wantonly but also inadvertently. There are several security vulnerabilities a web system can have. I will explain only a few of them.

- **HTML Injection**

  User input containing html code can destroy the correct view of a webpage. This security hole can also be used to display some Spam at a webpage. To resolve this problem in our system, at least partially, no special html characters are allowed in the user input. But this solution is not very satisfactory because it restricts the user. The solution should be optimized.

- **SQL Injection**

  This vulnerability allows hackers manipulating the SQL queries. This vulnerability is caused by a lack of masking or verification of metacharacters in user input. It allows an attacker to gain access to all data in the database and to modify this data. This would be restricted only if the database user has not all privileges, but this is not the case in our Web Submission System.

  To remedy this deficiency the function `mysql_escape_string($input)` provided by PHP can be used. This function adds unwanted characters with a backslash. However, these backslashes should be removed before displaying the input again.

  Since PHP 5.1 so called PHP Data Objects offers an optimized solution. They have been created to simplify and unify database entity access. One advantage of these objects is precisely to reduce the susceptibility of SQL Queries for SQL Injection.

  If such metacharacters are entered in inadvertently, it can cause wrong database entries.

- **Insecure Session Handling**

  Insecure Session Handling allows an attacker to access the session ID of a user. The possession of this ID allows the attacker to use the web application, as he/she would be logged normally.

A logged user can also be able to manipulate his/her own session tokens to get access, for example, to submissions he/she should not see.

There can be many more security holes, but at least these three should be fixed.

8.1.2 Protection against Hacker Attacks

The susceptibility of the system to hacker attacks needs to be checked. A user's privileges are dependent on his/her active role and they are checked in the RequestHandler. If a hacker knows the names of the classes of the model, he may be able to avoid the RequestHandler by calling the methods directly on the relevant objects.

8.2 Optional Extensions

These extensions are nice to have and do not affect the generic idea.

- Add a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) to the registration process to prevent that so-called "spam-bots" or "vandal-bots" automatically create accounts.
- Users who may view the related submissions of a submission could choose whether they want to be informed by email when a new entry on this topic is made. Implement an object, for example a Messenger, who is responsible for that.
- Add a form to allow users to enter their text directly at the web page and use the class PdfFileCreator to create automatically a PDF file from the entered text.
- Add a help-function to improve the usability of the system.
- Add a new variable to the class WorkflowItem. With this variable can be specified how the submissions from this item should be mentioned and displayed on the websites. For example, the user-friendliness of an application portal is increased when a submission in the first phase, in which the candidates submit their applications, is called "application" and the submissions of the committee members in the second phase are called "reviews".
- Offer a manager of an event the possibility to determine when an event should be listed on the page *events.php*. For example, in an application system it makes sense that an event is no longer displayed when the application deadline has passed.
- Offer a manger the possibility to delete a foreign submission. Consequently, the manager is able to delete spam or unacceptable submissions.
- Offer a manager of an event the possibility to define which sort of files may be uploaded. Thus, for example, the papers of JOT could only be submitted in PDF format.
- Offer the administrator the possibility to change the global roles. In some cases it might be useful that anyone can create an event. So the administrator has to add the required permission to the default role.

For the specific use of the Web Submission System many more enhancements are can be made, but probably with the loss of the generality.

9 Personal Experiences

Before I started this project, I had very little experience with programming in PHP.

Thus, I created a login class with the help of a tutorial, to get familiar with forms and the two required methods GET and POST, with session variables and the database support for PHP. The syntax of PHP is more or less the same as in Java, so it was no problem to learn it quickly.

Then I started developing a system especially for the professor hiring process, because it was easier for me to get the requirements a submission system has. But because I have not programmed such a complex system before, I had problems to define the objects and their responsibilities. In retrospect, the choice of objects seems very plausible, but it has many different ways to allocate the tasks. For example I was not sure, whether I should save only the user name of the manager in an Event or the manager as a User object. It seemed more logical to work with the whole object than just with a user name, but this caused some problems. If the manager of an event changed his/her data while a second person had access to the same event, this second person worked with an invalid manager object. Therefore I decided to work with user names. They do not change and they simplify the problem of simultaneous accesses to the system. So I did not have to change all occurrences of an object while changing an object.

The transition from the concrete system for applications to a generic system seemed like a step backwards. I could only copy the classes UserLogin, UserActivator, User-Registrar and User, the others I had to revise. Previously I had worked with a class called Application and a separate class called Review; these had to be merged now. Furthermore I had only defined roles for the whole system and now I wanted separate roles for each event. That means that first I had a working system and then during the changeover almost nothing worked anymore.

A few times I thought I would make a regress and so I was frustrated, but now I think that this is a normal experience during the development of a program. You are adding features to the system and then these features are changing the ideas and concepts that you had before.

But most of all I struggled with the installation of the development environments. At the beginning I was working with Eclipse for PHP. But this program supports no refactoring, which was really annoying and the plug-in for Simple Test could not be installed. So I switched to NetBeans IDE for PHP. The refactoring worked not always but most of the time. But here I had many problems to install PHPUnit and XDebug. To make both working took me a lot of time. Not till the half of my work I could write tests. The Selenium Framework did not run with NetBeans. So I could no test the web pages automatically. That would have been a great help. These problems are the reason why I've written a tutorial. No one should agonize before testing the system locally.

The third kind of problem that occurred has to do with the different operating systems. That my system works fine on my laptop, still does not mean that it would also work on the computer of my assistant. I work with a Windows operating system, he with Mac OS X. On the one hand, the installation of programs is different; on the other hand, there were problems with the code. For example, I used backslashes in the path for including a link. On Windows, both slash and backslash can be used as directory separator character, but in other environments like in MAC, it is only the forward slash.

All together this project was a good experience and I have learned a lot. I am grateful that I could write my thesis at the Software Composition Group.

Bibliography

Book:

[1] Matt Zandstra: *PHP Objects, Patterns and Practice. Build powerful code by mastering PHP's object-oriented enhancements, design patterns, and essential development tools* (3. Edition), Apress, 2011

Web pages:

[2] PHP – W3SCHOOLS
URL: http://www.w3schools.com/php/

[3] PHP – WEB SECURITY
URL: http://php-security.org/2010/05/01/article-php-web-security

[4] PHP LEARNING TRAIL – NETBEANS
URL: http://netbeans.org/kb/trails/php.html

[5] PHP: PHP – MANUAL
URL: http://php.net/manual/en/index.php

[6] PHP: PHPUNIT MANUAL
URL: http://www.phpunit.de/manual/3.6/en/index.html

[7] WIKIPEDIA - PHP
URL: http://de.wikipedia.org/wiki/PHP

[8] WIKIPEDIA – SQL INJECTION
URL: http://de.wikipedia.org/wiki/SQL-Injection