DPS – DISTRIBUTED AND PARALLEL SYSTEMS GROUP

University of Innsbruck
Distributed and Parallel Systems Group

Bachelor Thesis
Institute Library System

Nathalie Steinmetz
Rauschgraben 5 Top 5
A-6162 Mutters
Matrikelnr.: 0115415
nathalie.steinmetz@student.uibk.ac.at

Supervised by Univ.-Prof. Dr. Thomas Fahringer
and co-supervised by Marek Wieczorek

Innsbruck, October 14, 2006

Contents

List of Figures

Abstract

In the context of this thesis we have built an Institute Library System. The Web application uses the Apache Web server, PHP and MySQL as underlying technologies. We provide a short overview and discussion of these and of similar technologies. We describe the library architecture and the use cases that occur and we draw a corresponding database model. We provide a detailed user manual for the library application.

Acknowledgements

I would like to thank Marek Wieczorek for reviewing this thesis. He helped me in finding a proper structure and in presenting especially the diagrams in an adequate way.

I am thankful to my supervisor Prof. Thomas Fahringer for his patience. And finally thanks go also to Claude Kinarian for proof reading the thesis.

Chapter 1

Introduction

The Distributed Parallel Systems Group (DPS) is a research group of the Computer Science Institute of Innsbruck. The group has more than 300 publications in its institute library, which are only managed in a simple table processing system.

The administration of large publication collections without support of an appropriate library system is cumbersome. Only one person can manage the library "list" and potential library users have no possibility to search for publications. To borrow a book they either check it out without informing anyone or they inform the responsible person which needs to mark this manually in the publication list.

A computer-based application would be appropriate for the use cases such an institute library provides. Therefore the goal of this thesis is the development of a Software Framework for the DPS Library.

Computer-based Library System

A computer-based library system with an underlying database simplifies the library management. One or more responsible persons can administer the library content independently from each other. They don't need to therefore exchange any data lists, as the publications are saved in a "common" database. The database can regulate the access of lots of users. This allows the potential library users to easily search for publications in the database.

The management of library users and data such as "who has borrowed which book and needs to return it until when" can also be supported by a computer-based system. This way the library can be administrated in a transparent, user-friendly way and becomes a competent knowledge base which facilitates information retrieval.

The computer-based library system could be realized as Web application.

Web Application

A Web-based application is running on a Web server, and uses a Web browser as interface. The user does not need to install any software on his computer, all he needs is a browser that is connected to the internet. Additionally, as the Web application only uses a browser as user interface, it can run on any operating system that may be installed on the client's side.

Such Web applications have several advantages. [Graham, 2001] talks for example about the possibility of using a Web-based application anywhere, while software that is installed on a desktop computer can only be used on that computer. It refers also to the fact that with Web applications, all users use the same version and do not need to upgrade their product. Bug fixes and upgrades can be done on the server, fast and without interaction from the application users.

Institute Library System (ILS)

In the context of this thesis we have built a library Web application. In the following chapters we will explain the system functionality and architecture. We will provide an overview of some state-of-the-art Web technologies, will explain our choice of technologies for the ILS and will provide detailed information about the final implementation.

The thesis describes what steps need to be done to install a running library system and how the software needs to be configured. We will provide a user manual where we show how to use the library system Web interface. Finally, the appendices will list relevant fragments of the implementation source code and build scripts.

Chapter 2

Requirement Specification

The goal of this work is the development of a software framework for the institute library of the Distributed and Parallel Systems (DPS) group, a research group belonging to the Institute of Computer Science of the Leopold-Franzens University Innsbruck.

In this chapter we will explain the requirements to the framework, both concerning the library functionality (Section 2.1) and the technological side (Section 2.3). Section 2.2 will provide an overview of some system invariants.

2.1 Functionality Requirements

The Institute Library System (ILS) should be a system that allows managing both library publications and library users.

All library publications should be entered and saved in the ILS database. The system should provide search functionalities: a simple mode for keyword search and an expert mode for advanced search possibilities.

Library users should be able to login to the ILS and to borrow books. The Web application should become the first "contact point" for users when they search for a publication, borrow or return it. Before a user checks a book out of the library, he must borrow it online. Returning books works analogically. The user must bring the book back to the library and then return it online, so that it becomes available for other users again.

Only books shall be available to be checked out of the library. All other publications should be read or eventually copied in the library room. New books should not be checked out during a certain time period, e.g. one month. They shall first be available for all library users in the library room.

The publication and user database shall be managed by one or more persons with additional competences: the library administrators. They should be the ones who add new publications and users to the library and who manage the library content (publications and users). They should also be responsible for the allocation of user rights (e.g. administrator rights).

The ILS shall be able to send automatic emails to library users. These mails should be sent to remind users when they need to return a book to the library. It should thus be sort of an electronic reminder service.

2.2 Invariants

In the following, some invariants of the library system are mentioned. These functional rules describe conditions that need to be always fulfilled in the library system.

- Only the administrator/s can add, change and delete publications and users from the library database.
- Only the administrator/s can change the system's email reminder time interval.
- Only the administrator/s can change the time period after which the system changes a publication status.
- Only books can be checked out. Other publications can be read or copied at the library.

2.3 Technical Requirements

The ILS should be implemented as a Web interface with a database behind. For the Web interface HTML and PHP, Java Servlets or similar technologies shall be used. [Uni, 2006] is an example library system upon which we could base our ILS. The layout of the Graphical User Interface (GUI) should be based on the DPS homepage[1]. As internal library database any database system can be used. The use of BibTeX would also be welcome.

The implemented system should be able to execute the above mentioned system functions automatically. It should also allow to log all actions that users and administrators execute at the library interface.

[1] `http://www.dps.uibk.ac.at/`

Chapter 3

Architecture and Implementation

In this chapter, we will discuss the architecture and implementation approach to the Institute Library Software Framework.

In Section 3.1 we will provide an overview of the implemented system architecture. Section 3.2 explains our solution proposal for the library functionality, including use cases, an overview of the system entity structure, of publication states and of the database architecture.

Section 3.3 shows our technological approach. We give a short overview of some state-of-the-art technologies and explain the choice of technologies we made for our implementation. We provide also details about the implementation of the library system.

3.1 General Description

The Institute Library System (ILS) has been developed as a Web based system (see Figure 3.1). This means that library visitors and users can access the library Web interface online over the Internet, via their standard Web browser and from any computer they want. They don't need to run any separate application on their computer.

The access to the system is regulated by specified user and administrator rights and is protected by the use of passwords.

The functional unit of the system consists of a Web application, implemented in PHP, and an internal relational database, both running on a Web server.

3.2 System Architecture

In the following, we will explain the functionality and the architecture of the developed library system. Section 3.2.1 provides an overview of the library

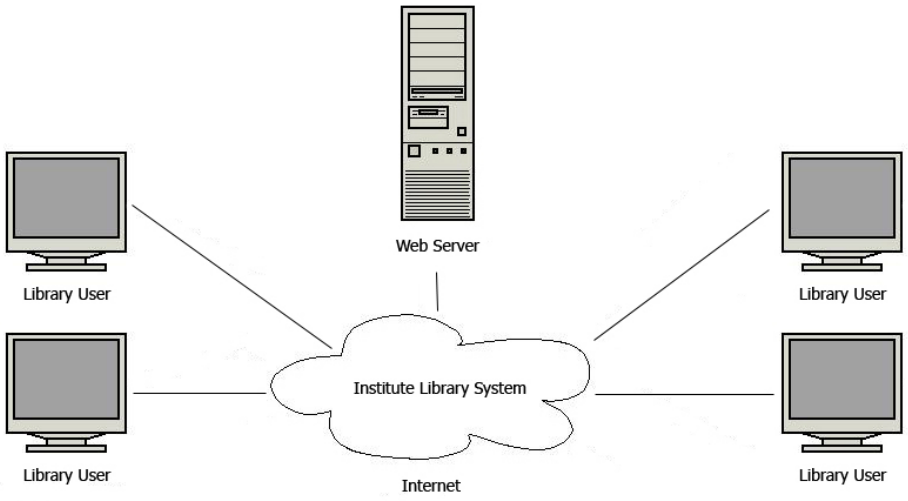

Figure 3.1: Web based Institute Library System

actors, Section 3.2.2 describes the general system functionality and Section 3.2.3 provides details of the resulting use cases. In Section 3.2.4 we explain the possible states of library publications and in Section 3.2.5 we show taxonomies of library publications with respect to their classification and type. Finally, Section 3.2.6 explains our database structure.

3.2.1 Library Actors

The DPS Library is a system that manages library publications and library users. The requirement specification (see Chapter 2) leads us to four actors that are concerned by the system:

- **Visitor** - A Visitor can search in the library's publication database.
- **User** - A user is a registered library user. He has a username and a password to log in to the library system. He can borrow books if they are available.
- **Administrator** - An administrator is a special kind of user. He is a user with additional capabilities, a so-called superuser. The administrator can add new publications and users to the library and make changes to the existing database content.

- **System** - The system shall send automatic reminder emails and change automatically the publication status.

3.2.2 Library Functionality

In the following, we show what actions can be performed by the library visitors, users and administrators and what actions can be performed by the system automatically. This leads to the use cases that we explain in Section 3.2.3.

Visitor Functions

All library visitors can use the system for searching publications in the DPS library. They can choose between a simple and an advanced search mode. The simple search mode allows only to search for one or more keywords, whereas the advanced search mode can be used to search for one or more determined authors, a title, a topic, etc.

Search results are represented in a list. By selecting a publication from this list, the visitor gets a form with detailed information about the object.

To be able to borrow a book, the library visitor needs to register to the DPS Library and get a username and password for the library system.

User Functions

If a publication is available, the registered user can borrow it. After having borrowed a book online, he can check it out personally at the library. Only books can be checked out from the library. Other publications, e.g. proceedings and manuals, can be read and copied at the library.

The user needs to return the book online, after he has brought it back to the library.

As we mentioned already above, each user has a username and password. While the username is fixed, the password can be changed by the users on their own.

Privileged Functions

The DPS Library is managed by one or more library operators, the so-called administrators. They can exercise some privileged functions.

Administrators add new publications to the library database, change the information of existing publications or delete publications from the database. They can see which publications are borrowed, by whom and until when.

They do the same for library users, adding new users to the system, changing user information and deleting users. They can also determine if a user will get automatic reminder emails from the system or not.

Concerning the system's automatic email reminder, administrators can change the time interval after which the system sends its reminding emails. They can also change the time period after which the system changes the status of a new publication.

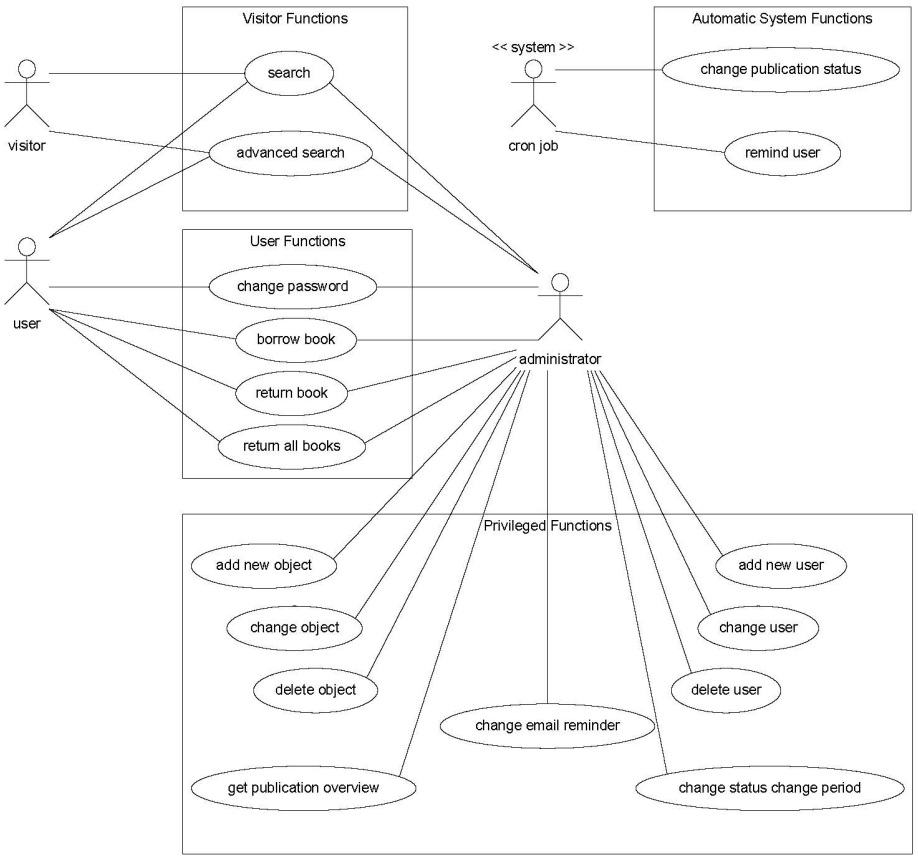

Figure 3.2: Use Case Diagram

Automatic System Functionality

The system has two functions that are automatically executed. One concerns the email reminder. The system reminds users via email when one of their borrowed books is due to be returned to the library. The other function concerns the status of publications. After a given time period (which can be changed by the administrator) the system changes the status of a new publication to "Available" or "Not Available" (see Section 3.2.4).

3.2.3 Use Cases

Diagram 3.2 shows the use cases that have been considered during the modeling of the Institute Library System . In the following sections we group the use cases in topics (introduced at Section 3.2.2 about the library functionality) and describe them in more detail.

3.2.3.1 Visitor Functions

Library visitors can use two different modes to search for publications, the *simple* and the *advanced* search.

- **Simple Search** - In the simple search mode, the visitor or user enters one or more keywords to a single field.
- **Advanced Search** - The advanced search mode allows visitors and users to search for e.g. one or more specified authors, a title, a year, a publisher, a location, an ISBN/ISSN, a publication type, a classification, one or more keywords.

3.2.3.2 User Functions

Registered users can borrow and return books.

- **Borrow Book** - Users can borrow books online and check them out personally at the library. Other types of library publications, like proceedings, manuals, etc. cannot be checked out.
- **Return Book** - Users can see a list with all the books that they have currently borrowed. They can then choose to return one or more of these books. Before they return the books online, they should bring them back to the library.
- **Return All Books** - Users can see a list with all the books that they have currently borrowed. They can return all these books at once. Before they return the books online, they should bring them back to the library.
- **Change Password** - Users can change their system password on their own and at any moment. All they need to do is to fill out a form where they need to enter the old password and, for security reasons, twice the new password.

3.2.3.3 Privileged Functions

Administrators have privileged functions that allow them to manage the library system.

- **Add New Object** - Administrators can add new publications into the library database via a simple form.
- **Change Object** - Publication information from any existing object in the database can be changed. To find the publication to be changed, the administrator can select the concerned object from a list of all publications, or he can search for it.

- **Delete Object** - The administrator can delete publications from the system database. Again, the object can be found either by selecting it from a list of all publications or by searching for it.
- **Get Publication Overview** - The administrator can get an overview of all publications with detailed information about the status and, in case the book is checked out, about who has borrowed the book until when.
- **Add New User** - Administrators can add new users to the library system. A user is identified by his last name, his first name, his system username and his email address. The system username needs to be unique and must not be longer than 16 letters.

  A flag is set to indicate whether this user shall be automatically reminded by the system when one of his borrowed books is due, or not. By default the user is reminded by the system.

  Another flag indicates whether this user shall be an administrator or not. By default the user does not become an administrator.
- **Change User** - The administrators can change the user information and the user rights of a specified user. They can select this user from a list of all library users.
- **Delete User** - Administrators can delete users from the system, so they simply become visitors.
- **Change Email Reminder** - The administrators can change the time interval that determines for how long library users can keep books. After this time interval the system sends a reminder email to the user. By default the time interval is set to 28 days.
- **Change Status Change Period** - The administrators can change the time interval that specifies after how many days the status of a new publication shall be changed from *New* to *Available* (for books) or *Not Available* (for other publications).

3.2.3.4 System Functions

The system has two automatic functions:

- **Remind User** - The system sends a reminder email to a user, when one of the books this user has checked out is due to be returned to the library.
- **Change Publication Status** - The system automatically changes the status of new publications after a given time period.

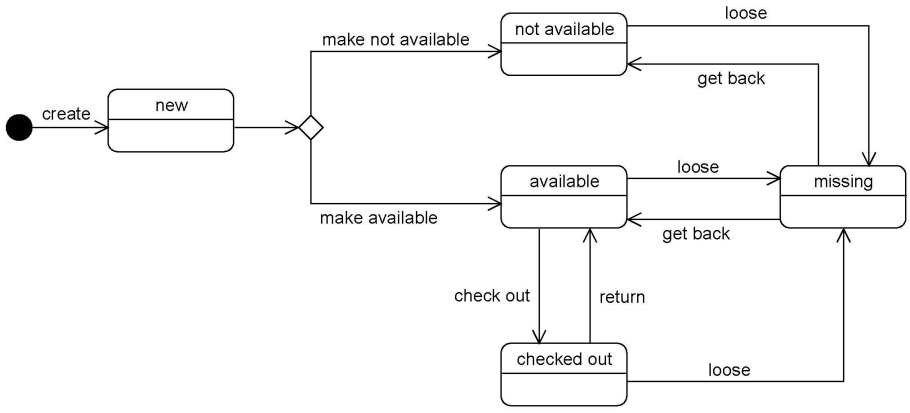

Figure 3.3: Publication States

3.2.4 Publication States

While all publications can be looked at in the library room, only books can be borrowed and checked out. Diagram 3.3 shows the different states publications can get in.

- **New** - A *New* publication cannot be checked out from the library. The system should automatically change the state of the publication to *Available/Not Available* after 1 month.
- **Available** - *Available* books are at the library and can be borrowed and checked out.
- **Not Available** - Publications other than books are *Not Available* to be checked out from the library.
- **Checked Out** - Books can be borrowed and *Checked Out* for four weeks (by default).
- **Missing** - A book is stated as *Missing* if it has been lost or if a user has not returned it after the due return date.

3.2.5 Publication Classification and Type

Publications in the library database get a classification attribute and a type attribute. The classification specifies the genre of the publication, while the type specifies its nature. Possible values for both, classification and type, can be seen in Figure 3.4.

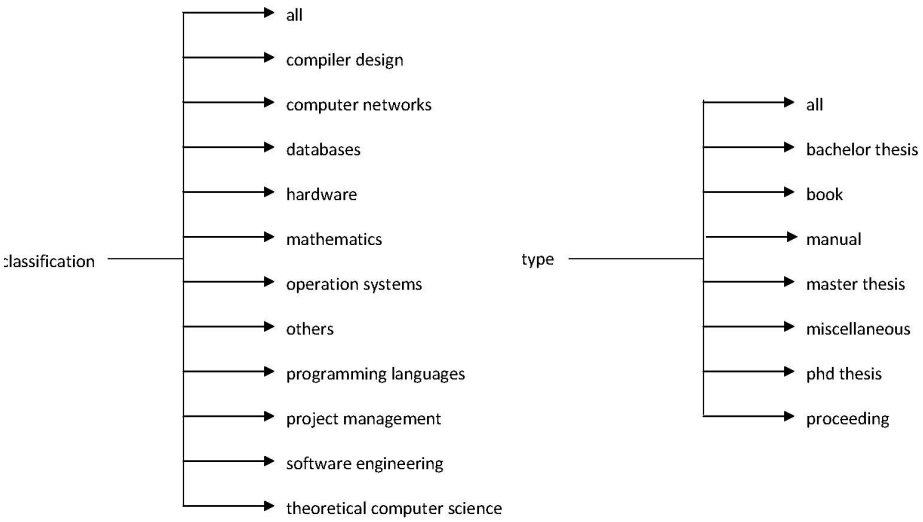

Figure 3.4: Publication Classification and Type Taxonomies

3.2.6 Database Structure

Our database structure is based on the standard guidelines for working with relational databases. We have modeled one database, *dpslibrary*, with four tables, *user*, *borrowing*, *publication* and *organisation*. Figure 3.5 shows the conceptual structure of the database architecture, including the associations between the different tables.

In the following sections we explain the four database tables (Section 3.2.6.1) and provide an overview of the database users and their rights on the database (Section 3.2.6.2).

3.2.6.1 Database Tables

Each one of the four tables in our database has an identifier as primary key that is used to identify the inserted rows (e.g. the attribute *uid* in the *user* table). In the following we explain the properties specific to each table.

User. The *user* table does not allow null values, i.e. each value of a row needs to be provided. The *username* must not exceed 16 characters and needs to be unique in the table. The *email* field, containing the email address does also need to be unique.

The *password*, initially specified by the administrator, can be changed later by the user. If a password gets lost, the administrator needs to create a new one.

The attribute *reminder* specifies whether a user shall get automatic reminder emails by the system (1) or not (0). By default the *reminder* value is 1. The

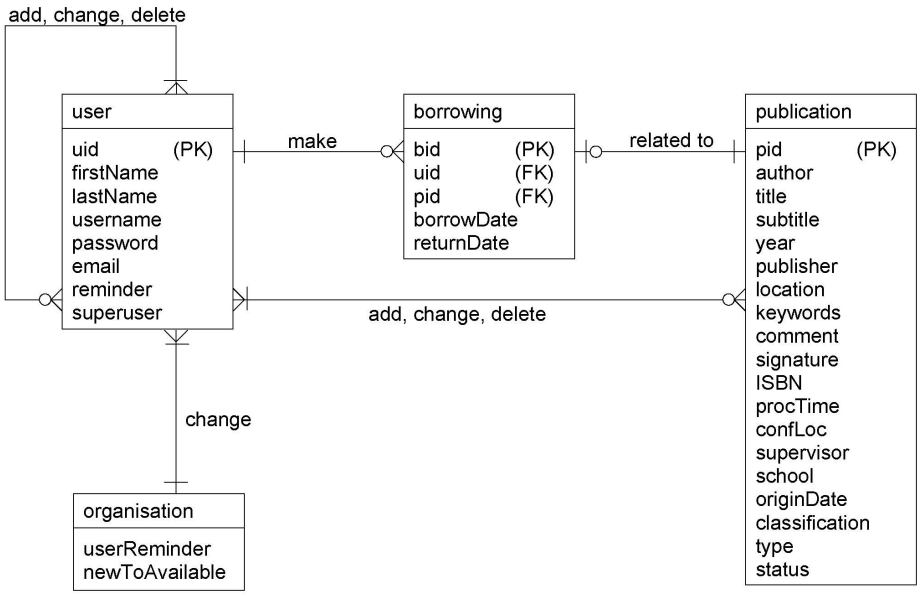

Figure 3.5: Database Structure

attribute *superuser* indicates whether a user is an administrator (1) or not (0). By default the *superuser* value is 0.

Publication. The *publication* table has six attributes that must be provided with a new row insert. These are the *title*, the *signature*, the *originDate*, the *classification*, the *type* and the *status*.

A unique *signature* should be applied by the library administrators to each library publication. The *originDate* (date when a new publication is added) is needed for the publication state. By default publications stay for 28 days in the state *New* and change then to *Available* or *Not Available*. The detailed meaning of publication *classification*, *type* and *status* has already been explained in Section 3.2.4 and Section 3.2.5.

The attributes *procTime* and *confLoc* are only allowed for proceedings, while the *supervisor* and *school* may only be used for theses.

Borrowing. The *borrowing* table links users to publications and uses both the user's and the publication's id as foreign keys. The *borrowDate* is set on the actual date when the publication is borrowed online and the *returnDate* is set 28 days later (28 being the default value for *userReminder* from the *organisation* table).

Organisation. The table *organisation* contains only the two attributes *user-Reminder* and *newToAvailable*. The first one indicates after how many days

users shall be sent a reminder email by the library system and the second one specifies how long a publication stays in the state *New*. Both values are by default *28*.

3.2.6.2 Database Users

Although the DPS Library may have many users, the database has only three. This means that not each library user has an own database account. The following paragraphs explain the access rights the three database users are granted. For more details please check Figure 6.1 and the SQL grant script at 6.1.2.

Library Default. When the searching functionality of the library system is used, the database is accessed via a default user. This one has only select rights on the database, what means that this user can only read and not change fields in the library tables.

Library User. Normal library users have restricted write rights on the database. They can change the publication status (e.g. if they borrow a book the status is set on *Checked Out*), the borrowing table and their own password.

Library Administrator. Library administrators have all privileges on the database. This means that they can read, change and delete all the tables and fields in the database. They also have the right to change the grant options of the database, i.e. they can define new access rights on the database.

3.3 System Implementation

In this section we explain how the DPS Library system is built up from the technological point of view.

As we have already mentioned in Section 3.1, the system consists of a Web application and a database, both running on a Web server. A Web application can be described as a program that sits on a Web server and is accessed via a Web browser.

What we expect from such a Web application is the creation of dynamic Web pages. Dynamic because we want the library user to interact with the library system. Mainly two technologies are use to generate such dynamic content: client-side scripts and server-side scripts or programs.

Client-side dynamic content is produced on the client's computer, as the name already indicates. The Web server and browser processes look like the following:

- the Web server retrieves a page and sends it to the client.
- the client's Web browser processes the embedded code (usually JavaScript) and renders the page to the client.

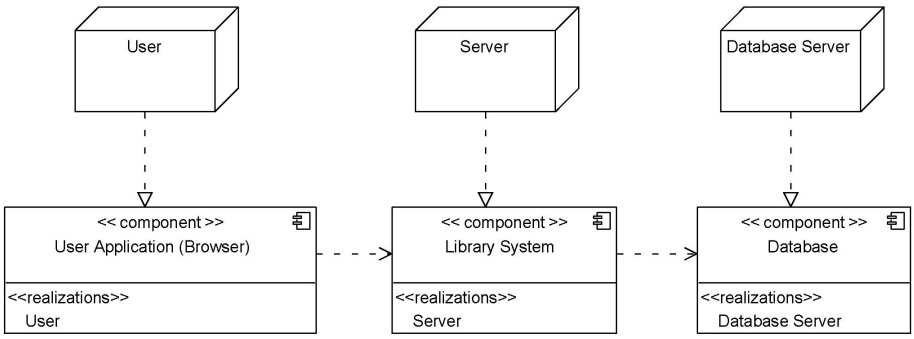

Figure 3.6: Institute Library System Components

Such client-side scripts can respond to user events like mouse clicks and form input. A JavaScript could, for example, verify the information that a user entered in a Web form, before the information is sent to the Web server. But such scripts do not support access to external databases, what is a requirement for building more complex Web applications. Additionally, client-side dynamic Web pages are not supported by all browsers. Some users, for example, disable the use of JavaScript in their browser out of security reasons.

Server-side dynamic Web pages are produced differently. The dynamic content is executed on the Web server and simple HTML pages are sent to the client:

- the client's Web browser sends user input, in form of an HTTP request, to the Web server. This request contains user input that has, e.g., been obtained from a submitted Web form.
- the server retrieves the requested script or program and executes it. This includes, the case given, making queries and updates on a database. Then it generates HTML Web pages as output.
- the server sends the HTML pages to the client's Web browser, which now renders the page for the client.

Server-side applications are mainly written in so-called scripting languages. These languages are very good in connecting components, as databases for example, and communicating with them. In Section 3.3.1.2 we will describe scripting languages in more detail.

The Institute Library System has been implemented as server-side Web application. Figure 3.6 shows the components that are implicated in the library application and how they depend on each other. As we see, the Web browser is the first component. The second component, in the middle, is the Web server on which runs the library system Web application. The third component is a database.

In the following sections we will provide more detailed information about the different system components. In Section 3.3.1 we will get an idea of some state-of-the-art technologies. We explain what technologies we have used for our implementation, outlining the reasons for this choice. Section 3.3.2 explains different details of our implementation: logging function, session variables, cron job, as well as database password protection and SQL scripts.

3.3.1 Technology Overview

We have used different popular technologies to implement the Institute Library System. In the following of this section we will explain those technologies with respect to which component they are situated in. We will provide an overview of Web languages, scripting languages, databases and Web servers and will explain our approach to each of these technologies.

3.3.1.1 User Application Component

For the Web interface we have used HTML 4.01 as specified in [Ragget et al., 1999] and CSS (Cascading Style Sheets) as defined in [Lie and Bos, 1999], both W3C recommendations. The use of a CSS stylesheet allowed us to only once define the layout and look of the DPS Library interface, so that during the further development of the system, we could adapt this style to newly created pages and could eventually change the look of the whole library pages in one step. We have based the style of the library system on the DPS homepage[1].

3.3.1.2 Library System Component

The middle component of our system is the Web application, which represents the core of our library system. The Web application consists of server-side scripts that are written in a so-called scripting programming language, or simply scripting language, and run on a Web server.

In the following of this section we will provide an overview of some prominent server-side technologies and Web servers. We will explain the choice we did in opting for PHP and Apache for the development of the Institute Library System.

Server-side Technologies

PHP, JSP and ASP actually belong to the most used server-side technologies. The following listing introduces them:

- **PHP**[2] - PHP (PHP: Hypertext Preprocessor) is an open-source widely-used scripting language. It is available for all state-of-the-art operating systems.

[1] `http://www.dps.uibk.ac.at/`
[2] `http://www.php.net/`

It is very well suited for Web development and can be embedded in HTML. [Theis, 2000] and [Krause, 2004] explain the structure of PHP, show the easy handling of Web forms and introduce the collaboration with different state-of-the-art database systems. PHP is used very often, in particular, with the MySQL database.

The scripting language is particularly recommendable when the application does not have an important business layer, as any business logic is embedded into the Web pages. This approach is ok for smaller projects, but may get unbearable for big enterprise systems. More information on PHP can be found in the PHP Manual [PHP, 2006].

- **JSP**[3] - JavaServer Pages (JSP) is a server-side technology developed by Sun Microsystems and is part of the Java 2 Platform Enterprise Edition (J2EE). Same as Java, JSPs can run on each server which provides a Java Virtual Machine.

  JSPs are, similar to PHP, HTML documents with embedded code, Java code rather. [Kaminaris and Annunziato, 2001] describes how JSPs are transformed internally to Servlets before being executed. Servlets are Java classes that accept requests and generate answers.

  The J2EE technology allows a very clear separation between business logics and Web page design. JSPs can for example interact with Enterprise JavaBeans (EJB), which is a specification that defines local and remote Java components and specifies the relationship between them. Due to the strict business logics separation, JSPs are most often used for Web-based enterprise applications. More information on JSPs and Java Servlets are available at [JSP, 2005].

- **ASP** - Active Server Pages (ASP) is a server-side technology which is proprietary to Microsoft. Whereas originally it used to only run on Microsoft servers, it has been ported to e.g. Apache servers in the meantime.

  ASP is not supported any more by Microsoft as it has been replaced by its successor technology ASP.NET. ASP.NET is a more complex technology, which works in a similar manner than the above described JavaServer Pages. As JSP, it separates the business logic from the Web pages design. More information on ASP.NET is available at [ASP, 2006].

To create the dynamic Web interface of our Institute Library System, we have chosen to use the open-source scripting language PHP.

One of the main reasons for this choice was the fact that PHP offers exactly the functionality we needed, not more and not less. On the contrary, JSP seemed us to be more advisable for bigger projects than ours. ASP did not come into a closer selection procedure, as the "classical" ASP is not supported anymore and ASP.NET is particularly meant, same as JSP, for more complex projects.

[3] `http://java.sun.com/products/jsp/index.jsp`

A second reason for our choice was the easy integration of the open-source database MySQL (see Section 3.3.1.3). PHP and MySQL are often used together for building Web applications (referring e.g. to [Theis, 2000]), what implicates that there is a long experience in connecting both and that lots of bugs, errors, and corresponding solutions, are known.

PHP as a scripting language cannot only be used for writing dynamic Web pages. It also provides a command line interface for developing e.g. shell applications or system administration tasks. This functionality was our third reason for working with PHP, as we needed to write and execute a cron job (see Section 3.3.2.4) for the support of automatic system functionalities.

Web Server

According to the Netcraft Web Server Survey [Net, 2006], the open-source Apache HTTP Server[4] is by far the most popular Web server (Market share of 61.64 percent in September/October 2006). It is followed by the Zeus Web Server[5], the Microsoft Internet Information Services (IIS) Server[6] and the Sun Java System Web Server[7] (formerly Sun ONE Web Server), all three being proprietary, non open-source, servers.

We have used the Apache Web Server for our implementation, as it is the most popular Web server and open-source at the same time. It supports all popular operating systems and, being embedded in the WAMP (Windows, Apache, MySQL, PHP) and LAMP (Linux, Apache, MySQL, PHP) packages, there again is a long experience in using the Apache server together with PHP and MySQL.

Now how can we install PHP on the Apache Web server? PHP can be used as both Apache module or CGI[8] (Common Gateway Interface) program. According to [CGI, 1998], CGI is a standard for interfacing external applications with information servers, such as Web servers. A CGI program is executed in real-time and outputs dynamic information. When PHP is installed as such CGI-program, the server starts a new process each time PHP code shall be executed.

The Apache Web server allows to directly embed the PHP interpreter, via the mod_php module. In this case the PHP module is constantly loaded in the Web server, which may bring advantages in performance.

We have installed PHP as CGI program, as otherwise it is not possible to use PHP for the development of shell scripts.

3.3.1.3 Database Component

The third component of our DPS library is the database system. We needed a database for the library publications, as well as for the library users and the

[4] `http://httpd.apache.org/`
[5] `http://www.zeus.com/products/zws/`
[6] `http://www.microsoft.com/windowsserver2003/iis/default.mspx`
[7] `http://www.sun.com/software/products/web_srvr/home_web_srvr.xml`
[8] `http://cgi-spec.golux.com/`

borrowing data. In the following listing we provide an overview of some possible database solutions:

- **Relational Databases** - A relational database consists mainly of interrelated relations/tables and their attributes/columns. It could thus also be defined as set of relations. For working with such a relational database we must build a relational model of our Institute Library System, and add the relations as tables to the database. It is a very appropriate way to e.g. model the relation between library users and publications: the borrowing.

  Two of the most popular free relational databases are MySQL[9] and PostgreSQL[10]. These databases are quite difficult to compare, as both are very stable and provide similar functionality. In both database systems, queries can be submitted in SQL (Structured Query Language). And while PostgreSQL offers a bit more on functionality, MySQL is more widespread, particularly through the delivery in the already mentioned WAMP and LAMP packages.

- **BibTeX** - BibTeX as database is not comparable to the relational databases like MySQL and PostgreSQL. It is a tool for formatting and saving lists of references which can be used through the document preparation system LaTeX. All publications are saved in one file, in normal characters and with tags identifying the different publications and attributes.

  The widely-used format for bibliographies makes it easy to cite sources. BibTeX bibliographies can be formatted in different manners, according to a given style file. All publication types that can be entered come with a predefined set of attributes which may be mandatory or optional.

  Although BibTeX would be a good choice for a mere publication database, it is not adequate for the management of library users and borrowing data.

We have chosen to use a MySQL database for the implementation of the DPS Library system. We preferred the MySQL database system over the PostgreSQL system because MySQL is quite widespread together with PHP and Apache.

Initially we would have liked to build a BibTeX file of all library publications, which could then be used by the members of the Distributed Parallel Systems Group for citations in their own publications. But as this would have meant to integrate two databases, BibTeX for publications and MySQL for users and borrowing data, we chose to go with MySQL for everything.

To get a better overview of the database structure and to better handle the MySQL administration, we have used the open source tool phpMyAdmin[11]. This tool allows the administration of MySQL through a Web interface. During the test period we managed the database (create, change and delete tables, fields and inserts, etc...) through it and have found it very practical and user friendly.

[9] `http://www.mysql.com/`

[10] `http://www.postgresql.org/`

[11] `http://www.phpmyadmin.net/home_page/index.php`

3.3.2 Implementation Details

In this section we provide details of our implementation. Section 3.3.2.1 goes into PHP security issues, Section 3.3.2.2 explains the system's logging function and Section 3.3.2.3 describes the use of session variables. In Section 3.3.2.4 we explain the use of a cron job to perform automatic system actions and in Section 3.3.2.5 we provide some details concerning the database implementation.

3.3.2.1 PHP Security

We have tried to make a quite secure configuration of PHP, referring to [PHP, 2006, Chapter IV]. This is important, as PHP can execute commands on the server and has access to network connections and data files.

register_globals. We have set the default `register_globals` value in the PHP configuration file to `Off`(since PHP version 4.2.0 it is off by default). When this value is on, internal variables are defined by the script itself. This way programmers cannot always know for sure where variables come from. When the `register_globals` value is disabled, the programmer can again exactly know all variables.

Global Arrays. When a user submits a form, all data in this form becomes available to the PHP script. The most secure way to access these variables is by using the reserved global arrays `$_GET`, `$_POST` or `$_COOKIE`. As we always know, in our implementation, where a variable comes from, we can get it in the appropriate way (see example PHP file at 6.2).

3.3.2.2 Logging Function

Our system requires each user to log into the system to be able to execute actions which alter the database, e.g. borrow and return books. We have implemented a PHP function (see file at 6.4) that logs all user logins and all executed database changes. All automatic system actions, like the sending of reminder emails, are also logged.

Each month a new log file is created. Section 6.5 shows such an example log file.

3.3.2.3 Session Variables

To allow users to log into the system and to be able to preserve data across subsequent accesses, we need session support. Thus we have implemented the ILS with PHP session variables. This way we can maintain a certain environment for logged-in users and provide a user-friendlier library system.

Whenever a visitor accesses the DPS Library website, PHP checks explicitly (via `session_start()`) if he has already a session id (a unique id). If not, he gets one. The id is being sent in a cookie and needs to be stored by the user's Web browser. If the id has already been sent, the saved environment is restored.

The use of session variables implies that the user needs a browser that accepts cookies. In most state-of-the-art browsers users can choose whether they accept cookies or not, so that this should not be a problem.

3.3.2.4 Cron Job

A cron job is a script that is executed regularly (in scheduled time intervals) by the cron daemon of the operating system. We have written a PHP script that needs to be executed once a day (see script at 6.3). Figure 6.3 illustrates the execution of the script.

The script is responsible for two tasks:

- **Send Reminder Email** - The script checks if a user needs to be sent a reminder email and sends it the case given.

  As sender address the email of the library administrator is taken. The email's subject is *DPS Library Reminder* and it indicates the due book and its return date.

- **Change Publication Status** - The script checks if a publication's status has to be changed from *New* to *Available* (in case of a book) or *Not Available* (for all other publications), and changes it the case given.

Both of these system actions are logged in the current log file.

3.3.2.5 Database Implementation Details

In the following we explain how we protect user passwords and provide an overview of our SQL scripts.

Password Protection Each DPS Library user has its own password. Before this password is being stored in the database, we encrypt the password using the MD5 algorithm (*RSA Data Security, Inc. MD5 Message-Digest Algorithm*), as specified in [AB, 2006, Chapter 12.9.2]. This way not even the administrators can access the user passwords.

SQL Scripts. We have provided SQL scripts for building the database, adding tables and fields, granting user rights, making some initial user inserts and again dropping the database (see Section 6.1 for the main scripts). Normally the library administrator should not need to alter the provided database structure. To make changes in the database content, like to change user information or the email reminding time, the administrator can use the normal DPS Library Web interface.

Chapter 4

User Manual

In this chapter we show how to use the developed Institute Library System (ILS). Section 4.1 describes the Graphical User Interface (GUI) of the DPS Library and Section 4.2 describes the error handling of the system.

In Section 4.3 we explain how to install and configure the ILS.

4.1 Graphical User Interface

The following will explain the usage of the DPS Library. This includes the visitor, user and privileged user functions.

As explained in the library functionality description (see Section 3.2.2), visitors can only use the library search functions, whereas registered users can borrow and return books and change their password. Privileged users can execute privileged functions. In the following manual we will show only once, that the visitor needs to login to the system to be able to execute user functions. For the rest of the manual this is assumed to be known.

For each operation that causes a change in the library database (like borrowing and returning books, adding, changing or deleting library publications and users), the user gets a confirmation after successful execution. In the following we will only once show such a confirmation and omit it for the rest.

Start Working

The main page of the DPS Library (Figure 4.1) provides an overview of the library functionality.

Search Publications

The DPS Library provides two different search modes. One is the simple search (Figure 4.2), which does search the database for given keywords. If more than one keyword is indicated, the search function links them by a boolean OR operation.

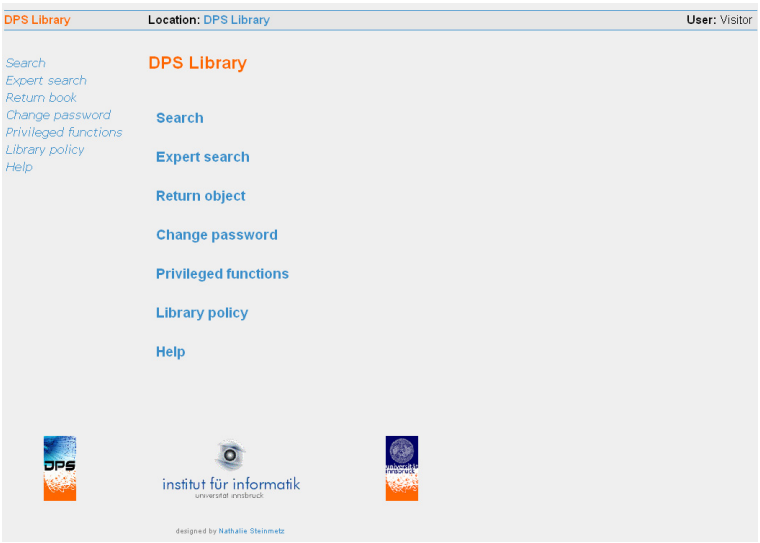

Figure 4.1: DPS Library

The simple search mode also offers the possibility to get a list of all library publications, by clicking on the *Show all objects* button.

The (advanced) expert search (Figure 4.3 and Figure 4.4) allows visitors to search for one or more specific authors and keywords, a specific title, type, classification and more. If more than one author or keyword is indicated, the expert search function links them by a boolean AND. For some of the input fields, grey text in brackets specifies how the input format must be.

The search results, whether coming from the simple search or expert search mode, are shown in a list (Figure 4.5). In this list the visitor can also see if a book is already checked out and if so, until when. If no result is found, this is indicated as shown in Figure 4.6.

Figure 4.2: Simple Search

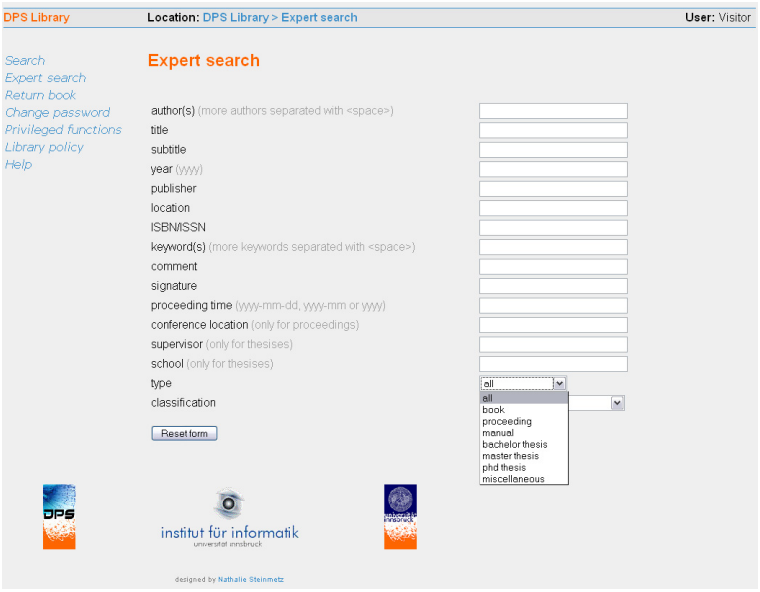

Figure 4.3: Expert Search (1)

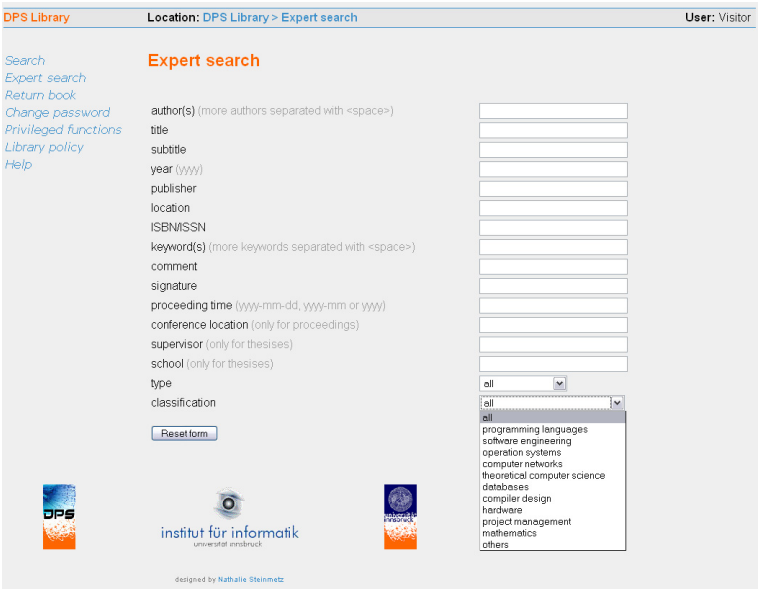

Figure 4.4: Expert Search (2)

Figure 4.5: Search Results

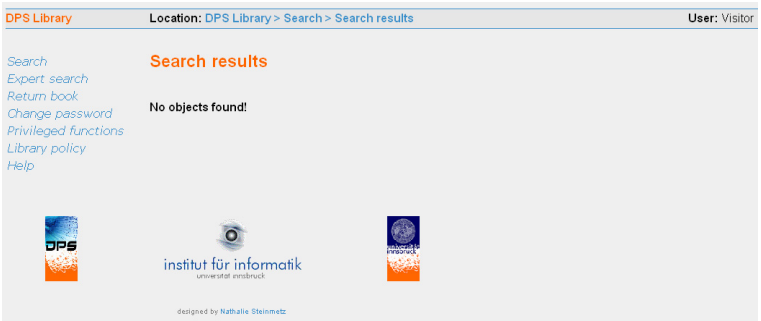

Figure 4.6: No Search Results

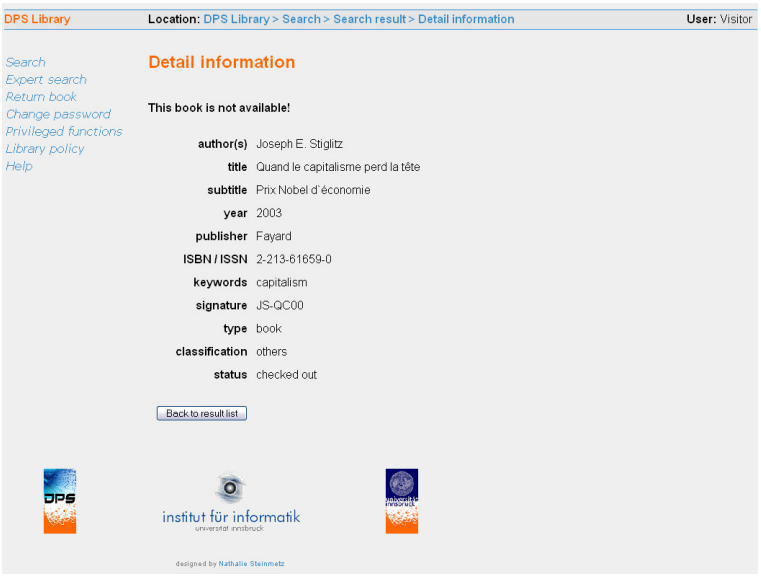

Figure 4.7: Not Available Object Detail

By clicking on a title in the result list, the visitor gets detailed information about the chosen publication. If this publication is not available to be borrowed (e.g. it is already checked out by another user), this is indicated (Figure 4.7). If the publication is available, the visitor can borrow it (Figure 4.8).

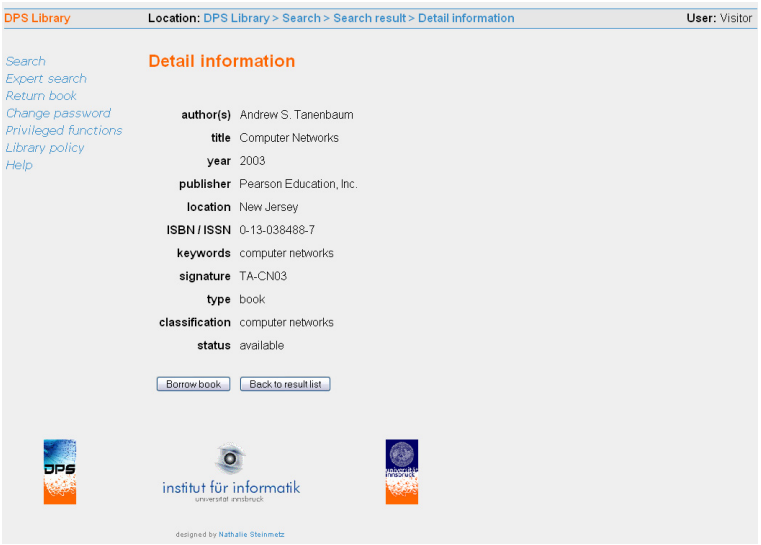

Figure 4.8: Available Object Detail

Borrow Books

Only registered library users can borrow books. When the visitor chooses to borrow a book, he first needs to login to the system to be able to (Figure 4.9).

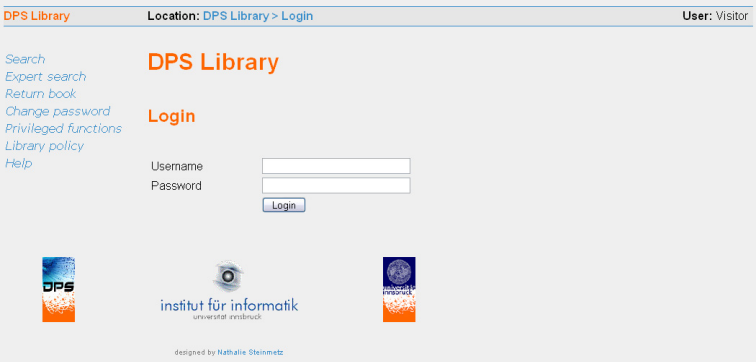

Figure 4.9: Library Login

The user needs to confirm if he wants to borrow this book (Figure 4.10). After having done so, he gets a confirmation from the system (Figure 4.11).

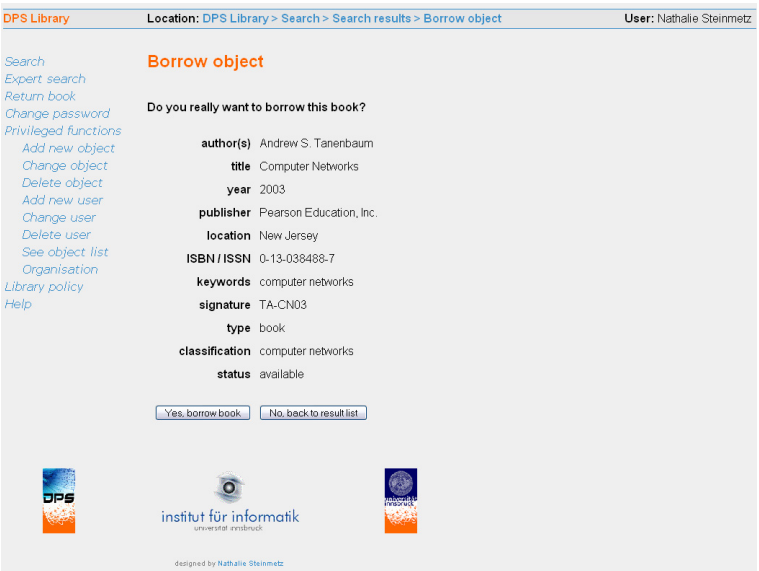

Figure 4.10: Borrow Object

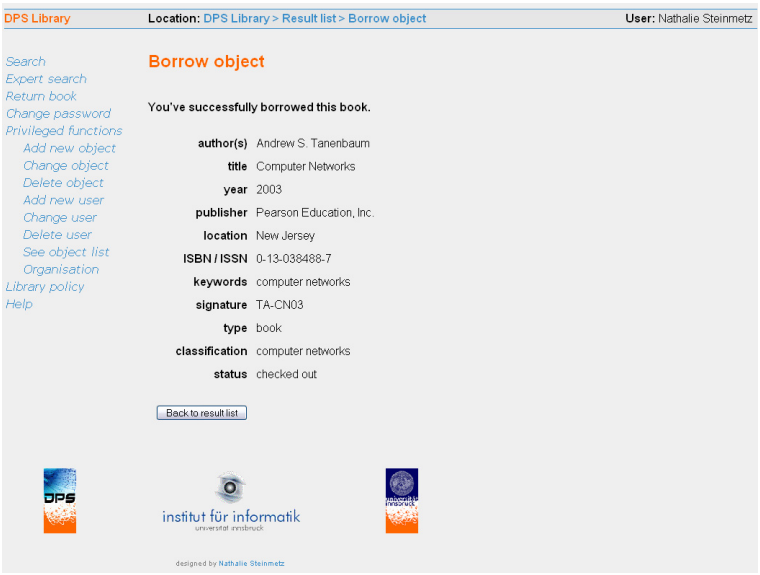

Figure 4.11: Borrow Object Confirmation

Return Books

Books can be returned either one by one or all at once. By clicking on the *Return book* link on the left part of the page or on the main DPS Library page, the user gets to a page where he sees all the books that he has currently borrowed (Figure 4.12).

Figure 4.12: Return Book List

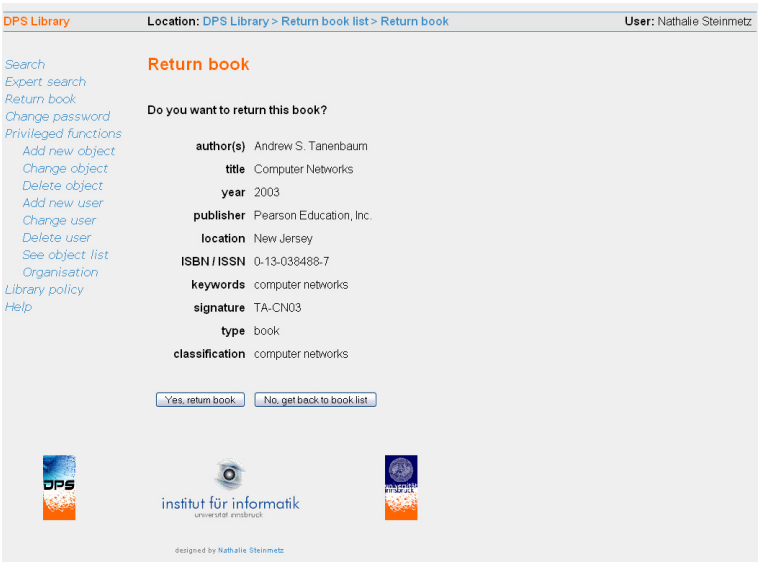

Figure 4.13: Return Book

The user can choose to return one of the enlisted books, by clicking on its title or signature. He gets a form with the details of the book and needs to confirm his wish to return it (Figure 4.13).

To return all borrowed books at once, the user can click on the *Return all books* button below the list. Again, this choice needs to be confirmed before being executed (Figure 4.14).

Figure 4.14: Return All Books

Change Password

The DPS Library user has only limited user account management capabilities. All he can change on his own is his system access password. By clicking on the *Change password* link on the left part of the page or on the main DPS Library page, he gets to a form (Figure 4.15) where he needs to enter his old password and twice his new password.

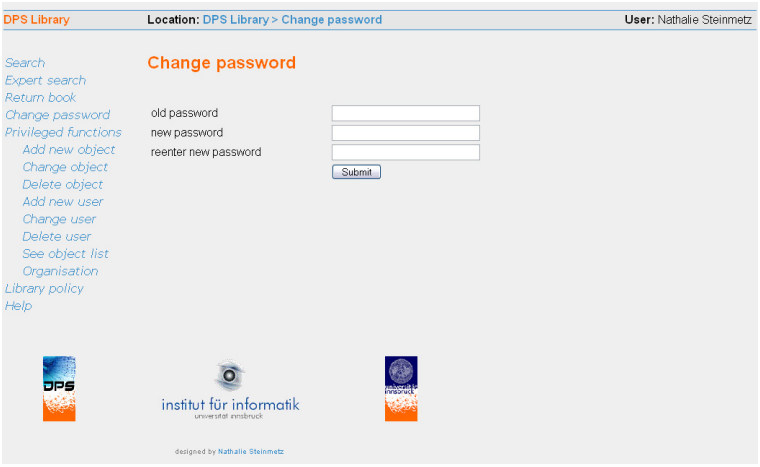

Figure 4.15: Change Password

Privileged Functions

The library administrators have privileged functions within the DPS Library system. Figure 4.16 provides an overview of these privileged functions.

Privileged Functions - Publications

The administrator can add new publications to the database, and can change or delete existing publications. Figure 4.17 shows the form that has to be filled out in order to add a new publication. The grey text in brackets behind some of the fields indicates whether a field is required, shows how the input needs to be formatted or specifies for which type of publication this field is allowed.

To change an existing publication, the administrator can either search for it or choose it from the complete publication list (Figure 4.18). If he searches for a publication he gets a list with results (Figure 4.19) or, in case his search is fruitless, is informed that there are no results to this search (Figure 4.20). By clicking on the title of a publication in the publication list the administrator gets to a form where he can change the publication information (Figure 4.21).

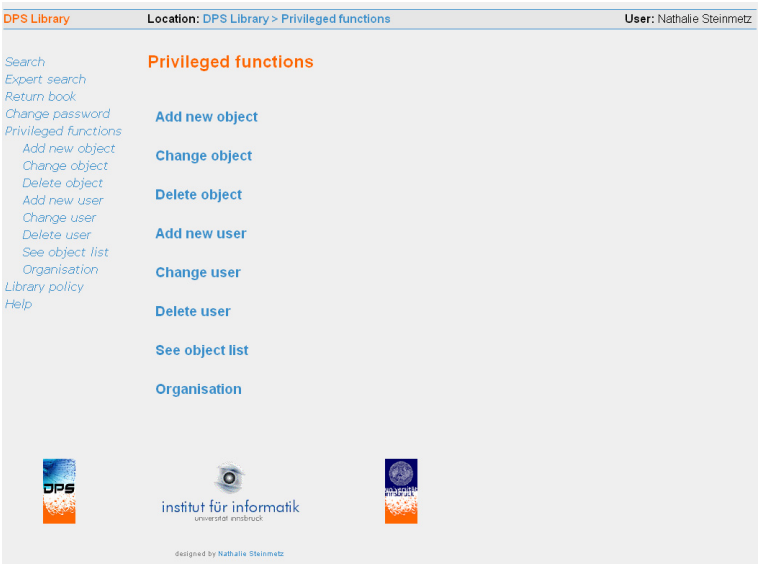

Figure 4.16: Privileged Functions Overview

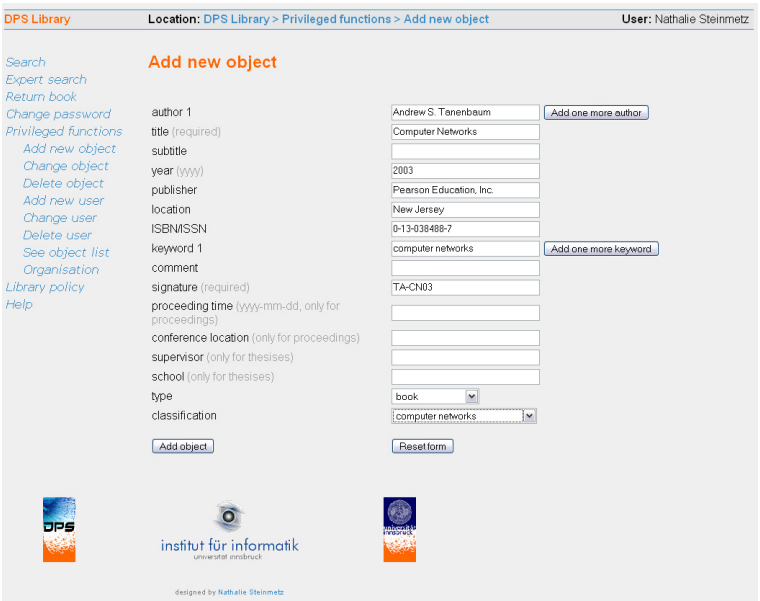

Figure 4.17: Add New Object

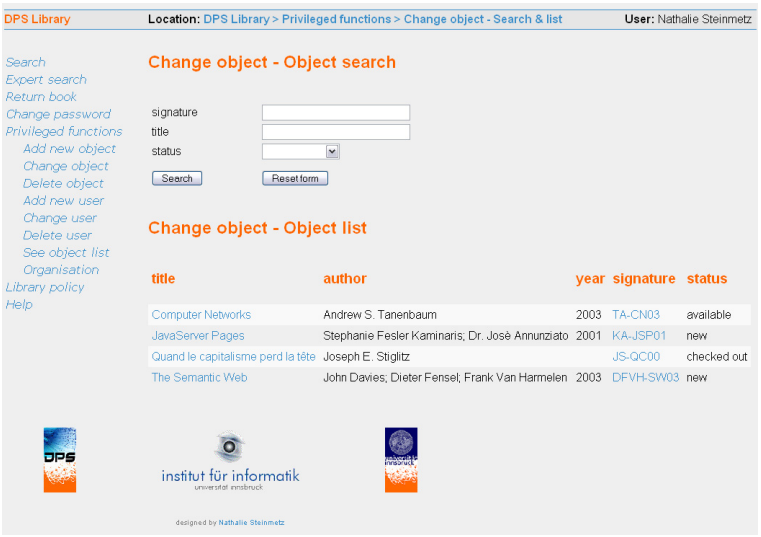

Figure 4.18: Change Object - Search & List

Figure 4.19: Change Object - Search Results

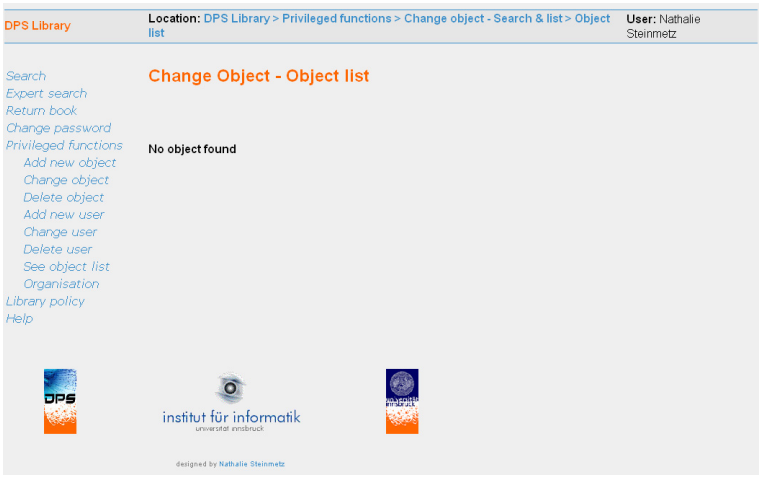

Figure 4.20: Change Object - No Search Results

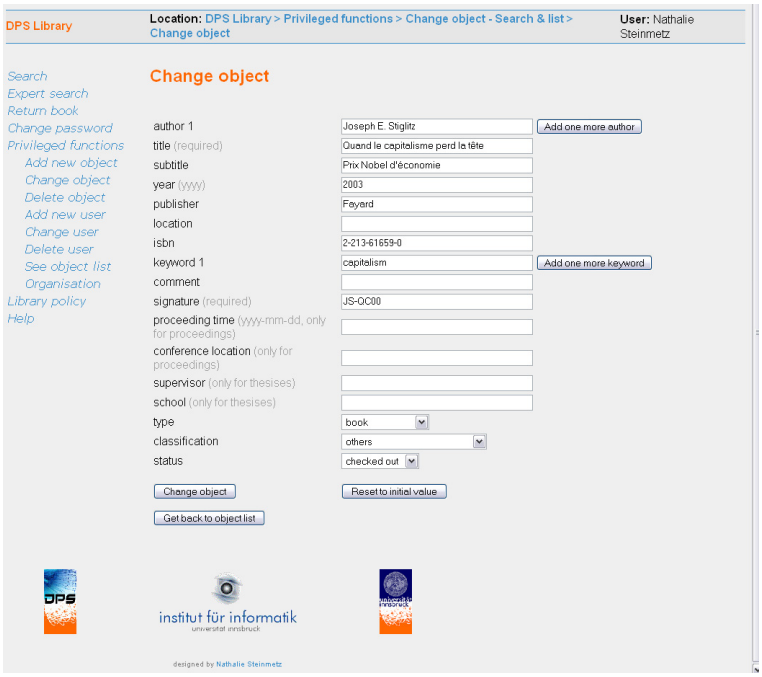

Figure 4.21: Change Object Information

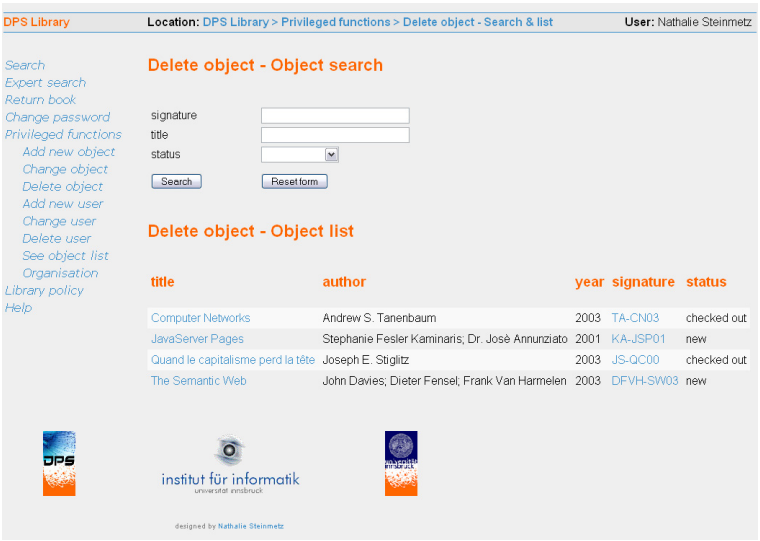

Figure 4.22: Delete Object - Search & List

To delete a publication from the database, the procedure is the same. The administrator can either search for it or choose it from the complete publication list (Figure 4.22). By clicking on the title of a publication in the publication list he gets the detailed publication information and can delete it (Figure 4.23). Before the book is deleted from the system, the action has to be confirmed by the administrator (Figure 4.24).

The administrator can get an overview of all library publications. The list provides information about the publications author/s, title, year, type and status (Figure 4.25). If a book is borrowed, the list specifies the corresponding user, borrowing date and return date. The list of publications can be ordered according to any of the mentioned columns (e.g. ordered by publication title in Figure 4.25). When the list is ordered by user, borrow date or return date, it shows only those publications that actually are borrowed (Figure 4.26).

Privileged Functions - Library Users

The administrator can add new library users to the database, and can change or delete existing users. Figure 4.27 shows the form that has to be filled out in order to add a new user. The grey text in brackets specifies which fields need to contain unique values (unique in the library system).

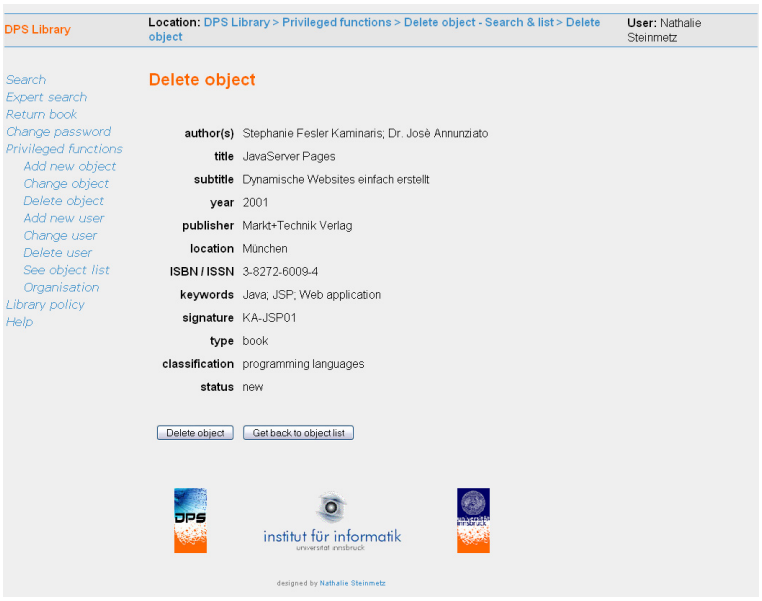

Figure 4.23: Delete Object

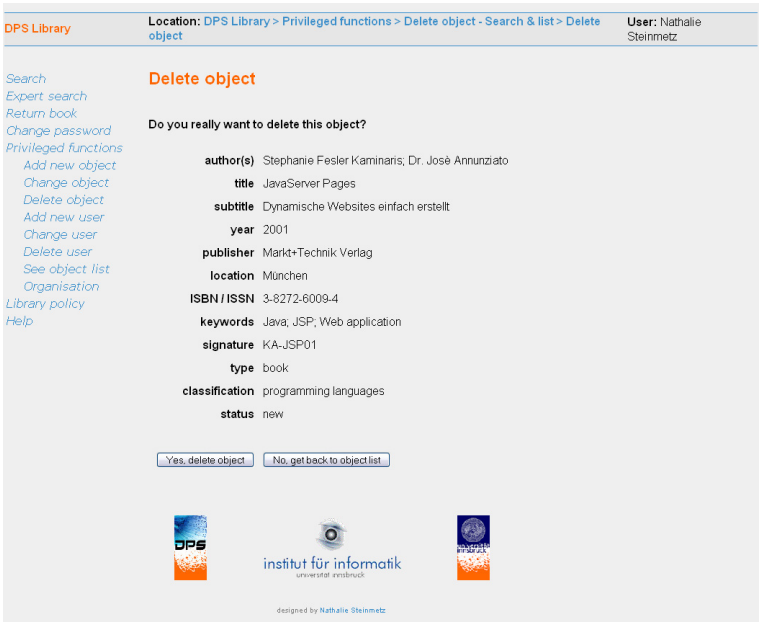

Figure 4.24: Delete Object - Confirmation Needed

Figure 4.25: Object Overview - Ordered by Title

Figure 4.26: Object Overview - Ordered by User

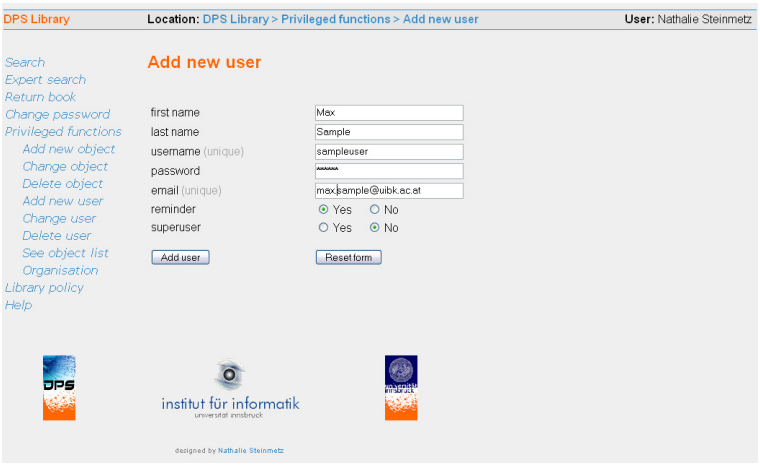

Figure 4.27: Add New User

To change the information of an existing user, the administrator can either search for it or choose it from the complete user list (Figure 4.28). If he searches for a user he gets a list with results (4.29) or, in case his search is fruitless, is informed that there are no results to this search (Figure 4.30). By clicking on a user in the user list the administrator gets to a form where he can change the user information (Figure 4.31).

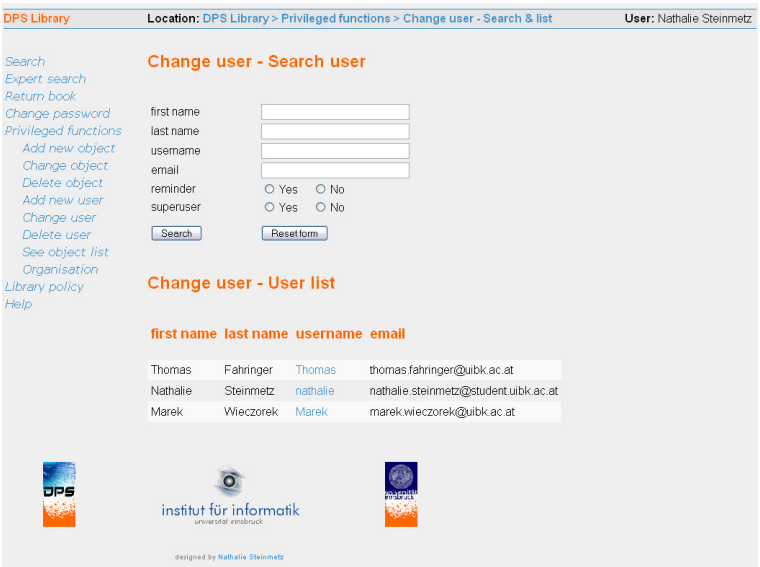

Figure 4.28: Change User - Search & List

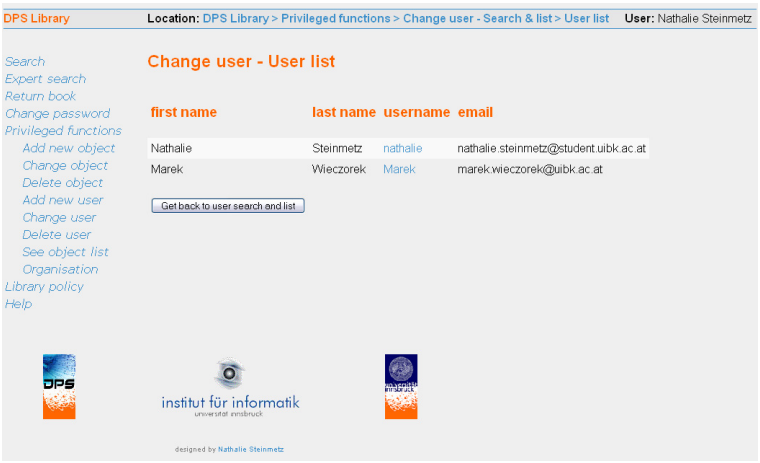

Figure 4.29: Change User - Search Results

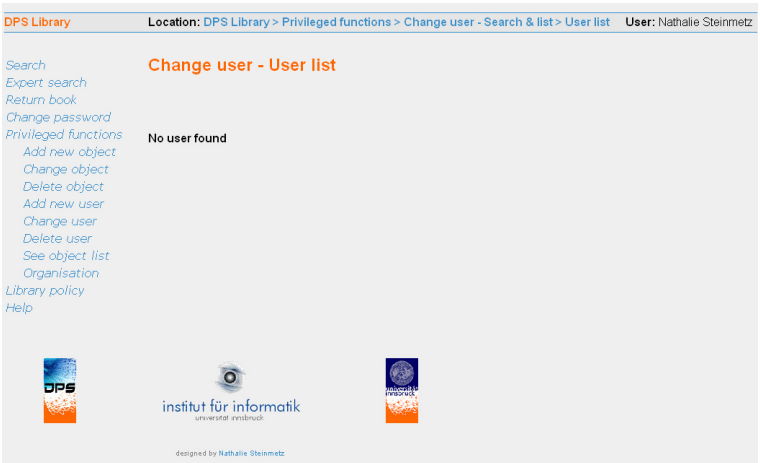

Figure 4.30: Change User - No Search Results

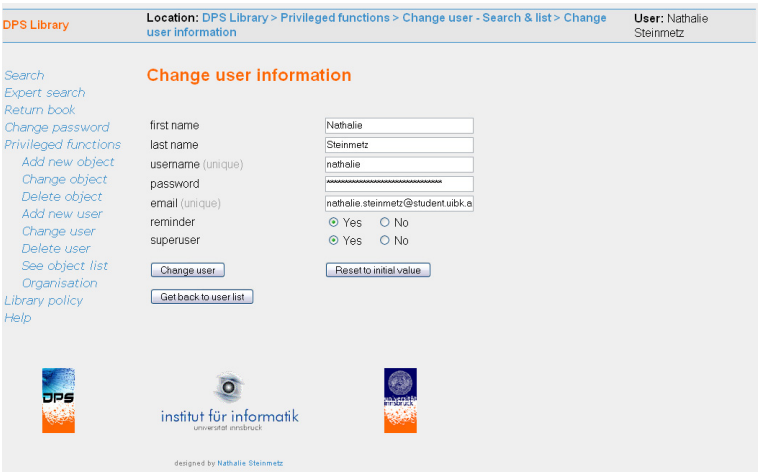

Figure 4.31: Change User Information

To delete a user from the database, the procedure is the same. The administrator can either search for it or choose it from the complete user list (Figure 4.32). By clicking on a user in the user list he gets the detailed user information and can delete him (Figure 4.33). This choice has to be confirmed before being executed by the system (Figure 4.34).

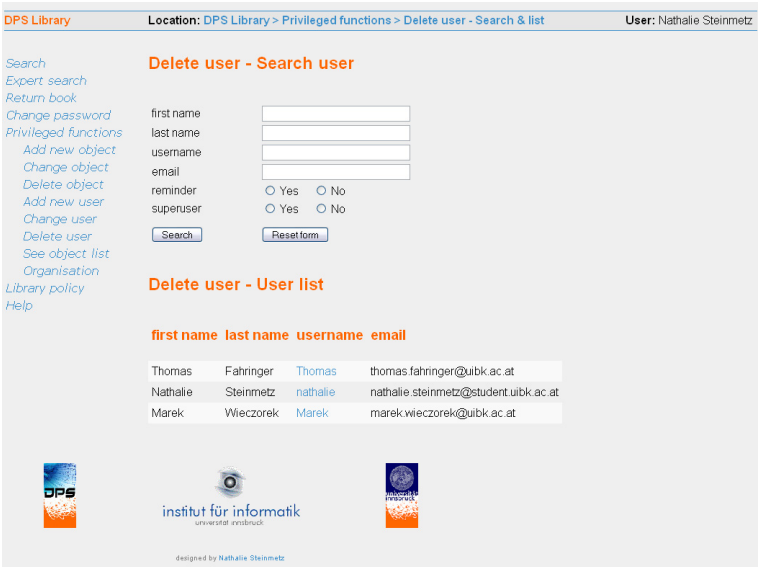

Figure 4.32: Delete User - Search & List

Figure 4.33: Delete User

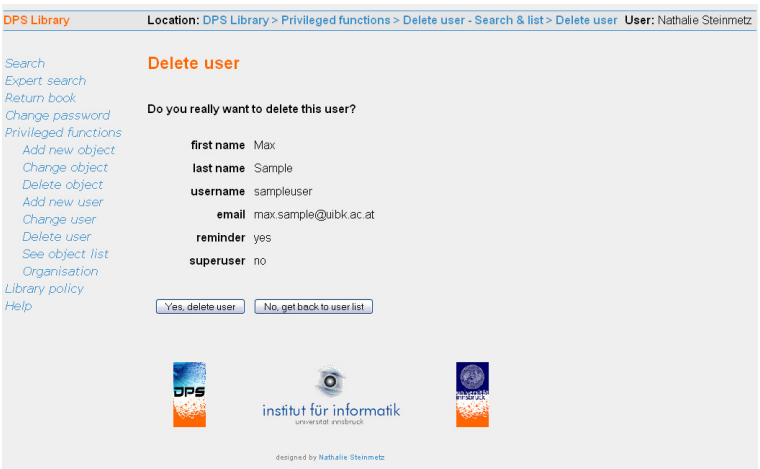

Figure 4.34: Delete User - Confirmation Needed

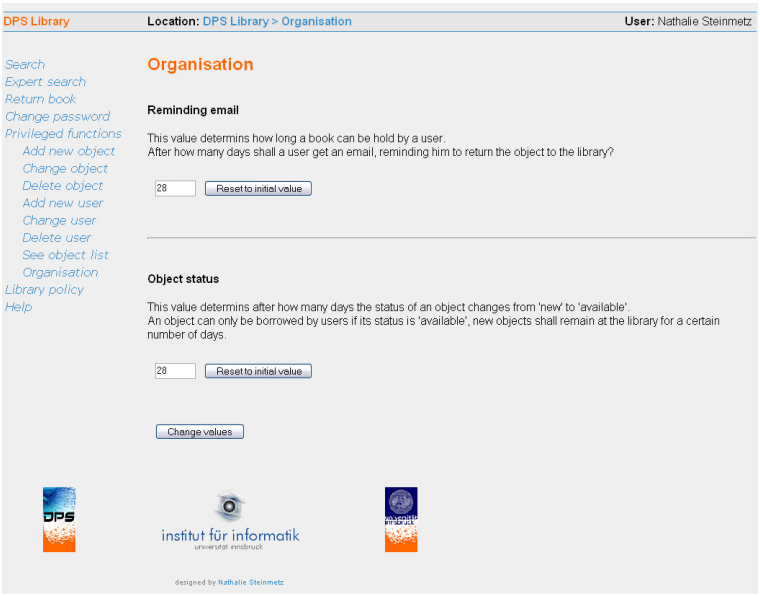

Figure 4.35: Organisation - Automatic System Functionality

Privileged Functions - Automatic System Functionality

Figure 4.35 shows two values concerning automatic system actions that the administrator can change.

The first one is the time period after which users are reminded when borrowed books are due to be returned to the library. After this time period, which by default is 28 days, the DPS Library system automatically sends an email to the corresponding users to remind them to bring back the books.

The second value concerns the time period after which a new publication's status is set to *Available* or *Not Available*. As long as a book is new, it cannot be borrowed and checked out. By default this time period is 28 days.

Library Policy

Figure 4.36 shows the library policy that informs users of the procedures how to check out, return and sort library publications.

Library Help

The DPS Library system provides help for the library users and administrators on how to use the library functionalities (Figure 4.37, Figure 4.38, Figure 4.39 and Figure 4.40).

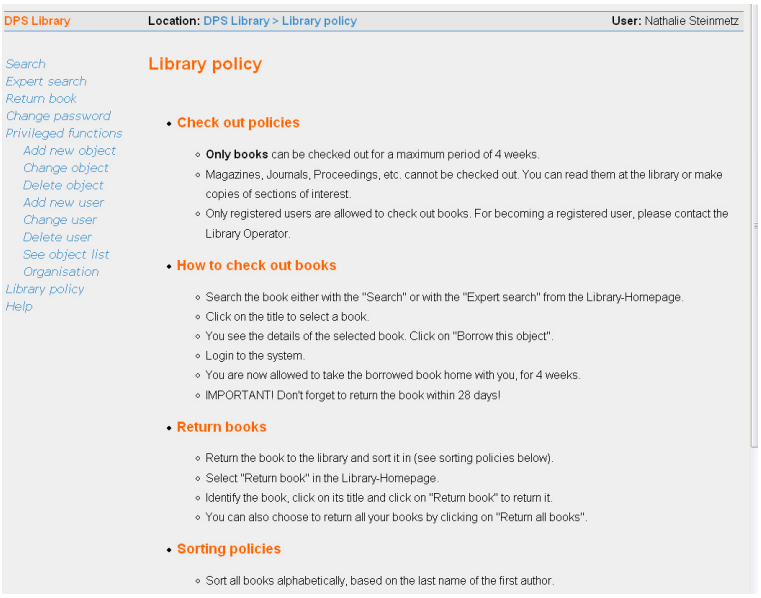

Figure 4.36: Library Policy

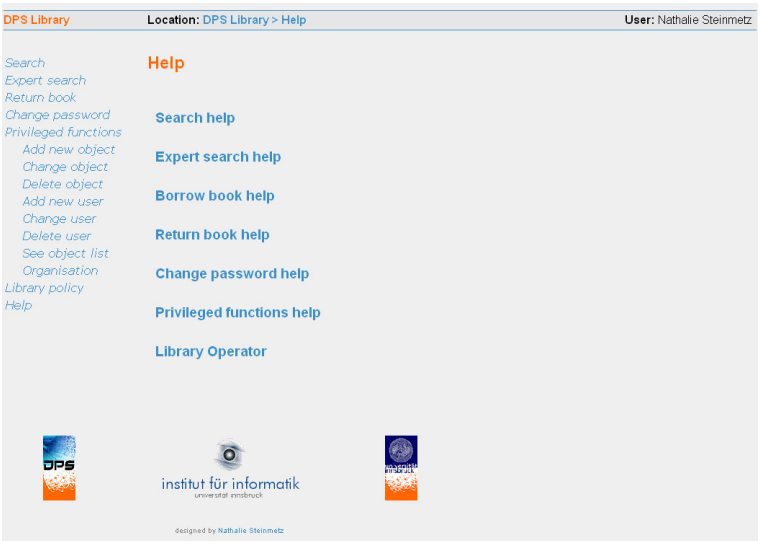

Figure 4.37: Library Help Overview

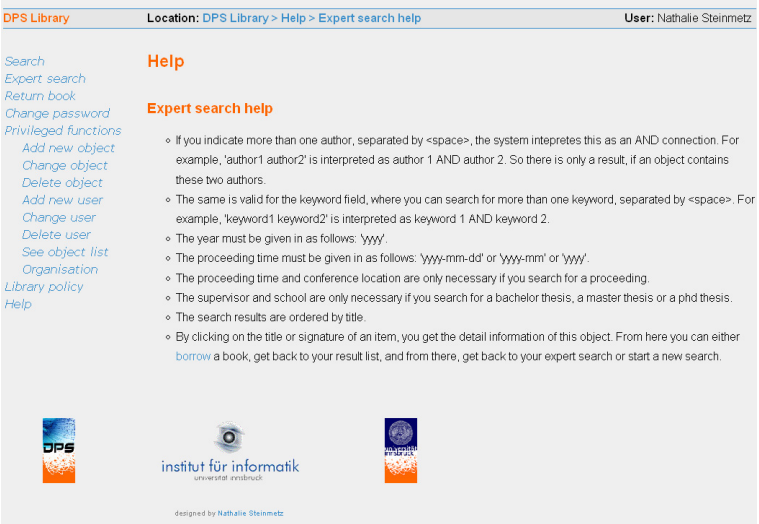

Figure 4.38: Expert Search Help

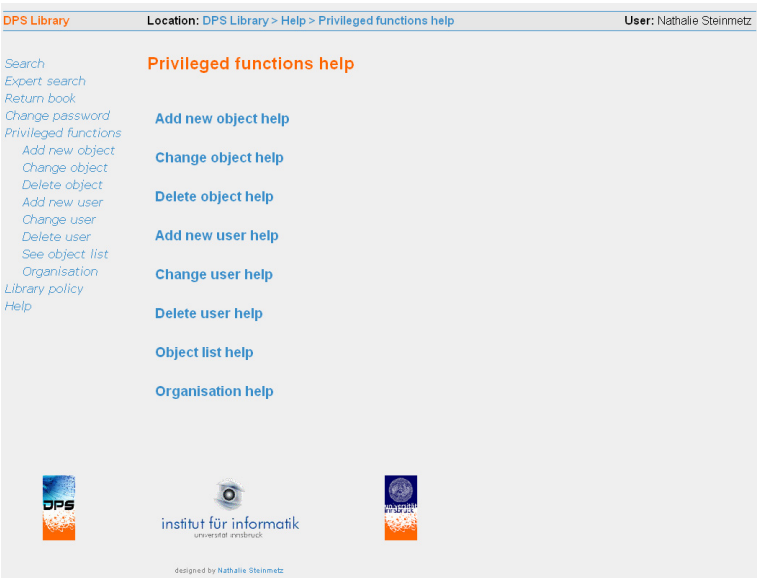

Figure 4.39: Library Privileged Functions Help Overview

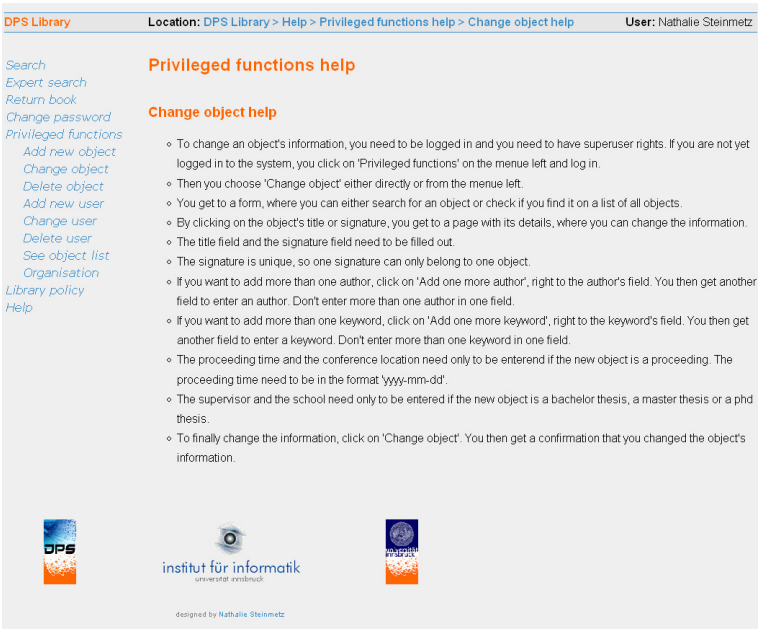

Figure 4.40: Change Object Help

4.2 Error Handling

The following provides an overview of the error handling of the Institute Library System. Section 4.2.1 shows how the user interface reacts on wrong or missing user input and Section 4.2.2 explains how the system logs all upcoming errors.

4.2.1 GUI Error Handling

When the PHP script detects an error, we redirect the browser to a specified page and submit an error code via `GET`. The error code leads then to an appropriate error message output.

The redirection is made using the `Location` header:

```
// the username must not be longer than 16 letters
 if (strlen($user_name) > 16){
   header("Location: add_user.php?error=4");
   exit;
 } // end if
```

The following sections provide an overview of some error messages:

No Administrator Rights

Figure 4.41 shows what happens if a normal user tries to access the privileged functions.

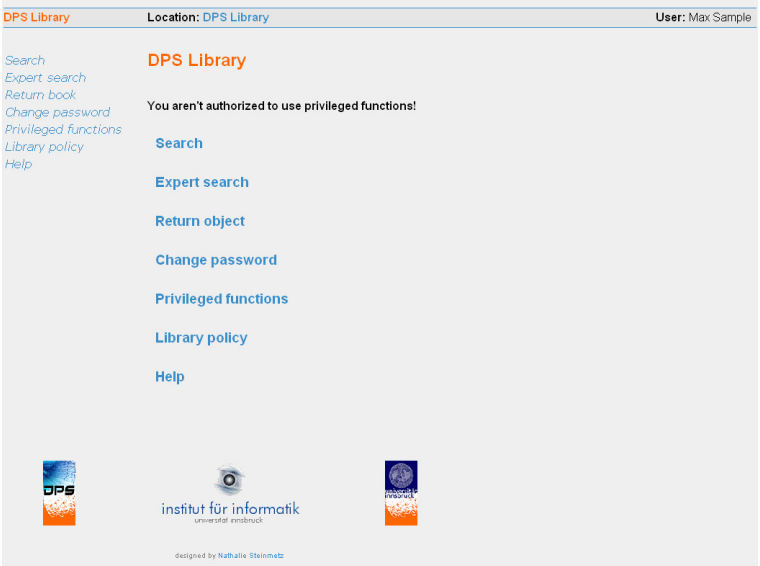

Figure 4.41: No Administrator Rights

Missing Input or Data

The errors in this section happen when mandatory input or some data is missing. In Figure 4.42 no search word was indicated, in Figure 4.43 the required field title is missing and in Figure 4.44 we see a message that indicates that the user has no books to return.

Wrong Input

In this section the errors belong to submitted wrong input data. Figure 4.45 indicates that either the given password or username must be wrong. In Figure 4.46 an inappropriate field has been filled out, *school* is not allowed as attribute for publications of type *book*.

Figure 4.47, Figure 4.48 and Figure 4.49 treat error messages that can arise when a user tries to change his system password.

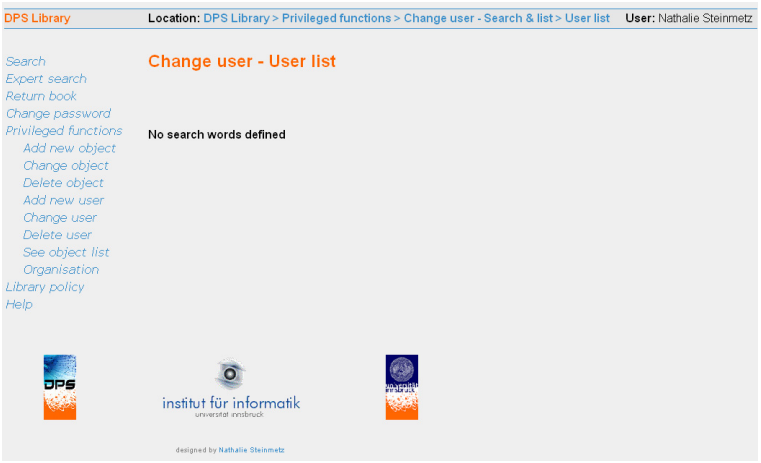

Figure 4.42: Missing Search Word

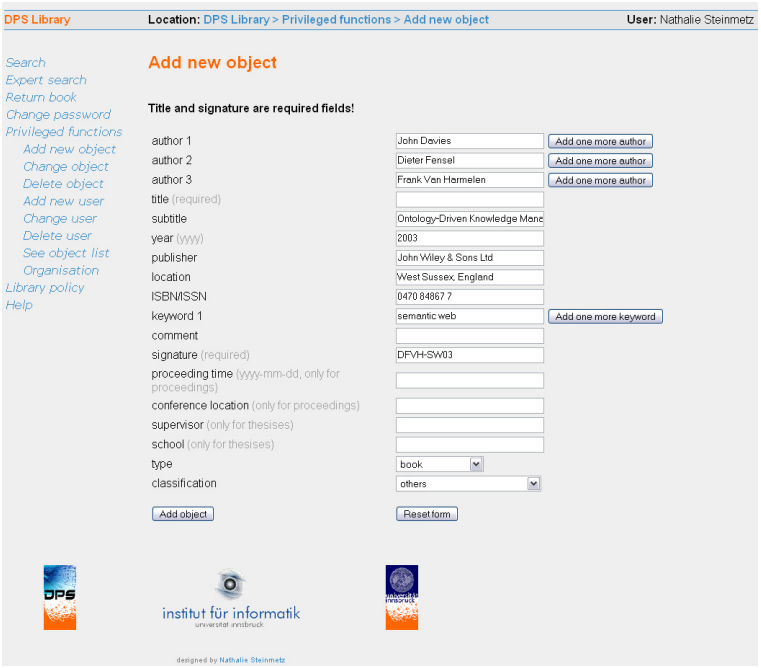

Figure 4.43: Required Field at New Publication Input

Figure 4.44: No Books to Return

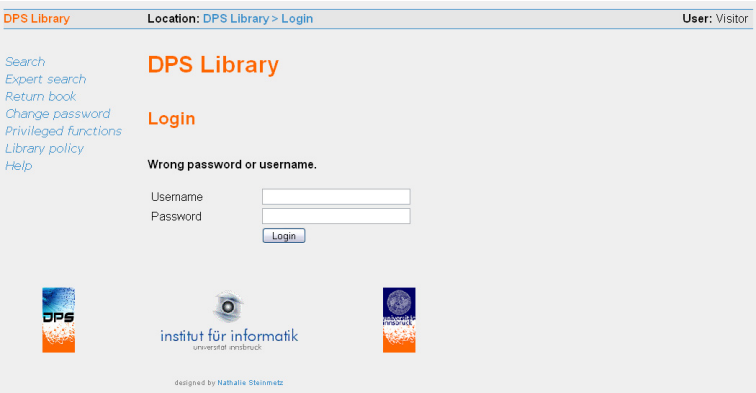

Figure 4.45: Wrong Password or Username

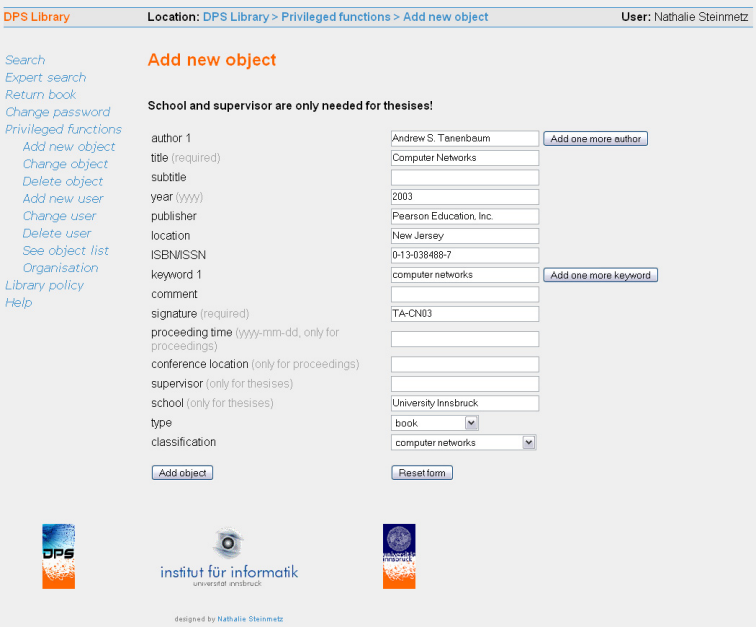

Figure 4.46: Wrong Fields Have Been Filled Out

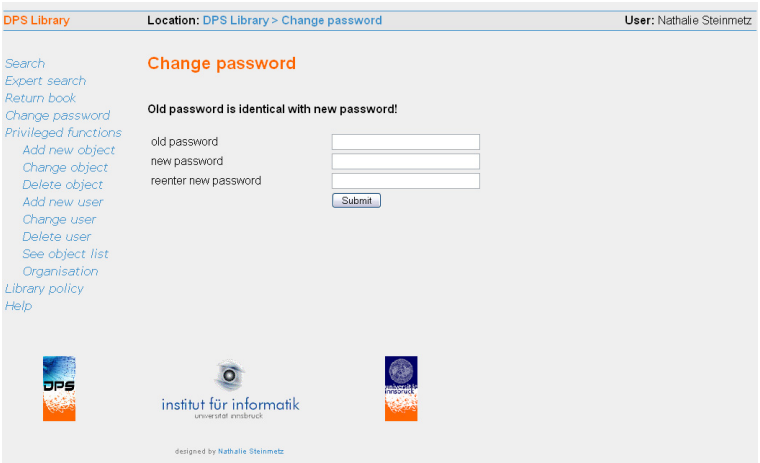

Figure 4.47: Old Password Is Identical With New Password

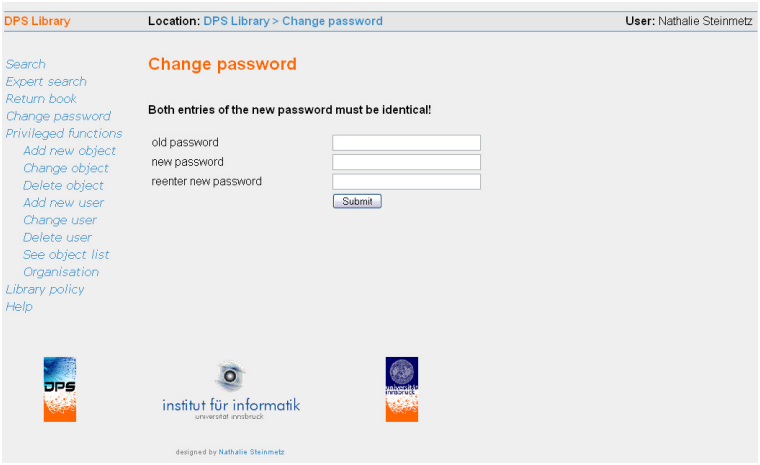

Figure 4.48: New Password Entries Are Not Identical

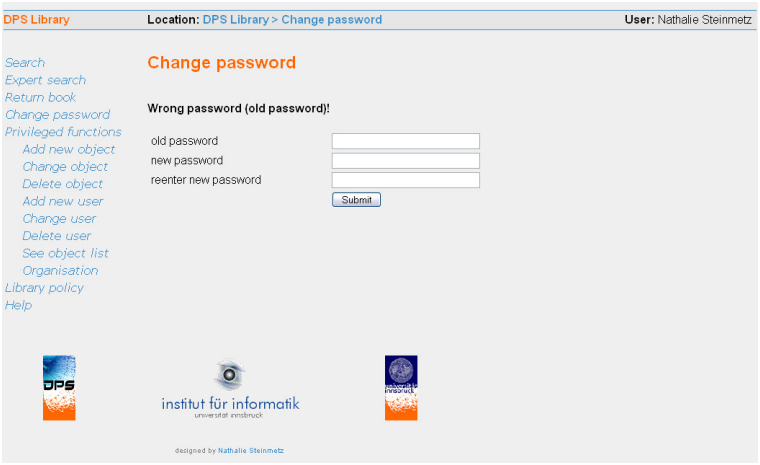

Figure 4.49: Wrong Password Entered

Database Errors

The following error messages are related to database errors. Figure 4.50 shows the error message that arises when PHP could not establish a connection to the database.

Figure 4.50: Connection Failed

The error messages in Figure 4.51 and Figure 4.52 appear when a database input could not be executed successfully. In this case, the database stays unaffected.

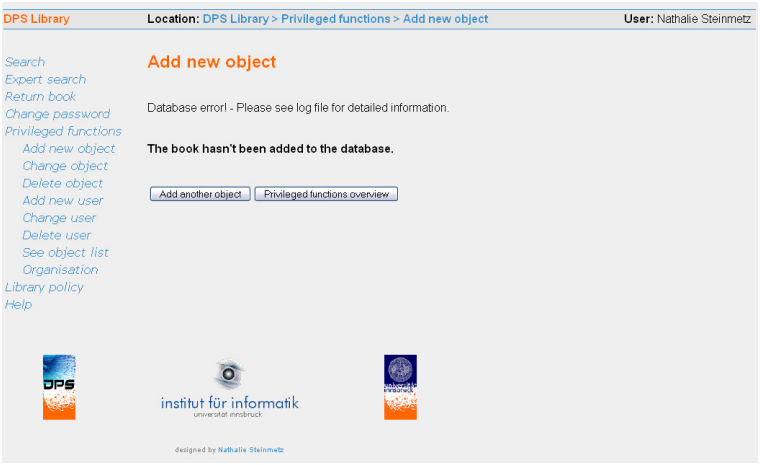

Figure 4.51: Publication Has Not Been Added to Database

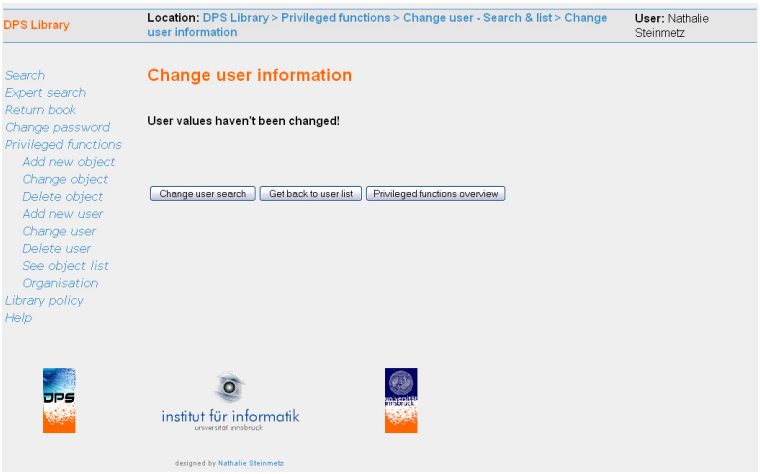

Figure 4.52: User Information Has Not Been Changed

4.2.2 System Error Log

We have two different errors logs in the system:

- **PHP Error Log** - This functionality is provided by PHP and needs to be activated in the PHP configuration file.

  If the value `log_errors` is `On`, any error that occurs during the execution of a PHP script is logged to the server's error log.

- **ILS Log Functionality** - Using the log functionality that we described in Section 3.3.2.2, we log each database error that happens during the execution of our PHP scripts. The following example shows how an upcoming error produces an error message and a call to the logging function:

```
// send query to database
  $res1 = mysql_db_query("dpslibrary", "SELECT username FROM
          user WHERE username = '$user_name';");
  if (mysql_errno() != 0) {
    // logging function call
    $string = "Error: " . mysql_errno() . ": " .
            mysql_error(). "\n";
    library_log($string);
    echo $string;
  } // end if
```

4.3 Software

In the context of this thesis we have built a Web application for an Institute Library. To install the library system, you need to download, install and configure the Apache Web Server, PHP and MySQL. The installation of phpMyAdmin is optional. The following listing explains where the software can be downloaded and under which license it is released:

- **Apache HTTP Server 2.0.52** - All Apache releases can be downloaded from `http://httpd.apache.org/download.cgi`. The Apache Web server is released under the Apache License[1].
- **PHP 4.3.10** - All available PHP releases can be downloaded from `http://at.php.net/releases/index.php`. PHP is released under the PHP License[2].
- **MySQL 4.0.21** - All available MySQL releases can be downloaded from `http://downloads.mysql.com/archives.php`. MySQL is released both under the General Public License (GPL) [3] and under a commercial license[4].
- **phpMyAdmin 2.6.1** - phpMyAdmin releases can be downloaded from `http://www.phpmyadmin.net/home_page/downloads.php`. The software is released under the GPL License[5].

The indicated program versions are the ones against which we tested our system. To install a stable system, we recommend using these versions.

The following section gives an overview of the installation and configuration of the above mentioned programs. The configuration we propose in the following, corresponds to the configuration of our test environment and is recommended for a stable system installation.

4.3.1 Installation and Configuration

Detailed installation instructions are available on the official Apache[6], PHP[7], MySQL[8] and phpMyAdmin[9] Web sites. We configured Apache and MySQL according to the official instructions and applied only for PHP an own configuration.

[1] `http://www.apache.org/licenses/LICENSE-2.0`
[2] `http://at.php.net/license/3_01.txt`
[3] `http://www.opensource.org/licenses/gpl-license.php`
[4] `http://www.mysql.com/company/legal/licensing/commercial-license.html`
[5] `http://www.opensource.org/licenses/gpl-license.php`
[6] `http://httpd.apache.org/docs/2.0/en/install.html`
[7] `http://www.php.net/manual/en/install.php`
[8] `http://dev.mysql.com/doc/refman/4.1/en/installing.html`
[9] `http://www.phpmyadmin.net/documentation/#quick_install`

PHP needs to be installed as CGI (Common Gateway Interface) or CLI (Command Line Interface) program, as we want to execute a PHP cron job. We have changed the following values in the PHP configuration file `php.ini`:

- `register_globals` - For security reasons the value `register_globals` was set to `Off`, as we described already in Section 3.3.2.1. This implies that global variables are no longer automatically registered from input data.
- `display_errors` - The value `display_errors` is set to `Off`, which entails that errors that occur during the execution of a PHP script are not displayed to the user interface (as part of the script output).
- `log_errors` - We set the value `log_errors` to `On`. Any errors that occur during the execution of our PHP scripts are written into the server's default error log.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have provided a Web application for the library of the Distributed Parallel Systems Group (DPS) of the Computer Science Institute of Innsbruck. The system runs on an Apache Web Server and is implemented using PHP. As internal database we have used MySQL. We provided a short overview of the used techniques, as well as of similar ones. The installation of PHP on the Apache Web server was uncomplicated. The integration of MySQL through PHP was quite easy and fast done. We did not encounter any major problems during the implementation.

Although, with hindsight, we still think that our decision concerning the technologies we used was the right one, we would now plaid for an integration of BibTeX to the system. In the scientific work areas it is quite common to write publications with the document preparation system LaTeX. In that case the use of BibTeX eases the making of citations and the embedding of references quite a lot.

In the thesis we described the library functionality and the use cases that occur within the library. We drew an appropriate database model and explained how we granted the database user rights. A detailed user manual explains how to use the library Web interface.

The Institute Library System has been successfully adapted by the DPS research group. The library currently manages 342 publications and 28 users. The system runs stable since July 2005 and no major problems have yet been reported.

5.2 Future Work

We actually see two possible feature extensions to the library system. One of these concerns the use of BibTeX as publication database: a BibTeX generator could be added to the existing system. The generator could be embedded into the system and be programmed in a way that it produces a new BibTeX database each time the MySQL publication database is altered.

Another system extension could allow the use of a barcode scanner for books. This way new publications could be easily added to the library database without needing to manually enter publication information. The DPS Group actually intends to add this feature in the near future.

Chapter 6

Appendices

6.1 MySQL Scripts

The following scripts are used to build the database, to grant user rights and to insert some initial data, as described in Section 3.2.6 and Section 3.3.2.5.

6.1.1 build.sql

The following script is used for building the dpslibrary database and creating tables and fields. This implementation is based on the model showed in Figure 3.5 (Section 3.2.6).

```
/* dpslibrary database */

CREATE DATABASE dpslibrary;

/* Tables */

CREATE TABLE user (
  uid           SMALLINT(5) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstName     VARCHAR(50) NOT NULL,
  lastName      VARCHAR(50) NOT NULL,
  username      VARCHAR(16) UNIQUE NOT NULL,
  password      VARCHAR(50) NOT NULL,
  email         VARCHAR(50) UNIQUE NOT NULL,
  reminder      TINYINT(1) NOT NULL DEFAULT '1',
  superuser     TINYINT(1) NOT NULL DEFAULT '0');


CREATE TABLE publication (
  pid           SMALLINT(5) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  author        TEXT,
  title         VARCHAR(250) NOT NULL,
  subtitle      TEXT,
```

```
  year          YEAR,
  publisher     VARCHAR(200),
  location      VARCHAR(200),
  ISBN          VARCHAR(13),
  keywords      TEXT,
  comment       TEXT,
  signature     VARCHAR(15) UNIQUE NOT NULL,
  procTime      DATE,
  confLoc       VARCHAR(200),
  supervisor        VARCHAR(100),
  school        VARCHAR(200),
  type          ENUM ('all', 'book', 'proceeding', 'manual',
            'bachelor thesis', 'master thesis',
            'phd thesis', 'miscellaneous') NOT NULL,
  classification    ENUM ('all', 'programming languages',
                      'software engineering', 'operation systems',
                      'computer networks', 'theoretical computer
                       science', 'databases', 'compiler design',
                      'hardware', 'project management',
                      'mathematics', 'others') NOT NULL,
  status        ENUM('new', 'available', 'not available',
             'checked out', 'missing') NOT NULL,
  originDate        DATE NOT NULL);


CREATE TABLE borrowing (
  bid           SMALLINT(5) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  borrowDate        DATE NOT NULL,
  returnDate        DATE,
  user          SMALLINT UNSIGNED NOT NULL,
  publication       SMALLINT UNSIGNED UNIQUE NOT NULL,
  FOREIGN KEY (user)        REFERENCES user (uid),
  FOREIGN KEY (publication)     REFERENCES publication (pid));

CREATE TABLE organisation (
  userReminder      SMALLINT(5) UNSIGNED NOT NULL,
  newToAvailable    SMALLINT(5) UNSIGNED NOT NULL);
```

6.1.2 grant.sql

Figure 6.1 visualizes the granting of access rights on the *dpslibrary* database.

```
/*Access rights on the dpslibrary database*/

/*Default*/
GRANT USAGE ON dpslibrary.* TO 'libraryDefault'@'localhost'
    IDENTIFIED BY 'default';
GRANT SELECT ON dpslibrary.* TO 'libraryDefault'@'localhost';

/*Superuser*/
```

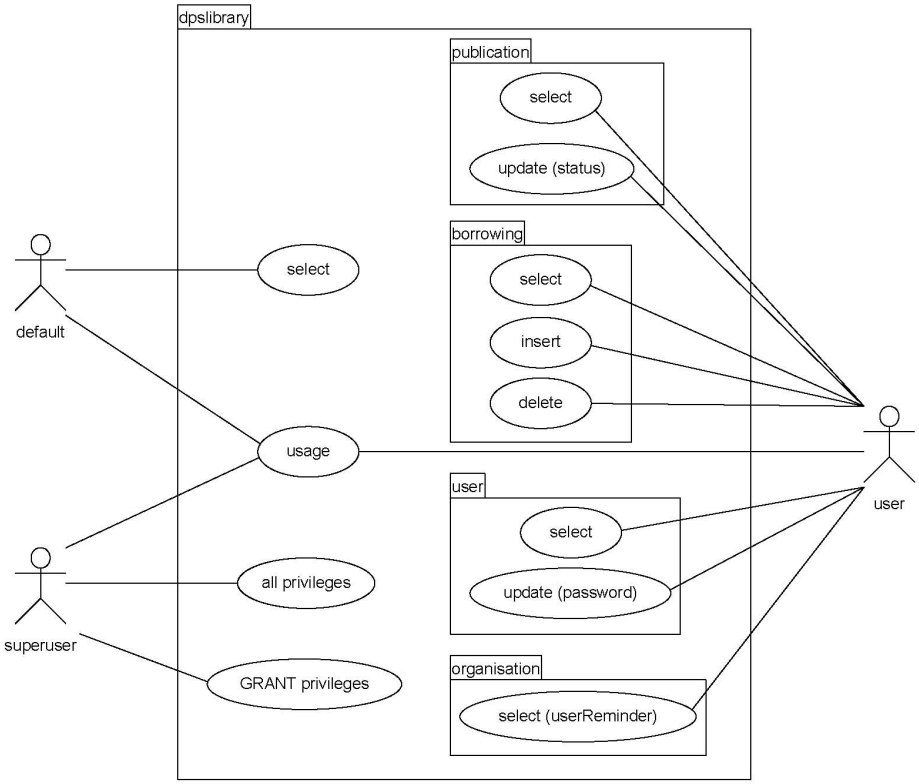

Figure 6.1: Use Case Diagram - Database User Rights

```
GRANT USAGE ON dpslibrary.* TO 'librarySuperuser'@'localhost'
    IDENTIFIED BY 'superuser';
GRANT ALL PRIVILEGES ON dpslibrary.* TO
    'librarySuperuser'@'localhost' WITH GRANT OPTION;

/*User*/
GRANT USAGE ON dpslibrary.* TO 'libraryUser'@'localhost'
    IDENTIFIED BY 'user';
GRANT SELECT, UPDATE (status) ON dpslibrary.publication TO
    'libraryUser'@'localhost';
GRANT SELECT, INSERT, DELETE ON dpslibrary.borrowing TO
    'libraryUser'@'localhost';
GRANT SELECT, UPDATE (password) ON dpslibrary.user TO
    'libraryUser'@'localhost';
GRANT SELECT (userReminder) ON dpslibrary.organisation TO
    'libraryUser'@'localhost';
```

6.1.3 insert.sql

The following script is used to insert some initial user data into the database.

```
/* Initial inserts Institute Library*/

/* USERS */

INSERT INTO user VALUES (1, 'Marek', 'Wieczorek', 'Marek',
    md5('marek'), 'marek.wieczorek@uibk.ac.at', 0, 1);
INSERT INTO user VALUES (NULL, 'Thomas', 'Fahringer', 'Thomas',
    md5('thomas'), 'thomas.fahringer@uibk.ac.at', 0, 1);

/* ORGANISATION */

INSERT INTO organisation VALUES (28, 28);
```

6.2 Example PHP file

The following example file extracts user information from a form and adds a new library user to the database. The activity diagram in Figure 6.2 illustrates what happens in the PHP file. This example file shall reflect the php issues we discussed in Section 3.3.1.2 and Section 3.3.2.1.

```
<?php
  // start session
  session_start ();

  // set actual page session variable
  $_SESSION["act_page"] = "priv_add_user_conf";

  // start session, check if the user is already logged in and if he
  // is a superuser or not
  include ("../check_user.inc");
  include ("../library_functions.inc");

  // check if the logged-in is a superuser
  if (!$user_superuser && isset($_SESSION["user_id"])){
    header("Location: ../User/dps_library.php?error=1");
    exit;
  } // end if

  else {
    // extract values from POST
    $first_name="";
    if (isset($_POST["firstName"]))
      $first_name = $_POST["firstName"];
    $last_name="";
    if (isset($_POST["lastName"]))
```

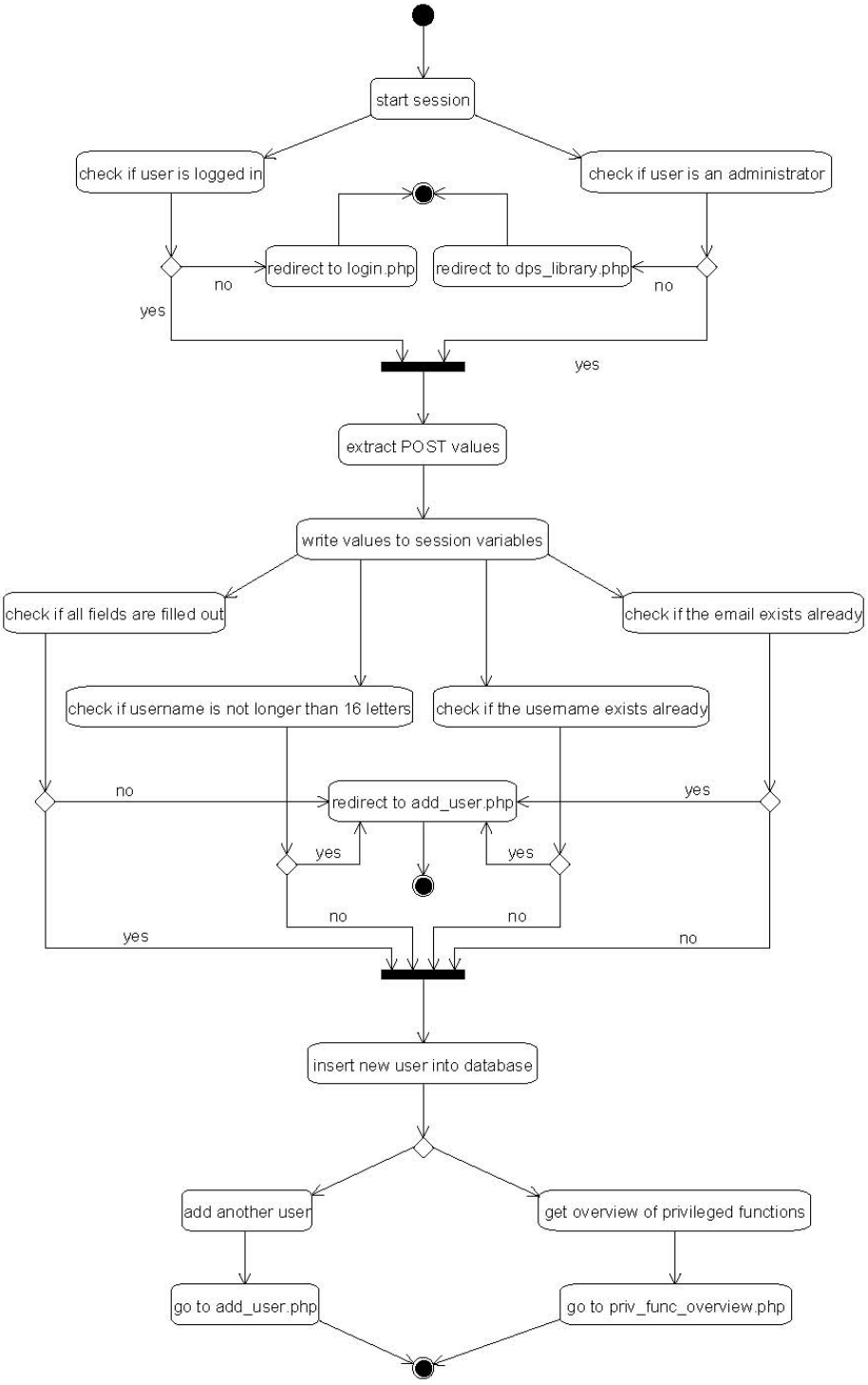

Figure 6.2: Activity Diagram - Add New User to Database

```
  $last_name = $_POST["lastName"];
$user_name="";
if (isset($_POST["username"]))
  $user_name = $_POST["username"];
$password="";
if (isset($_POST["password"]))
  $password = $_POST["password"];
$email="";
if (isset($_POST["email"]))
  $email = $_POST["email"];
$reminder=1;
if (isset($_POST["reminder"]))
  $reminder = $_POST["reminder"];
$superuser = 0;
if (isset($_POST["superuser"]))
  $superuser = $_POST["superuser"];

// write values to session variables
$_SESSION["firstName"] = $first_name;
$_SESSION["lastName"] = $last_name;
$_SESSION["username"] = $user_name;
$_SESSION["password"] = $password;
$_SESSION["email"] = $email;
$_SESSION["reminder"] = $reminder;
$_SESSION["superuser"] = $superuser;

// all the fields have to be filled out
if (($first_name == "") || ($last_name == "") ||
        ($user_name == "") || ($password == "") || ($email == "")){
  header("Location: add_user.php?error=2");
  exit;
} // end if

// the username mustn't be longer than 16 letters
if (strlen($user_name) > 16){
  header("Location: add_user.php?error=4");
  exit;
} // end if

// start database connection for the superuser, with error handling
$db = mysql_connect("localhost", "librarySuperuser", "superuser");
if (!mysql_select_db ("dpslibrary", $db)){
  die ("Connection to database failed - superuser");
} // end if

// send query to database
$res1 = mysql_db_query("dpslibrary", "SELECT username FROM user
        WHERE username = '$user_name';");
if (mysql_errno() != 0) {
  // logging function call
```

```
      $string = "Error: " . mysql_errno() . ": " . mysql_error(). "\n";
      library_log($string);
      echo ("Database error! - Please see log file for detailed
          information.<br><br>");
    } // end if

    // an error code is set, if the chosen username exists already.
    // The user is sent back to the previous page.
    $num1 = mysql_num_rows($res1);
    if ($num1 > 0){
      // close database connection
      mysql_close($db);

      header("Location: add_user.php?error=1");
      exit;
    } // end if

    // send query to database
    $res2 = mysql_db_query("dpslibrary", "SELECT email FROM user
            WHERE email = '$email';");
    if (mysql_errno() != 0) {
      // logging function call
      $string = "Error: " . mysql_errno() . ": " . mysql_error(). "\n";
      library_log($string);
      echo ("Database error! - Please see log file for detailed
          information.<br><br>");
    }

    // an error code is set, if the chosen email exists already.
    // The user is sent back to the previous page.
    $num2 = mysql_num_rows($res2);
    if ($num2 > 0){
      // close database connection
      mysql_close($db);

      header("Location: add_user.php?error=3");
      exit;
    } // end if

    // close database connection
    mysql_close($db);
  } // end else
?>

<!doctype html public "-//W3C//DTD HTML 4.01//EN">

<html>

  <head>
```

```
  <title>DPS Library</title>

  <meta http-equiv="keywords" content="Distributed Parallel Systems
          Group - Library" />
  <meta http-equiv="generator" content="PHP Designer 2005" />
  <meta http-equiv="Content-Style-Type" content="text/css">

  <link rel="stylesheet" href="../format.css" type="text/css" />

</head>

<body>

<?php
  // including vertical menue
  include("menue1.inc");
?>

    <span class="crumbTrail"><a class="crumbTrail"
            href="../User/dps_library.php">DPS Library</a> &gt;
            <a class="crumbTrail" href="priv_func_overview.php">
            Privileged functions</a> &gt; <a class="crumbTrail"
            href="add_user.php">Add new user</a></span>

<?php
  // including vertical menue
  include("menue2.inc");
?>

        <table cellpadding="0" cellspacing="0" border="0"
                width="100%">
          <tr>
            <td>

              <h2>Add new user</h2><br>

              <?php
                // start database connection for the superuser,
                // with error handling
                $db = mysql_connect("localhost", "librarySuperuser",
                        "superuser");
                if (!mysql_select_db ("dpslibrary", $db)){
                  die ("Connection to database failed - superuser");
                } // end if

                if ($reminder == 'yes')
                  $reminder = 1;
                else
                  $reminder = 0;
```

```
if ($superuser == 'yes')
  $superuser = 1;
else
  $superuser = 0;

// check if the first_name contains a single quote " ' "
if(strpos($first_name, "'") != 0) {
  $first_name = str_replace("'","\'", $first_name);
}
// check if the last_name contains a single quote " ' "
if(strpos($last_name, "'") != 0) {
  $last_name = str_replace("'","\'", $last_name);
}
// check if the user_name contains a single quote " ' "
if(strpos($user_name, "'") != 0) {
  $user_name = str_replace("'","\'", $user_name);
}
// check if the password contains a single quote " ' "
if(strpos($password, "'") != 0) {
  $password = str_replace("'","\'", $password);
}
// check if the email contains a single quote " ' "
if(strpos($email, "'") != 0) {
  $email = str_replace("'","\'", $email);
}

// build sql insert into user string
$sqlInsertUser = "INSERT INTO user VALUES ('NULL',
        '$first_name', '$last_name', '$user_name', ";
$sqlInsertUser .= "md5('$password'), '$email',
        '$reminder', '$superuser');";

// send query to database
mysql_db_query("dpslibrary", $sqlInsertUser);
if (mysql_errno() != 0) {
  // logging function call
  $string = "Error: sqlInsertUser: " . mysql_errno()
          . ": " . mysql_error(). "\n";
  library_log($string);
  echo ("Database error! - Please see log file for detailed
      information.<br><br>");
} // end if

// if insert was successful
$num = mysql_affected_rows();
if ($num>0) {
  // logging function call
  $string = "Inserted new user " . $user_name . "\n";
  library_log($string);
```

```
  // send select user query to database
  $res = mysql_db_query("dpslibrary", "SELECT firstName,
          lastName, username, email, reminder, superuser
          FROM user WHERE username = '$user_name';");

  // extract results
  $first_name = mysql_result($res, 0, "firstName");
  $last_name = mysql_result($res, 0, "lastName");
  $user_name = mysql_result($res, 0, "username");
  $email = mysql_result($res, 0, "email");
  $reminder = mysql_result($res, 0, "reminder");
  $superuser = mysql_result($res, 0, "superuser");

  echo "<strong> You've successfully added this user to
          the database. </strong>";
  echo "<br><br>";

  echo "<table cellspacing='10'>";
  echo "<tr> <td> <div align='right'><strong> first
          name </strong></div> </td>";
  echo "<td>", $first_name, "</td> </tr>";
  echo "<tr> <td> <div align='right'><strong> last
          name </strong></div> </td>";
  echo "<td>", $last_name, "</td> </tr>";
  echo "<tr> <td> <div align='right'><strong> username
          </strong></div> </td>";
  echo "<td>", $user_name, "</td> </tr>";
  echo "<tr> <td> <div align='right'><strong> email
          </strong></div> </td>";
  echo "<td>", $email, "</td> </tr>";
  echo "<tr> <td> <div align='right'><strong> reminder
          </strong></div> </td>";
  if ($reminder == 1)
    echo "<td> yes </td> </tr>";
  else
    echo "<td> no </td> </tr>";
  echo "<tr> <td> <div align='right'><strong> superuser
          </strong></div> </td>";
  if ($superuser == 1)
    echo "<td> yes </td> </tr>";
  else
    echo "<td> no </td> </tr>";
  echo "</table> <br><br>";
} // end if
else {
  // logging function call
  $string = "Error: Couldn't insert new user " .
          $username . "\n";
  library_log($string);
  echo "<br><strong> The user hasn't been added to
```

```
                    the database. </strong>";
                } // end else

                // close database connection
                mysql_close($db);
              ?>

              <br><br><br>

              <table>
                <tr>
                  <form action="add_user.php" method = "GET">
                    <td><input type="submit" name="add_button"
                            value="  Add another user  "></td>
                  </form>
                  <form action="priv_func_overview.php" method = "GET">
                    <td><input type="submit" name="priv_func_button"
                            value= "Privileged functions overview"></td>
                  </form>
                </tr>
              </table>

            </td>
          </tr>
        </table>

      </td>
    </tr>
  </table>

  <?php
    // including links
    include("menue3.inc");
  ?>

</body>

</html>
```

6.3 PHP Cron Job

The following cron job needs to be executed once a day. It is responsible for two tasks, described in Section 3.3.2.4. The activity diagram in Figure 6.3 illustrates how these tasks are executed by the cron job.

```
<?php
  // function to write the current date and time, and a given
  // string into a logfile
  function library_log($string) {
```

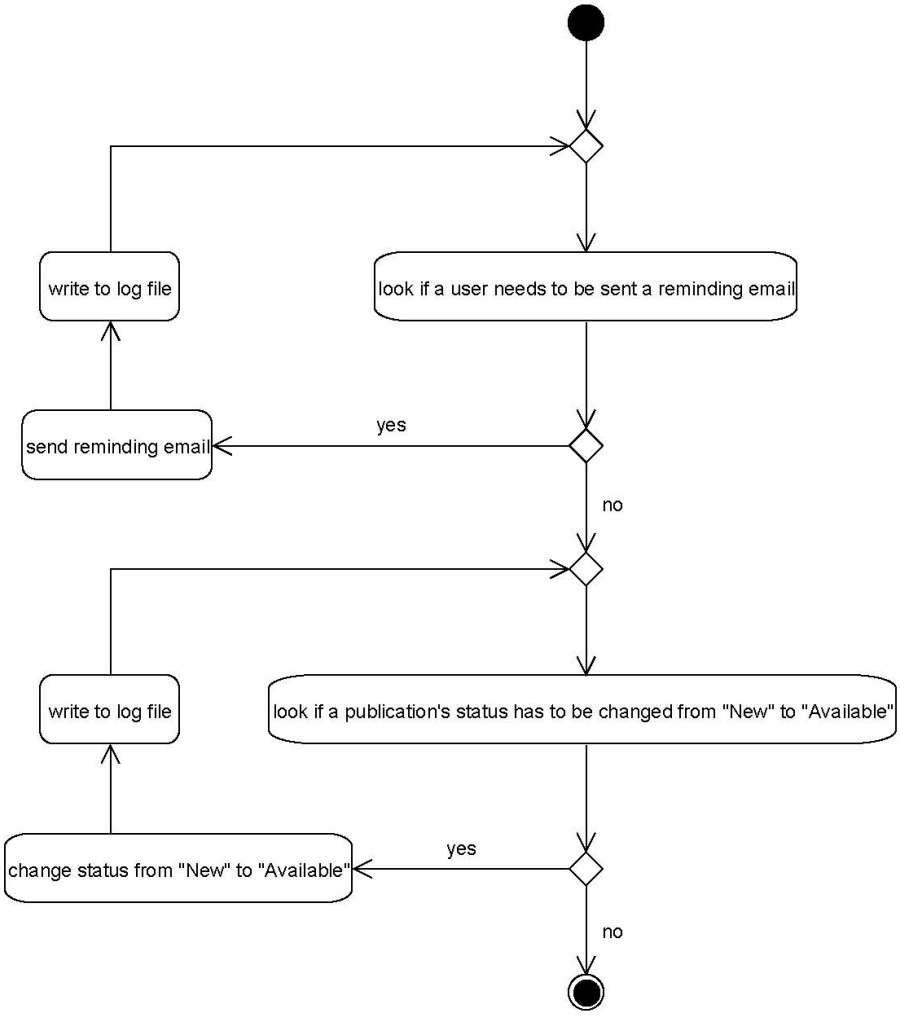

Figure 6.3: Activity Diagram - PHP Cron Job

```
$date = date("d.m.Y:");
$time = date("H:i:s");

// the actual date is written into the logfile
$logInfo = "[$date$time] Cron: ";

if ($string != "")
  $logInfo .= " - $string";

// each month a new logfile is created
```

```
  $month = date("m");
  $year = date("Y");
  $filename = "Log/log_".$month."_".$year.".txt";

  // the logfile is always opened for append, which means that
  // all the new logs are appended to the existing
  $fp = fopen($filename,"a");
  if ($fp) {
    fputs($fp,"$logInfo\n");
    fclose($fp);
  } // end if
  else
    echo "Logfile couldn't be opened to append!";
} // end function library_log

// start database connection for the superuser, with error handling
$db = mysql_connect("localhost", "librarySuperuser", "superuser");
if (!mysql_select_db ("dpslibrary", $db)){
  die ("Connection to database failed - superuser");
} // end if

/*-------------------------------------------------------------------
--------- Look if a user shall be sent a reminding mail ------------
-------------------------------------------------------------------*/

// select all the publications that are borrowed and for which a
// remindDate is defined
// send select borrowing query to database
$res = mysql_db_query("dpslibrary", "SELECT returnDate, user,
    publication FROM borrowing WHERE returnDate != '0000-00-00';");
if (mysql_errno() != 0) {
  // logging function call
  $string = "sqlSelectRem: " . mysql_errno() . ": " .
      mysql_error(). "\n";
  library_log($string);
  echo $string;
} // end if

// get number of results
$num = mysql_num_rows($res);
if ($num > 0) {
  for ($i=0; $i<$num; $i++) {
    // extract returnDate from borrowing results
    $returnDate = mysql_result($res, $i, "returnDate");

    // make a timestamp out of the returnDate, to be able to better
    // work with it
    list($year, $month, $day) = explode("-", $returnDate);
    $returnDateTStamp = mktime(0, 0, 0, $month, $day, $year);
```

```
// get a timestamp of today's date
$actualDate = time();

// if today is later or equal to the return date, the user must
// be reminded to return his publication
if ($actualDate >= $returnDateTStamp){
  // extract user and publication from borrowing results
  $user = mysql_result($res, $i, "user");
  $publication = mysql_result($res, $i, "publication");

  // select the email address from the user which is to be
  // reminded
  // send select user query to database
  $res2 = mysql_db_query("dpslibrary", "SELECT email FROM user
      WHERE uid = $user;");
  if (mysql_errno() != 0) {
    // logging function call
    $string = "sqlSelectEmail: " . mysql_errno() . ": " .
        mysql_error(). "\n";
    library_log($string);
    echo $string;
  } // end if

  // extract email from user results
  $email = mysql_result($res2, 0, "email");

  // select the title from the publication which shall be
  // returned
  // send select publication query to database
  $res3 = mysql_db_query("dpslibrary", "SELECT title FROM
      publication WHERE pid = $publication;");
  if (mysql_errno() != 0) {
    // logging function call
    $string = "sqlSelectTitle: " . mysql_errno() . ": " .
        mysql_error(). "\n";
    library_log($string);
    echo $string;
  } // end if

  // extract title from publication results
  $title = mysql_result($res3, 0, "title");

  // send an email to the user which needs to be reminded,
  // including the return date and the corresponding publication
  // title
  mail($email, "DPS Library Reminder", "The object " . $title .
      " is due to be returned on " . $year . "-" . $month . "-"
      . $day . " to the library.",
      "From: Marek.Wieczorek@uibk.ac.at\r\nReply-To:
      Marek.Wieczorek@uibk.ac.at");
```

```
      // logging function call
      $string = "Cron job: sent mail to user with id " . $user . "
          concerning the return of publication with id " .
          $publication . "\n";
      library_log($string);
    } // end if
  } // end for
} // end if

/*-------------------------------------------------------------------
---------------- Look if a publication's status has to -------------
----------------  be changed from 'new' to 'available' -------------
-------------------------------------------------------------------*/

// check after how many days an object's status shall be changed
// from 'new' to 'available'
// send select organisation query to database
$res = mysql_db_query("dpslibrary", "SELECT newToAvailable FROM
    organisation;");
if (mysql_errno() != 0) {
  // logging function call
  $string = "sqlSelectNewToAv: " . mysql_errno() . ": " .
      mysql_error(). "\n";
  library_log($string);
  echo $string;
} // end if

// extract newToAvailable from organisation results
$newToAvailable = mysql_result($res, 0, "newToAvailable");

// select all the publications whose status is 'new'
// send select publication query to database
$res = mysql_db_query("dpslibrary", "SELECT pid, type, originDate
    FROM publication WHERE status='new';");
if (mysql_errno() != 0) {
  // logging function call
  $string = "sqlSelectStatus: " . mysql_errno() . ": " .
      mysql_error(). "\n";
  library_log($string);
  echo $string;
} // end if

// get number of results
$num = mysql_num_rows($res);
if ($num > 0) {
  for ($i=0; $i<$num; $i++) {
    // extract publication results
    $pid = mysql_result($res, $i, "pid");
    $type = mysql_result($res, $i, "pid");
```

```
$originDate = mysql_result($res, $i, "originDate");

// make a timestamp out of the originDate, to be able to
// better work with it
list($year, $month, $day) = explode("-", $originDate);
$originDateTStamp = mktime(0, 0, 0, $month, $day, $year);

// build the exact date on which the status shall be changed
// (86400 == 24*60*60)
$changeStatusDate = $originDateTStamp +
    ($newToAvailable * 86400);

// get a timestamp of today's date
$actualDate = time();

// if today is later or equal to the date on which the status
// shall be changed, the status is changed
// if the object's type = book, the status is changed to
// 'available'
if (($actualDate >= $changeStatusDate) && ($type == "book")){
  // build sql update publication string
  $sqlUpdateObj = "UPDATE publication SET status = 'available'
      WHERE pid = '$pid';";

  // send query to database
  $res2 = mysql_db_query("dpslibrary", $sqlUpdateObj);
  if (mysql_errno() != 0) {
    // logging function call
    $string = "sqlUpdateObj: " . mysql_errno() . ": " .
        mysql_error(). "\n" . $sqlUpdateObj . "\n";
    library_log($string);
    echo $string;
  } // end if

  $num2 = mysql_affected_rows();
  if ($num2 > 0) {
    // logging function call
    $string = "Cron job: status from publication with id " .
        $pid . " changed to available.\n";
    library_log($string);
  } // end if
  else {
    // logging function call
    $string = "Error, Cron job: status from publication with
        id " . $pid . " couldn't be changed to available.\n";
    library_log($string);
  }
} // end if

// if today is later or equal to the date on which the status
```

```
      // shall be changed, the status is changed
      // if the object's type != book, the status is changed to
      // 'not available'
      if (($actualDate >= $changeStatusDate) && ($type != "book")){
        // build sql update publication string
        $sqlUpdateObj = "UPDATE publication SET status =
            'not available' WHERE pid = '$pid';";

        // send query to database
        $res2 = mysql_db_query("dpslibrary", $sqlUpdateObj);
        if (mysql_errno() != 0) {
          // logging function call
          $string = "sqlUpdateObj: " . mysql_errno() . ": " .
              mysql_error(). "\n" . $sqlUpdateObj . "\n";
          library_log($string);
          echo $string;
        } // end if

        $num2 = mysql_affected_rows();
        if ($num2 > 0) {
          // logging function call
          $string = "Cron job: status from publication with id
              " . $pid . " changed to not available.\n";
          library_log($string);
        } // end if
        else {
          // logging function call
          $string = "Error, Cron job: status from publication with id
              " . $pid . " couldn't be changed to not available.\n";
          library_log($string);
        }
      } // end if
    } // end for
  } // end if

  // close database connection
  mysql_close($db);
?>
```

6.4 PHP Log Function

The following PHP script shows the logging function that we have described in Section 3.3.2.2.

```
<?php
  // function to write the current date and time, and a given string
  // into a logfile
  function library_log($string) {
```

```
    $date = date("d.m.Y:");
    $time = date("H:i:s");
    $ip = getenv("REMOTE_ADDR");
    $site = $_SERVER["REQUEST_URI"];

    // the actual date, the user ip address and the requested site
    // are written into the logfile
    $logInfo = "[$date$time] - $ip - \"GET /$site\"";

    // the username from the currently logged-in user and the
    // parameter string are written into the logfile
    $username = "";
    if (isset($_SESSION["user_id"])) {
      $username = $_SESSION["user_username"];
      $logInfo .= "\nUser: $username";
    } // end if

    if ($string != "")
      $logInfo .= " - $string";

    // each month a new logfile is created
    $month = date("m");
    $year = date("Y");
    $filename = "../Log/log_".$month."_".$year.".txt";

    // the logfile is always opened for append, which means that all
    // the new logs are appended to the existing
    $fp = fopen($filename,"a");
    if ($fp) {
      fputs($fp,"$logInfo\n");
      fclose($fp);
    } // end if
    else
      echo "Logfile couldn't be opened to append!";
  } // end function library_log
?>
```

6.5 Example Log File

In the following we show an example log file, created by the logging function that we have described in Section 3.3.2.2.

```
[21.08.2006:21:45:49] - 10.0.0.140 -
    "GET //DPSLibrary/User/login_check.php" - User nathalie logged in.

[21.08.2006:21:55:30] - 10.0.0.140 -
    "GET //DPSLibrary/borrow_obj_conf.php"
User: nathalie - borrowed object with publication id 42
```

```
[22.08.2006:10:09:33] - 10.0.0.140 -
    "GET //DPSLibrary/User/login_check.php" - User nathalie logged in.

[22.08.2006:10:33:16] - 10.0.0.140 -
    "GET //DPSLibrary/PrivUser/change_obj_conf.php"
User: nathalie - Successfully changed the object with id 2

[22.08.2006:10:33:57] - 10.0.0.140 -
    "GET //DPSLibrary/PrivUser/change_obj_conf.php"
User: nathalie - Successfully changed the object with id 5
```

Bibliography

[CGI, 1998] (1998). Common gateway interface. URL: http://hoohoo.ncsa.uiuc.edu/cgi/intro.html.

[JSP, 2005] (2005). *JavaServer Pages (JSP) v2.0 Syntax Reference*. URL: http://java.sun.com/products/jsp/docs.html.

[ASP, 2006] (2006). The official microsoft asp.net 2.0 site. URL: http://asp.net/.

[PHP, 2006] (2006). *PHP Manual*. URL: http://www.php.net/manual/en/.

[Net, 2006] (2006). September 2006 web server survey. URL: http://news.netcraft.com/archives/web_server_survey.html.

[Uni, 2006] (2006). University of vienna, library of the institute of scientific computing. URL: http://www.par.univie.ac.at/~bib/ips-lib/main.html.

[AB, 2006] AB, M. (2006). *MySQL 3.23, 4.0, 4.1 Reference Manual*. Available from: http://dev.mysql.com/doc/#manual.

[Graham, 2001] Graham, P. (2001). The other road ahead. URL: http://www.paulgraham.com/road.html.

[Kaminaris and Annunziato, 2001] Kaminaris, S. F. and Annunziato, J. (2001). *Jetzt lerne ich JavaServer Pages*. Markt+Technik Verlag, first edition.

[Krause, 2004] Krause, J. (2004). *PHP 5 - Grundlagen und Profiwissen*. Carl Hanser Verlag, first edition.

[Lie and Bos, 1999] Lie, H. W. and Bos, B. (1999). Cascading style sheets - level 1. Technical report. Available from: http://www.w3.org/TR/REC-html40/.

[Ragget et al., 1999] Ragget, D., Hors, A. L., and Jacobs, I. (1999). Html 4.01 specification. Technical report. Available from: http://www.w3.org/TR/REC-html40/.

[Theis, 2000] Theis, T. (2000). *PHP 4*. Galileo Computing. Open book available from: http://www.galileocomputing.de/openbook/php4/.