CIS:
A Web-Based Course Information System

Holger Gast, Albrecht Haug, Rüdiger Loos,
Volker Simonis, and Roland J. Weiss

WSI 2004-1

February 2004

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Sand 13, 72076 Tübingen, Germany

Email:
{gast,haug,loos,simonis,weissr}@informatik.uni-tuebingen.de

ISSN 0946-3852

Abstract

This report surveys the design and implementation of CIS, a web-based Course Information System. CIS has been developed for the Computer Science I/II courses held between 2000 and 2003 by Prof. Dr. R. Loos, which were attended by 300 to 450 students. It maintains and presents each student's submissions and grades and holds related information such as worksheet texts, submission deadlines and the assignment of students to teaching assistents. In short, it covers most of the administrative data that comes up in regular university courses.

CIS is designed to be used by first-year students conveniently. It aims at modelling real-world procedures, so that the system behaviour can be explained in well-known analogies. It is minimalistic, in the sense that it only takes on the routine work, while leaving the teacher free in any questions of structuring the contents of the course.

Our problem statement and analysis focuses to two aspects: The requirements on the central data base and the interfaces for three groups of users: Students, teaching assistants, and teachers/administrators. The actual implementation is straightforward, and we only mention particular decisions taken herein.

CIS has been in use at the Wilhelm-Schickard Institut for three years, in courses organized both by the authors and others. The experiences indicate that the system can be considered reliable and mature. As the effort of setting up CIS is small, it has become feasible to employ it for several advanced courses with fewer than 20 students.

Contents

List of Figures

Chapter 1

Introduction and Problem Statement

The administrative overhead for university courses becomes a serious issue as soon as the list of enrolled students extends to more than one or two pages. Traditionally, the information management for weekly grades on homework and participation is handed on to the teaching assistants (TAs) for their particular groups of students. However, this strategy achieves a partial solution only: Each TA will still spend part of her time and energy to track down late homework and create reports on the students' achievements, rather then helping students in following the course. Furthermore, the consistency of grading policies still needs to be enforced by regular meetings with organizational rather than educational contents.

CIS embodies a radically different strategy: All course-related data is stored in one central repository where it can be accessed by both students and TAs for their specific purposes. Reports can be generated automatically and from up-to-date sources. The user interface is entirely web-based, that is any browser can be used to connect. CIS handles not only the assigned grades, but is designed to serve as a central information repository for the students also. All homework assignments are available and submissions can be handed in electronically. Grades appear in each students personal data sheet as soon as they are entered by the TA.

Beyond these immediate answers to the original administrative tasks, CIS also allows for more flexible schemes in homework assignments. Electronic submissions can be assigned to TAs for grading based on consideration of load-balancing and exceptional absences without any noticeable delay for the students. Also the rigid assignment of a TA to her group can be loosened up to the point where some TAs offer regular offices hours instead of a weekly group meeting and participate in the grading of other groups where necessary.

1.1 Design Goals

CIS is designed for the environment of a first year course in computer science where some students may not have had extensive experience with computers. Therefore special care needs to be taken to ensure transparent and comprehensible behaviour of the system. Beyond this basic objective, we have established a number of technical requirements to ensure smooth operation of the resulting system.

Automate routine work, treat special cases by hand A project such as CIS quickly extends to unmanageable size with the accumulation of different users' needs, so the scope of our implementation needs restriction. The most prominent design goal for CIS is therefore to automate the larger part of administrative work, but allow manual intervention for special situations.

Model real-world procedures CIS is designed to be explicable to students in terms of situations that do happen in everyday course work. Most importantly, for every policy implemented by the system, there is a reasonable rule that could be enforced in conventional course administration.

Reassuring behaviour Students take personal interest in their achievements in courses. Resulting anxieties need to be taken serious if CIS is to be used comfortably. Specifically, students must get immediate feedback to each of their transactions and they need to be thoroughly informed about the current state of data regarding their submissions and grading.

Uptime CIS must be available every day of the week without interruption.

Security of personal data The personal data of students needs to be protected from third-party access. Specifically, the connection between grades and names needs to be handled carefully. Ideally, that connection should be accessible for administrators only. However, fully anonymous users are no option, since the final credits for the course are given based on the stored data.

Protection from hacking In particular the database may be subject to attacks, both from students spying on fellow students and those who attempt to modify their grades. The implementation tools should therefore share the considerations about security and every action triggered by users needs to be logged.

1.2 Information Base

CIS is meant to contain all relevant data for a course. For a minimal system, at least the following items should be available.

- Homework grades for students.
- Worksheets and solutions.
- Directions to contact TAs and lecturers.
- A bulletin board for announcements.
- Handouts and citations of literature.

Not all of these items need to be stored in a database. Some of them are updated infrequently, so that conventional hypertext documents are a suitable means of representation.

1.2.1 Homework Assignments

For every homework assignment five steps need to take place, each of which can be facilitated by a centralized information system.

1. The assignment is made available to the students.
2. The students pursue a solution and may request help from TAs.
3. The students hand in their final solution.
4. The TAs receive the solution for grading.
5. The TAs submit the grades.

For CIS, worksheets are prepared as PS/PDF files and they are published on the web. Each assignment is listed in the database with its maximal score. The deadlines and extensions for worksheets are also stored. Hence, a complete and up-to-date description of the assignments in the course is available online.

TAs usually have office hours and/or meetings with groups of students. The week's schedule of office hours should be available on the web such that a student can request immediate help from some TA any time rather than waiting for the next scheduled group meeting to come up.

In computer science courses, many assignments require solutions consisting of computer files and grading is facilitated largely if the TA does have for instance program source available for compilation and testing. CIS is able to collect solutions from students for further processing by the administrators and TAs. It enforces the stored deadlines and accepts no late submissions. The non-electronic submissions need a placeholder in the database.

Right after grading, the TAs store the results in the database, where they can be accessed by both students and teachers. Grades are to be considered personal data. Clearly, students can only read their own grades. Furthermore, a TA can enter grades only for solutions assigned to her, analogously to traditional course work, where a TA has direct access only to the grades of her own group.

1.2.2 Administrative Information

The organizational structure of a course must be transparent and accessible to students. A web-based presentation can greatly enhance the information flow compared to traditional weekly group meetings with TAs. Most notably, changes and corrections to homework assignments and schedules will be accessible to all students simultaneously. The following data needs to be stored for administrative purposes:

- Personal data of students, i.e. name, birth date and student id for assigning credits at the end of the course.
- Email of students and TAs for urgent broadcast messages and individual notification.
- Names, office hours and email addresses of TAs.
- Bulletin board for current announcements.
- Assignment of TAs to student groups and teams.

At German universities, the students receive a certificate of successful participation (so called *Schein*) at the end of a course, provided they have fulfilled the criteria. It is very convenient to be able to print the necessary data into forms by standard text-processors, most of which can import tabular data as generated by database systems.

During the semester, students can be expected to visit the course homepage on a regular basis. However in the spring and summer breaks, email is usually a more reliable form of information transport. Also personal notifications about inadequate submissions have often proved to solve upcoming grading problems before submission deadlines.

General announcements are usually short messages that should be visible immediately on the course's homepage. They are meant to contain information that is invalid or useless after a certain date, and they should expire automatically. This procedure frees the administrator from the burden of regular updates and furthermore the texts of the messages need not be deleted from the database, but remain accessible for later reference.

The information about which TA is responsible for which group of students must be up-to-date and frequent changes are to be expected in the beginning of a course. It is also desirable to allow teamwork among students, i.e. only one solution to homework assignments is submitted by each team. Beyond educational purposes, this eliminates the time needed to grade copies of solutions.

1.3 User Views

CIS provides access to the information base for three groups of users: Students, TAs and administrators. Administrators are typically the lecturer responsible for and research assistants assigned to the course. Their respective views on the data must be chosen with regard to utility and security. It can be assumed that the groups are disjoint.

1.3.1 Students

The students are the primary user group of the CIS. Their interface must enable them to trigger the transactions needed for successful participation in the course in a convenient fashion . The results of each transaction must be apparent in the system's feedback. The following actions are essential.

- Sign in for the course and submit personal data.
- Modify the email address and password.
- Download homework assignments.
- Upload solutions.
- Query grades.
- Contact the TA and the administrators.
- Access current administrative information.

In Germany, there is no formal registration for courses at the beginning of a semester. To establish the user-group of CIS, the students sign in for the course directly on the web, submitting their personal data to be printed on the course certificate and their email address (if any). The email address can be expected to change during the semester, hence students need to modify that entry by themselves.

Solving homework assignments from the students' point of view consists of downloading the exercise text and submitting an electronic solution. It must be possible to update previous submissions with enhanced versions. The least recent version before the deadline is used for grading, but the five previous submissions are left on disk in case of technical failure. Submissions are considered a critical part of the system. Hence, each submission must be safely stored to disk before any acknowledgement is returned.

A student can access the results on her exercises immediately after the TA has entered them. Sums for worksheets and grand total are computed automatically. Visual indications can be given whether the required minimal scores have been achieved for each entry. Grades are considered strictly personal data. No information on grades of other students, including statistics, is available to users in the student and TA groups.

For questions and other requests for help, all TAs are listed with their office hours and email addresses. Organizational information and current announcements are displayed at each login to the system.

1.3.2 Teaching Assistants

The teaching assistants access the system only for grading purposes. The administrators assign the submissions to TAs based on a suitable strategy. Conventionally, a TA is responsible for a fixed group of students during the semester. However, more flexible schemes such as load balancing can be supported without difficulty. The following tasks can be expected to be carried out by TAs on a regular basis:

- Online grading, evaluating one submission at a time.

- Online grading, entering all grades for the current worksheet.
- Download all assigned submissions and upload results.
- Changes to previously assigned grades during office hours.

Each of these procedures has the same effects on the final database state. The assigned grades are recorded in the database with the responsible TA and the transaction is logged on disk for later validation of the date and time in case of technical failure.

Online grading of electronic submissions can be carried out entirely in a web-based interface. The TA requests submissions one by one and enters the assigned grade into a form displayed along with the submission. For each submission, the grade is transferred to the database where it is accessible to the submission's author immediately.

For grading at home, a TA can download all submissions currently assigned to her. She receives a (zip-) archive containing all submissions together with a form in a text file. The assigned grades are written into that form for all submissions by means of a text editor and the resulting file is uploaded to the server. The form must contain redundant information, so that minor changes such as deletion of fields and swapping of lines can be detected as errors.

A TA can change the grades she assigned herself during office hours, in case a student rightly complains about a decision. This process can be carried out by an online interface, since we can assume that a computer with access to the web is available in the office. In the same dialog, the TA can also enter the grades of all of her students for a given worksheet.

1.3.3 Administrators

The administrators enter and maintain the organizational data in the information base. They have full access to all parts of the information base and need a dedicated web-interface only for routine tasks. In particular they are responsible for the following:

- Correct personal data entered by the students during subscription.
- Publish homework assignments, modifications and solutions.
- Distribute electronic submissions to TAs for grading.

The subscription can be carried out over the web by each student separately and largely anonymously. However, later corrections of personal data may not change the identity of the participants during the semester. Hence, they need to be done by administrators, rather than allowing the students to update stored personal data directly.

The weekly homework assignments are published on the web by uploading the exercise text (preferably in PS and PDF formats) and entering the maximal scores for each assignment to the data base. If errors are detected, the administrators can post a corresponding announcement and possibly give extensions to worksheets or individual exercises.

After each submission deadline, the administrators decide on the distribution of submissions to the TAs. The scheme used herein should not rely on a specific course organization, but the implementation must be flexible enough to accommodate several approaches.

In Spring Term 2001 TAs were not assigned to fixed groups of students but offered regular office hours instead. Hence, CIS needed to implement a rather flexible scheme of distribution. That scheme prescribes that each TA is made responsible for grading a number of exercises. The incoming solutions are then distributed evenly among the responsible TAs. The scheme does not guarantee perfect load-balancing, yet it provides the administrators with enough flexibility to create a fair distribution of work. During the next terms, we resorted to a more classical distribution scheme. Each TA was assigned a fixed set of students at the start of the term for which she was resposible. This included two hours per week for discussing assignments and their solutions, as well as grading the assignments of the students belonging to the TA's group.

Chapter 2

System Architecture

CIS is a web-based information storage system running on a dedicated host. It consists of three large parts: The HTML web-pages with embedded PHP-scripts, a relational database, and a directory in the local file system in which file uploads are stored. The software architecture is built around the database content: The web interface conceptually presents a view onto the current database state and the user can trigger transactions to modify that state.[1]

2.1 System Components

The main system components and their interactions are shown in figure 2.1. All of the required software systems are freely available and have been widely applied and tested in practical projects (see the respective web-sites [2, 11, 7]). The data management is mainly done by the MySQL [7]

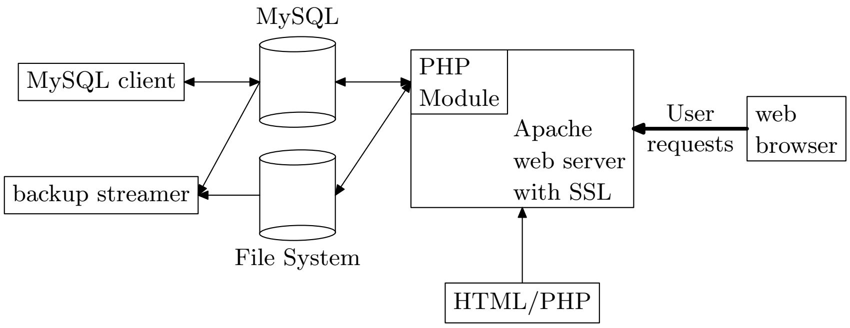


Figure 2.1: The main system components

relational database engine. It implements a sufficient part of the ANSI SQL92 standard [5] and can be accessed by clients both through TCP connections and local UNIX sockets. There are several MySQL clients available through which the system administrator can issue SQL queries directly, such that the non-standard modifications of the database state can be carried out in a convenient fashion, specifically without any programming efforts.

The Apache web-sever [2] is extensible via modules, a number of which is available for download. We make use of the SSL (Secure Socket Layer [3, 9]) and PHP (PHP: Hypertext Preprocessor [11])

[1]We shall use the term "transaction" for single actions in CIS, although in the current implementation, there is no guarantee that they will have the common attributes of transactions [4] in case of system failure.

extensions. PHP is a scripting language specifically designed for dynamic web-pages. PHP programs can be embedded into HTML pages, and they are executed within the web-server. Unlike in CGI-scripts, only the dynamic parts of a page need to be generated by the PHP code, the surrounding HTML code is passed to the web-browser unmodified. The PHP interpreter comes with a library for access to a MySQL database and the file system.

The user accesses the system by any web-browser with support for https connections. The "user requests" in figure 2.1 can be reading access as well as submissions of form data (via the HTTP `POST` method [6]). Hence, every transaction needs to be coded in PHP in some HTML page. Indeed, most of the existing pages implement exactly one transaction, and the system is extensible by simply adding new pages with required functionality.

The backup system writes the data repository and the uploads-section of the file system to tape. Currently, backup is done on a daily basis by means of UNIX `cron`-jobs.

2.2 Database Design

The development of the database schema parallels the requirements stated in section 1.2. We start with the discussion of user authentication and user groups, treat homework assignments, submissions and grading and conclude with the auxiliary administrative information. The corresponding entity relationship diagrams are introduced along the way. The final design is shown in appendix A.3.

2.2.1 User Authentication and Identity

Section 1.3 introduces the specific needs of three disjoint groups of users: The students, the teaching assistants, and the administrators. Figure 2.2 shows corresponding entity types *student*, *TA*, and *admin*.

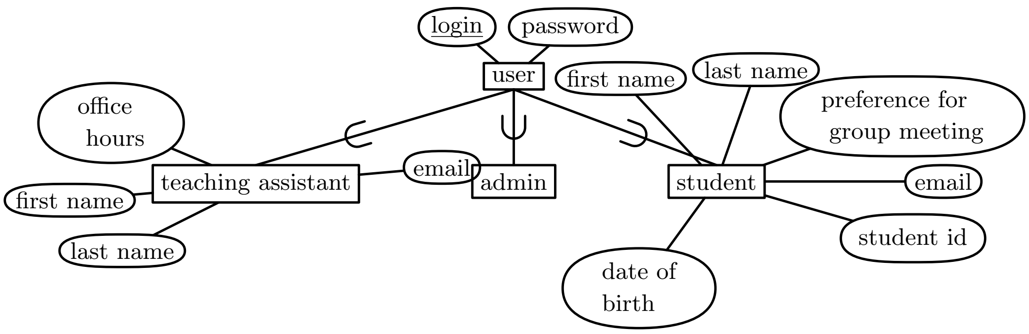


Figure 2.2: CIS users: Groups and identity

The three user groups have a common super-type *user*, which is used for the authentication within CIS: Every *user* has a login name (also called the *nickname*) and a password (more precisely, the database contains a hashed version of the password). Every user belongs to a unique group.

The administrators currently have no attributes. For every teaching assistant, CIS contains the name, the office hours and email address. That information is meant to be displayed on the web-pages, so that students can contact their TA without difficulty.

The attributes of students mostly concern their identification, i.e. the name, student id and date of birth. When signing up for the course, a student may also state her preferred times for the weekly group meetings, so that the actual schedule can try to minimize conflicts in the students' schedules (see section 2.5.3).

2.2.2 Homework Assignments

CIS aids in the administration of weekly homework assignments. In the ER diagram in figure 2.3 we assume that assignments are given in the form of worksheets, which are available in PS and PDF formats. A worksheet consists of several independent problems to be solved, each of which has an associated maximal score. From time to time, a problem turns out too complex for the given amount of time. Since the CIS enforces submission deadlines, extensions to single problems must be stored in the attribute *due date*. CIS considers *problem* a weak entity, because its key *serial number* may be counted from one for each worksheet.

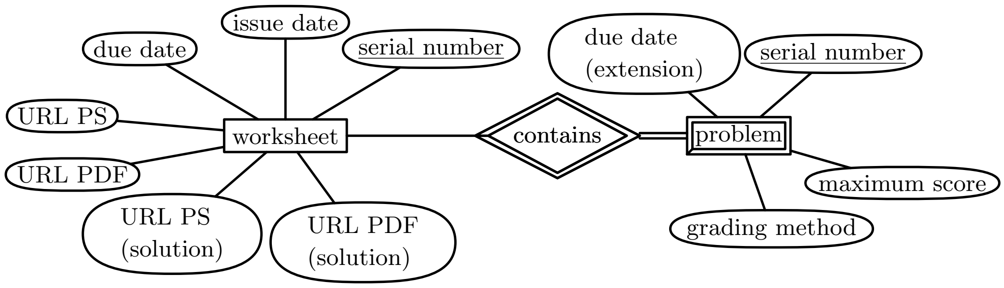


Figure 2.3: Worksheets and assignments

In summer term 2001, an attempt has been made to automate a large part of the grading process. It amounts to running submitted programs on a large number of test cases to give the teaching assistant an idea of the correctness. The attribute *grading method* contains a description of the procedure to be applied to submissions for this problem. The default is forwarding to a teaching assistant.

2.2.3 Submissions and Grades

For every problem on the worksheets, a student may submit up to five solutions, the most recent one is considered for grading. The idea is that students may submit partial solutions during the week and update them with enhanced versions later on, which encourages early submission and takes some load from public computer pools shortly before the deadline.

Every submission consists of a single file, which is stored in the server's local file system. The complete path to that file is stored in the *file* attribute of the submission and the *submission date* is set to the current date and time. In case of a paper submission, a *submission* with an empty *file* is created as soon as the teaching assistant enters the attained score.

The remaining three attributes concern the grading result. We store the assigned score, the TA responsible for the assignment, and the timestamp of the last update. The ER diagram is depicted in figure 2.4.

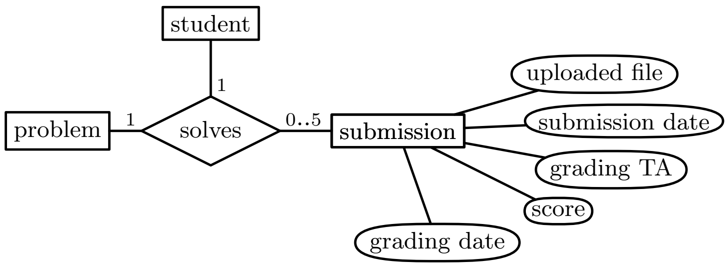


Figure 2.4: Submissions and Grades

2.2.4 Managing Groups of Students

Conventionally, a teaching assistent is responsible for a group of students throughout the semester. CIS supports the management of these groups, although it does not provide any means to determine the groups at the beginning of the semester from the students' preferred meeting times (section 2.2.1). That optimization problem is supposed to be solved off-line by the administrators (see section 2.5.3).

Figure 2.5 shows the entity *group* and the relationships to *teaching assistant* and *student*. A teaching assistant may be responsible for any number of groups, but every group needs exactly one TA and every student is assigned to at most one group. A group has as attributes the regular meeting time and an artificial *identifier*, which is used as a key. In most settings, also time and place together constitute a key, provided that the description of the meeting point is accurate enough (so that no two groups can meet at the same place during a period of time). Experience shows, however, that an artificial identifier also facilitates communication and is quickly accepted by the students.

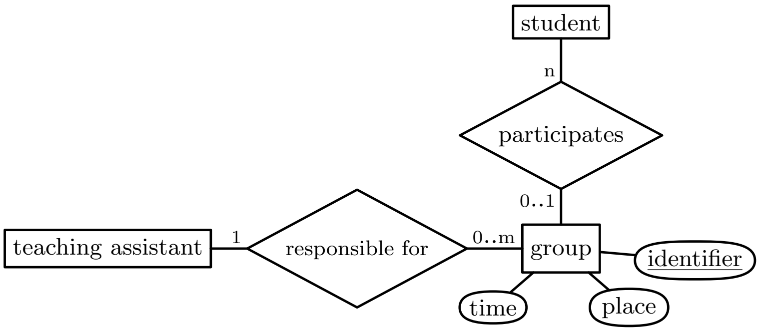


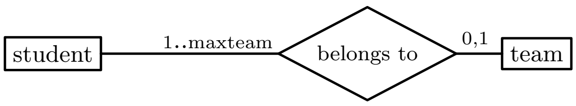
Figure 2.5: Groups of students

2.2.5 Teams of students

If it is desirable that teams of students work together and submit a single solution, CIS must offer support for this policy. In other words, a student must receive grades for submissions which she has not handed in herself. Unlike in conventional scenarios, where a team is personally known to the responsible teaching assistant, there are some subtle fallacies to be overcome.

1. The specified cardinality ratio must be strictly enforced, i.e. a team must never consist of more than the specified number of members.
2. The statement by one student A, that she is in a team with some other student B, must not transfer B's grades to A. As a first principle, any statement by A concerning team mates may only result in *giving* A's grades to other students.
3. The statement by A must not bind B, that is B can always reject the offered grades from A. Otherwise, because of the maximum size of teams, B may not be able to specify her real partner C.
4. Teams may change from one worksheet to the next, in case a student drops the course or other problems occur.

Specifically consideration 2 shows that the relation among team members is really directed. Therefore we cannot simply model it as follows:



CIS conceives teams as consisting of students who donate the grades of their submissions to the other team members. However, this relation is not transitive, i.e. a student does not forward grades given to her by other team members. Furthermore, a student gets only one grade for each problem, i.e. only if a student does not submit a solution herself, she can receive a donation from another team member.

The ER diagram is shown in figure 2.6. The cardinality ratio 1–`maxteam`, together with the restriction to a non-transitive relation, is sufficient to ensure that grades are not shared among excessively large groups of students, i.e. each grade given by a teaching assistant is at most attributed to `maxteam` students. However, intuitively a team is understood as a closed group of students no larger than `maxteamsize`. This requirement can be stated more precisely: The connected components of the undirected *shares with* relation must have a size smaller than `maxteam`.

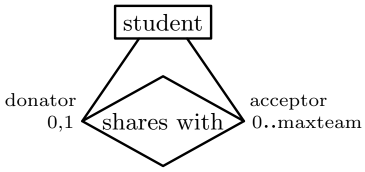


Figure 2.6: Teams in CIS

CIS currently implements the special case `maxteam` $= 2$, which obviously simplifies the enforcement of the last mentioned restriction, since transitivity in connected components can be neglected: If a student A shares her grades with a student B, than B can only shares her grades with A, if at all.

2.3 Relational Schema

The mapping from the ER model given in the previous section to a relational schema expressible in the SQL data definition language is mostly mechanic (section 6.8 in Elmasri/Navathe [4]): As a rule of thumb, an entity is mapped to a table and the entity's attributes become columns of the table. Relationships are implemented by foreign keys, i.e. a relation between two entities is established by adding the first entity's primary key to the table of the second entity.

We have chosen the tables' and columns' names as the names of the respective entities and attributes from the ER diagrams, so the reader will follow the SQL queries in the subsequent sections without difficulty. The full SQL definition can be found in appendix A.1.

There are two noteworthy exceptions from the standard mapping outlined above:

- The table `submission` has an internal column `submission`, which serves as the primary key. Since submissions sometimes are referred to in external files (section 3.3.1), this gives a concise way of identification. Furthermore, the redundancy between the `submission` and candidate key $(student, worksheet, problem)$ can be used for error detection in that context.
- The entity *submission* is realized by two tables `submission` and `grade`. This way, the `grade` table can be read-only for the student data base user, even though MySQL does not support access control at the column level.

2.4 Web Interface

The web-interface of CIS provides a collection of HTML pages with PHP scripts to invoke the implemented transactions. In general, there is a separate page for each transaction. Besides, there is a PHP framework to handle user authentication and navigation, and that part is shared among all pages.

The common behaviour is also visualized for the user in a header appearing at the top of each HTML-output page. A snapshot is shown in figure 2.7, with the main areas highlighted: In the upper right corner, the header contains the fields for login name and password for authentication (fig. 2.7 (a)), or the user name, if authentication is completed (fig. 2.7 (b)). At the bottom there is a two-level menu for navigation.

(a)

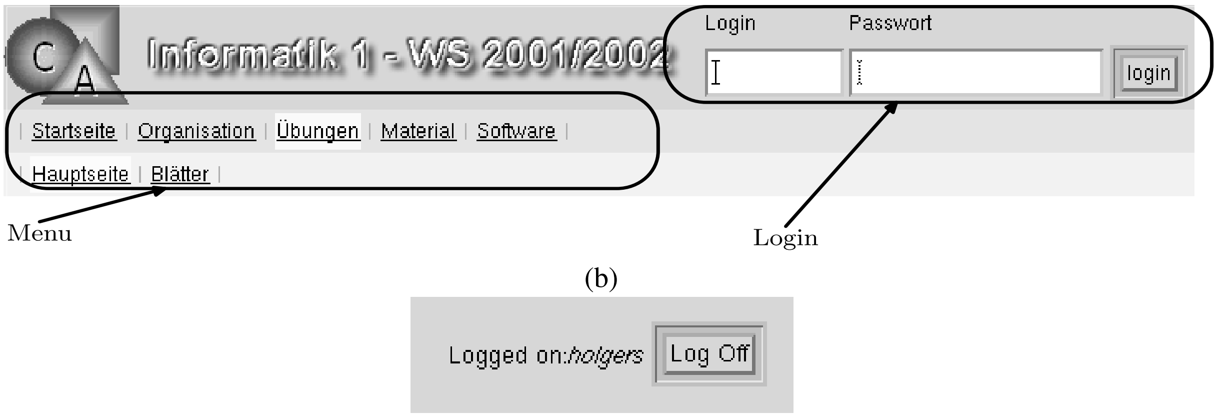


Figure 2.7: Header of the web-pages

2.4.1 User Authentication and Privileges

User authentication is session-based, that is the user logs on to the system with login name and password and is considered authenticated until she explicitly logs off or the session is timed out. At login time, the user's group is determined from the *user* table (section 2.2.1) and she gains access to the transactions available to that group.

There are two standard error screens concerning the authentication: The first one informs the user that the authentication has failed because of an invalid login name or password. The second screen is displayed in case a user accesses a page which is not currently available to her, because she belongs to the wrong group or is not logged on at all.

2.4.2 Navigation

Most of the navigation within the CIS interface is done through the two-level menu in the standard page header (figure 2.7). Since the transactions are mostly self-contained and require only one page, there is seldom need for a higher degree of cross-referencing.

The displayed menus are context sensitive:

- The items in the second line depend on the choice in the first line.
- The current choices in both lines are highlighted.
- If a user is authenticated, new items may appear at the right to the publicly available ones in both lines, giving access to the restricted transactions.

The menu mechanism is fully data-driven (see section B.9), such that the contents can be easily adapted and extended. The menu structure currently provided by CIS is detailed in appendix B.

2.5 System Deployment

System deployment can be divided into two main categories. First, a server machine has to be configured that will be used to serve all client requests (see figure 2.1). This has to be done only once. Second, for each course the initial state of the course system has to be established. We will give a rough outline of the basic steps for both of these two tasks.

2.5.1 Setting up the Server Machine

The main software components required on the server machine were already mentioned in section 2.1. In this section we give a few more details on the software installation.

All data that is handled by CIS should be stored on a separate partition or even hard disk. This simplifies both backup tasks and security precautions. Most notably, this data consists of the MySQL databases, the web-pages for the managed courses as well as the special files associated with each course.

MySQL A standard MySQL installation suffices for CIS.

Apache We need an Apache installation that supports SSL connections and PHP. In our setup, we achieved this goal by employing the following Apache modules:

1. Module `mod_ssl` is used for secure connections. This module requires an installation of the OpenSSL library.[2]
2. The PHP module that enables dynamic content in web-pages.

PHP A standard PHP installation suffices for CIS. However, we still register globals, so maybe one has to enable this option for an existing PHP installation.

Once the server machine is set up properly, it can host an arbitrary number of courses taking place concurrently.

2.5.2 Initializing CIS

Currently, CIS requires some manual work in order to get an initial system state from which the course can progress. A detailed description of these steps is provided in the CVS module `cicinstall`. Here, we sketch the major procedures.

Create the Course Database This is achieved by running the script `schema.sql`. Attention: Adjust the name of the courses database, otherwise data from another course may be compromised.

Create Branch All changes to CIS for a course should be localised to its own branch. We employ CVS for version management purposes. Eventually, bug fixes and functional additions to CIS should be merged in to the main development trunk.

Adjust Branch Usually, this involves changing the courses name and description, the homepage logo, and so on. Most of the internal changes are automated by just setting the variable `$dbname`, which is used to parameterize course specific section (e.g. file and directory names).

Create Server Directories As mentioned throughout this report, CIS stores parts of its data in the server file system. Therefore, for each course several directories have to be created. They are made up of the handout directories for students and TAs respectively, and the handin directory.

[2] Furthermore, notice that you need a server certificate and key by which the server identifies itself to connecting clients.

Checkout Course Branch Once all the previous steps are carried out, we can checkout the new branch for the current course on the server machine. It will be the server root directory for the Apache web server. This working copy should be kept up to date with the local copies of administrators handling the course system.

Initialize Database In order to start working with CIS, an initial user account has to be created. It should belong to the administrator group:

```
INSERT INTO benutzer(gruppe,login,passwd)
VALUES ('admin','...',PASSWORD('...'));
```

Now the system is ready to go. After the students have enrolled through the web interface, a final major tasks has to be performed. This is assigning each student his teaching assistant/group.

2.5.3 Assigning Students to Groups

A major challenge at the start of each course has been to find a fair distribution of students to available groups according to the students' preferences. In Winter Term 2002/2003 the Parallel Computing Group used CIS and they provided a Java program to solve the distribution problem based on a network flow graph with a minimal cost flow algorithm [1].

The input to the distribution generator is extracted from the database which contains the necessary information gathered when students sign in to CIS. The program needs a list of all students and their ranked group preferences. The generator outputs the assignment of students to one of their preferences, trying to respect their priorities. Currently, an SQL script is emitted that updates the student table (see section A.1).

Chapter 3

Available Transactions

The functionality of CIS is based on a set of transactions, each of which queries or modifies the database or file repository. This organization of the software reflects the stateless nature of the HTTP protocol used for web-access: The user-requests are broken down into small, self-contained tasks, which are processed independently. The only state shared between all pages is the information on the user's authentication. Likewise, the menu structure used to access the single pages can be chosen as desired for a particular course. The current two-level menu can be found in appendix B.

In this chapter we describe the available transactions. Although we omit the fairly obvious PHP implementation, we do include the SQL queries from the source code for a precise definition of the database accesses. The queries' meaning can be understood from the database design in section 2.2, the full SQL data definition is included in appendix A.1.

We will use the following common format for the description of transactions.

Transaction: *Name which is used for reference in the text*
Group: *Group which has the permission to use the transaction*
Source: *HTML/PHP file containing the implementation*
Form Data: *The input provided in HTML forms, if any*
Response: *Description of the HTML response of the transaction*
Effect:
The modification to the data base state and file repository
Queries:
SQL source of the queries for a more precise definition of the effect and output

An entry **Group** states that the transaction is only available to authenticated users from that group (student, teaching assistant or administrator). Since authentication is session-based, any transaction with non-empty **Group** has access to the user id *user*, which is a small integer constant and can be used in queries.

3.1 General Contents

CIS serves as a central information repository for one course. Therefore, it also contains conventional hyper-text documents with largely static contents. However, some of that contents is also represented in the database, for instance the names and office hours of the teaching assistants. Those pieces of information are queried dynamically to establish a consistent and up-to-date display.

Contact Information for Students It is desirable to have teaching assistants and students discuss open questions via email between regular meetings. Hence, CIS outputs a view on the entire table `TA`.

Transaction: *Query TA Contact Information*
Source: `contact.html`
Response: A list of the teaching assistants with their email address, office hours and group meeting.
Queries:

```
SELECT firstname, lastname, email, group.room, time, day.short
FROM TA LEFT JOIN Group ON Group.ta=TA.user NATURAL JOIN Day
ORDER BY firstname, lastname
```

Remark: The table `Day` is just a mapping from the weekdays' codes to their names.

Schedule of office hours When office hours of teaching assistants may be visited by any student in the course, not only by the TAs group, a quick overview over the weeks schedule may facilitate matters.

Transaction: *Overview office hours*
Source: `sprechstd.html`
Response: The TAs' office hours are presented in the form of a schedule. Each hour of the weekdays received a field and those where a TA is available are marked green. The office hours must be stored in the format *`Day,start-end`* and separated by semicolons. Whitespace may precede and follow the comma. For instance `Fr, 11-13; Mo, 12-14` would be a legal entry in a TA's the `office_hours` attribute.
Queries:

```
SELECT office_hours FROM TA
```

Announcements from the Administrators The bulletin board of CIS is the place for administrators to post announcements. Each posting carries an expiration date and is removed from the display afterwards. It does not vanish from the database, though, so that it can be referred to in case discussions about its contents arise later on. Since the postings might well concern questions of grading, extensions to problems and final exams, the history is important.

Transaction: *Read bulletin board*
Source: `index.html`
Response: The current announcements posted on the bulletin board by the administrators. Only the non-expired entries are selected and the most recent postings are shown at the top.
Queries:

```
SELECT date,text
FROM cur_info
WHERE NOW()<expire
ORDER BY date desc
```

Enrolling for the Course At the beginning of the semester, the participants of the course register with CIS by entering their personal data (see section 2.2.1). They also choose a login name and initial password for authentication in CIS. The procedure is implemented as three distinct steps:

1. Query information via an HTML-form. Until the data is complete and valid (e.g. login names must be kept unique), the form is presented to the user. The missing or invalid fields are marked with a red bullet.
2. When the information is complete, a tabular summary of all fields is presented and the user is asked for confirmation. She can also choose to go back to step (1) with the current data.
3. After confirmation, a new CIS-user and student entry are created and the user is informed about the successful completion. The CIS-account is immediately ready and the user can authenticate using the chosen login name and password.

Transaction: *Enroll for the course*
Source: `subscribe.html,subscribe_check.html,subscribe_complete.html`
Form Data: The personal data for the entities *student* and *user* (see section 2.2.1).
Response: Welcome message in `subscribe_complete.html`
Effect:
A new user with the given login and password is created in the table `user`. If this has been successful, a new student with the just-established user-id is inserted to table `student`.
Queries:

```
INSERT INTO user (group,login,passwd)
VALUES ('student', 'login', PASSWORD('password')')
```

This query has establish a new `user`-id *user*, which is the foreign key in the `student` table

```
INSERT INTO student (user,firstname,lastname,dateofbirth,studentid,email,
                    uegr1,uegr2,uegr3,uegr4,uegr5) VALUES
       (user,'firstname','lastname','date of birth','studentid',
       'email or NULL if email is not given',
        'pref1','pref2','pref3','pref4','pref5')
```

Announcement of Groups After registration deadline, the administrators decide about the weekly group meetings of students and teaching assistants. The students are distributed into groups based on the preferred time and day they stated during registration. The outcome of that decision can be read directly from the database, so it can be published on the web.

Transaction: *Display groups with meeting time and responsible TA*
Source: `aha_gruppen.html`
Response: A list of groups with the TAs. Each line contains a link to the list of students in the group. The students in each group are listed as well.
Queries:

```
SELECT firstname, lastname, Group.group,
       time, Day.short, room
FROM TA INNER JOIN Group ON Group.ta=TA.user NATURAL JOIN Day
ORDER by day.short, hour, lastname, firstname
```

For each of the groups, the following query retrieves its members.

```
SELECT firstname, lastname
FROM student
WHERE group=group
ORDER BY 1,2
```

Transaction: *Private information of group assignment*
Group: Students
Source: `matr_gruppe.html`
Form Data: The student id
Response: The group and TA the student is assigned to
Queries:

```
SELECT group,day.short,hour,room,TA.vorname,tutor.nachname,tutor.email
FROM student NATURAL JOIN group NATURAL JOIN day
INNER JOIN TA ON TA.user=group.ta
WHERE student.id='student id'
```

3.2 Transactions for Students

The students are the main user group of CIS. They subscribe for the course, download worksheets and request information on the organizational issues. Their most important concern, however, is the upload of submissions and their personal grades stored in the database.

3.2.1 Change of Personal Data

The database contains personal data from students as needed for issuing credits for the course. Currently, this includes the first and last name, and the date of birth (see section 2.2.1). Since submissions and grades are closely tied to CIS-users, the identifying data of students must not be changed arbitrarily, because otherwise the achievements in the course could be handed over to a different person. Instead, the page to display the currently stored personal data contains a link with an administrator's email address to request required changes.

Transaction: *Display personal data*
Group: student
Source: `showpersonal.html`
Response: Current state of identifying personal data and email address of an administrator to request necessary changes.
Queries:

```
SELECT firstname,lastname,login,dateofbirth,studentid,email
FROM user NATURAL JOIN student
WHERE user.group='student' AND student.user='user'
```

Students may change their passwords for CIS with the following transaction.

Transaction: *Change CIS-password*
Group: student
Source: `passwd.html`
Form Data: old password, new password (twice)
Response: Message on success or failure of the transaction
Effect:
Changes the *passwd* attribute of the current user in the `user` table if the old password matches the stored one and the new password has been entered correctly twice.
Queries:

Update password securely, i.e. with check of old password
`UPDATE user SET passwd=PASSWORD('`*new password*`')`
`WHERE user=`*user* `AND passwd=PASSWORD('`*old password*`')`

3.2.2 Handouts with restricted access

Some of the material offered for a course may be subject to copyright and other formal restrictions, so one may want to distribute them only to members of the course. Such a policy is implemented by the script `handout.php`. The script requires a separate directory, which is not publicly accessible via the web, as a pool for the restricted material. When a user requests `handout.php?file=`*`filename`* with her browser, the script returns the desired file only after checking the user's permission. The current policy is to allow download for authenticated users, who have either submitted some solution to a problem within the last three weeks or have the flag `gethandouts` in the table `user` set to `yes` (see appendix A.1 for the table definition).

These conditions aim at determining whether the user is indeed an active member of the course. If they are not met by a user, the page `handoutrestrict.html` is returned instead of the desired file, explaining the failure and the exact terms of downloading handouts.

Transaction: *Get handout*
Group: student, teaching assistant
Source: `handout.php, handoutrestrict.html`
Form Data: As `GET` parameters: *file*
Response: The desired file, if the user is allowed to download handouts. The file is returned with HTTP header information on text or binary content and the file name for saving:

```
Content-Type:  text/plain or application/octet-stream
...
Content-Disposition:  attachment; filename="file"
...
```

The fields for cache control have been omitted here.
Queries:

(1) Policy: If a student has submitted a solution within the last 3 weeks, allow the download

```
SELECT count(*)
FROM submission
WHERE student=user AND d_submit >= DATE_SUB(NOW(),INTERVAL 25 DAY)
```

(2) Allow the download if flag in the `user` table is `yes`

```
SELECT gethandouts
FROM user
WHERE user=user
```

The list of handouts can be seen by any authenticated student.

Transaction: *Display list of handouts*
Group: student, teaching assistant
Source: `handouts.html`
Response: The list of available restricted handouts, each with a link to `handouts.php` with `GET` parameter `file=`*filename*.
Queries:

The list of handouts is currently not implemented in a database table. Instead,
the directory with the handout files also contains a file with a short

textual description for each handout. The structure of this directory is maintained by suitable transactions (see. section˜3.4.5). This implementation will be replaced in the near future.

3.2.3 Worksheets, Submissions and Grades

CIS manages the worksheets and homework assignments of a coure. The regular tasks to be supported have been outlined in section 1.2.1 and the resulting ER schema is found in section 2.2.2. For the student group, there must be transactions to download worksheets, submit solutions and receive their current grades.

Transaction: *Display available worksheets*
Source: `blaetter.html`
Response: A list of the worksheets handed out so far (attribute *published*=`yes`). For each worksheet, the contained problems are shown.
Queries:

```
SELECT w.w_sheet, w.d_avail, w.d_due,
   w.url_ps, w.url_pdf,
   w.url_sol_ps, w.url_sol_pdf,
   p.problem, p.max_score,p.d_due
FROM worksheet AS w
 LEFT OUTER JOIN problem AS p USING (w_sheet)
WHERE w.published='yes'
ORDER BY w_sheet desc
```

Submission of solutions For each problem in table `problem`, CIS accepts one file as a solution (section 2.2.3). That file is stored in a designated directory, i.e. CIS creates a file

$$\textit{local file} = \textit{submission directory}/\textit{CIS-userid}/\textit{submission file}$$

The *submission file* is built from the worksheet and problem numbers plus the date and time of submission.[1] Then a new entry in table `submission` is created containing a reference to that file.

Transaction: *Submit solutions*
Group: student
Source: `handin.html`
Form Data: For each uploaded file f the *worksheet*, *problem*
Response: List of currently submitted solutions and error messages if any.
Effect:
Stores uploaded (temporary) file f into local file *local file* and runs query for each upload.
Queries:

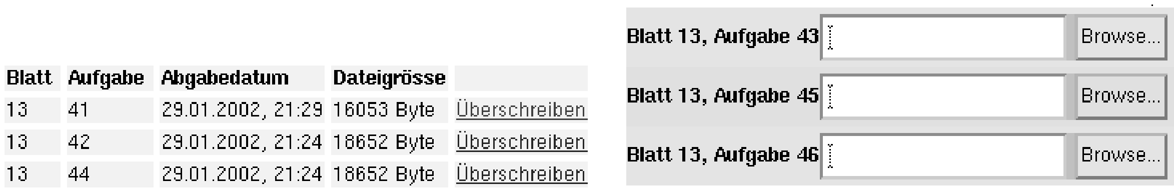
Try to insert a new row (with `UNIQUE` constraint, see section A.1)

```
INSERT INTO submission (w_sheet,problem,student,file)
VALUES ('worksheet','problem','user','local file')
```

[1] This redundancy with the data in the `submission` table was originally intented to allow recreation of the submission table from the stored files. Though the case has never occurred, the precaution still seems sensible, since the set of submissions, besides the related `student` table, is the part of CIS most crucial for reliability.

Issue an error message to output if query had no effect.

A student must be informed about her current submissions and those problems which she has not yet solved. In that context, only the problems with a deadline in the future are relevant, the others can be tracked with transaction *Check grades*, page 26.



Blatt	Aufgabe	Abgabedatum	Dateigrösse	
13	41	29.01.2002, 21:29	16053 Byte	Überschreiben
13	42	29.01.2002, 21:24	18652 Byte	Überschreiben
13	44	29.01.2002, 21:24	18652 Byte	Überschreiben

(a) (b)

Figure 3.1: The student's submissions on open problems

Transaction: *Query current submissions*
Group: student
Source: `handin_cnt.html`
Response: For each of the currently open problems from worksheets the output page displays the following items:

- If the student herself or her partner (see section 2.2.5) has submitted a solution, the submission date and time, and the file size are shown to help the student identify the file. If the submission is by the student herself (instead of her partner, section 2.2.5), there is also a link for replacing the file (transaction *Replace Submission*, page 26).
- In case the student herself has not submitted, an HTML form element for uploading a file is appended.

Queries:

```
SELECT p.w_sheet, p.problem, s.student,s.d_submit,s.file,
       p.p,p.q,pp.student,pp.d_submit
FROM (worksheet AS w INNER JOIN problem as p ON w.w_sheet = p.w_sheet)
LEFT OUTER JOIN submission AS s
   ON p.w_sheet=s.w_sheet
     AND p.problem=s.problem
     AND s.student='user'
 LEFT OUTER JOIN partner AS pa
    ON pa.q='user'
      AND pa.w_sheet=b.w_sheet
 LEFT OUTER JOIN submission AS pas
    ON pas.w_sheet=pa.w_sheet
       AND pas.problem=p.problem
       AND pas.student=pa.p
-- select only currently open problems
WHERE (NOW() >= w.d_avail AND (NOW() < w.d_due OR NOW() < p.d_due))
```

For each problem in the worksheets, there is one currently valid solution. It does not get overwritten with new solutions, unless the student explicitly requests the substitution by the following transaction.

The previous solution becomes a backup. In the current version of CIS, the backup is not accessible to the teaching assistants, but a corresponding transaction can be easily added.

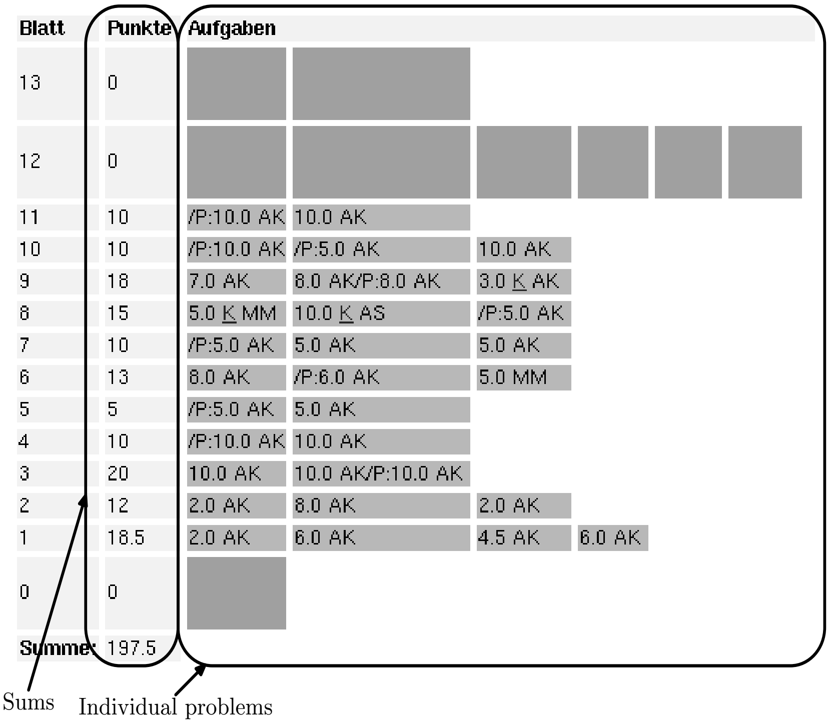
Transaction: *Replace Submission*
Group: student
Source: `handin.html`
Form Data: For each uploaded file *f* the *worksheet*, *problem*
Response: List of currently submitted solutions and error messages if any.
Effect:
This transaction moves the current submission for each pair of *worksheet* and *problem* into table `prev_submissions`, deletes all but the most recent 5 submissions from that table and then treats *f* as a new submission for the *worksheet* and *problem* (transaction *Submit solutions*, page 24).
Queries:

For each uploaded file
(1) Get the previous submission for the problem
```
SELECT submission,file,d_submit
FROM submissions
WHERE student=user and w_sheet=worksheet AND problem=problem
```
(2) Store in the table `prev_submissions`
```
INSERT INTO prev_submissions(student,w_sheet,problem,submission,file,d_submit)
VALUES ('$s','$b','$a','$ab','$fi','$d')
```
(3) Find the previous submissions by date
```
SELECT submission,file FROM prev_submissions
WHERE student='$s' AND w_sheet='$b' AND problem='$a'
ORDER BY d_submit DESC
```
(4) Skip 5 entries from query (3) and delete all remaing
```
DELETE FROM prev_submissions WHERE submission='$ab'
```

Query the grades The students will want to check their current grades in the course regularly, so it is important to provide a concise presentation. Figure 3.2 shows a screenshot of the resulting overview.

Transaction: *Check grades*
Group: student
Source: `checkgrades.html`
Response: A detailed presentation with the current user's grades. The output table contains one row per worksheet with one cell per problem. The cell shows the grade assigned to the student's solution and to that of her partner, if any. Furthermore, the initials of the responsible TA (table `grades`, section 2.2.3) are included. Sums of scores are computed per worksheet and overall.

There are visual marks for the status of each problem: gray means "no solution submitted", green means "grade is available" and red means "submission not yet graded".
Effect:
Queries:
```
SELECT p.w_sheet,p.problem, p.max_score,
       s.d_submit, g.score, ta.firstname, ta.lastname,
       pas.d_submit, pag.score, pat.firstname, pat.lastname,
```



Blatt	Punkte	Aufgaben			
13	0				
12	0				
11	10	/P:10.0 AK	10.0 AK		
10	10	/P:10.0 AK	/P:5.0 AK	10.0 AK	
9	18	7.0 AK	8.0 AK/P:8.0 AK	3.0 K AK	
8	15	5.0 K MM	10.0 K AS	/P:5.0 AK	
7	10	/P:5.0 AK	5.0 AK	5.0 AK	
6	13	8.0 AK	/P:6.0 AK	5.0 MM	
5	5	/P:5.0 AK	5.0 AK		
4	10	/P:10.0 AK	10.0 AK		
3	20	10.0 AK	10.0 AK/P:10.0 AK		
2	12	2.0 AK	8.0 AK	2.0 AK	
1	18.5	2.0 AK	6.0 AK	4.5 AK	6.0 AK
0	0				
Summe:	197.5				

Figure 3.2: Summary of a student's grades

```
        pa.p, pa.q
FROM -- for each problem from the worksheets ...
  problem as p
  -- retrieve the students submissions and grade, if any ...
  LEFT OUTER JOIN submission AS s
    ON p.w_sheet=s.w_sheet AND p.problem=s.problem AND s.student=user
  LEFT OUTER JOIN grades AS g USING (submission)
  LEFT OUTER JOIN TA ON g.ta=TA.user
  -- and those of the partner, if any
  LEFT OUTER JOIN partner AS pa ON pa.q=user
            AND pa.w_sheet=p.w_sheet
  LEFT OUTER JOIN submission AS pas
    ON p.w_sheet=pas.w_sheet AND p.problem=pas.problem AND pas.student=pa.p
  LEFT OUTER JOIN grades AS pag ON pag.submission=pas.submission
  LEFT OUTER JOIN TA AS pat ON pag.ta=pat.user
-- order for tabular rows and columns
ORDER BY p.w_sheet DESC, p.problem
```

3.2.4 Evaluation

At the end of the term, students fill in a questionnaire to evaluate the performance of teachers, teaching assistants and the general success of the course. The traditional way of handling the answers was by manual processing and counting of answers, which was tedious and error-prone, and an evaluation took about four weeks to complete. With CIS in place, there is no need to distribute questions on paper anymore: The questionnaire can be read and answered online as an HTML form and the answers can be read immediately, which makes more than one evaluation per term feasible. The set of questions use in between 2001 and 2003 in shown in appendix C (in German).

There are three problems to be addressed in order to make the results reliable:

1. The survey must be strictly anonymous and the students must be intuitively sure that their answers cannot be traced back to them.
2. Only students from the course itself are allowed to participate.
3. No student may file more than one set of answers.

The first point has been treated by making the `evaluation.cgi` script independent and separate from the remainder of CIS: Being in a common environment with well-known look-and-feel would have suggested a connection to the students' nicknames, and accidential login-data stored in cookies would have troubled the students. The last two points are solved by the concept of *transaction IDs* (or TANs for short): In the course meeting itself, each student receives a randomly generated string of characters that allows him or her to answer the questions online exactly once. The implementation abstracts over the actual questions to be reusable in different contexts.[2] There are three general forms of questions, depending on the expected form of answer; each question automatically contains cases *not-applicable* and *don't-know*.

ranges present several radio-buttons with values $min \dots max$.

selections present a given number of check-boxes.

values are distinct (textual) values, represented as radio-buttons.

multi-values same as **values**, but with check-boxes.

text fields are used for free-form comments which may be most helpful for the teacher in improving the course over time.

Each of these questions can be generated by a function call and uses `<tr>`, `<td>` tags for formatting. For instance, the following questions appear aligned in the HTML-form:

```
<table border="1" rule="all">
<tr><th><th><th><th>0-25%<th>25-50%<th>50-75%<th>75%-alle
<?
qst_sel_n("tutorials",
          "How often did you meet the TA in her office hours?",4);

qst_val(2,"questions",
        "Have you taken the opportunity to ask questions in the course?",
        "yes","no");
?>
</table>
```

Each function call to a `qst_...` function also registers the defined form-field for checking and storing the value. Hence, once the questionnaire is described by the function calls, the data processing is automatic. The data is stored in a database table whose `CREATE`-statement can be output by the `evaluation.cgi` script.

[2] `evaluation.cgi` can be obtained from the authors separately.

3.3 Transactions for Teaching Assistants

One of the main goals of CIS is the maintenance of a central repository for the students' grades. The benefits of such an effort are twofold: First, the processing of the data can be largely automated, e.g. when the final scores need to be computed. Second, the lecturer of the course can get an up-to-date account of the students' achievements any time. For instance, she may wish to consult the overall results in a particular problem from the worksheets to estimate the understanding of the concepts in question, which in turn can help her design the teaching schedule.

The teaching assistants play a crucial rule in this scheme: They have to enter scores and grades of every single problem for every student as soon as the grading is finished. If they are to benefit from the automatic processing of the data at all, the input-procedure must not be complicated nor cumbersome.

This section gathers the transactions of teaching assistants. They are concerned with accessing the electronic submissions and entering the grades to the central repository. We shall see that there is more than one way to go, and the TAs may choose according to personal preferences and working conditions.

3.3.1 Offline grading at home

The teaching assistants will have different preferences concerning the time and environment they use for grading. For some, this means working at home with a dial-in connection or even without access to the Internet. It is particularly important for this group to minimize online-time, so that CIS allows teaching assistants to download all submissions for one worksheet in one transaction and to upload the resulting grades in one file. That latter transfer must be implemented carefully: Typing errors must not lead to corrupt database states, but they should be detected as errors.

Transaction: *Download submissions of groups*
Group: teaching assistant
Source: `correct.html`
Form Data: The serial number *w* of the selected worksheet.
Response: A `.zip` file with the submissions. The name of the file is `worksheet`*w*`.zip` and the contained files are decompressed to a subdirectory named `worksheet`*w* so that different downloads are surely disjoint. The individual file names in the archive consist of the student's first and last name, and the serial numbers of the worksheet and problem. There is also a content description with one line per submission: That line consists of the file-name, and again the serial numbers and the student's user id. At the end of each line, a field is left blank for the grade, which can be filled in with any text-editor and used for uploads with transaction *Upload grades* below.
Queries:

```
SELECT firstname,lastname,w_sheet,problem,file,suffix
FROM Student
   INNER JOIN Group USING (group)
   INNER JOIN Submission ON student.user=submission.student
WHERE ta='u' AND w_sheet='w' AND file LIKE "/data/handin%"
ORDER BY submission.problem,lastname,firstname
```

Grading can be done with two short online sessions: First, the TA downloads the submissions for a worksheet. Second, after assigning the grades, she writes them into the content description file contained in the download-archive and uploads that file to the information server.

Transaction: *Upload grades*

Group: teaching assistant
Source: `correct.html`
Form Data: The content file from transaction *Download submissions of groups* with the blank *grade* field filled in.
Response: Result of processing the input file, with detailed error messages for each inconsistent line.
Queries:

For each line in the input file, split the line into the fields *submission*, *worksheet*, *problem*, *student* and *grade*.
Then check the following query returns 1.

```
SELECT count(*)
FROM submission
WHERE submission='submission' AND w_sheet='worksheet'
  AND problem='problem' AND student='student'
```

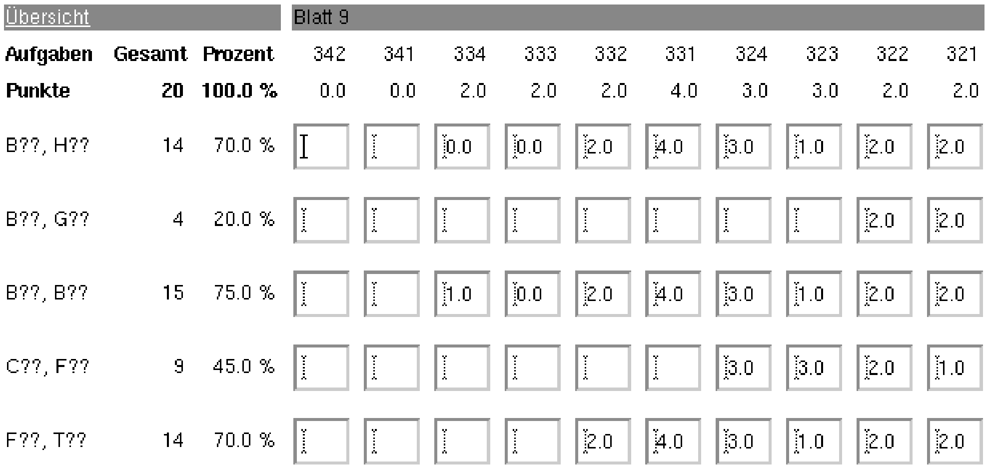
If this consistency check is successful and the *grade* is a number, execute the replacement of any previously assigned grade.

```
REPLACE grades (submission,grade,comment)
VALUES ('submission', 'grade', 'comment')
```

Remark: If `UPDATE` is used instead of `REPLACE`, no involuntary changes of earlier entries will take place. Due to the uniqueness of the primary key `grades.submission`, `REPLACE` has that effect otherwise.

3.3.2 Online transmission of grades

Submissions on paper have not been registered with CIS, and so they have not yet been assigned a unique submission id. Hence, the process described in the preceding section does not apply. Instead, CIS includes a dialogue for entering the grades for all students in a group for a single worksheet. It is assumed that the unknown submissions have been handed in in some other fashion, and CIS includes an empty *file* field in the `submission` table.



Übersicht			Blatt 9									
Aufgaben	**Gesamt**	**Prozent**	342	341	334	333	332	331	324	323	322	321
Punkte	**20**	**100.0 %**	0.0	0.0	2.0	2.0	2.0	4.0	3.0	3.0	2.0	2.0
B??, H??	14	70.0 %			0.0	0.0	2.0	4.0	3.0	1.0	2.0	2.0
B??, G??	4	20.0 %									2.0	2.0
B??, B??	15	75.0 %			1.0	0.0	2.0	4.0	3.0	1.0	2.0	2.0
C??, F??	9	45.0 %							3.0	3.0	2.0	1.0
F??, T??	14	70.0 %					2.0	4.0	3.0	1.0	2.0	2.0

Figure 3.3: Dialog for grades of one worksheet

Transaction: *Enter grades for worksheet*
Group: teaching assistant

Source: `gradesws.html`
Form Data: A serial number of a *worksheet* and possibly a list of quadruples

$$(student, worksheet, problem, grade)$$

If there is no submission for the combination of $(student, worksheet, problem)$, it is created with an empty file-reference.

The HTML-form for generating the input quadruples shows only students from the `user`'s group and *problem*s from a particular *worksheet* (screenshot in figure 3.3).

Response: A new HTML-form for the selected *worksheet*, with one text field for each problem and student. The text fields contain the currently valid grades.

Effect:

The quadruples are taken to describe new grades for the $(student, worksheet, problem)$ combination. If no submission exists for that combination, a new record in table `submission` is created with an empty `file` attribute, otherwise the old submission is retrieved. In any case, the record stored in `grades` for the submission is updated with the new value.

Queries:

Make sure that a submission exists for the worksheet and problem;
if it does, this query has no effect
`INSERT INTO submission (student,w_sheet,problem)`
`VALUES (`*user*`,`*worksheet*`,`*problem*`)`
Get the auto-generated *submission* identifier `mysql_insert_id()`
`REPLACE grades (submission,grade,ta)`
`VALUES ('`*submission*`',`*grade*`,`*user*`)`

3.3.3 Handouts and Passwords

The handouts with restricted access are of course available to the teaching assistants. The transactions differ from those in section 3.2.2 only in the technical detail of giving access to teaching assistants rather than students. The attribute `gethandouts` of the `user` records of all teaching assistants must be set to `yes`.

Since it is very likely that students will forget their passwords for CIS from time to time, teaching assistants are given the right to change password entries in the `user` table.

Transaction: *Set password for students and teaching assistants*
Source: `adminpasswd.html`
Group: teaching assistant, admin
Form Data: The *login* name and a new password *passwd* to be set. For increased robustness, the transaction also requires the expected *group* of the user whose password is being changed.
Response: Indication of success or failure.
Effect:

Updates the password for user with *login* name with a hashed version of *passwd* if the current `user` is allowed to change that password. The implemented policy is that both administrators and teaching assistants may change any student's password. A teaching assistant may of course change her own password, but only the administrator has access to all teaching assistants' passwords.

Queries:

`UPDATE user`
`SET passwd=PASSWORD('`*passwd*`')`

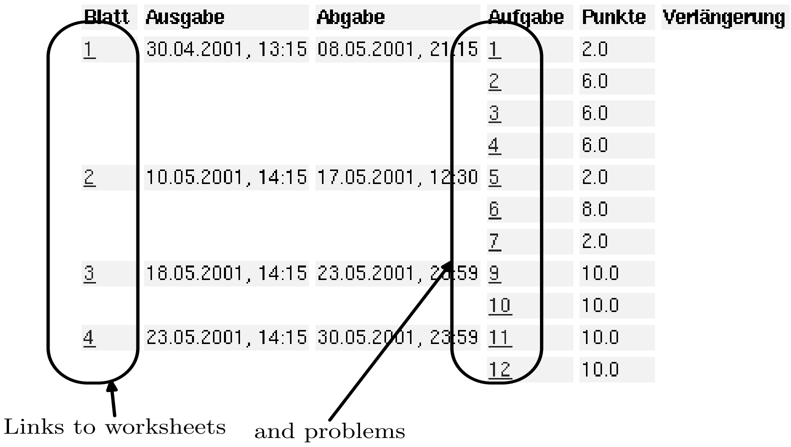


Figure 3.4: Central dialogue for administration of worksheets

```
WHERE login='login' AND group='group'
```

3.4 Transactions for Administrators

The group of administrators is responsible for the content of CIS's web-pages and the organization of the course. Their regular tasks consist mostly of the publication of worksheets and the management the teaching assistants. Depending on the structure of the course, this may include distributing the students' submissions for grading.

Besides, the administrators should be able to assist students in their problems with CIS, thus increasing the acceptance of the system as a reliable tool for participation in the course.

3.4.1 Worksheets and Solutions

Worksheets can be handled by administrators in dialogue-style interactions. The starting point is an overview table of the available worksheets with the contained problems (see figure 3.4). The transactions relating to worksheets come in two groups:

- Addition and removal of worksheets and problems.
- Modification of previously entered data.

While addition of new items is done using HTML forms directly on the main worksheets-page, modification and removal transactions are accessible via hyperlinks (figure 3.4). These links are contained in the overview and employ the HTTP `GET` method to parameterize the modification dialogues with the desired worksheet and problem serial numbers.

Transaction: *Enter new worksheet*
Group: administrator
Source: `worksheets.html`
Form Data: The attributes of a worksheet record (section 2.2.2, figure 2.3).
Response: The modified overview over the worksheets.
Effect:

If a worksheet with the desired serial number does not yet exists, a new record in the table `worksheet` with the given attributes is created. The attribute *published* is set to *yes* to enable the display.[3]

Queries:

```
INSERT INTO worksheet (published,w_sheet,d_avail,d_due,url_ps,url_pdf)
VALUES ('yes','worksheet serial no', 'date of publication',
        'submission deadline','URL of PS file','URL of PDF file')
```

Transaction: *Add problems to a worksheet*
Group: administrator
Source: `worksheets.html`
Form Data: A worksheet number *ws* and a list of problem numbers $p_1, \ldots, p_n$ with associated maximal scores $s_1, \ldots, s_n$. The actual input is a string with format p_1:s_1,p_2:s_2, ... ,p_n:s_n. If the string cannot be parsed correctly, the transaction is abandoned.
Response: The modified overview over the worksheets.
Effect:
If the selected worksheet *ws* exists, the problems $p_1, \ldots, p_n$ are created in table `problem`.
Queries:

```
INSERT INTO problem (w_sheet,problem,max_score)
VALUES ('ws','p1','s1'), ... , ('ws','pn','sn')
```

Modifying worksheet and problem attributes The attributes of the worksheet and problem entities (figure 2.3) can all be edited with HTML-forms `text` input elements.[4] The forms are requested using the links in the overview. The form is parameterized by the worksheet (and problem) numbers, which are provided by the HTTP `GET` method in those links.

Transaction: *Modify attributes of a worksheet*
Group: administrator
Source: `worksheets_bedit.html`,`worksheets.html`
Form Data: A worksheet number *ws* and values for the attributes of a worksheet.
Response: The modified overview over the worksheets.
Effect:
Updates the attributes of worksheet *ws* in table `worksheet` with the given values. Only the non-empty input values are considered for modifications, the other attributes are skipped.
Queries:

```
UPDATE worksheet SET d_avail='publishing date' WHERE  w_sheet='ws'
... for the other attributes of a worksheet
```

Similarly, the attributes of a problem can be modified.

Transaction: *Modify attributes of a problem*
Group: administrator
Source: `worksheets_bedit.html`

[3]This introduces a short time of inconsistency, in which a worksheet with empty file link is display. The next major revision will contain a separate switch for that attribute.

[4]The dates and times can be entered in the format *day*.*month*.*year*,*hour*:*minute* Functions named `make_db_date()` and `format_db_datetime()` are called to create and interpret the internal representation in the format *yyyymmddhhmm*.

Form Data: The serial numbers *ws* and *pr* of a worksheet and contained problem, together with a subset of a problem's attributes (figure 2.3).
Response: The modified overview over the worksheets.
Effect:
Changes the attributes of problem *pr* on worksheet *ws* to the given values. Only non-empty values are considered for updates, the other inputs are skipped.
Queries:

```
UPDATE problem SET max_score='new max score'
WHERE problem='pr' AND w_sheet='ws'
```

... for the other attributes of a problem

Distributing the submissions to the teaching assistants CIS currently supports two ways to assign submissions to teaching assistants for grading. First, the administrators may take the decision explicitly, i.e. for each problem the submissions are distributed evenly among a number of teaching assistants. Second, a teaching assistant may be made responsible for a group of students for the whole of the semester. The TA then regularly receives the submissions of her group. The latter functionality does not need new data structures (section 2.2.4).

3.4.2 Teaching Assistants

The table of teaching assistants must be filled at the beginning of the semester and occasionally updates, e.g. of the email addresses and of the office hours, will become necessary. CIS provides a single page for performing these tasks. That page contains an HTML form with all attributes of a teaching assistant (section 2.2.1) and buttons to trigger creating of a new record and update of the field values. Furthermore, one can move forward and backward between the existing records to view and edit them one by one.

Transaction: *Update attributes of teaching assistants*
Group: administrator
Source: `admintutor.html`
Form Data: The user id *u* of the teaching assistant and new values for the attributes of table `TA` (section 2.2.1).
Effect:
The attribute values of the TA identified by *u* are set to the new values.
Queries:

```
UPDATE TA SET officehours='office hours', ... WHERE user='u'
```

Transaction: *Enter a new teaching assistant*
Group: administrator
Source: `admintutor.html`
Form Data: Values for the attributes of table `TA` (section 2.2.1) and a login name and initial password.
Effect:
A new row is created in both the `user` and `TA` tables with `user.user=TA.user`. The attribute values of the TA identified by *u* are set to the given values. The *office hours* should be in the format recognized by transaction *Overview office hours*, page 20.
Queries:

```
INSERT INTO user (group,login,passwd)
```

```
VALUES ('ta','login', PASSWORD('passwd'))";
```
If this query created a new row with user id *uid* run:
```
INSERT INTO TA(user,firstname,lastname,email,officehours)
VALUES ('uid','first name','last name', 'email or NULL', 'office hours');
```

3.4.3 Queries via Email

At some occasions it becomes necessary that the administrators extract data from the CIS database, e.g. for generating the final list of grades with a word-processor. A very convenient strategy is to send the results as text-files via email. It is clear that such transactions must be implemented with care, in order to protect the students' personal data. Three measures can be taken towards that end:

- Queries cannot be entered online, but only selected from a given list $Q_1, \ldots, Q_n$.
- Similarly, the results of the queries can only be send to an address from a fixed list $E_1, \ldots, E_m$.
- Encryption, possibly with the recipient's public key.[5]

Transaction: *Mail results of a query to an administrator*
Group: administrator
Source: `mailquery.html`
Form Data: The index *q* of the desired query and index *m* of of the destination email address.
Effect:
The query Q_q is executed and the result is stored as a text-file with tabulator-delimited fields. That file is send to the email address E_m as an attachment. The body of the email contains the date and time that the query was run to facilitate distinction between different versions of the data base.[6]
Queries:
```
The selected query Q_q is run.
```

3.4.4 Bulletin Board

One of the most frequent tasks of an administrator is to update the electronic bulletin board, which is meant to serve as a central repository for all urgent announcements concerning the course (section 1.2). Entries can be created, deleted, and modified through the CIS's web-interface.

Transaction: *Create entry in bulletin board*
Group: administrator
Source: `adminsp.html`
Form Data: The text *t* of the new announcement and the *date* it is to expire.
Effect:
Creates a new entry on the bulletin board, which remains visible until the specified date of expiration (see also transaction *Read bulletin board* , page 20).
Queries:
```
INSERT INTO cur_info (expire,text) VALUES ('date','text')
```

The modification and deletion transactions are invoked conveniently from a list of the existing announcements, each of which is displayed in a separate HTML-form. Their internal `id` attributes, which serve as artificial primary keys, are contained as `hidden` form fields.

[5]This can be implemented without much effort, if a suitable command-line tool such as PGP [10] is available on the server. In our setting, the mail messages were sure to be transported within a LAN between servers owned by the computer science department.

[6]Currently, the `nmh` package [8] is used for creating and sending the MIME-compliant email message.

Transaction: *Modify entry in the bulletin board*
Group: administrator
Source: `adminsp.html`
Form Data: The *id* of the entry to be updated, the new date where it *expires* and the new *text*.
Effect:
Modifies entry *id* according to the given attributes. Note that the `timestamp` attribute `date` of table `cur_info` is implicitly updated as well (appendix A.1).
Queries:

```
UPDATE cur_info SET text='text' WHERE id='id'
UPDATE cur_info SET expire='expire' WHERE id='id'
```

Removal of announcements should usually not be necessary, since they expire automatically at the stored point in time. Nevertheless, CIS provides the transaction for completeness.

Transaction: *Delete entry from bulletin board*
Group: administrator
Source: `adminsp.html`
Form Data: The *id* of the announcement to be deleted.
Queries:

```
DELETE FROM cur_info WHERE id='id'
```

3.4.5 Handouts

Handouts with restricted access can be used to limit distribution of copyrighted material to active participants of the course (section 3.2.2). The distributed files are kept outside of the main `HTTP` document tree and can be accessed only by the script `handout.php`.[7]

Transaction: *Publish new handout*
Group: administrator
Source: `adminho.html`
Form Data: A file *file* with the document to be published and a short description *descr* of its contents.
Effect:
Moves the uploaded *file* to the directory, where the handout documents are stored and adds an entry to the list of handouts with the one-line description *descr*, the name of the uploaded file, and the current date as the date of publication.
Queries:
The data structures for handouts are kept on disk, see footnote 7.

Transaction: *Delete handout*
Group: administrator
Source: `adminho.html`
Form Data: The serial number n of the handout to be deleted.
Effect:

[7] Currently, handouts are not organized in a database table, but handout number n has associated a proxy file `ho.`n in a subdirectory `Handouts/` of the directory with the CIS-pages. That file contains the name of the handout document, a one-line description of the contents, and the date of publication. It is expected that the handouts will be stored in the database in the next version of CIS.

Removes the description of handout n from the list and deletes the references file from the handout directory.

Queries:

The data structures for handouts are kept on disc, see footnote 7.

In section 3.2.2 we had established two criteria for downloading handouts, one of which must be fulfilled to enable access. The first one is based on recent submissions, the second one checks a flag in the `user` table. The administrators can update that flag with the following transaction.

Transaction: *Modify flag for access to handouts*

Group: administrator

Source: `aha_access_handouts.html`

Form Data: The user id *user* and the new value *f* (=`yes` or `no`) for the flag.

Queries:

```
UPDATE user SET gethandouts='f' WHERE user='user'
```

The input of *user* and *f* in above transaction is done by selection from an overview over all student users. Each student is shown with login, first- and lastname and least recent submission.

Transaction: *Summarize users for handout administration*

Group: administrator

Source: `aha_access_handouts.html`

Queries:

```
SELECT login, concat(s.lastname,', ',s.firstname) AS name,
       s.user, gethandouts, max(UNIX_TIMESTAMP(d_submit)) AS lastSubmission,
FROM student s LEFT OUTER JOIN submission a ON s.user=a.student
  INNER JOIN user ON s.user=user.user
GROUP BY s.user
ORDER BY name
```

3.4.6 Computing the final scores on homework

The final scores on homework assignments can be computed within CIS with a simple SQL-query: We just have to gather the grades for a student's submissions (due to the `UNIQUE` on table `submission`, see appendix A.1, we are sure to account only one submission per problem).

```
SELECT s.user,s.lastname,s.firstname,s.studentid,
       sum(g.score) as score
FROM student AS s
  INNER JOIN submission AS u ON s.user=u.student
  INNER JOIN grades AS g ON u.submission=g.submission
GROUP BY s.user,s.lastname,s.firstname,s.studentid
ORDER BY s.lastname,s.firstname;
```

But also more complicated computations can be arranged for. For instance, in summer term 2001 the students were allowed to name a team-mate for each worksheet, who would also receive their grades (section 2.2.5). The policy implemented in the following query is to transfer a grade only if the receiving student has not handed in a solution herself (line marked `(*)`).

```
CREATE TABLE results
SELECT s.user,s.firstname,s.lastname,s.studentid,
      sum(g1.score) as 'own',
      sum(g2.score) as 'foreign',
      sum(g1.score)+sum(g2.score) as Score
```

```
FROM student AS s
     CROSS JOIN problem AS p
     LEFT OUTER JOIN submission AS u1
          on p.blatt=u1.blatt and
             p.problem=u1.problem and
             s.user=u1.student
     LEFT OUTER JOIN grades AS g1 ON u1.submission=g1.submission
-- fetch grade from partner, if it exists
     LEFT OUTER JOIN partner AS p
          ON p.blatt=p.blatt AND s.user=p.q AND
             isnull(g1.score)                            -- (*)
     LEFT OUTER JOIN submission AS u2
          ON u2.problem=p.problem and
             u2.blatt=p.blatt and
             u2.student=p.p
     LEFT OUTER JOIN grades AS g2 ON u2.submission=g2.submission
GROUP BY s.user,s.firstname,s.lastname,s.studentid;
```

Chapter 4

Summary and Conclusions

4.1 Summary

We have presented CIS, a web-based Course Information System to facilitate the purely administrative tasks arising with courses given at universities and colleges. Our central objective has been to support traditional forms of organization by means of computer technology, rather than enforcing new and artificial policies. Most notably, we designed CIS so that it can be used conveniently by first-year students without prior exposition to computers.

We have based our design decisions on the prevailing needs of three groups of users: The students, the teaching assistants and the administrators. Their specific responsibilities and regular activities have been analyzed in chapter 1. The implementation is structured as a set of largely independent transactions, which we have described in chapter 3. In factoring single tasks, we have aimed at modeling real world procedures as closely as possible to support intuitive system behaviour, specifically towards the student users.

CIS's architecture is designed in a modular way around a central database, whose schema we have described and motivated in detail in section 2.2. Every transaction modifies a very limited part of the stored data, such that its effects can be understood intuitively. Yet, by grouping related transactions on a single web-page, CIS nevertheless features a compact and consistent user-interface, which appears to the user as a pool of dialogues. The single dialogues are selected by means of a top-level menu-structure (section 2.4.2).

CIS has been build as an open system, that can be extended – and restricted – in almost arbitrary ways. Hence, it can be perceived as a framework for implementing course information systems. It provides the infrastructure for user-authentication, menus, handouts and worksheets. CIS builds on relational database technology, and the schema should be extensible in a gradual manner. The tools we have chosen, i.e. the database engine MySQL [7] and the Apache web-server [2] (with a PHP-interpreter module), are free and can be installed on a standard PC with modest difficulty.

4.2 Experiences

We have been using CIS for three years, both for large undergraduate and smaller advanced courses. It has proven to be a reliable and managable tool to support the routine tasks in organizing courses. Our experiences are encouraging: Even the first-year students in the "Computer Science I/II" courses were able to handle CIS without much difficulty. The availability of electronic submissions have been a great help to the teaching assistants for grading. The central storage of all grades has made the relative success of the students on single problem assignments more transparent to the lecturer than any weekly meeting of teaching assistants could possibly have done. As a final convenience, the evaluation of grades at the end of the course was a matter of a few database queries (section 3.4.6) and left room in the meeting with teaching assistants to discuss the more problematic cases individually.

4.3 Future Work

During all course we have observed two main time periods that required increased efforts from the lecturers and administrators using CIS, the beginning and the end of the course, respectively.

The administrative work at the beginning of the course is caused by the manual installation procedure (see section 2.5.2), assigning students to TAs, and handling change requests from students that are not satisfied with their assignment. The assignment problem has been solved recently by applying an appropriate graph algorithm (see section 2.5.3), and the other two tasks can be automated with reasonable effort.

At the end of the course, students are usually categorized into three main categories: passed, failed, and border cases. According to these categories, the final decision can be coordinated with the TAs based on the experiences from group meetings. This categorization can be achieved by SQL queries. However, they usually differ among courses. Therefore, a parameterized back-end for generating this categorization would be helpful.

A more technical observation has ignited a major redesign: Our approach of identifying a transaction with a whole web-page in the system has turned out to be too coarse-grained. Therefore, we started to produce so called *document fragments*, which we collect in a library. A document fragment captures a transaction of reasonable size, and web-pages in CIS are now composed from fragments. This also allows the CIS maintainer to adapt the default web-pages more easily. Furthermore, the static and dynamic contents are maintained in different files, such that the functionality of single fragments can be upgraded without touching the including web pages.

Also, internationalization of the user interface would be necessary in order to distribute CIS to universities outside of Germany.

Finally, one feature in the database schema was sorely missed. We have to add support for subproblems on worksheets. Currently, the grading scheme is designed for toplevel problems only.

Appendix A

SQL Data Definition and ER Diagram

⟨Listing A.1: Database creation⟩ ≡

```
USE mysql;
DROP DATABASE info2_SS03;
CREATE DATABASE info2_SS03;
USE info2_SS03;
```

Code extracted from file `cicinstall/schema.sql`, lines 2 to 5.

A.1 Tables

⟨Listing A.2: Table user⟩ ≡

```
CREATE TABLE benutzer (
   benutzer    SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
   gruppe      ENUM('Student','Tutor','Admin'),
   login       CHAR(15),
   passwd      CHAR(30), -- CRYPT() hash
   gethandouts ENUM('nein','ja') default 'nein',
   PRIMARY KEY (benutzer),
   UNIQUE INDEX (login)
);
```

Code extracted from file `cicinstall/schema.sql`, lines 49 to 57.

⟨Listing A.3: Table student⟩ ≡

```
-- Only root should be able to access this table.
CREATE TABLE student (
  benutzer    SMALLINT UNSIGNED NOT NULL,
  vorname     VARCHAR(30) NOT NULL,
  nachname    VARCHAR(30) NOT NULL,
  geb         DATE,
  matrikelnr  DEC(7,0),
  account     CHAR(8), -- WSI-Account
  email       VARCHAR(70),
  forum       ENUM('ja','nein'),
  fach        ENUM('Informatik','Bioinformatik', 'Nebenfach'),
  uegru       SMALLINT UNSIGNED,
  uegr1       SMALLINT,
  uegr2       SMALLINT,
```

```
    uegr3      SMALLINT,
    uegr4      SMALLINT,
    uegr5      SMALLINT,
    PRIMARY KEY (benutzer)
    -- FOREIGN KEY (benutzer) REFERENCES benutzer
  );
```
Code extracted from file `cicinstall/schema.sql`, lines 72 to 91.

⟨Listing A.4: Table tutor⟩ ≡
```
  CREATE TABLE tutor (
    benutzer   SMALLINT UNSIGNED  NOT NULL,
    vorname    VARCHAR(30) NOT NULL,     -- saves one join
    nachname   VARCHAR(30) NOT NULL,
    email      VARCHAR(70),
    abg_email  VARCHAR(70),              -- deliver zip file if NOT NULL
    sprechstd  VARCHAR(80),
    technical  ENUM('ja','nein'),
    PRIMARY KEY (benutzer)
    -- FOREIGN KEY (benutzer) REFERENCES benutzer
  );
```
Code extracted from file `cicinstall/schema.sql`, lines 97 to 107.

⟨Listing A.5: Table admin⟩ ≡
```
  CREATE TABLE admin (
    benutzer   SMALLINT UNSIGNED  NOT NULL,
    PRIMARY KEY (benutzer)
    -- FOREIGN KEY (benutzer) REFERENCES benutzer
  );
```
Code extracted from file `cicinstall/schema.sql`, lines 111 to 115.

⟨Listing A.6: Table sheet⟩ ≡
```
  CREATE TABLE blatt (
     blatt     SMALLINT UNSIGNED NOT NULL,
     d_ausgabe DATETIME,
     d_abgabe  DATETIME,
     d_loesung DATE,
     url_ps    VARCHAR(80),
     url_pdf   VARCHAR(80),
     url_mlps  VARCHAR(80),
     url_mlpdf VARCHAR(80),
     published ENUM('ja','nein'),
     PRIMARY KEY (blatt)
  );
```
Code extracted from file `cicinstall/schema.sql`, lines 120 to 131.

⟨Listing A.7: Table problem⟩ ≡
```
  CREATE TABLE aufgabe (
    aufgabe  SMALLINT UNSIGNED NOT NULL,
    blatt    SMALLINT UNSIGNED NOT NULL,
    punkte   DECIMAL(3,1),
    d_abgabe DATETIME,
    text     LONGBLOB,
    loesung  LONGBLOB,
```

```
  test     LONGBLOB,
  prog     TINYINT DEFAULT 1,
  kmeth    TINYINT,
  kprog    VARCHAR(50),
  PRIMARY KEY (blatt, aufgabe)
  -- FOREIGN KEY (blatt) REFERENCES blatt(nr)
);
```

Code extracted from file `cicinstall/schema.sql`, lines 135 to 148.

⟨Listing A.8: Table group⟩ ≡

```
CREATE TABLE uegru (
 uegru  SMALLINT UNSIGNED NOT NULL,
 tag    SMALLINT,
 stunde TIME,
 raum   CHAR(10),
 tutor  SMALLINT UNSIGNED,
 PRIMARY KEY (uegru)
);
```

Code extracted from file `cicinstall/schema.sql`, lines 280 to 287, referenced in listing Listing.A.15.

⟨Listing A.9: Table day⟩ ≡

```
CREATE TABLE tag (
  tag  SMALLINT NOT NULL,
  kurz CHAR(5),
  PRIMARY KEY (tag)
);
```

Code extracted from file `cicinstall/schema.sql`, lines 292 to 296, referenced in listing Listing.A.15.

⟨Listing A.10: Table submission⟩ ≡

```
CREATE TABLE abgabe (
  abgabe    SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  student   SMALLINT UNSIGNED NOT NULL,   -- solutions
  aufgabe   SMALLINT UNSIGNED NOT NULL,
  blatt     SMALLINT UNSIGNED NOT NULL,
  suffix    VARCHAR(6),                    -- remember file extension
  datei     VARCHAR(80) NOT NULL,         -- reference to file
  d_eingang TIMESTAMP,
  PRIMARY KEY (abgabe),
  UNIQUE (student, aufgabe, blatt)      -- comment for multiple submissions
  -- FOREIGN KEY (student) REFERENCES student,
  -- FOREIGN KEY (blatt, aufgabe) REFERENCES aufgabe (blatt, nr),
  -- FOREIGN KEY (student,aufgabe,blatt)
  --   REFERENCES korrektur (student,aufgabe,blatt)
  --   ON DELETE CASCADE
);
```

Code extracted from file `cicinstall/schema.sql`, lines 180 to 195.

⟨Listing A.11: Table for previous submissions⟩ ≡

```
CREATE TABLE abgabe_alt (
  abgabe    SMALLINT UNSIGNED NOT NULL,
  student   SMALLINT UNSIGNED NOT NULL,   -- solutions
  aufgabe   SMALLINT UNSIGNED NOT NULL,
```

```
    blatt     SMALLINT UNSIGNED NOT NULL,
    datei     VARCHAR(80) NOT NULL,          -- reference to file
    d_eingang TIMESTAMP,
    INDEX (student, aufgabe, blatt)
  );
```
Code extracted from file cicinstall/schema.sql, lines 204 to 212.

⟨Listing A.12: Table grades⟩ ≡
```
  CREATE TABLE korrektur (
    abgabe      SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,  -- link to submission
    d_korrektur TIMESTAMP,
    punkte      DECIMAL(3,1),
    tutor       SMALLINT UNSIGNED NOT NULL, -- for further requests
    kommentar   TEXT,
    PRIMARY KEY (abgabe)
  );
```
Code extracted from file cicinstall/schema.sql, lines 220 to 227.

⟨Listing A.13: Table partner⟩ ≡
```
  CREATE TABLE partner (
    p     SMALLINT UNSIGNED NOT NULL,        -- donor
    blatt SMALLINT UNSIGNED NOT NULL,
    q     SMALLINT UNSIGNED NOT NULL,        -- receiver
    PRIMARY KEY (blatt, p)
  );
```
Code extracted from file cicinstall/schema.sql, lines 231 to 236.

⟨Listing A.14: Table corrector⟩ ≡
```
  CREATE TABLE aufgabentutor (
    blatt   SMALLINT UNSIGNED NOT NULL,
    aufgabe SMALLINT UNSIGNED NOT NULL,
    tutor   SMALLINT UNSIGNED NOT NULL,
    PRIMARY KEY (blatt,aufgabe,tutor)
  );
```
Code extracted from file cicinstall/schema.sql, lines 240 to 245.

⟨Listing A.15: Table correction queue⟩ ≡
```
  CREATE TABLE korrekturschlange (
    abgabe     SMALLINT UNSIGNED NOT NULL,  -- determines order on form
    tutor      SMALLINT UNSIGNED,
    d_entnahme DATETIME,                 -- NULL if not yet retrieved
    PRIMARY KEY (abgabe)
    --  FOREIGN KEY (abgabe) REFERENCES abgabe,
    --  FOREIGN KEY (tutor) REFERENCES tutor (id)
  );
```
⟨Expanded in listing Listing.A.15, page 44⟩

Code extracted from file cicinstall/schema.sql, lines 257 to 245.

⟨Listing A.16: Table news⟩ ≡
```
  CREATE TABLE aktuell (
    id     SMALLINT UNSIGNED AUTO_INCREMENT,
    date   TIMESTAMP,
    expire DATETIME,
```

```
  text   TEXT,
  PRIMARY KEY (id)
);
```

Code extracted from file cicinstall/schema.sql, lines 269 to 275, referenced in listing Listing.A.15.

⟨Listing A.17: Table pin⟩ ≡

```
CREATE TABLE PIN (
  pin      CHAR(20) NOT NULL,
  used     TINYINT DEFAULT 0,
  benutzer SMALLINT UNSIGNED,
  PRIMARY KEY (pin)
);
```

Code extracted from file cicinstall/schema.sql, lines 61 to 66.

A.2 Grants

⟨Listing A.18: Grants for students⟩ ≡

```
GRANT INSERT, SELECT, DELETE ON abgabe         TO student@localhost;
GRANT INSERT, SELECT, DELETE ON abgabe_alt     TO student@localhost;
GRANT SELECT                 ON korrektur      TO student@localhost;
GRANT SELECT                 ON personal_input TO student@localhost;
GRANT SELECT                 ON aufgabentutor  TO student@localhost;
GRANT SELECT                 ON tutor          TO student@localhost;
GRANT SELECT                 ON benutzer       TO student@localhost;
GRANT SELECT                 ON blatt          TO student@localhost;
GRANT SELECT                 ON aufgabe        TO student@localhost;
GRANT SELECT, INSERT, DELETE ON partner        TO student@localhost;
GRANT SELECT                 ON aktuell        TO student@localhost;
```

Code extracted from file cicinstall/schema.sql, lines 371 to 381, referenced in listing Listing.A.15.

⟨Listing A.19: Grants for teachings assistants⟩ ≡

```
GRANT SELECT                 ON abgabe            TO tutor@localhost;
GRANT UPDATE, INSERT, SELECT ON korrektur         TO tutor@localhost;
GRANT UPDATE, DELETE         ON korrekturschlange TO tutor@localhost;
GRANT SELECT                 ON tag               TO tutor@localhost;
GRANT SELECT                 ON login             TO tutor@localhost;
GRANT SELECT                 ON aktuell           TO tutor@localhost;
-- needed for corrections
GRANT SELECT                 ON personal_output   TO tutor@localhost;
GRANT SELECT                 ON aufgabe           TO tutor@localhost;
GRANT SELECT                 ON blatt             TO tutor@localhost;
GRANT SELECT                 ON aufgabe           TO tutor@localhost;
GRANT INSERT, UPDATE, DELETE ON korrektur         TO tutor@localhost;
GRANT SELECT                 ON blatt             TO tutor@localhost;
GRANT SELECT                 ON aufgabe           TO tutor@localhost;
GRANT INSERT                 ON abgabe            TO tutor@localhost;
GRANT INSERT, UPDATE, DELETE ON korrektur         TO tutor@localhost;
GRANT SELECT                 ON student           TO tutor@localhost;
GRANT SELECT                 ON uegru             TO tutor@localhost;
GRANT SELECT                 ON benutzer          TO tutor@localhost;
GRANT SELECT, UPDATE         ON tutor             TO tutor@localhost;
```

Code extracted from file cicinstall/schema.sql, lines 347 to 366, referenced in listing Listing.A.15.

⟨Listing A.20: Grants for admins⟩ ≡

```
GRANT INSERT, SELECT, UPDATE, DELETE ON *       TO admin@localhost;
GRANT INSERT, SELECT, UPDATE         ON aktuell TO admin@localhost;
```

Code extracted from file `cicinstall/schema.sql`, lines 334 to 335, referenced in listing Listing.A.15.

⟨Listing A.21: Grants for system logon user⟩ ≡

```
GRANT SELECT, INSERT ON benutzer TO logon@localhost;
GRANT INSERT         ON student  TO logon@localhost;
GRANT SELECT, UPDATE ON PIN      TO logon@localhost;
```

Code extracted from file `cicinstall/schema.sql`, lines 340 to 342, referenced in listing Listing.A.15.

A.3 ER Diagram

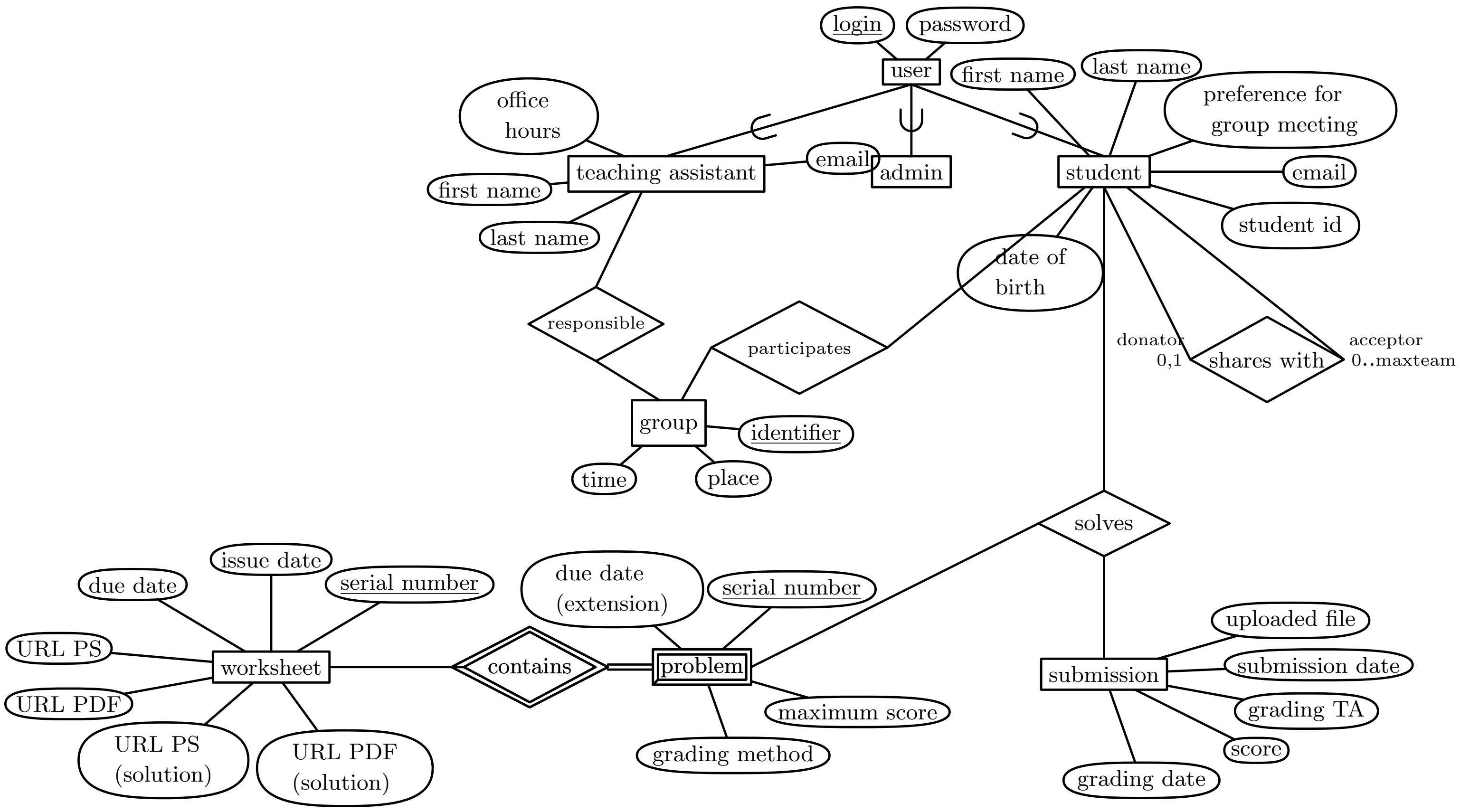


Figure A.1: The ER diagram of the data base

Appendix B

Website Menu Structure

B.1 Toplevel Menu Structure

- Homepage
- Organization
- Homework
- Material
- Software
- Administration
  Only available to users belonging to the administrator's group.
- Teaching Assistants
  Only available to users belonging to the teaching assistant's group.

B.2 Homepage Menu

- Homepage
  The current contents of the bulletin board is displayed on this startup page of the system.

B.3 Organization Menu

- Organization
  - Enroll for the Course
    See transaction *Enroll for the course*, page 21.
  - Change Password
    Students and teaching assistants can change their passwords (transaction *Change CIS-password*, page 22).
  - Contact Information
    Displays email addresses and office hours of teaching assistants and lecturers (transaction *Query TA Contact Information*, page 20).
  - TA–Groups
    The assignment of students to groups and teaching assistants (transaction *Display groups with meeting time and responsible TA*, page 21).

B.4 Homework Menu

- Homework
  - Worksheets
    Information on download of worksheets, see section 3.2.3.
  - Submit Solution
    Upload solution files for problems, transaction *Submit solutions*, page 24.
  - Check Personal Grades
    Overview for students, transaction *Check grades*, page 26.

B.5 Material Menu

- Material
  - Material
    Links to material augmenting the course.
  - Handouts
    Handouts made available for students by the lecturer, usually code or documents.
  - Handouts for TAs
    TAs have a dedicated handout section not visible to students which can contain early solutions ideas and so on.

B.6 Administration Menu

- Administration
  This sub-menu is visible only to authenticated administrators.
  - Worksheets
    The administrators can add and modify the available worksheets, see section 3.4.1.
  - Teaching assistants
    Enter and update data of teaching assistants, section 3.4.2.
  - Enroll for the Course
    The administrator can add students to the database even after regular enrollment time has expired.
  - Announcements
    Manage the contents of the bulletin board displayed at system startup (see section 3.4.4).
  - Email-queries
    See transaction *Mail results of a query to an administrator*, page 35.
  - Passwords
    The administrator may change any user's password, transaction *Set password for students and teaching assistants*, page 31.
  - Handouts
    Upload Handouts with restricted access, see section 3.4.5.

B.7 Teaching Assistants Menu

- Teaching Assistants
  This part of the menu is visible only to authenticated teaching assistants.
  - Change Personal Data
    The TAs are allowed to make corrections to their personal data like email address or first name and surname.
  - Grade Worksheet
    An overview of all students for the TA is given, and she can select one worksheet for grading.
  - Download Solutions
    The TAs can download the archive of all currently handed in solutions on a per worksheet basis.
  - Passwords
    TAs may change passwords of their students.

B.8 Software Menu

- Software
  This page contains links to local and external copies of software relevant for the course. Furthermore, installation instructions are provided for the supported platforms.

B.9 Configuring the Menu

The menu structure is encoded using PHP arrays in file `info_lib.php`. Each array entry consists of the text that will be displayed in the menu of the web interface (see section 2.4) as well as the file to which this item should be linked.

The character in front of the item text encodes the availability of the item itself. The availability is based on the current session's authentication state. Table B.1 shows the employed encoding.

Character	Availability
`-`	always
`n`	not authenticated
`s`	students only
`t`	TAs only
`a`	administrators only

Table B.1: Character encoding for availability of menu items

The following code shows the current toplevel menu structure. Notice that submenus are generated the same way.

⟨Listing B.1: Data Structure for Toplevel Menu⟩ ≡

```
global $std_menu;
$std_menu =
array( "-Startseite" => "index.html",
       "-Organisation"  => "orga.html",
       "-&Uuml;bungen" => "uebungen.html",
       "-Material" => "material.html",
       "-Software" => "software.html",
       "aAdministration" => "admin.html",
       "tTutorseite" => "tutor.html"
);
```

Code extracted from file `cicsource/info_lib.php`, lines 41 to 50.

Appendix C

Questions for Evaluation

Einschätzung der Vorlesung (Antwort: zutreffend 1–5, nicht anwendbar, weiss nicht)

1. Die Vorlesung ist inhaltlich interessant.
2. Ich besuche die Vorlesung gerne.
3. Der Dozent selbst hält die Vorlesung gerne.
4. Der Dozent kennt sich mit dem Vorlesungsthema aus.
5. Der Dozent drückt sich klar und verständlich aus.
6. Die Vorlesung ist gut strukturiert.
7. Der Dozent vermittelt die Schlüsselkonzepte, Begriffe und Methoden.
8. Der Dozent geht während oder nach der Vorlesung auf Fragen ein.
9. Fragen werden zufriedenstellend beantwortet.
10. Der Dozent ist regelmäßig für Fragen verfügbar. (Sprechstunde, Übung,etc.)
11. Die Übungen prüfen mein Verständnis und Wissen vernünftig.
12. Die Übungen können im gegebenen Zeitrahmen von vorbereiteten Studenten bewältigt werden.
13. In der Vorlesung lerne ich etwas Neues dazu.
14. Ich würde diese Vorlesung anderen Studenten empfehlen.
15. Ich würden den Dozenten anderen Studenten empfehlen.
16. Die Vorlesung regt mich dazu an, mich auch weiter mit dem Thema zu beschäftigen.

Textfragen für Kommentare und Hinweise:

17. Was war Ihrer Meinung nach das Beste an der Veranstaltung?
18. Was würden Sie ändern?

Persönliche Situation:

19. Wieviele Vorlesungen haben sie besucht?
20. Wieviele Übungen haben sie besucht?
21. Welchen Anteil des Vorlesungsstoffes haben Sie grundsätzlich verstanden?

22. Welchen Anteil des Vorlesungsstoffes konnten Sie sofort aktiv anwenden?
23. Haben Sie in der Vorlesung Fragen gestellt? (ja/nein)
24. Haben Sie in der Übungen oder außerhalb der Veranstaltungen Fragen gestellt? (ja/nein)
25. Warum haben Sie die Vorlesung besucht? Erforderlich Thema Dozent Sonstiges
26. Haben Sie eine Vorlesung zu diesem Thema schon früher gehöhrt? (ja/nein)
27. War für Sie das Buch/Skript zur Vorlesung hilfreich?
28. Ausführungen (Text zu vorhergehenden Fragen, falls gewünscht)

Acknowledgement

We want thank the Parallel Computing Group at the University of Tübingen for using CIS in their own courses. Most notably, they developed the student assignment tool. It is an invaluable help to relax some of the pressure at the beginning of a course. Markus Eiglsperger and Nina Lehmann have suffered most from dedicating course management to CIS. Markus Eiglsperger is also responsible for the student assignment tool.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] Apache Software Foundation. http://www.apache.org.

[3] Netscape Communications. SSL 3.0 specification. http://www.netscape.com/eng/ssl3.

[4] Ramez Elmasri and Navathe Shamkant B. *Fundamentals of database systems* . Benjamin/Cummings, Redwood City, CA u.a., 2nd edition, 1994.

[5] International Organization for Standardization. *ISO 9075:1992(E) Database Language SQL*, 1992.

[6] Network Working Group. Hypertext transfer protocol – http/1.1 (rfc 2616). ftp://ftp.isi.edu/in-notes/rfc2616.txt, June 1999.

[7] The MySQL Database. http://www.mysql.com.

[8] nmh – Message Handling System. http://www.nongnu.org/nmh.

[9] The OpenSSL Project. http://www.openssl.org.

[10] PGP Corporation Homepage. http://www.pgp.com.

[11] PHP: Hypertext Preprocessor. http://www.php.net.