



ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

**Fakultät für
Ingenieurwissenschaften
und Informatik**
Institut für Datenbanken und Infor-
mationssysteme

Document Maintenance With Multiple Access Strategy

Bachelorarbeit an der Universität Ulm

Vorgelegt von:

Leo Hnatek

leo.hnatek@uni-ulm.de

Gutachter:

Prof. Dr. Manfred Reichert

Betreuer:

Rüdiger Pryss

2013

„Document Maintenance With Multiple Access Strategy“
Fassung vom 20. Mai 2013

Bei der Erstellung dieser Bachelorarbeit wurde zur Erstellung der Grafiken die Software Balsamiq Mockups verwendet. Die Lizenz dazu wurde vom Institut für Datenbanken und Informationssysteme bereitgestellt.

Satz: PDF- \LaTeX 2_ε
Druck: Kommunikations- und Informationszentrum kiz | Medien

© 2013 Leo Hnatek

Dieses Werk ist unter der Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany License lizenziert: <http://creativecommons.org/licenses/by-nc-sa/3.0/de/>

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Structure	2
2	Concept	3
2.1	Exemplary Usage	3
2.2	Definitions	4
2.3	Existing Document Distribution Systems	5
2.4	Parties Involved	5
3	Requirements	7
3.1	Data-Oriented	7
3.2	User-Oriented	8
3.3	Infrastructure-Oriented	8
3.4	Overview	9
4	Design	11
4.1	Requirement Analysis	11
4.2	Mock Ups	14
4.2.1	End User Interface	15
4.2.2	Handler Interface	15
4.2.3	Administrator Interface	17
5	Architecture	19
5.1	Technologies	19
5.1.1	JSON	19
5.1.2	File Structure	21
5.1.3	Meta Data Creation	23
5.1.4	Email	24

Contents

5.1.5	Lightweight Director Access Protocol	24
5.1.6	Common Unix Printing System	25
5.1.7	Google Web Toolkit	26
5.1.8	SmartGWT	26
5.2	Component Diagram	26
5.3	Web Interface	28
5.3.1	Walkthrough	28
5.3.2	Client-Server Model	33
5.3.3	Client Structure	33
5.3.4	Server Structure	35
5.3.5	Server Components	36
5.3.6	<i>StartUp And Timed</i>	39
5.3.7	Configuration	40
5.3.8	Target System	41
5.4	Console	42
5.4.1	Walkthrough	42
5.4.2	Function	44
5.5	File Browser	44
5.6	Backup	44
6	Conclusion	47
6.1	Requirement Alignment	47
6.2	Outlook	49
6.3	Discussion	50
	Bibliography	51

1 Introduction

This chapter will introduce the thesis, illustrate its motivation and present the outline of the following chapters.

1.1 Motivation

This thesis should yield a system that is capable of providing access to documents. Such documents and the file access thereto are of central importance. The access to these documents should not be limited by a single interface or access manner.

Databases and database management systems (DBMS) using the Structured Query Language (SQL)[11] and following the ACID paradigm¹ are commonly used to manage data. Such databases are robust and performant but minimize flexibility regarding the data structure, as it has to be determined at the creation of a system and should not be changed over time. Additionally, the data can only be accessed by the database management system (DBMS). Hence, the data cannot be accessed if the database management system is not running, because in major systems like MySQL, PostgreSQL or DB2 the data is stored in a binary manner to increase performance.

As this thesis aims to provide a system wherein access is independent of single components, especially of the database server such traditional SQL servers cannot be used.

Moreover, single data attributes can not be inserted, in performance-oriented SQL servers, even if the DBMS was not running as it would require knowledge over the internal database structure, as well as complex insertion and modification methods, whereas the focus should lie on the documents, their consistency, rather than on the meta information.

In consideration of these goals, the implementation of a system with multiple access strategy and easy modification will be discussed in the following chapters of this thesis.

¹ACID is short for: Atomicity, Consistency, Isolation, Durability. These are properties a relational database generally possesses.

1.2 Structure

Figure 1.2 shows a summary of the thesis, its chapters as well as its sections, and subsections. The different chapters are grouped by color.



Figure 1.1: A structured overview of the thesis.

2 Concept

This chapter will discuss the practical purpose of document maintenance systems, and the context within which they are used, as well as introduce existing implementations for such a system.

2.1 Exemplary Usage

The aim of this thesis is to create a document management system that provides its functionality via multiple access. As a practical implementation, a system was developed that provides the possibility for students to gain access to past examinations and other forms of documents.

For most students taking examinations is one of the most challenging situations in the course of their studies while oral examinations in particular are less predictable compared to written ones, therefore students often ask fellow students who already passed the upcoming examination for advice on how to handle the stress and what the lecturer might ask. This procedure is very inefficient and unfair because only students with connections can get help. So to help students manage the stressful situation of an examination and to help them prepare properly, many student associations gather intelligence on how to prepare for oral as well as written examinations. In order to distribute this knowledge, students may write a detailed report of an oral exam and submit it to the student association. Many lecturers agree with these principle of transparency and also provide past written examinations to the associations. To have a certain amount of control over the distribution of these documents, both student associations and lecturers came th the agreement that it would be best if the documents were provided in printed form only.

2.2 Definitions

In this section roles for users interacting with the system will be introduced, and the documents these users utilize with will be defined.

Administrator

An *administrator* is a person who manages and maintains the system as described in chapter 5, thus maintaining and organizing documents and importing them into the system. They have more rights than the end user. A student member of a student association is planned to hold the role of the administrator.

End User

An *end user* is a student who accesses the system by searching for *documents*. They might perform the searching directly with the system, or require the help of a handler, as described below. In any case, the final goal of an end user is that the sought document is provided to them.

Handler

A *handler* is a student assigned to hold opening hours and explicitly interact with the system to search and print *documents*.

Provider

Any person who provides a *document* to an *administrator* is defined as a *provider*.

Document

The minutes of examinations, i.e. the records of past examinations which people created for the purpose of better preparation for future examinations will be, in short, called *documents*. In general, *documents* can be any kind of document, as in the commonly used sense, that is provided to the administrator and consensually imported by them.

Collection

The aggregated documents as requested by an end user, either put together by the end users themselves or by handlers, is defined as a *collection*.

2.3 Existing Document Distribution Systems

Some associations keep the documents in written form, i.e. in filing cabinets. When students request to get a copy of a document, they have to find a copy machine and make a paper copy. This procedure is very inefficient. Especially if many students concurrently want to access the same document at once, as they have to wait until it is available.

Other associations do have their documents, such as examinations, digitalized. To access those documents, files are selected separately, opened with a file browser and printed one by one. Alternatively one can use a shell script to do the work in an automatized way. In order to import the documents, an administrator has to scan them, copy them and move them to the right destination in a folder for which the person also has write permissions. Few associations have a front end for users to search for the documents. The distribution of documents requires a long time; because importation of new documents usually does not happen, neither in an atomized manner nor digitally, the amount of available documents is often low and increases only slowly. Above all, different associations have different ways of organizing documents, meaning that a student has to know them all to gain access to the wanted documents. The adaptation to thesis processes is unnecessary for providers and end users. Additionally, an non-unified process of importation requires lecturers to know about them, too. This can decrease the amount of documents which are willingly provided. Hence, having one single system for documents across all student associations and faculties would provide transparency. Furthermore, having a centralized system would make maintenance easier and less redundant, as currently two different associations may accidentally have the same document, not knowing about the other's existence.

2.4 Parties Involved

As Figure 2.1 simply illustrates, documents are expected to be supplied in some way to the system by either lecturers themselves or examinees. The system then should be able to distribute documents to future examinees in a printed form. The black-box system designed here will be introduced in chapter 4 and further discussed in chapter 5 of this thesis.

2 Concept

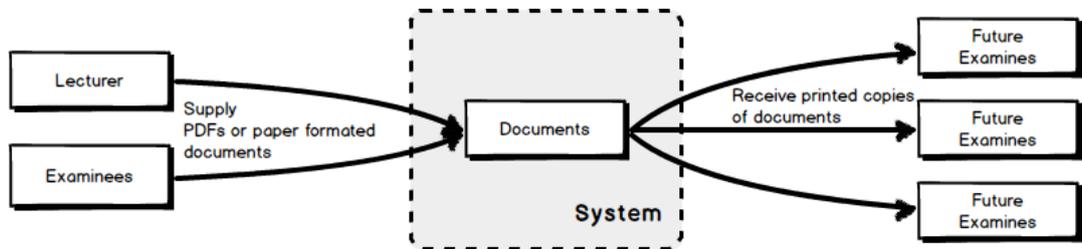


Figure 2.1: Parties involved with the system.

3 Requirements

This chapter will illustrate which requirements will have to be met by the implementation of the document management system. These requirements are based on the present state of the student associations, normal students and lecturers at the Ulm University, and their agreements. This status quo may change, wherefore the implemented system should be as adaptive as possible.

3.1 Data-Oriented

Data Storage The documents that are provided should be stored in a format that is as platform-independent as possible. The single files will not have to be changed over time, so a format can be chosen to represent the documents, in a manner that can be displayed on any device, as accurately as possible in comparison to the printed ones. Furthermore, it should save disk space and be flexible enough to handle converted pictures, as well as text and vector graphics.

Data Structure In order to gain flexibility, the files should be stored in a manner whereby they can be transferred to a new implementation of the system without much effort towards changing them. The structure within which the documents are stored should be clear and understandable to both handlers and administrators of the system. Furthermore, this structure must not be dependent on a specific implementation, such as a proprietary compressed file format, as different access methods will be provided to work in a multiple access manner. Optionally, this structure should be easy to backup and reproduce.

Document Style Ideally, the documents being provided to users should have a unified style so that end user can focus on the content of the documents rather than being troubled by differing layouts or fonts of documents.

3.2 User-Oriented

User Access By having many users from different associations, the system will have to provide the option for multiple users to access it. Additionally, the administrator might change over time, even frequently, as the administrator's role is held voluntarily by students, whose availability is likely to be limited to only a few terms, as well as limited by a time constraint during these terms due to their own studies. Therefore, the user management should be as manageable as possible and the user logins should preferably correlate to logins used by other university systems, so that no ghost accounts exist and redundancy can be kept as low as possible.

User Interface It is key to this thesis that not only one interface can be used to interact with the system and achieve all tasks, but multiple ways can be used to interact with it, making it user-friendly, efficient and failsafe. This way, advanced administrators can use methods to import documents efficiently i.e. with batch processes, whereas inexperienced administrators are provided with an interface that is easy to use and understandable. Fail safety should be provided by the multiplicity of access, though possibly constraining usability or efficiency if the typically used interface fails.

3.3 Infrastructure-Oriented

Printing The system should provide the functionality of printing documents as the documents will be handed to students in a printed form. It should be possible to print a collection instead of printing file by file. To save paper, the system should optionally be capable of post-processing the collections before printing, in order to print both sides of each single sheet of paper.

Maintenance The administrators should be able to import documents and also to manage the documents that are archived within the system. These procedures should be as efficient and intuitive as possible. The system should also be capable of doing this work not only from one single workstation but instead remotely, and optionally, concurrently.

3.4 Overview

In this section the requirements to the system, that will be implemented over the course of this thesis, are summarized and classified.

Classification	Description Of The Requirement	Type
Data Storage	Make use of a platform-independent file format.	functional
	Save disk space by keeping the files small.	functional
	Choose a file format that is flexible enough to handle vector graphics, text, as well as embedded graphics.	functional
Data Structure	Data is portable, i.e. it can be transferred to another system.	functional
	Data is stored in a structured manner, thus understandable by handlers and administrators.	functional
	The data is stored in a manner, independent of proprietary file formats.	functional
Document Style	The documents could have a unified document layout.	optional
User Access	The system is accessible in more than one way.	functional
	Existing accounts of existing systems are used, thus preventing ghost accounts.	functional
	Handling accounts is easy.	non-functional
User Interface	The system has more than one interface for interaction.	functional
	At least one interface is user friendly.	functional
	At least one interface is efficient.	functional
	The system is in some way failsafe.	non-functional
Printing	Documents can be printed.	functional
	Options are provided to save paper.	optional
Maintenance	Documents can be imported.	functional
	Documents can be managed.	functional
	Interaction with the system can happen concurrently.	optional
	Interaction with the system can happen remotely.	functional

4 Design

With the requirements discussed in the previous chapter 2 and a given infrastructure, choices have to be made in regards to the demonstrative implementation; choices, which should satisfy as many needs as possible. Potential options will be discussed and illustrated in this chapter.

4.1 Requirement Analysis

In this section the requirements discussed in chapter 3 will be analyzed and possible implementation strategies will be provided.

Data Storage

As described in the Data Storage section 3.1, once the documents are created, they are not expected to change over time. Therefore, PostScript and the Portable Document Format (PDF) are good choices as both store the files in a way that the printed collection will look exactly like the digital representation, and the files will be consistent over different operating systems or software used to display them. Finally, the PDF has the advantage that modern browsers can display the format natively and that text editing tools as well as modern scanning hardware often support PDF output.

Data Structure

The implementation of the demonstrative system will run on a server belonging to the official infrastructure of Ulm University. The maximum number of end users will be less than the total number of students, a number which is below 9.589, as this was the total number of students in the winter semester of 2012/2013. In fact, it is very unlikely that all the students access the system, especially not at the same time, so the number of hits can be expected to remain below 10.000 over the course of a term. It will more likely even be less than 500 hits at the same time, as less than 100 documents were handed out by the Computer Science student association during

4 Design

the winter semester of 2012/2013. The amount of available documents for Computer Science students are currently below 1000. Accordingly, there is no obvious necessity for a database in terms of performance. Without a database the documents can be stored as separate files in a file system. This way, they can be accessed by users without a running database management system or special interface tools such as phpMyAdmin. Document meta data should be provided, so an *Extensible Markup Language (XML)* or *JavaScript Object Notation (JSON)* file can be used to store the necessary data in a structured way; alternatively, simple lists can be used or the filenames themselves could hold the needed values.

User Access

The system's expected end user group consists of students, though some students will also have the role of administrators who add and manage the documents. In fact any role will be occupied by a student, although student's field of study may vary. Thus the user login must be universal and applicable to all students. This can be achieved by using the account provided by the "Kommunikations und Informationszentrum"[15] of Ulm University, as every student is assigned such an account. The necessity of authentication is not mandatory for all roles, as for lookups i.e. to see which documents are obtainable, it is not necessary to authenticate oneself, whereas a login is compulsory for the administrative tasks. To access the KIZ login, the university provides an LDAP server where one can lookup people and see the groups a person is in. This lookup can be performed for generic values, such as name and room number for lecturers, staff members and student fellows by anybody from anywhere. The lookup for student details needs to be granted first. As well as ask if a username / password input is correct. In order to perform a certain role, the person can be assigned to different groups, namely the administrator group and the handler group. This way, the overhead for user management is minimal. Alternatively separate logins could be created, meaning that these accounts would not depend on the university's LDAP. For a single administrator or root access, this single, independent login is purposeful in terms of fallback safety but not preferable in general.

User Interface

As some users of current systems only access digital files, doing so with the new system would on one hand make the change for previous users easier, and on the other hand provide a good fallback strategy if there is no way of using another graphical user interface. Therefore, the documents should be arranged hierarchically and

named in an informative way. Thus ensuring a comprehensible overview via the file browser. For handlers and administrators who prefer to use the console, a well working script with filter functionality can be provided, allowing an experienced administrator to perform maintenance work quickly without a web interface, as well as remotely by using a secure shell. Finally, the main interface for both end users and administrators should be a web interface, as it is accessible remotely, platform-independent, and, if implemented properly, the most intuitive interface of all three.

Document Style

In order to provide a unified reading experience for students reading the minutes of examinations, a template could be developed for students creating them. These templates should be found in the document system or at least be provided on the student associations' websites. Optionally an additional system interface could allow students to create a document online. The text input would be processed by the same template that the students could download to create the minutes offline, which ideally would be a LaTeX template with which the document submitted online can easily be transformed into a PDF, and then handed to an administrator for further processing. Another template format could be a standard format such as the Open Document Format or one compatible with Microsoft's office suit.

Printing

In order to hand the documents over to the end users, they have to be printed, thus requiring a print job to be created. This print job should contain the files to be printed and maybe an additional cover page. To save money, the documents can be joined and printed double sided. A collection can be created by a handler and then printed, which is, in essence, the way most student associations handle it today. Alternatively, one could submit a collection by selecting the wanted documents online, creating a collection with the files involved. Now a handler would have to approve or disapprove the print of that job. An email ticket system would also be possible to create collections that are directly printed. One or several printers should be available, giving the system the option to selectively print on different printers. The configuration of this behavior should be modifiable via configuration files.

Backups

As all documents are in fact single files, they can be backed up by using traditional strategies such as incremental backup with the Grandfather-Father-Son strategy, or

4 Design

via snapshots. The backup method is not significant for this thesis and thus will not be discussed further. It will instead be handled by the server and the backup strategies applying thereto.

Maintenance

The easiest way for students to submit a document is to write it on the computer. From there it can be either printed and handed over or sent to an email address. Therefore the simplest work flow would consist of the following: a student creates a report on an examination, exports a PDF and sends it to a student association. Existing minutes can be scanned, thus digitalized and then sent to the same email address. Later the PDF file has to be extracted and imported into the working system, along with all the necessary meta information. After importing the file, it should still be possible to modify the meta data, as an error may have occurred while importing it. Having an email account should not be the only way to import files, as a failure of the email server would otherwise hinder the import of new documents. By using files and a structured file system, files can simply be copied into the file hierarchy and thus imported into the system. In this case, the proper meta information has to be provided in some other way. An application or script could also do the same thing, namely moving the file to a certain location in the file system. However, that way the import process would be more controlled compared to if the administrator did it by hand. All these processes should ideally be doable remotely. The mentioned web application would be accessible remotely, but remote logins, remote desktops, or a remote shell could also make the script or file browser interface more practicable.

4.2 Mock Ups

Given that the web as a platform is usable with different devices and operating systems and because most users will use this interface to interact with the system, the highest emphasis was put on building the web interface. Nevertheless, the shell script is expected to be used by advanced administrators and was implemented keeping that in mind.

In this section, the idea for the web interface's design as the main interface will be discussed.

4.2.1 End User Interface

As end users would appreciate if they could search for documents themselves to check whether the ones they are searching for are available, an interface should provided options enabling them to do so. Figure 4.1 shows an interface which allows for searches via

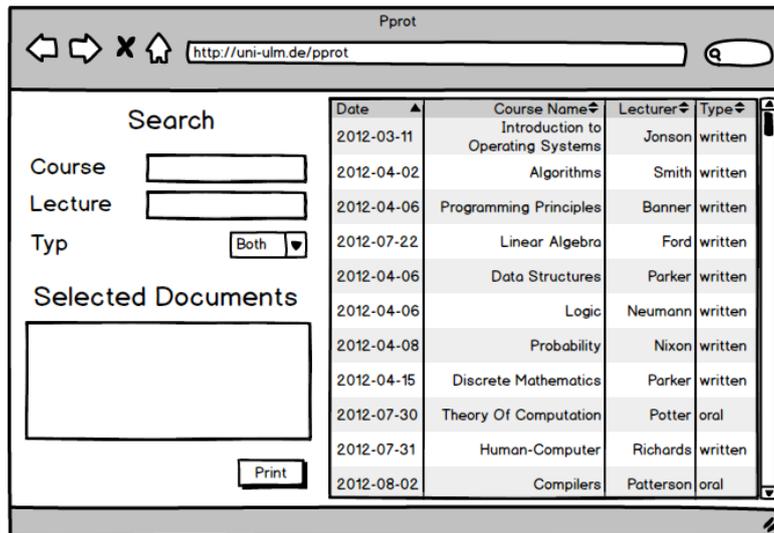


Figure 4.1: The search interface to be provided to the end user.

the course name or the lecturer, as well as by type, e.g. a document containing details regarding an *oral* or *written* past examination. The system can be designed to provide orders from the end users; users then should be able to select the documents they want to have. Afterwards the end user will see an overview of the collection and can create a print request. Otherwise the search interface will just allow an end user to view available documents, whereas the print request can only be issued by handlers. As the focus of the thesis lies more on the aspect of providing access to the documents rather than supporting maximum functionality of each role, the creation of collections by end users is regarded with lower priority, and thus was not implemented, yet.

4.2.2 Handler Interface

The role of the handler is to provide documents to the students by printing them out and handing them to the students asking for them. To be able to do so, for one, they should

4 Design

have access to the search interface just as the end users do, allowing them to search for documents and create a collection of documents to be printed.

If the system is implemented in a way that end users preselect the documents they want and then send a print request, the handler will be able to evaluate it. This request consists of a collection i.e. a list of documents that can be displayed to the handler in the form of a summary, as shown in Figure 4.3. Another interface lists end users' requests, along

Date	Customer ID	Documents	Price
2013-01-06	john.smith@uulm.edu	6	70ct
2013-01-06	frank.miller@uulm.edu	3	30ct

Delete Selected Delete All Total Tasks: 2

Figure 4.2: The Overview Interface shows a list of collections ordered by end users.

with an identifier and a time stamp (see Figure 4.2). This way an end user approaching a handler can identify that collection via the *Customer ID* and the *Request Date*. To print this collection or view more details, the end user's request can be opened, which leads to the aforementioned interface seen in Figure 4.3. This summary, again, shows the identifier provided by the end user, the date of the request as well as the collection's documents.

For the sake of preventing abusive behavior against the system, it might be an option to charge a small fee for each document printed. This fee can then also cover the costs for running the system and the printing costs. This fee should then appear in the summary, so that it can be charged before printing the collection.

Finally, the handler can selectively modify the collection by deleting single documents or entire requests, or print the displayed collection.

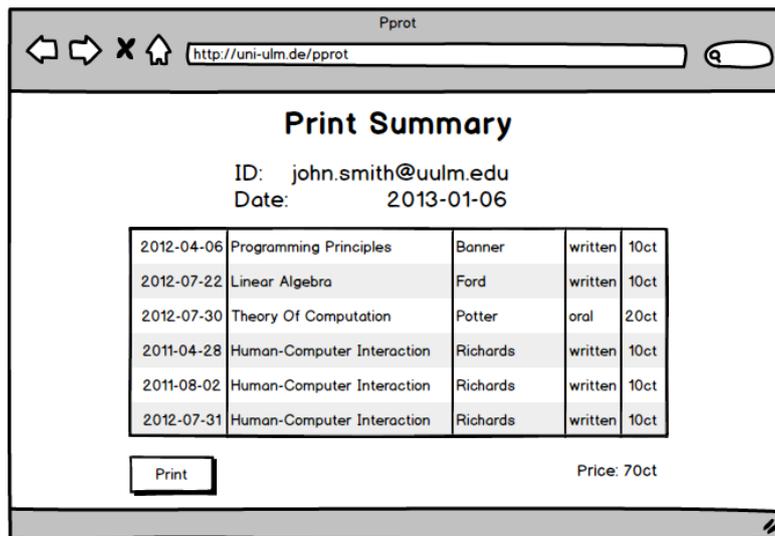


Figure 4.3: The Summary Interface lets a handler print the documents within in a collection, as well as view and modify its details.

4.2.3 Administrator Interface

In order to search for documents, create collections and finally print them, the administrator first has to import these documents into the system. They also have to be maintained i.e. if meta data has changed or was found to be wrong, the administrator must be able to alter these values. It might also be necessary to remove the document from the system completely which is to be done by an administrator as well. An interface that allows the administrator to do so is to be created. Figure 4.4 showing how a document can be imported. On the left an email inbox is displayed; the corresponding attached files are displayed on the right. After reviewing the file and determining the necessary meta data, the file can be imported.

To modify the meta data, a separate interface should be provided, allowing each attribute to be edited. Changes can be applied by editing entries in a list, as seen on the right hand side in Figure 4.5. Applying the changes will save the whole set and update the representing data set of the meta data.

4 Design

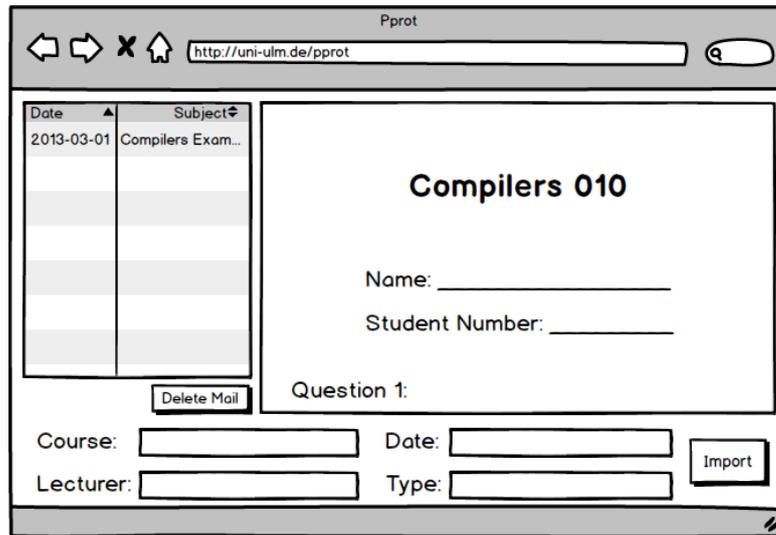


Figure 4.4: The Import Interface lets the administrator import documents to the system.

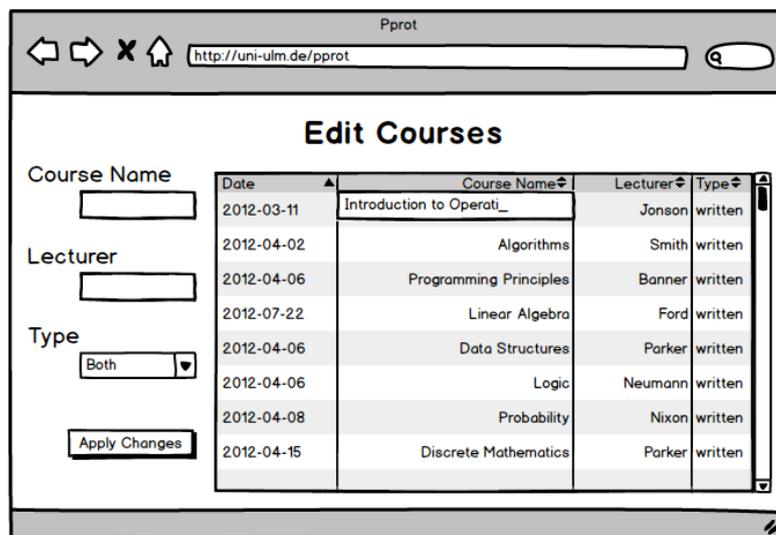


Figure 4.5: The Edit Interface lets the administrator modify a documents meta data.

5 Architecture

In this chapter, the technologies used to implement the system will be discussed and shown. Furthermore, the system's detailed structure and capability is going to be illustrated. The different interfaces will be shown in action and their interconnection discussed, as well as the roles of the human counterparts.

5.1 Technologies

In this section, the utilized technologies will be introduced, as well as their usage within the system's implementation.

5.1.1 JSON

As pointed out by Adam Lith and Jakob Mattsson, Carlo Strozzi first used the term NoSQL[16] in 1998 for his relational database that did not offer an SQL interface. Such databases do not primarily use tables to store their data, but rather via loosely-structured collections of data. They are more flexible as their fields are loosely defined.

For the purpose of the implemented system such flexibility is most useful. CouchDB[8] is a representative of the NoSQL family. While not a typical Database Management System such as MySQL[6], DB2[4] and so on, it still provides an ACID semantic[10] by using a Multi-Version Concurrency Control[2]. Data is *eventually consistent*[3] and can be processed using map/reduce operations.

In this thesis, CouchDB can be used to store the document's meta data. In the system's implementation it has not been used, as the requirements for a server running the system should be kept as low as possible, while also maintaining the concept of information storage by loosely defined structures - the meta data is therefore stored in single JSON files. As CouchDB uses the same format to store its data, using JSON to represent the meta

5 Architecture

data makes it possible to adopt to a CouchDB database in the future. Until then, the meta data file is created by a script which iterates through the file structure in order to output the necessary information. The structure of the JSON corresponds to the following:

```
1 [{"file": "BASENAME/TYPE/COURSE/YYYY-MM-DD.LECTURER.S.pdf",
2   "date": "YYYY-MM-DD",
3   "type": "TYPE",
4   "solution": true,
5   "lecturer": ["LECTURER1", "LECTURER2"],
6   "courses": ["COURSE1", "COURSE2"]},
7 {...}, ...]
```

file

Type: *String* - This element represents the location of the file to be described. The name of this file follows the structure described in section 5.1.2.

date

Type: *String* - The date is represented according to the ISO 8601 standard[14], therefore consisting of four digits representing the year, and two for both month and day. Each time unit is separated by a "-".

type

Type: *String* - A document can belong to one of several categories; these may depend on whether it is an examination or lecture note, e.g. for examinations the content of the type field is either "*oral*" or "*written*". As for now the system also supports the type "*other*" but it could be any string.

solution

Type: *boolean* - If the document also contains solutions, such as solutions to a written past examination, this entry is to be added and set to true. If there are no solutions, then the entry is optional.

lecturer

Type: *String Array* - The lecturer defines who held the lecture corresponding to the document. In single cases, more than one lecturer can be involved in one course, for example a course can be taught in a lecture series. The array elements are names;

more precisely, last names. This might lead to conflicts but the format is flexible enough to change the value to “*Last Name, First Name*” in the future.

courses

Type: *String Array* - As, for example, an oral examination can include different topics from different courses at once, the meta data has to support such cases. There all course names have to be added, which is saved in an array of strings. For the names of courses, the ones from the course catalog can be used and shortened if too long.

5.1.2 File Structure

It is to be mentioned that for the system, the documents are of the greatest importance. This is because if there were meta data but no actual documents, the end user could not get the document they wanted. However, if a document’s meta data is missing, it can still be found, and can be updated by the administrator over time, eventually allowing for a flawless entry. In short: If there is no file, any meta data is useless.

For the current implementation, the information in the JSON file used to provide the document’s meta data is created by a script analyzing the file structure.

A well-formed file structure is also important when using the file browser to navigate to a document, when requiring it to be done in an intuitive way. Figure 5.1 shows the abstract file hierarchy.

The following guidelines apply to all file and directory names

All file names, no matter whether directory or PDF, must not contain spaces. Instead an underscore “_” is used.

If a file contains more than one value for an attribute, such as more than one lecturer, these values are separated by a “,” and will be used to create the array values.

Files containing different attributes should separate these values by a “.”.

The following categories describe the markup seen Figure 5.1

Association

Example: “*Fachschaft_Informatik*”. The association field allow the system to be used by different student associations. Each association is assigned a distinct name. The

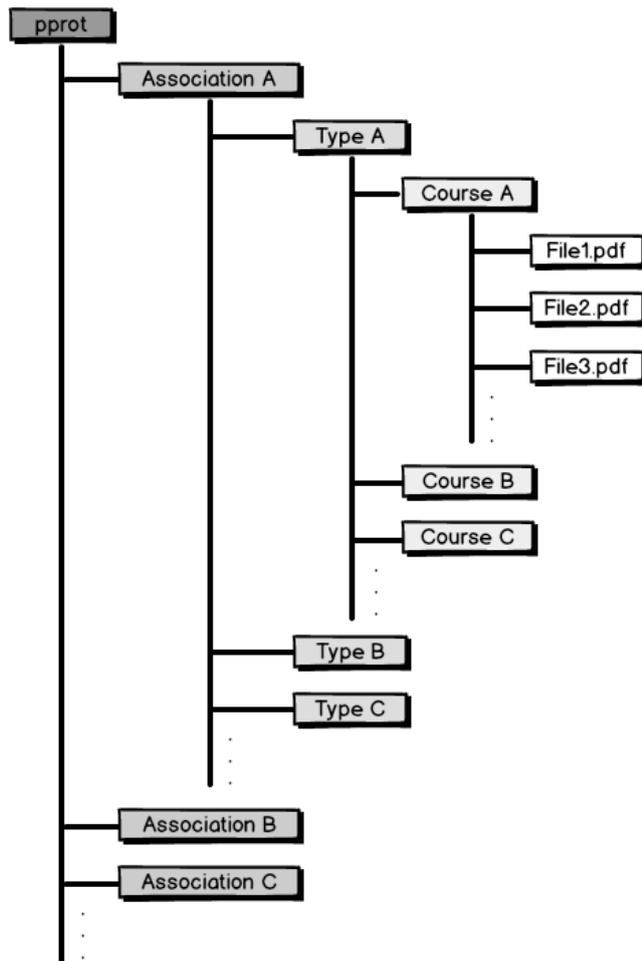


Figure 5.1: The file structure to be used for the system.

association's documents are all in one subdirectory. This makes it possible to grant a group of people file access to the whole directory to i.e. making file access possible for a person belonging to a group in the LDAP tree, rather than having all documents in one directory and handling the file attributes for each document independently.

Type

Can be one of the following: *written*, *oral*, *other*. To search explicitly for written or oral examinations, one can open the corresponding subdirectory. The *other* might contain protocols of experiments. If the number of these protocols exceeds a threshold, they might be put into a separate folder.

Course

Type - Example: “*COURSE1,COURSE2*” - “*Algebra_1,Algebra_2*”. If a document consists of two courses’ examinations it has to be mapped with the directory’s name. This is done by giving it both of those names, sorted by name and separated by comma. This will then represent an array of the same form.

File

Type - Example: “YYYY-MM-DD.L1,L2.pdf” - “2013-05-09.Goethe.pdf”. Similarly to the *course directory*, an examination can respond to more than one lecturer. These names are also separated by comma; and the filename includes a date. The ISO 8601[14] formatted date is used as the first part of the name so that listing all the files in one directory will present them sorted by date directly.

5.1.3 Meta Data Creation

In section 5.1.2 it was briefly addressed that all meta data can be extracted from the underlying file structure. The following example may illustrate this further.

This is the path and name of a file:

```
.../Computer_Science/written/IT_Security/2013-01-30.Smith,Clark.pdf
```

From this, all the meta data used in section 5.1.1 can be extracted because the document was named according to the guidelines discussed in section 5.1.2. This is performed by a shell script which outputs the correct JSON file. Which is, in this case, the following:

```
1 { "file": "...",
2   "date": "2013-01-30",
3   "type": "written",
4   "lecturer": ["Smith", "Clark"],
5   "courses": ["IT Security"] }
```

The JSON file can be created and modified by and for each association individually, meaning that if a person with write permissions to the association’s directory updates information to a file by altering its filename, that person can call the script to update the meta data file. To avoid misuse, the steps of altering a path and executing the meta data file should

5 Architecture

happen atomically, thus the usage of scripts or programs to follow this procedure is to be preferred.

5.1.4 Email

Documents can be sent to the associations via email. This is a unified way of receiving documents as emails can be accessed across platforms and across devices with various tools. Certain devices (e.g. a multipurpose scanner with network access) are also able to provide documents in this way. To access the email, an administrator can use an email program and download the messages or rather use the front-end designed for the exact purpose of receiving emails, viewing the PDF attachments, extracting them, and moving the file to the right place in the file hierarchy according to its meta data.

5.1.5 Lightweight Director Access Protocol

As its name implies, this protocol provides access to a directory service. These directory services are hierarchically structured sets of records and can be used by, for example, for telephone directories. The protocol is called LDAP because it is a lightweight alternative to the DAP, which is specified by the X.500 Directory Access Protocol[22]. As the DAP in its early versions did not support TCP/IP, the IETF released a specification for the LDAP protocol with the RFC1487[23][1].

Today, DAP is capable of using TCP/IP instead of relying on the OSI protocol stack[22][21], though in the case of the underlying system, LDAP will be used. Microsoft's Active Directory[5] also utilizes LDAP version 2 and 3 to represent its records[12], meaning that LDAP can be used on different platforms, in the case of Windows even natively.

To understand the hierarchical structure, one can look at the output of a query:

```
1 dn: uid=Leo Hnatek,ou=people,dc=uni-ulm,dc=de
2 uid: Leo Hnatek
3 cn: Leo Hnatek
4 sn: Hnatek
5 givenName: Leo
```

```
6 o: Uni
7 objectClass: person
8 objectClass: organizationalPerson
9 objectClass: inetOrgPerson
10 objectClass: uuPerson
11 ou: Abteilung II-3 Internationale Angelegenheiten
12 mail: leo.hnatek@uni-ulm.de
13 roomNumber: Pav. Ehem.Rektoramt
14 telephoneNumber: +49 731 50-10
```

In this case, a person can be found via the *Distinguished Name dn*. The person has a relative distinguished name, belongs to an organization, and an organizational unit, and has two Domain Components. As one can see above, the distinguished name consists of four components making it distinct. The record itself also contains the email address, telephone and room number. LDAP can also provide an authentication method. This and elements added to a record, such as a group membership can be used to identify a user and determine a user's authority. If, for example, a student becomes the administrator of the system for association A, an attribute saying "userGroup: associationA.Administrator" can be added to that student's record. By accessing the LDAP for both a PC login and checking the group attribute with the web interface, only one single modification to the student's account has to be made and no extra account has to be created.

5.1.6 Common Unix Printing System

The Common Unix Printing System (CUPS)[18] is a service that can run on Unix-like operating systems such as Mac OS, Linux or Solaris. In the case of the implemented system, the accessibility from remote hosts and the option to print using the command line are especially useful. By using the command line tool `lpr`, one can add documents to be printed on a printer that is set up. Here printers on remote destinations can be added, as CUPS supports protocols such as the Internet Printing Protocol. Such remote printers can be selected by adding parameters to the print command.

5.1.7 Google Web Toolkit

The Google Web Toolkit (GWT)[13], is a toolkit for the development of websites. With the GWT, a developer can write Java source code and compile it into Java Script which can be rendered by the browser on the client side. But GWT also supports the server side where standard Java libraries can be used to implement a server utilizing these libraries to perform complex operations. Therefore, lots of functionality is already given, such as libraries to access LDAP or IMAP[19][7]. *Asynchronous Remote Procedure Calls (Async. RPCs)* are used to send messages back and forth between client and server. They form the interface between the Java-based server and Java Script-based client. Such usage of libraries and RPCs will be discussed later in this chapter.

5.1.8 SmartGWT

On top of GWT, one can use different libraries for both the client and server. Such libraries offer widgets to create a rich UI. Vaadin[17] and SmartGWT[20] are only two of such libraries. They facilitate the creation of the UI by extending the suit of forms and layouts, but also offer connectivity to data management. For example, one can easily use a database as a data source for a List Grid by using a wrapper class. Additionally, by only modifying this data source class one can modify the underlying method of accessing the data. For instance, the database can easily be exchanged with a JSON file. SmartGWT was used along with GWT itself for the implementation of the web interface of this thesis' system.

5.2 Component Diagram

The system as seen in Figure 2.1 does not involve the roles of the administrator or handler, nor does describe inner processes.

Figure 5.2 shows insights into the system as described with Figure 2.1. However it does it in more detail. The system expects documents to come into the system from the left. From there, the administrator takes care of them, by scanning the paper form documents and sending them to an email address or receiving an email by a provider they can import them into the working set of documents. They can also modify existing ones as described before. The inner system handling of the documents and the method of providing them

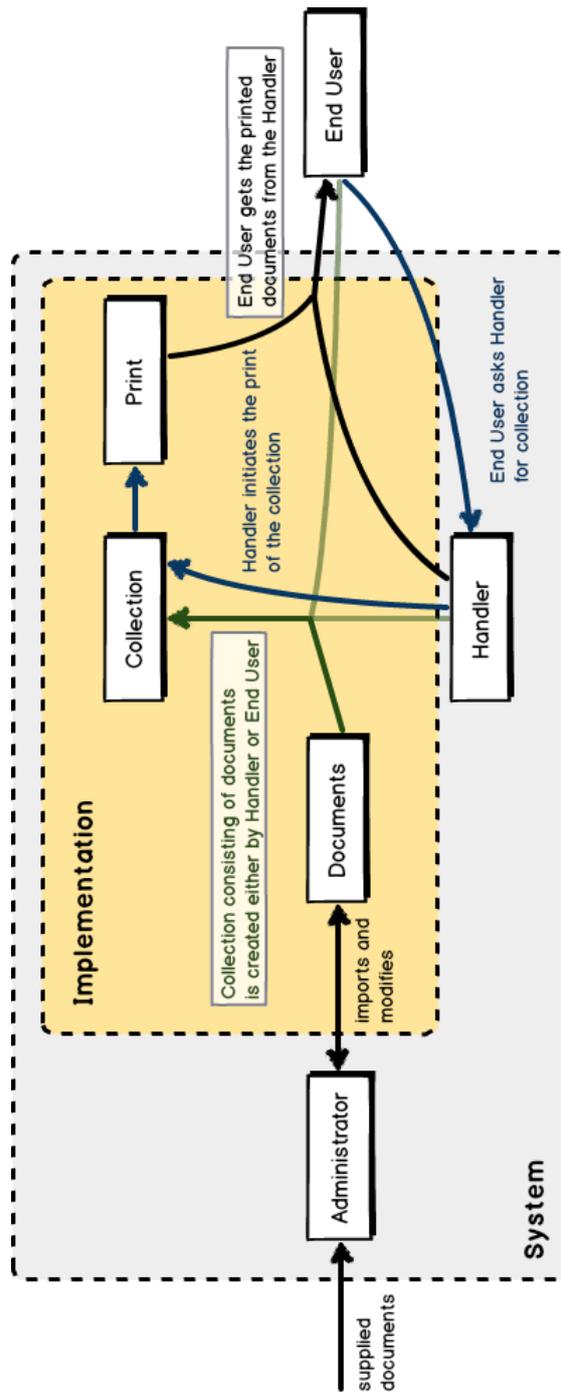


Figure 5.2: The system as seen in the Figure 2.1, but with a deeper look into the system.

to end users depends on the interface used. Explanatory information about the different interfaces can be found later in this chapter. The people involved in this work flow still keep the same though. The end users either can create a collection containing different documents themselves or ask the handler to do so. The handler finally prints the documents and hands them over to the corresponding end user.

5.3 Web Interface

The web interface is the anticipated main interface to be used by students of any role as it is to be the system's most intuitive, yet also productive interface.

5.3.1 Walkthrough

The implemented interfaces will be shown in this subsection and explained in further detail.

End User Interface

After browsing to the front page, the screen as shown in Figure 5.3 is displayed. From there, the end user can filter for the searched value as shown in Figure 5.4. The document type can be chosen from a list of given options, and the date can be set by choosing from a calendar. If only documents with solutions are requested then they can be filtered by clicking the check box and for the remaining fields, the results can be filtered by entering a substring of the desired result.

If the end user needs a more specific structure of the data, they can sort the results by columns, such as descending first by lecturer, then by available results and ascending by date. Figure 5.5 shows such a multi-column sort.

Finally, the end user can select as many documents they want. These will appear in the list below, and can be removed by clicking on a single column or all at once by clicking the button labeled *remove all*. It is possible to configure the system in such a way that only a maximum number of documents can be selected before printing to prevent misuse. If it is wanted to enable the end user with a print request, the list of selected documents can be submitted to a collection for further processing by clicking the *print-Button*. Figure 5.6 shows a selection of documents.

5.3 Web Interface

Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
08/05/2011	Muendlich	Kestler	Evolutionaere Algorithmen	<input type="checkbox"/>
07/01/2011	Muendlich	Slomka	Rechnerarchitektur	<input type="checkbox"/>
04/15/2011	Muendlich	Neumann	Ausgewaehlte Methoden und Anwendungen der Computer	<input type="checkbox"/>
04/07/2011	Klausur	Toran	Formale Grundlagen der Informatik	<input type="checkbox"/>
04/07/2011	Muendlich	Borchert	Objektorientierte Programmierung mit C++	<input type="checkbox"/>
04/04/2011	Klausur	Penso	Kombinatorik	<input type="checkbox"/>
03/30/2011	Klausur	Weber	Sicherheit in IT Systemen	<input type="checkbox"/>
03/28/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
03/28/2011	Klausur	Weber	Web Engineering	<input type="checkbox"/>
03/16/2011	Muendlich	Schwenker	Data Mining,Neuroinformatik 2	<input type="checkbox"/>
03/03/2011	Muendlich	Schweiggert	Systemnahe Software 1	<input type="checkbox"/>
03/01/2011	Muendlich	Weber	Mobile Ubiquitous Computing	<input type="checkbox"/>
02/28/2011	Klausur	Palm	Neuroinformatik1	<input type="checkbox"/>
02/21/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/21/2011	Klausur	Ohlebusch	Algorithmen der Sequenzanalyse	<input type="checkbox"/>
02/07/2011	Muendlich	Hauck	Architekturen fuer Verteilte Internetdienste	<input type="checkbox"/>

0 Protokoll(e) ausgewählt

Multilevel Sort Auswahi aufheben Drucken

Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
zu druckende Protokolle auswählen				

Figure 5.3: The screen as it appears to the end user after start.

Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
04/04/2011	Klausur	Penso	Kombinatorik	<input type="checkbox"/>
04/17/2010	Klausur	Maier	Kombinatorik	<input type="checkbox"/>
02/27/2010	Klausur	Maier	Kombinatorik	<input type="checkbox"/>
06/15/2009	Muendlich	Maier	Kombinatorik	<input type="checkbox"/>

0 Protokoll(e) ausgewählt

Multilevel Sort Auswahi aufheben Drucken

Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
zu druckende Protokolle auswählen				

Figure 5.4: The end user interface after filtering for a lecturer.

Handler Interface

The handler interface can be implemented as the means to provide a list of collections as described in the design chapter 4. This has not yet been implemented as it's not yet clear if the system will be used in that way, i.e. it is not clear whether end users will be allowed to create collections, or whether collections will be created by handlers themselves. These

5 Architecture

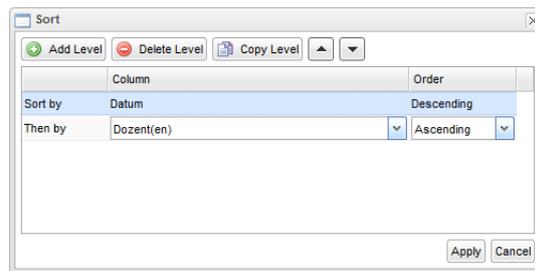
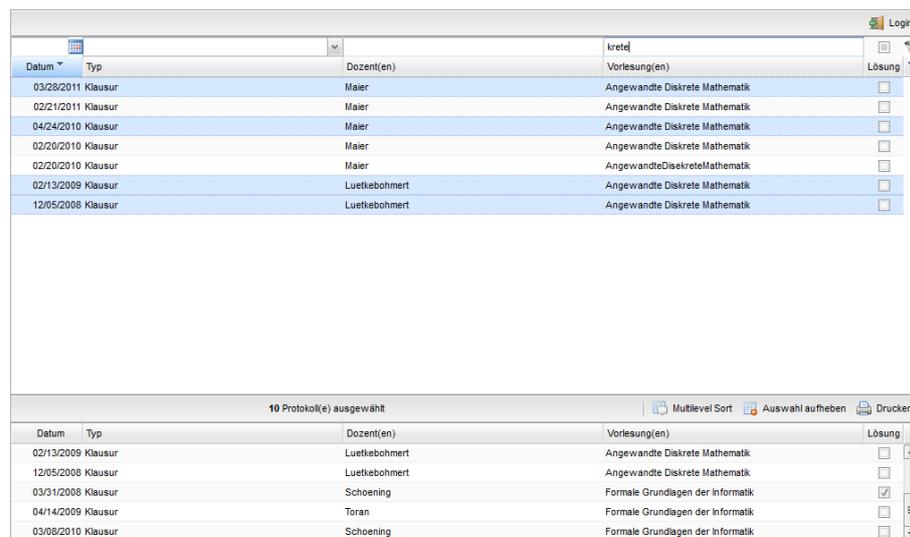


Figure 5.5: In order to sort by more than one column one can add more sort criteria.



Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
03/28/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/21/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
04/24/2010	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/20/2010	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/20/2010	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/13/2009	Klausur	Luetkebohmert	Angewandte Diskrete Mathematik	<input type="checkbox"/>
12/05/2008	Klausur	Luetkebohmert	Angewandte Diskrete Mathematik	<input type="checkbox"/>
10 Protokoll(e) ausgewählt				
Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
02/13/2009	Klausur	Luetkebohmert	Angewandte Diskrete Mathematik	<input type="checkbox"/>
12/05/2008	Klausur	Luetkebohmert	Angewandte Diskrete Mathematik	<input type="checkbox"/>
03/31/2008	Klausur	Schoening	Formale Grundlagen der Informatik	<input checked="" type="checkbox"/>
04/14/2009	Klausur	Toran	Formale Grundlagen der Informatik	<input type="checkbox"/>
03/08/2010	Klausur	Schoening	Formale Grundlagen der Informatik	<input type="checkbox"/>

Figure 5.6: The search interface to be provided to the end user.

two options in the workflow are displayed in Figure 5.2. Instead, the handler can use the interface as described for an end user but with the difference that the print button will show a print dialog to choose the printer to print to. This means that the end user comes to a handler during their opening hours and asks for documents.

Administrator Interface

As for the administrator's tasks the interface has to meet the needs. Before interacting with the system, an administrator will have to login, by selecting the top right corner's login button and inputting their user name and password. The user will then have to choose the administrator role and click *ok*. It would also be possible to determine the account type by the LDAP group the user is in which would make it impossible for a single person to have

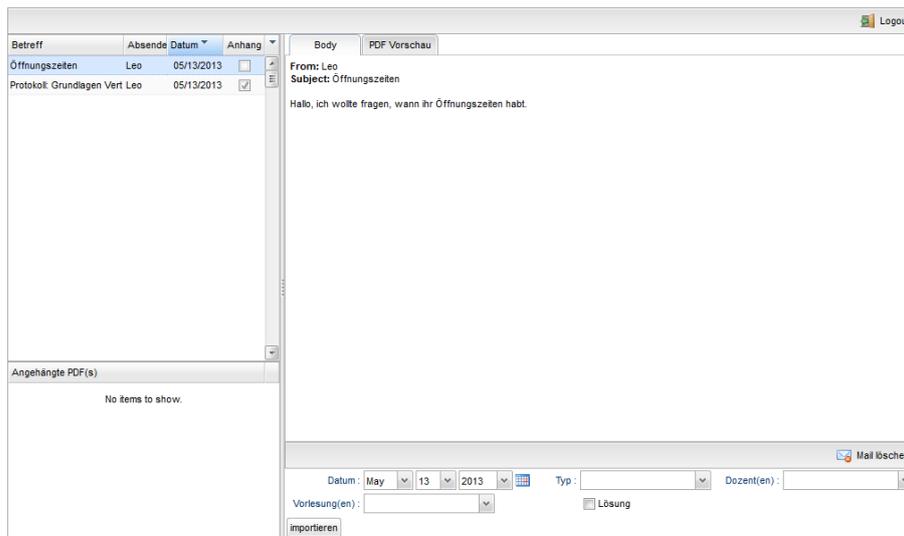


Figure 5.7: The administrator's interface after opening the import function.

more than one role, so the first option is to be preferred. The administrator is tasked with the editing of the documents in the repository and importing new ones. Figure 5.7 shows the screen that will appear when an administrator wants to import a document. On the left, one can see the inbox of an email account configured for the association the administrator is representing. Alternatively, depending on the amount of messages, a single email account could be used for all associations together, which would make it easier for a provider administrators to coordinate document but would weaken security.

After selection of a message, one can see its body on the right as well as whether it contains PDF attachments. If so, those attachments would appear in the list on the lower left side as seen in Figure 5.8. By selecting a file, it can be viewed in the preview-tab. Meta data can be added to the document and by clicking the *import* button, the document will be added to the document repository. With the import interface, it is also possible to sort the email, meaning that emails with a file attachment can be found easily. It is also possible to delete mails after an import to allow for a smooth work flow.

In case the administrator needs to modify meta data because errors occurred while importing, the administrator can use the modification interface as seen in Figure 5.9. There, as already seen in the end user interface, documents can be filtered to find those matching a pattern. By clicking a single entry, a whole column can be edited and changes will be

5 Architecture

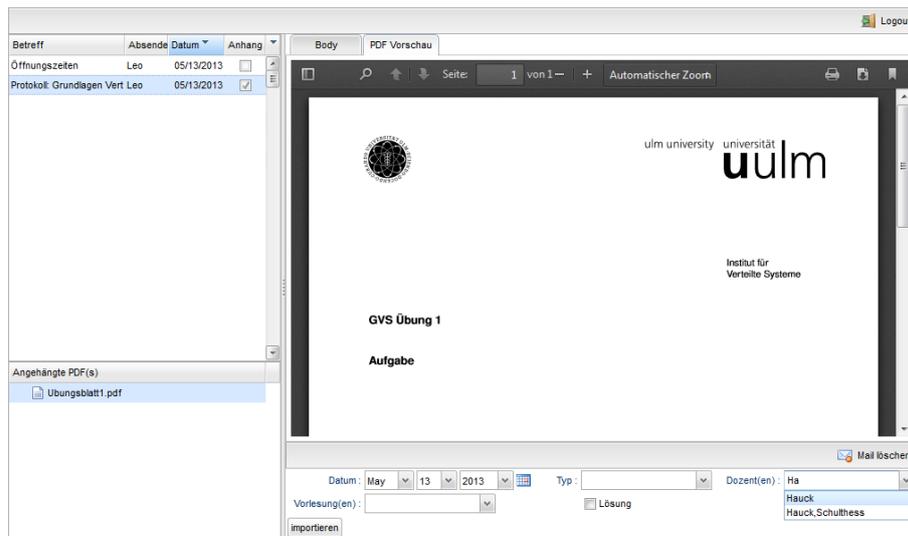


Figure 5.8: After choosing an attached file, the PDF can be previewed and meta data can be added.

highlighted afterwards. After ensuring the edit process is finished, it can be submitted by clicking *save*, whereafter the changes will be transmitted and applied on the server.

Datum	Typ	Dozent(en)	Vorlesung(en)	Lösung
08/05/2011	Muendlich	Kestler	Evolutionaere Algorithmen	<input type="checkbox"/>
07/01/2011	Muendlich	Slomka	Rechnerarchitektur	<input checked="" type="checkbox"/>
04/15/2011	Muendlich	Neumann	Ausgewaehlte Methoden und Anwendungen der Compute	<input type="checkbox"/>
04/07/2011	Klausur	Schoening	Formale Grundlagen der Informatik	<input type="checkbox"/>
04/07/2011	Muendlich	Borchert	Objektorientierte Programmierung mit C++	<input type="checkbox"/>
04/04/2011	Klausur	Penso	Kombinatorik	<input type="checkbox"/>
03/30/2011	Klausur	Wieber	Sicherheit in IT Systemen	<input type="checkbox"/>
03/28/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
03/28/2011	Klausur	Weber	Web Engineering	<input type="checkbox"/>
03/16/2011	Muendlich	Schwenker	Data Mining, Neuroinformatik 2	<input type="checkbox"/>
03/03/2011	Muendlich	Schweiggert	Systemnahe Software 1	<input type="checkbox"/>
03/01/2011	Muendlich	Weber	Mobile Ubiquitous Computing	<input type="checkbox"/>
02/28/2011	Klausur	Palm	Neuroinformatik 1	<input type="checkbox"/>
02/1/2011	Klausur	Maier	Angewandte Diskrete Mathematik	<input type="checkbox"/>
02/1/2011	Klausur	Oehbusch	Algorithmen der Sequenzanalyse	<input type="checkbox"/>
02/07/2011	Muendlich	Hauck	Architekturen fuer Verteilte Internetdienste	<input type="checkbox"/>
12/01/2010	Muendlich	Hauck	Multimediammunikation	<input type="checkbox"/>
11/12/2010	Muendlich	Kestler	Evolutionaere Algorithmen	<input type="checkbox"/>
11/12/2010	Muendlich	Kestler	Evolutionaere Algorithmen	<input type="checkbox"/>
11/10/2010	Muendlich	Toran	Algorithmen und Datenstrukturen	<input type="checkbox"/>
10/31/2010	Klausur	Toran	Logik	<input type="checkbox"/>
10/30/2010	Muendlich	Weber	Mobile Ubiquitous Computing, Multimediasysteme	<input type="checkbox"/>

Figure 5.9: In order to maintain the documents and change meta data if needed, the administrator can use the modification interface.

5.3.2 Client-Server Model

As the web-based interface uses GWT, it was realized in a client-server manner.

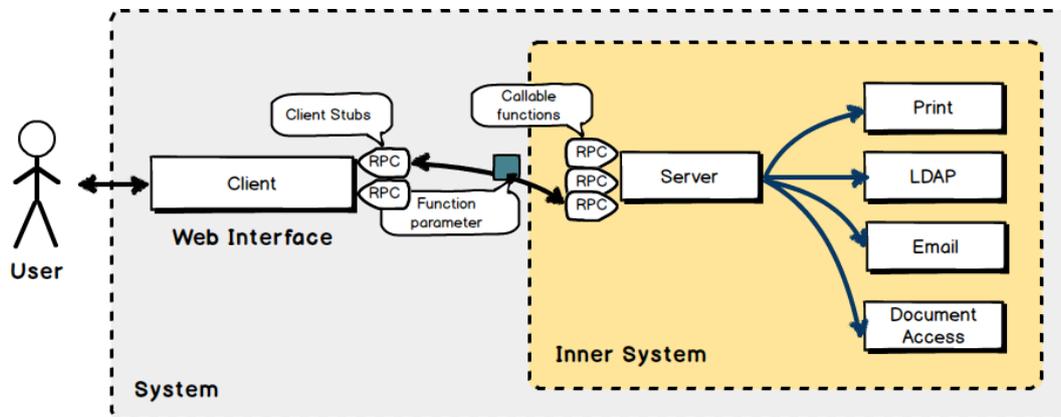


Figure 5.10: The user in a certain role interacting with the system's web interface. The client interacts with the server via Remote Procedure Calls.

In Figure 5.10 one can see that a person interacting with the system's client-side uses the web interface for interaction. The client offers easy access to the system where more complex functions can be performed by the server. For example: a user may want to log into the system as an administrator to import documents. For this the client-side calls procedures on the server-side of the system remotely. The server, unlike the client, can access services such as LDAP for authentication, and emails via IMAP to import documents. Procedures with parameters and procedure results are transferred back and forth between client and server offering the needed functionality.

Therefore, requests and results ought to be embedded with a serializable data structure. Non-primitive structures of this kind are defined as shared classes and are accessible by both client and server.

5.3.3 Client Structure

The client represents the user's front end. It needs to be used by all roles, thus needs to implement different interfaces.

Each interface consists of a representing class, which contains only one object. To the outside it offers a `getInstance()` method, therefore implementing the Singleton pattern.

5 Architecture

In this manner, it is not necessary to create an instance of that class every time a context change or rather interface switch happens. Note that the code for the client runs inside a client's browser. So unlike the server, the objects created for the client are only readable from one single machine, that is the user's browser. Therefore it is not only harmless but efficient to use the Singleton pattern in this case.

If an interface-class needs to access any form of data, this class has a *DataSource* class only for this purpose to improve modularity. Figure 5.11 shows this relation.

A remote procedure call has to be used for each interaction between client and server.

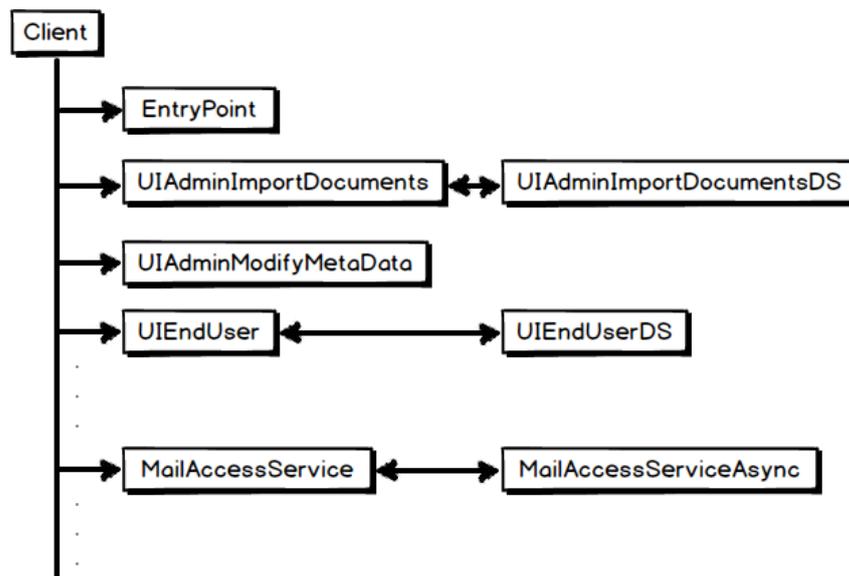


Figure 5.11: Client interface structure.

These RPCs are grouped into functional context classes. The access to mail, for example, requires more than just one method, such as `fetchMails`, `deleteMail` and `saveAttachment`. These methods are grouped in the `MailAccessService` class and have an asynchronous counterpart `MailAccessServiceAsync`, as well as an implementation of the methods to be run on the server called `MailAccessServiceImpl`.

There is an `EntryPoint` class that defines which interface is to be selected. It offers a login option which causes the container within the web interface to apply the instance of the respective role's interface.

5.3.4 Server Structure

The server basically has two main functions. One is to execute calls from clients and to interact with them by returning procedure calls, and the other is to execute preset functions. The later is performed repetitively in a loop. The time when a function is to be called can be defined in a configuration file. Options such as an email account's user name and password, as well as commands sent to the operating system for printing can be set in this file.

In order to synchronize emails, one might want to do this either in the mentioned timed manner or maybe do it each time a client calls a fetch method. Such an email access function can then be defined in an access class that can be called from different places on the server. Note that it might lead to inefficiency on the server side if all the mails are fetched on the client but can rather be cached and updated periodically or on deletions of emails.

The server does more than just email access but this was taken as an example. Section

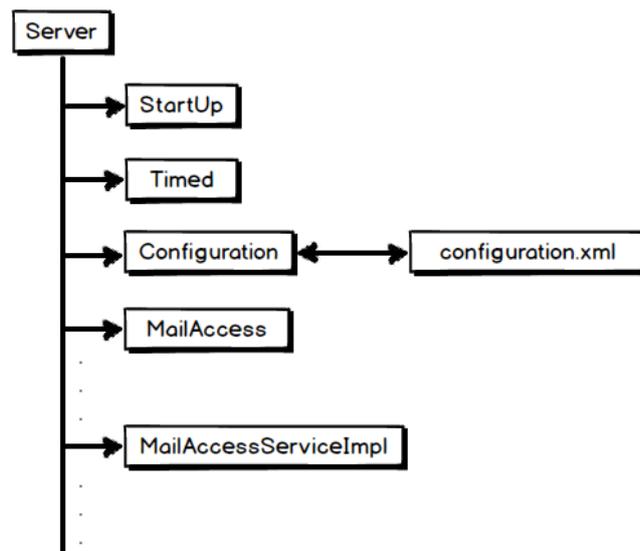


Figure 5.12: The server's structure with the configuration file, implementations of client services, and helper classes.

5.3.5 describes more functional components of the server. Figure 5.12 shows that every service method on a client has an implementation class on the server. The configuration file, wrapper class and a *StartUp* class are also shown. The *StartUp* class defines tasks that are executed once upon the system's start. The helper classes are also shown, as well

as the configuration wrapper class which offers a method to search for attributes with an internal `Properties` instance.

5.3.5 Server Components

In this subsection, the components of the server are illustrated, of which there are four, as described below.

File Access

- Moves and renames files.
- Creates file structures (directory trees).

Mail Access

- Accesses the inbox of email accounts.
- Deletes mails.
- Saves attachments to files.

User Authentication

- Validates username and password.
- Identifies user's role.

Print

- Interacts with printers.
- Can print given files either by filename or by a given collection containing the filenames.

Configuration

Gives one the ability to save and load properties of the server.

StartUp

Defines what happens on the start of the server.

Timed

Contains processes that are executed after a specific time interval.¹

In Figure 5.13, 5.14, and 5.15, one can view the interaction between the system's components.

¹The process's time interval can be defined in the configuration.

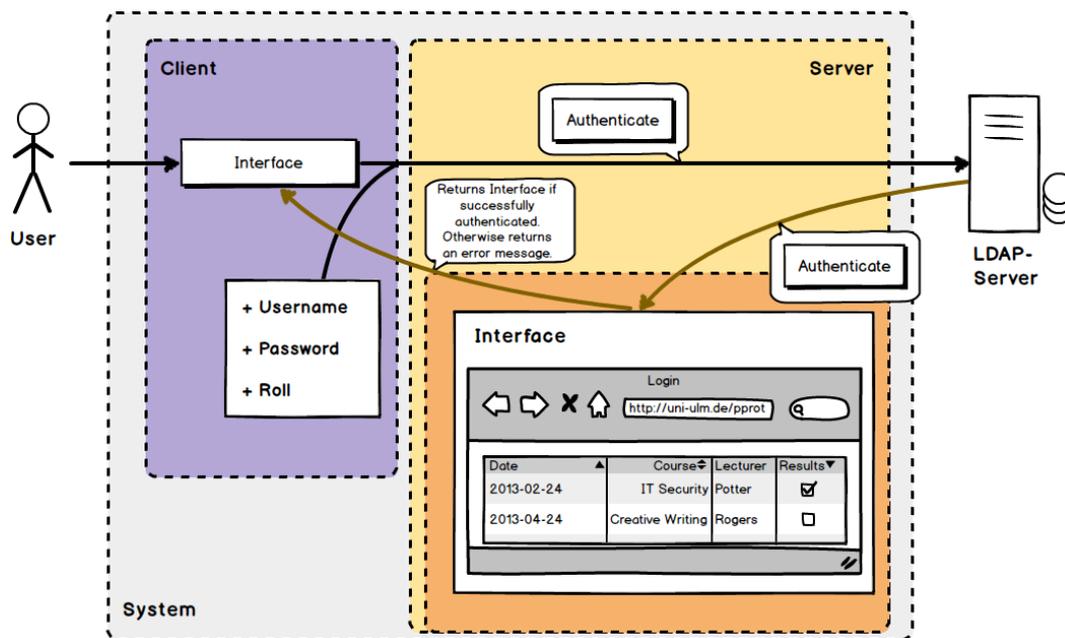


Figure 5.13: The authentication of a user involves the system's Authentication class as well as the LDAP server the login details are being validated against.

First, in Figure 5.13 a user whose role has not been identified yet, wishes to sign in. They therefore input their login details as well as their anticipated role into the client. The client runs on the user's browser which is not considered trustworthy enough to allow the user take on a role; therefore the given credentials have to be verified by the server. For this authentication process, the login information is sent to the server using a remote procedure call. The class used for this is the *AuthenticateUserServiceImpl* class. It connects to the LDAP server which has the capability to authenticate the user. The LDAP server also has the information whether the user belongs to the group they have provided. By verifying the username / password combination and providing the group information, the system's server can certify that the input data is valid and grant access to the restricted area. If the login was successful, the system's server returns the right interface to the client which will then display it; otherwise the client will show a fail message.

After the user has taken on the role of the administrator, they might import documents. For that they call up the email list. The mail's attachments will automatically be downloaded in a temporary import directory. If the file is imported, it will be moved to the right place inside the file hierarchy using methods from the File Access class. In case the user is logged in as a handler, they can print documents directly for end users or access collections created by

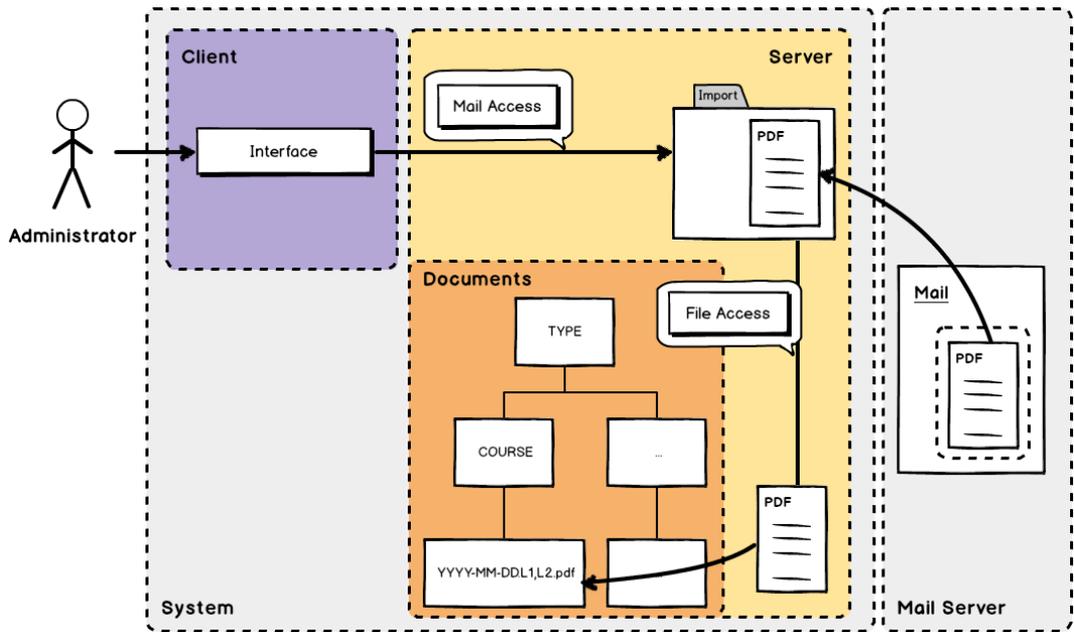


Figure 5.14: Importing a document from a mail attachment into the system.

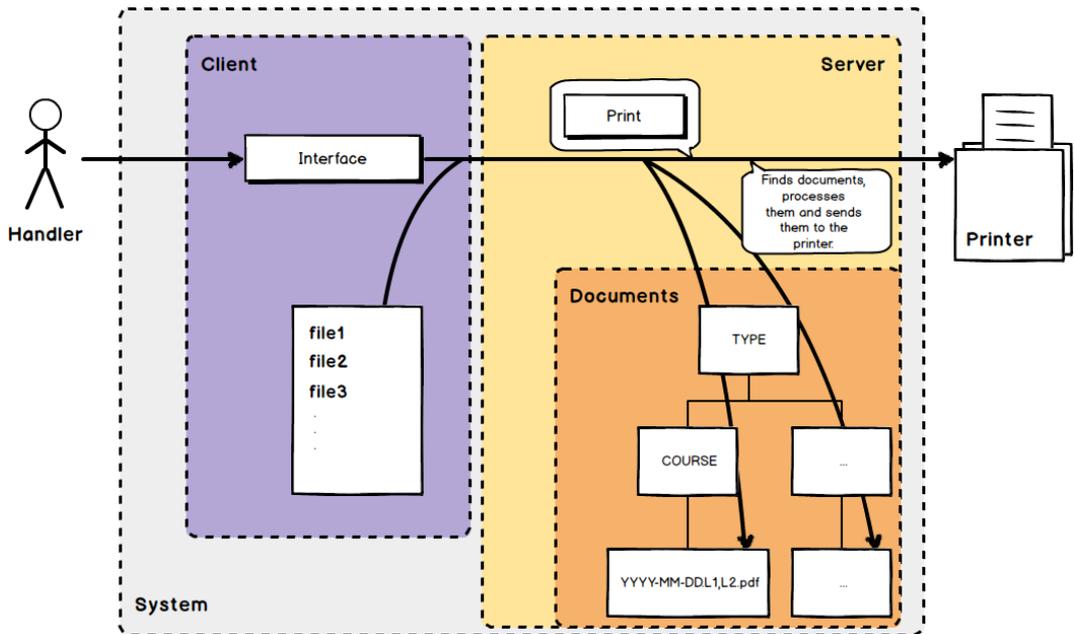


Figure 5.15: A list of files or a collection is printed.

end users. To achieve this, the client passes on a list of files to the server. The server then looks them up, returns with the filenames and processes them by utilizing a system call that can be set in the configuration file. This is in fact an `lpr` command first piped through a ghost script, but this can easily be altered via the configuration file. The server could also access collection files listing filenames to be printed as created by end users. In any case, the print class can call print commands to print documents on paper.

5.3.6 *StartUp* And *Timed*

To interact with the operating system the document system is running on, it is possible to use the *StartUp* and *Timed* classes. The configuration file contains entries for this purpose; one of these is used for the startup command and is called `startup.command`. The value of this key is called with a system call and therefore runs with the web server's privileges. The *Timed* class implements processes that are going to be executed in repetition, such as the fetching of new emails. The option to this is set in the configuration file and in general should have a meaningful structure, namely `timed.CLASSREF.SUBFUNCTION.OPTION`. In the case of email fetching this could be: `timed.mail.fetch.interval`.

Processes on operating system level can be set in the configuration file. For example, the JSON file containing all the documents' meta data is executed once after every alternation of the file structure which implies a change of the meta data, but can also be called repeatedly. This is because the system allows an administrator to place a document at the right position in the file structure. They then have to execute the update script to make the changes visible to the web interface. If they forget it, this update can be performed by the *Timed* class calling the script every n minutes. The respective configuration entry for this would be: `timed.commandA` and `timed.commandA.interval`. As these names imply, one can add up to 26 commands (commandA to comandZ) that are going to be executed by the system with system calls after the given interval².

The JSON file update script example is optional as the administrator should not forget to execute every necessary step or could be called with the *StartUp* script every time the system is started.

²This results in 26 command / interval pairs; each single command can call a shell script executing an arbitrary amount of other commands to be executed. That said, 26 is the amount of loops / timed intervals that can be defined.

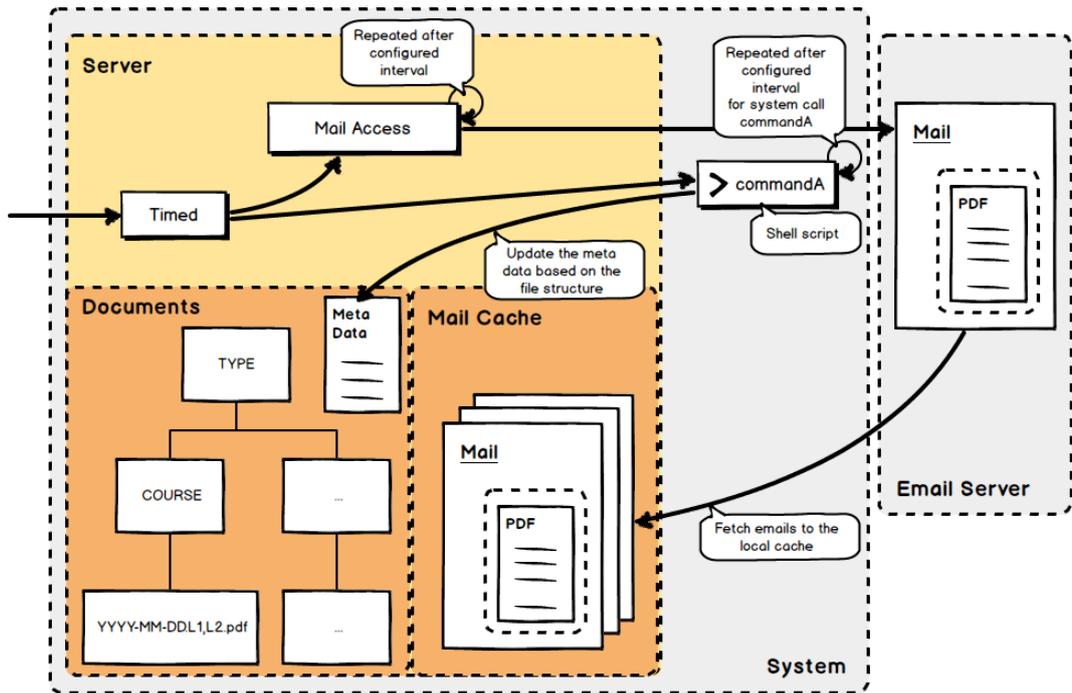


Figure 5.16: System's behavior with timed actions.

In Figure 5.16, one can see the repeated execution of two processes: one is the system fetching mails to fill the cache with them, and the second is the calling of an updated script after a preset time to update the documents' meta data.

5.3.7 Configuration

The configuration file is an XML file that defines key-value pairs; these keys can be accessed with the *Configuration* wrapper class and will return the XML file's values as a *Property* class instance.

Such a list of key-value pairs may consist of the following:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<properties>
...
```

```

<entry key="mail.imap.user">user</entry>
<entry key="mail.imap.password">passwd</entry>
<entry key="mail.imap.host">mail.uni-ulm.de</entry>
<entry key="mail.imap.port">143</entry>
<entry key="mail.imap.folder">INBOX</entry>
<entry key="mail.imap.starttls.enable">true</entry>
<entry key="mail.imap.starttls.required">true</entry>
<entry key="mail.filter">.*FIN.*</entry>
...
<entry key="timed.mail.fetch.interval">10</entry>
...
</properties>

```

Here only options for the mail access are shown. As an example, a username and password, as well as host address and port number are set in the configuration as access information to connect to the email server. It is to be noted that the class that is using these attributes is called `MailAccess` so the key is `mail.SUB1.SUB2.SUB3` where options are grouped logically, depending on their connection to each other.

5.3.8 Target System

As the web interface is created with the GWT the only requirement to run the interface's client is the same for any other GWT project, e.g. Apache Tomcat server[9]. For the web interface's server, a Java runtime is needed but that is also a requirement of the Tomcat server, and therefore covered. A script is run, for meta data updates, written in Ruby[24]. Another script with the same purpose is written as a shell script; so that the system can be deployed on a Windows server with Ruby and Tomcat or on any Unix-like system that possesses `find`, `sed`, and a shell as well as Tomcat. The anticipated server is a Solaris system so the only thing needed is an environment to run GWT projects. The necessary Java libraries the system uses to fetch emails or connect to an LDAP server can be bundled along with the `.war`-file.

In conclusion, all that is needed to run the web interface is a Tomcat server with write permissions to create files on a Unix-like system.

5.4 Console

The console tools allow handlers and administrators to use the shell for the printing of documents. This allows for a minimum of processing power and can be used when the web interface is not running or reachable.

5.4.1 Walkthrough

The two main actions with the console tools are printing and importing. As these tools are not meant to be used by end users, the user can be expected to be more advanced and capable of using the program properly. The printing script has the following options:

```
pprot [OPTIONS] FILTER
```

OPTIONS

```
-l, --list           Lists filenames instead of printing.  
-p, --printer       Specifies the printer to be used.  
-r, --regex         Makes use of regular expressions.  
-i, --interactive   Lets you choose files interactively.
```

FILTER

```
- Any name of lecturer  
- Any name of course  
- Dates  
- Types  
- Available solutions
```

Every filter can match a substring of the searched term, except if the regular expressions option is used, then the regular expression needs to explicitly contain wild cards if searching for substrings.

This help screen shows how documents can be both searched and printed. For a search query, one might for example enter `pprot --list 201 Ana | wc -l` which means that all documents will be listed that apply to the search query, e.g. documents such as “Analysis I, Analysis II” from the year 2011 onwards would be listed, matching any type. As the command is executed in a shell, pipes can be used to interact with the output. In this example, the printed result of the given command would be the amount of documents matching the criteria mentioned.

An other exemplary usage would be the input of `pprot -r -i 20.*-0[34]-.* Smith` into the console. This would let the person using the system print all documents from the lecturer Smith pertaining to lectures held after the year 2000 but only from March and April interactively. To input documents another script can be used:

```
iproto [OPTIONS] FILE
```

OPTIONS

```
-a, --association  
-t, --type  
-c, --course  
-d, --date  
-l, --lecturer  
-s, --solutions
```

This tool is very straight forward. It takes all the meta data given and moves the given file to the right position in the file hierarchy. Afterwards, it executes the update script which updates the web server’s JSON file. Note that if no options are given it will be put to the root of the document tree. There it can be viewed by the administrator for example when editing the meta data with the web interface. Because the file is not in any type or lecture subdirectory, the correlating field will be empty so that the files with missing meta data can be found easily.

5.4.2 Function

The console tools are written in Ruby; to make use of them one has to have a Ruby interpreter, allowing the scripts to be used on any operating system the Ruby interpreter is available for. To use them one has to login onto the machine remotely, e.g. by using `ssh`, a secure shell, that requires the user to enter the login details. This login corresponds to the LDAP username and password that are also checked with the web interface. To access a script the rights to execute it needs to be granted to the user. If the user is part of the handler group they will be able to use the `pprot` command, otherwise they need to be an administrator to import documents. Therefore, the same user can use the web interface and the console interface in the same role, having the same rights.

The print script for ones lists all filenames, splits them into strings, filtered by the substrings given, and outputting the results to the standard output if the list option is given; otherwise it passes the filenames to an `lpr` command with possible additional printer information.

The import script matches the given parameters and moves the given file to the right position in the file hierarchy.

5.5 File Browser

As it was shown in the File Structure section, the filenames indicate their content, and are structured such that they can be found by searching for type first, then by course, and finally, by date and lecturer. To find a specific document using the file browser, one can navigate to the right directory and there search for a specific file if wanted. To print files one can open the file with a PDF viewer and print it from there. To import a file, it can be renamed and put into the right directory.

5.6 Backup

The static part of the system, namely the web server and console tools, might be altered occasionally during the lifetime of the system, however this is not expected to happen very often. Not much data have to be saved if one wanted to back them up; there is the `.war` file for the web interface and the `pprot` and `iprot` tool for the console interface.

5.6 Backup

The saved documents are all PDF files in a hierarchical structure form. They can be saved by using backup strategies that have been proved successful for file systems in general. Principles such as the Grandfather-Father-Son principle can be used to save the documents, whereby with each increment only new documents need to be added if no old document names have been modified, which should be the case if the import was done right or was corrected before the backup started. This is the expected standard case, so backups can be done efficiently in regards to memory.

6 Conclusion

The conclusion discusses possible features which can be added to the system for optimization purposes and put the implemented system into context with the idea and task tackled in the thesis.

6.1 Requirement Alignment

In this section the requirements from section 3.4 are discussed in regards to their successful realization.

Classification	Achieved	Method
Data Storage	+	PDF format has been chosen. By doing so, the format is flexible enough to handle vector graphics, text, as well as embedded graphics. It is also platform independent, meaning that there are PDF reader for almost any device. If used properly it also saves the data with little overhead.
Data Structure	+	The file tree can be transferred and copied completely while the structure is kept.
	+	The file tree and the files themselves are named after the pattern described in section 5.1.2, thus finding single documents is comprehensible by administrators and handlers.
	+	As the documents do not rely on a database but are instead accessed on file system level, storage does not require a proprietary format.

6 Conclusion

Classification	Achieved	Method
Document Style	o	Some student associations, such as the computer science department's student association of Ulm University already has document templates for providers, thus new templates did not need to be created.
User Access	+	The system can be accessed via <i>ssh</i> , logging into a PC within the network that has access to the documents, and the web interface can be used to access the documents.
	o	LDAP accounts, that are used within the university network of Ulm University can be used in theory, though the functionality has not been tested, yet.
	-	The LDAP accounts cannot be managed from within the implemented system. Though this is due to the nature of the global accounts. Ghost accounts can be prevented by using these global accounts, but their management has to be done from outside the system.
User Interface	+	Three interfaces were implemented, which allow interaction with the system.
	+	The web interface is user friendly.
	o	The console interface is efficient when used properly. <i>iprot</i> has yet to be implemented, as a console import tool was not demanded by the student associations.
	+	The system is failsafe, meaning that if the web server crashes, an administrator can still use <i>ssh</i> to connect to the system.
Printing	+	Documents can be printed using the <i>pprot</i> tool or using the print function of a PDF viewer.
	o	In order to save paper, documents can be piped through command line tools so that documents are printed on both sides. The other interfaces do not offer this functionality, yet. This could be implemented in the future, though.

Maintenance	+	Documents can be imported and managed as described in section 5.1.2.
	o	Documents can always be printed concurrently if the file structure is not modified. However, if two administrators use the web interface to manage documents, concurrency might be an issue. As the number of administrators will be low, modification of documents even rare, this is not expected to be a problem.
	+	The command line tool can be accessed via <i>ssh</i> , the web interface over the internet and a remote login via <i>Remote Access</i> is also possible to interact remotely with the system.

6.2 Outlook

The minimum requirements for the implemented system are a server that runs Ruby (if the console interface is wanted or the server runs Windows), otherwise a server running a Unix-like operating system with the support for GWT projects, such as TomCat, is needed. This assumes that the main i.e. web interface is going to be used, though through the modularity of the system the interfaces can be chosen individually. With that said one can see that the implementation is fairly portable. Furthermore, while it can be used to provide the documents from student associations to students, it could also be used to print *lecture notes* or *exercise sheets*.

Future work might focus, for instance, on the creation of collections. End users could create a collection and get an email ticket for this request which they have to confirm, after which the documents could be printed directly. This would improve the work flow as handlers would not have to acknowledge each request, however this might increase abusive behavior towards the system.

To counter abusive behavior, fees could be charged for every sheet printed. This way, end user could still create collections that would be printed by the handler, but students would presumably print less unnecessary documents.

Thirdly, the import method of documents could be improved by providing a document creation page. There, meta data could be input directly by the provider. Additionally the provider could use a text box on that page to input the document's content directly. This content could be used in combination with a LaTeX template to generate a standardized document in PDF format. By providing a preview function the provider could submit their

6 Conclusion

document after they are satisfied with it and submit it to an administrator, thus lowering the threshold involved in creating a document, and the number of documents imported could be improved.

6.3 Discussion

The goal was to design a system that can tolerate the failure of different access methods, as well as to provide an exemplary implementation of such a system that is document oriented. The implementation has three interfaces, all of which are accessible in different ways, such as via file browser, *ssh* and the web interface. So if the web browser fails, one could still use *ssh* to connect to the server or access the file system directly in order to access the documents. Instead of using *ssh*, one can also use the Remote Desktop software on Windows servers and execute the shell script from there. The failure of the whole system making it impossible to operate with it or the failure of the file system was not taken into account as the server can be viewed as failsafe in regard to this matter as this problem was not part of the thesis's problem set. Therefore, the exemplary implementation fulfills the criteria of the thesis's requirements and the design of the system realizing the import, maintenance, printing, deletion, and user login via three ways of accessing the system meet with the anticipated goals of providing and implementing a document maintenance system with multiple access strategy.

Bibliography

- [1] APPLE, C. ; ROSSEN, K.: *X.500 Implementations Catalog-96*. United States, 1997
- [2] BERNSTEIN, Philip A. ; GOODMAN, Nathan: Concurrency Control in Distributed Database Systems. In: *ACM Comput. Surv.* 13 (1981), June, Nr. 2, 185–221. <http://dx.doi.org/10.1145/356842.356846>. – DOI 10.1145/356842.356846. – ISSN 0360–0300
- [3] BURCKHARDT, Sebastian ; LEIJEN, Daan ; FÄHNDRICH, Manuel ; SAGIV, Mooly: Eventually consistent transactions. In: *Proceedings of the 21st European conference on Programming Languages and Systems*. Berlin, Heidelberg : Springer-Verlag, 2012 (ESOP'12). – ISBN 978–3–642–28868–5, 67–86
- [4] CORPORATION, International Business M.: *IBM Database 2 (Homepage)*. <http://www-01.ibm.com/software/data/db2>. Version: May 2013
- [5] CORPORATION, Microsoft: *Active Directory Collection*. [http://technet.microsoft.com/en-us/library/cc780036\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc780036(ws.10).aspx). Version: May 2013
- [6] CORPORATION, Oracle: *MySQL (Homepage)*. <http://www.mysql.com>. Version: May 2013
- [7] CRISPIN, M.: *Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension*. United States, 2003
- [8] FOUNDATION, Apache S.: *Apache CouchDB Project*. <http://couchdb.apache.org>. Version: May 2013
- [9] FOUNDATION, Apache S.: *Apache Tomcat (Homepage)*. <http://tomcat.apache.org>. Version: May 2013
- [10] GRAY, Jim: Readings in database systems. Version: 1988. <http://dl.acm.org/citation.cfm?id=48751.48761>. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1988. – ISBN 0–934613–65–6, Kapitel The transaction concept: virtues and limitations, 140–150

Bibliography

- [11] GROFF, James ; WEINBERG, Paul: *SQL The Complete Reference, 3rd Edition*. 3. New York, NY, USA : McGraw-Hill, Inc., 2010. – ISBN 0071592555, 9780071592550
- [12] HOWES, Tim A. ; GOOD, Gordon ; SMITH, Mark: *Understanding and Deploying LDAP Directory Services*. 1st. Alpel Publishing, 1998. – ISBN 1578700701
- [13] INC., Google: *Google Web Toolkit (Homepage)*. <https://developers.google.com/web-toolkit>. Version: May 2013
- [14] *Data elements and interchange formats - Information interchange - Representation of dates and times*. 1988
- [15] KIZ, Universität U.: *Kommunikations- und Informationszentrums Universität Ulm*. <http://www.uni-ulm.de/einrichtungen/kiz.html>. Version: May 2013
- [16] LITH, Adam ; MATTSSON, Jakob: *Investigating storage solutions for large data - A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data*, Diplomarbeit, 2010
- [17] LTD., Vaadin: *Vaadin (Homepage)*. <https://vaadin.com>. Version: May 2013
- [18] MILLER, Frederic P. ; VANDOME, Agnes F. ; MCBREWSTER, John: *CUPS: Printer (computing), Unix- like, Operating system, Server (computing), Client (computing), Spooling, Internet Printing Protocol, Command- line interface, Line Printer Daemon protocol*. Alpha Press, 2009. – ISBN 6130225768, 9786130225766
- [19] NEWMAN, C.: *IMAP URL Scheme*. United States, 1997
- [20] SOFTWARE, Isomorphic: *SmartGWT (Homepage)*. <http://smartgwt.com>. Version: May 2013
- [21] *Message handling system and service overview*. 1999
- [22] *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*. 2012
- [23] YEONG, W. ; HOWES, T. ; KILLE, S.: *X.500 Lightweight Directory Access Protocol. RFC 1487 (Historic)*. <http://www.ietf.org/rfc/rfc1487.txt>. Version: July 1993 (Request for Comments). – Obsoleted by RFCs 1777, 3494
- [24] YUKIHIRO MATSUMOTO, et a.: *Ruby (Homepage)*. <http://www.ruby-lang.org>. Version: May 2013

Name: Leo Hnatek

Matrikelnummer: 643371

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den

Leo Hnatek