Alexei Scerbakov, BSc

Using cloud services in a modern Learning Management System

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme: Software Development and Business Management

submitted to
Graz University of Technology

Supervisor
Assoc.Prof. PhD Martin Ebner

Institute of Information Systems and Computer Media
Head: Prof. PhD Frank Kappe

Faculty of Computer Science and Biomedical Engineering

Graz, September 2015

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present diploma thesis.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Diplomarbeit identisch.

Graz, am _____ _____

Datum Unterschrift

Acknowledgements

I want to thank my family and my girlfriend for all the support they provided me with over the course of my university studies. Without them it would not have been possible for me to achieve this goal in my life!

Also very big thanks to Martin Ebner, for making this thesis possible and being a good boss and also a good friend.

Thanks to all my colleagues in the department of social learning who became friends of mine over the past years and always are there to help me or teach me new things.

A big thank you to all of my friends inside and outside of university for making my life as a student an experience I will never forget.

Thank you from the bottom of my heart!

Kurzfassung

So genannte Cloud Services bieten eine Möglichkeit, Daten der Nutzer im Netz zu speichern und zu verarbeiten. Sie sind in den letzten Jahren erschienen und werden vor allem genutzt, um verschiedene Arten von Daten zu verarbeiten. Die Funktionalität von modernen Lernmanagement-Systemen kann mit ihrer Hilfe wesentlich erleichtert werden.
Diese Arbeit zielt darauf ab, einen technischen Kontext zu bieten und analysiert die Vorteile der Integration von Cloud Services in ein modernes Lernmanagement-System. Typische Szenarien der Nutzung von Cloud Services in einem Lernmanagement-System werden näher ausgeführt und erklärt. Eine Anzahl beliebter Cloud Services wird spezifiziert und nach ihrem Nutzen für E-Learning-Aufgaben evaluiert. Der Hauptbeitrag dieser Arbeit besteht aus einer Reihe von Softwarekomponenten, welche für die praktische Nutzung im Lernmanagement-System TeachCenter implementiert wurden. Die Software-Architektur und Fragmente vom Quellcode dieser Komponenten werden bereitgestellt. Die Resultate dieser Arbeit, mögliche weitere Entwicklungen und bestehende Probleme werden am Ende der Arbeit diskutiert. Voller Erfolg und Benutzer-Akzeptanz wurden in allen Szenarien, die Dropbox und Google Drive verwenden, erreicht. Der Nutzen anderer Cloud Services kann zurzeit als Teilerfolg mit Verbesserungspotential betrachtet werden.

Abstract

So called cloud services provide a possibility to store and handle user data on the web (cloud). They emerged in the last few years and became popular for different kinds of data processing. Functionality of modern Learning Management Systems can be essentially amended by them. This master thesis aims to provide a technical context and analyses benefits of embedding cloud services into a modern Learning Management System. Typical scenarios for using cloud services in a Learning Management System are elaborated on and described in detail. A number of cloud services are specified and their benefits for e-learning tasks evaluated. The main contribution of this thesis is the implementation of a set of software components for practical use in a special Learning Management System called TeachCenter. Software architecture and snippets of sources of such components are provided. Results of this thesis, possible further developments and existing problems are discussed at the end of the thesis. Full success and user acceptance were achieved in all scenarios using Dropbox and Google Drive. The usefulness of other cloud services can be currently characterized as partially successful and has room for improvement.

Contents

List of Figures

1 Introduction

Todays university life is unimaginable without Learning Management Systems (LMS). Most universities are using LMS to facilitate the teaching and learning process (Lonn and Teasley, 2009). The tasks an LMS carries out can be essentially amended by cloud services, a technology that came up in the last few years and is increasingly gaining popularity.

(Some fragments of this thesis were taken from (A. Scerbakov, Ebner, and N. Scerbakov, 2015), which was released in the Journal of Computing and Information Technology in March, 2015.)

1.1 Technical context

Basic terminology that is used throughout the whole thesis is defined and explained in more detail in this section.

1.1.1 Internet

The Internet is a huge computer network connecting several million computers world-wide. It brings together multiple hardware and operating system platforms from dozens of different manufacturers. Communication between these different platforms is possible through a mutual way of exchanging data - TCP/IP, which is an acronym for Transmission Control Protocol/Internet Protocol.

TCP/IP specifies the data transport layer of communication, which treats the data transaction between two computers as a stream, called a transport data unit. There is a number of so-called Internet data service protocols

that are built on top of TCP/IP, each of these protocols is designed for some particular purpose - to support distributed collaborative hypermedia systems (HTTP), Internet news systems (News), file transfer systems (FTP), and so on.

The Hypertext Transfer Protocol (HTTP) is the most popular data service protocol. It is designed to support communication between clients and servers. Clients send requests to a server (HTTP request). The server responds by sending back relevant data to the clients (HTTP response). Some requests can also demand more complex server functionality such as storing data into a database, querying the database, removing data from the server, and so on.

The World Wide Web (WWW or simply Web) is one of the Internet services based on HTTP. It can be seen as a globally distributed collection of multimedia documents written in a markup language called HyperText Markup Language (HTML). From another perspective, it can be said that the Web is a huge number of HTTP servers providing HTML pages to WWW clients or browsers. The Uniform Resource Locator (URL) is one of the basic Web concepts. It may be seen as a unique ID of any resource available via the Internet. HTML allows to embed URLs into HTML documents to form a global net of interlinked Internet documents.

HTML is the WWW de facto standard for describing how information is structured and should be visualized. It allows different vendors to develop WWW browsers that are fully compatible, and visualize HTML documents almost identically on different hardware and software platforms.

A Web-based application is any software package that essentially uses HTTP for its functionality. Web-based applications often run in the context of a Web browser as opposed to ordinary applications that run in the context of a particular operating system. Web-based applications are also known as Web apps.

Since the WWW is based on the client-server architecture, there are two main methods for developing Web apps:

1. Server-side programming
2. Client-side programming

1.1.2 Server-side Programming

All modern HTTP servers support the Common Gateway Interface (CGI). The CGI provides the mechanism of executing arbitrary programs by the server, generates relevant data, and returns the data to a client as an HTTP response. The reason for running such add-ons is to return dynamically generated data directly to the client in the form of a dynamic HTML document. CGI applications may communicate to a file system and other software packages installed on the server. For example, CGI applications may provide Internet access (for instance an interface) to a local database, expert system, and so on. Thus, the CGI interface provides a powerful mechanism for building Web apps.

1.1.3 Client-side Programming

Internet Browsers are much more complex software systems than just a HTML visualization application. The architecture of a modern Internet browsers includes a number of so-called virtual machines which are able to interpret special imperative code embedded into an HTML text, known as scripts. The most popular client-side scripting language is called JavaScript. An important JavaScript concept is called Dynamic HTML. JavaScript dynamically modifies the appearance of a host HTML document by changing its Document Object Model. JavaScript and Dynamic HTML (DHTML) provide a new architecture for developing Internet-based information systems, called asynchronous JavaScript and XML (AJAX). Client-side scripts may send HTTP requests to a server to fetch data and dynamically alter the appearance of the current HTML document.

1.1.4 Learning Management System

Technologies, especially web technologies, are shaping the way of teaching and learning today (Saeed, Y. Yang, and Sinnappan, 2009). During the last years a dramatic shift happened from teaching without Internet technologies to teaching with a ubiquitously available Internet.

Technology changes rapidly, overwhelming educators and learners alike. At the turn of the millennium, Learning Management Systems were introduced aiming to assist lecturers in their main activities (Maurer, 1996). With an enormous speed lecture after lecture went digital and today almost every university uses its own LMS containing mainly lecture content (presentations, handouts), administrative notes for the lecture, and further tools (e-assessment, uploading tools, calendar, and so on) (Nishantha et al., 2009).

Traditionally LMS provide the following functionality:

- Distribution of training materials: LMS can be seen as a structured repository of educational materials prepared by teachers and available for students. Therefore, the question of effective tools for uploading new materials, modifying existing ones, notifying users about modifications and downloading materials is of utmost importance.
- Collaboration with different aspects of training: In many respects, educational materials are not only a result of the teachers' work, but a result of a collaboration between teachers and students during the classes. As a consequence, students must have the possibility to comment on materials, ask questions in context of particular fragments and share their experience with other students.
- Discussion about the content and flow of training: A well-known functionality of an LMS. All online courses are now provided with discussion forums, chats, announcement boards and other communication tools.
- Upload and evaluation of student assignments: One of the commonly accepted ways of checking student's knowledge in context of e-learning is uploading student assignments and evaluating such assignments by teachers or tutors. Uploading can be done using different scenarios. Uploading to protected areas where only a restricted user group has access to or uploading to shared repositories. In addition, uploads can be done on behalf of a single user or users can be requested to work in groups and prepare collaborative uploads.

E-learning History
After the huge LMS hype, the Web 2.0 movement (O'Reilly, 2006), described as the area where users tend to be more active in the World Wide Web

through participation in Wikis, Weblogs or even Social Media, hit the field of education under the name of e-learning 2.0 (Downes, 2005). Since then, the number of possibilities of how the web enhances teaching and learning has exploded (Ebner, 2007). Wikis (Augar, Raitman, and Zhou, 2004) (Raitman, Augar, and Zhou, 2005), Weblogs (Farmer and Bartlett-Bragg, 2005), Podcasts (Evans, 2008) as well as the use of Social Media (Ebner, 2013) for education attracted many lecturers as well as learners and changed the way how web technologies are used in everyday life. Ever since, collaboration, cooperation, and communication between students-teachers as well as students-students can be done in an entirely new way (Schaffert and Ebner, 2010). That is by including the use of semantic technologies (Klamma et al., 2007). For example, applications such as Etherpad or Google Drive allow for a timely interaction with each other. In addition mobile technologies open the opportunity for participation from nearly anywhere (Ally, Grimus, and Ebner, 2014).

1.1.5 TeachCenter

History:
Graz University of Technology has a long history and profound experience in technology enhanced learning. First experiments were carried out, almost simultaneously with general recognition of WWW potential for university education (Dietinger and Maurer, 2014), to enhance online education by developing special authoring tools (Maurer and N. Scerbakov, 1996) and information systems with a special focus on e-learning needs (Andrews, Kappe, and Maurer, 1996). Such experiments provided an essential experience and know-how for developing a large Content Management System called "Hyper-G" (Andrews, Kappe, Maurer, and Schmaranz, 1994) as well as the Learning Management System "WBT-Master" (Helic, Maurer, and N. Scerbakov, 2004). A logical follow up development is the so-called TeachCenter (TC), which serves as a heavily used Learning Management System at Graz University of Technology since 2007.

After a pilot phase, testing different use cases (Ebner, N. Scerbakov, and Maurer, 2006) (Ebner and Waldner, 2008), it became a university wide service and is used today by more than 15,000 students and 2,000 lecturers.

Between 2008 and 2010 a number of external Web services were integrated by using their Application Programming Interfaces, following the approach of Edupunk (Ebner, Holzinger, et al., 2011). For example plagiarism control, Facebook functionality, different programming languages compilers, database management system and so on. Today, it differs significantly from other popular LMS in a number of following aspects.

Architecture:
TeachCenter is built on the base of an architecture known as AJAX. As a consequence, data processing is performed on the client side by means of JavaScript and dynamic HTML. As a result, the system demonstrates good performance under heavy load. The system can be seen as a number of HTML5 files with the inclusion of Cascading Style Sheets (CSS) and JavaScript. The JavaScript files communicate to the server using the JavaScript Object Notation (JSON) format. Server functionality is essentially simplified and implemented as a number of Java servlets.

In this architecture Java servlets do not handle any data processing, but retrieve data from a database and wrap it in accordance with JSON standards. It should be especially noted that the TeachCenter works with three hundred to five hundred users simultaneously and the response time for any action is under two seconds. This creates comfortable environment for end users (Ramsay, Barbesi, and Preece, 1998).

Another well-known advantage of AJAX is the utilization of modern user interface elements. For example, most editing in TeachCenter is done using user interface features such as drag and drop, event-based functionality and so on. The well-known disadvantage of AJAX, security problems in case of script manipulation, is partially mitigated in TeachCenter by the usage of so-called tokens. Users get a unique token the moment they successfully log in to the system, this token is stored on server and client-side. When the user performs an action, both tokens are checked for equality. As a result, some actions that are not supposed to be done by the user are easily identifiable and can be prevented.

Infrastructure:
Normally, LMS offer courses consisting of HTML pages that need to be authored by the teacher. A course can be seen as a combination of HTML

pages and authoring them is one of the main tasks to be performed by teachers as they work with an LMS.

A few years of experience at the Graz University of Technology have shown that the need for easy authoring is the main obstacle for the acceptance of a system used by teachers. In order to improve acceptance by teachers, the TeachCenter was built using a modular concept. Courses consist of a number of predefined areas and additional tools. For example, there is a special area for doing announcements for students, a special area for describing administrative issues such as a course calendar, objectives of a course and so on. Such areas demand a minimum amount of authoring from the teacher-side. Teachers can easily add new announcements and modify or delete existing ones. As an example of additional tools, there are student projects, where students upload any files teachers demand, and the quiz tool, where teachers define questions to users, which have to be answered within the stipulated time. The areas and modules can be easily switched on or off to create a course as a collection of areas and tools that satisfy the teachers' needs.

One of the most important tasks an LMS has to fulfil is file management and distribution. Teachers must have extremely simple and powerful tools to upload files to the server and students must have equally convenient tools to download files to their computers. The TeachCenter provides a special area responsible for such file management. It is called course library.

1.1.6 Groovy

Groovy is a programming language for the Java Virtual Machine. It first appeared in 2003 and is used as the server-side language of choice for the add-ons for TC described in this thesis. It was chosen because it compiles straight to Java bytecode and has the same syntax as Java so it can be used by Java developers. Compared with Java, Groovy has a simpler syntax which makes the development process faster. Groovy scripts can be easily integrated in a working system and also are easy to remove in case they are not needed any more. This was another reason for choosing Groovy for add-on implementation in TC.

1.1.7 Internet services and cloud services

Online communication and collaboration can only happen if the data is also available online for each participant. Therefore, the number of so-called cloud-based services and appropriate devices, and the idea to grant users access to hosted centralized data centers with a variety of clients is growing (Hao et al., 2009).

The National Institute of Standards and Technology (NIST) gives a definition of cloud computing that is based on five essential characteristics, three service models, and four deployment models (Mell and Grance, 2011).

Essential Characteristics:

1. *On-demand self-service*: The user can use the offered resources when needed, without obligation to contact the cloud service provider.
2. *Broad network access*: The service is reachable through a network, for example the World Wide Web.
3. *Resource pooling*: The resources offered by the cloud service provider are dynamically distributed among the users.
4. *Rapid elasticity*: Resources of the service are scaled down or up dynamically, giving the user the impression of unlimited resources.
5. *Measured service*: The resources of a cloud service are managed and automatically limited to ensure a smooth experience for privileged users.

Service Models:

1. *Software as a Service (SaaS)*: The service offered is accessible through a special interface by the user. The user has no access and does not manage the hardware or the software running on the service providers' servers. In this thesis mostly cloud services offering this service model will be used.
2. *Platform as a Service (PaaS)*: The user has the ability to upload custom applications to the cloud service server and run them. The user therefore has access over a certain part of the software stack but not the hardware stack.

3. *Infrastructure as a Service (IaaS)*: In this service model the user has control over the most part of the software stack out of the three service models. For example including the operating system, storage or even certain network components.

Deployment Models:

1. *Private cloud*: The access to the cloud service is restricted to a special user group, such as the employees of a single company. The infrastructure can also be hosted by the consuming company.
2. *Cloud community*: Similar to the private cloud, access to this model is restricted, with the difference that the user group in this model is determined by a shared interest. An example would be a group of researchers from all over the world working on the same project.
3. *Public cloud*: This deployment model offers access to the cloud service for the public and is the one that is used by all cloud services in this thesis.
4. *Hybrid cloud*: A combination of two or more cloud infrastructures. They remain independent but work together.

Cloud sharing services such as Dropbox (Drago et al., 2012) follow a centralized approach – each single user's data are stored in a shared folder on the web, where a group of users has access as well as the rights to edit and change the data. The fact that after each edit all changes are automatically pushed to all group members helps making such a service valuable and useable.

Today almost all large software companies offer such a service – Apple's iCloud, Microsoft's SharePoint, Google's Google Drive, and Dropbox (Hu, T. Yang, and Matthews, 2010) are just the most recognized ones.

It is easily imaginable that lecturers as well as learners are asking for integration of such cloud-based services into the LMS (Stantchev et al., 2014).

Cloud services provide a solid basis for doing distributed calculation and collaborative development. A file uploaded to a WEB repository may be accessed via a number of interface solutions, and can be processed by a number of applications.

Moreover, developing a Web application based on a cloud service normally requires essentially less efforts since the cloud platform:

- Utilizes already advanced mechanisms for accessing and uploading the cloud content.
- Provides advanced data processing tools for developers.
- Syndicates a large community of developers that form virtually one developing team.

1.1.8 Cron

Cron is a software on the Linux operating system, to periodically run commands at certain, user chosen, times. It is used for a number of cloud service add-ons in TeachCenter, to make synchronization between the service and TC possible. For example the synchronization of a Dropbox folder with the according TC folder can be carried out once a day with a so-called ”Cron Job”. When a folder in any TC add-on is marked for synchronization, the folders owner can chose a time schedule which is used to keep the folder synchronized with the according cloud service folder.

1.2 Goals

The goals of this master’s thesis are as follows:

- To investigate the role and possible usage of cloud services in a modern LMS.
- To develop an architecture of packages integrated in the LMS and reusing functionality of cloud services via a respectful Application programming interface (API).
- To develop a user interface solution providing seamless integration of cloud services in the natural functionality of a modern LMS.
- To develop a software based on modern software engineering principals, practically integrating functionality of cloud services into a particular LMS.

- To investigate possible advantages and possible disadvantages of using the developed software by a large number of users in real university environment.

1.3 Research Question

Inclusion of cloud services into a heavy used LMS to facilitate certain common use cases and observe the user acceptance of this new cloud service based solutions.

1.4 Overview

The thesis starts with an overview of popular cloud services which were considered to be integrated into TC (see section 2). The focus will be on the advantages these cloud services have to offer to ease certain tasks or add new functionality to TC. It will be analysed if the cloud services offer a way to integrate them into an external system, hence have an API.

Furthermore, the implementations which were done based on this thesis will be discussed and analysed in detail. In the end the results that were gathered concerning the cloud services that were offered in the TeachCenter will be reviewed.

2 Overview of chosen cloud services

In this chapter a look will be taken on chosen cloud services that were used in this thesis to discuss the possibilities they offer to either add a new functionality to TeachCenter or ease the usage of an existing functionality. Each cloud service will be discussed separately to be able to formulate expectations and later analyse whether they have been fulfilled. There are a lot of different cloud services available nowadays under different user agreements (Freeware, Shareware, commercial and local implementations). All this applications provide similar functionality. Overview of all these services is not possible within one thesis of limited size. Moreover overview of all services is not needed because some of them simply can not be used in public e-learning because of license agreements. That is why only the more common solutions according to (Drago et al., 2012):

- Dropbox
- Google Drive
- Microsoft OneDrive

as well as two other cloud services, namely Etherpad and Firefox Pocket were selected for further investigation and implementation.

2.1 Dropbox

DropBox is one of the most popular cloud services nowadays (Drago et al., 2012). This cloud sharing service can be seen as a structured remote repository of files. The repository may be accessed and edited via the Internet. Each user obtains a limited space on a remote server, and is

free to create an own repository within this space. Dropbox utilizes a rather conventional ”files in folders” data structuring paradigm. It does not provide online data editing tools, but offers rich file replacing functionality including versioning and roll back operations. Dropbox would be a rather conventional internet service providing file hosting, but it goes much further by providing:

- a number of Dropbox clients that allow to see the content of a Dropbox as a part of the local file system on any operating system, as can be seen in figure 2.1. For example, on the Windows operating system the content of a user's Dropbox is visualized as a special folder and can be accessed as well as edited by means of the Microsoft Explorer or any other file management utility.
- a synchronization procedure that allows to automatically download and upload new files, to keep the content of a remote Dropbox and a part of the local file system identical. As soon as a Dropbox client is started, the application automatically requests the cloud services server on modifications of the Dropbox content since previous operations, and downloads all the modifications automatically. In addition all the modifications done locally on selected files are automatically mirrored to the server.
- a rather rich developer API that provides programming access to the whole functionality of Dropbox. A file may be uploaded to the Dropbox cloud service server by any application that may generate a ”POST” HTTP request to an Internet server. Similarly, content of a Dropbox may be read by an application by means of a ”GET” HTTP request. Since the topic of this thesis deals with integration of popular cloud services into an LMS, more attention has to be put to the API of such services. Dropbox supports an API via Representational State Transfer (REST) Web service, all the server functionality may be accessed via HTTP requests having different end points, structure and content.

The Dropbox REST API allows developers to access and integrate the functionality of Dropbox into other applications such as LMS in this particular case. Dropbox API methods include looking up the folder structure of an account, downloading files, retrieving user information, and so on. One

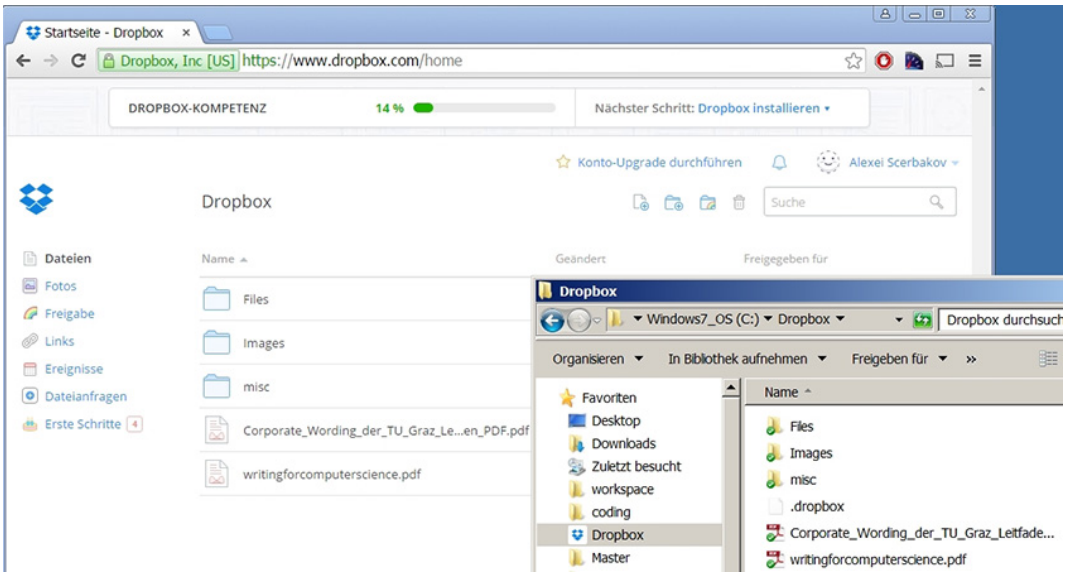

Figure 2.1: Local and remote dropbox content.

of the most important components of any API is an authentication mechanism. From one point of view this mechanism must secure and protect the application from leaking sensitive data, such as user credentials or user specific data(structure of folders), from another point of view the security considerations should not affect the functionality of the systems from a user's point of view. The authentication mechanism as well as particular functions of the Dropbox API are described in more detail later in this thesis (see chapter 4).

Cloud services make data and data processing mechanisms available anytime and anywhere. Dropbox fully supports this. Data placed onto a Dropbox account are available anytime, anywhere and on any platform. Another basic advantage of cloud services is a possible collaboration of users during such remote data processing. Unfortunately, Dropbox provides rather primitive data sharing functionality. Therefore, certain Dropbox folders may be shared with other users, and files may be uploaded, accessed and deleted by different users concurrently.

In terms of the NIST cloud computing definition (Mell and Grance, 2011) Dropbox is a software as a service in a public cloud.

2.2 Etherpad

Etherpad is more of an online editor of remote textual documents than a remote repository of files. Etherpad, compared to the cloud services listed before, has a primitive data structuring component. The system simply creates a so-called ”pad” having a unique id. Each pad is associated with exactly one textual document. The documents may be created online on Etherpad or may be uploaded from a local computer. Editing is supposed to be done by a number of users in a team using online tools offered by Etherpad, as can be seen in figure 2.2. Finally the document can be downloaded from Etherpad as a Portable Document Format (PDF) file or a plain text file.

The developer of this service paid special attention to the process of concurrent editing of textual documents. The system supports a time line of changes made on the document, colourization of updates done by different users, as well as a versioning and roll back functionality. Documents can be shared using different types of pads, a particular pad can be ”public” or ”user group” specific. Any user with a public pads Uniform Resource Identifier (URI) may access and edit the document. If a pad is defined with the attribute ”user group”, only members of this previously defined user group may access and edit the document.

The Etherpad REST API allows developers to access and re-use the functionality of Etherpad. API methods include creating a pad, setting access rights, uploading files, converting resultant files into PDF format, and so on. One of the most important components of any API is an authentication mechanism. From one point of view this mechanism must secure and protect the application from leaking sensitive data, such as user credentials or user specific data, from another point of view the security considerations should not affect the functionality of the systems from a user’s point of view. The authentication mechanism as well as particular functions of the Etherpad API are described in more detail later in this thesis (see chapter 4).

It should be especially noted that while Dropbox is a proprietary server that hosts tremendous amount of user accounts and provides just online tools for such users, Etherpad is an open source software package that can be installed on any Internet server. An Etherpad instance was hosted at the

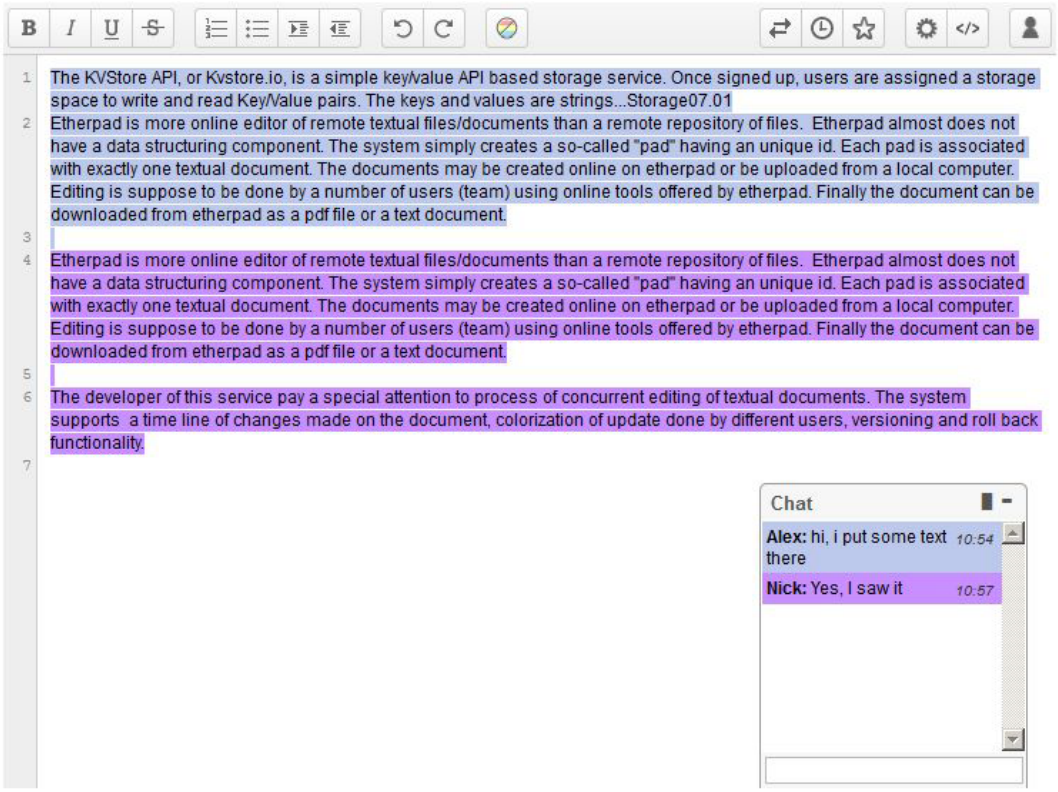

Figure 2.2: Etherpad collaborative authoring.

Graz University of Technology to be used by TeachCenter LMS accessible at (*Etherpad of Graz University of Technology* 2015).

2.3 Google Drive

API wise Google Drive is the most advanced cloud service nowadays. This cloud service combines the functionality of Dropbox and Etherpad, and even adds some functionality that was unavailable in the two previously discussed services. From one perspective, Google Drive can be seen as a cloud repository of files. Similarly to Dropbox, the repository may be accessed and edited via the Internet. The Google Drive account can be seen as a space on a remote server, where an own repository of files may be created. Furthermore Google Drive supports a more complex data structuring model than for example Dropbox - all data objects on the account space are called items and have a unique id. Items may have parents that can be other items. As a result items are structured into a net where items may have multiple parents and children. It is also interesting to note that items may be single files, folders and even applications such as Picasa Picture Albums, Calendars, and so on.

Google Drive provides very powerful data editing tools, as can be seen in figure 2.3. The cloud service also provides a comprehensive editor for text documents. In addition the application supports versioning and roll back mechanisms. Google Drive offers new Google specific Multipurpose Internet Mail Extensions (MIME) types like Google Document, Google Excel and Google Presentation, and allows conversion of this formats into PDF and Microsoft Office files. Google drive provides two important groups of functionality. First it makes data and data processing mechanisms available anytime anywhere and on almost any platform. In addition another basic advantage of Google Drive is a possible collaboration of users during such remote data processing. Google Drive provides rather comprehensive data sharing functionality based on user groups and user roles associated with each item. User roles are defined by an owner of an item, user groups are formed by means of a special invitation mechanism. Users may communicate as they edit documents via a system of comments, which can be seen

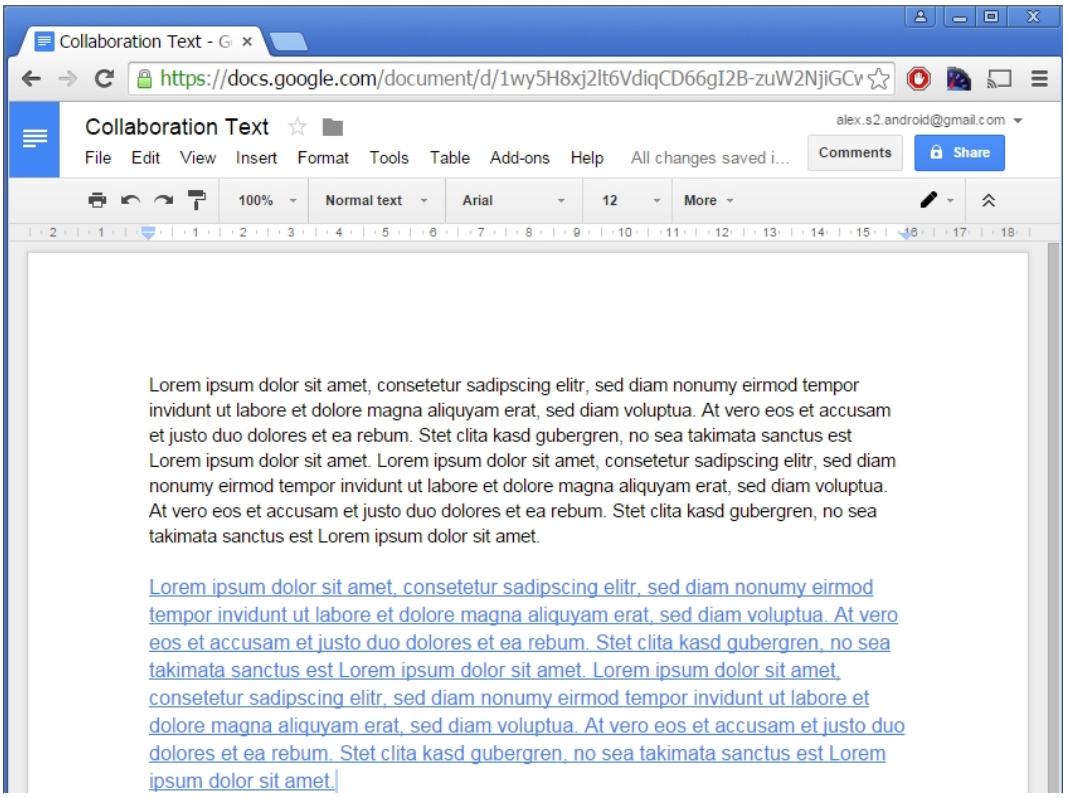

Figure 2.3: Remote editing of a Google Documents file.

as an asynchronous message exchange mechanism attached to a particular data item.

Similar to Dropbox, Google Drive provides local clients, toolboxes and an API that considerably improves user experience and acceptance of Google services. As a result, Google offers to users:

- a number of Google Drive clients that allow to see a Google Drive content as a part of the local file system on any operating system.
- a synchronization procedure that allows not only automatically download as well as upload of new, but also automatically download new versions of existing files as they are edited online.
- the Google Drive developer REST API, which provides programmable access to the whole functionality of Google Drive. This includes creat-

ing items, uploading items, conversion formats, reading folder content, setting user roles, inviting users to edit items, and so on.

The authentication mechanism and particular functions of Google Drive API are described later in this thesis (see chapter 4).

2.4 Microsoft OneDrive

Microsoft (MS) OneDrive combines rich data structuring and data processing facilities. OneDrive provides users with a remote repository of files that are structured similar to Google Drive - as a net where data items has unique IDs, and may have a number of parents and children. Additionally items may be associated with tags (key words) that can be used to search items in the repository. Similar to Google Drive, items may be single files, folders and even applications such as OneNote, Calendar, and so on. OneDrive is mainly oriented towards file formats supported by the popular Microsoft Office applications. For each of such Office file formats, the system offers very powerful editing tools having almost the same graphical user interface as a local Microsoft Office editor, as can be seen in figure 2.4. The application supports history of modifications, versioning and roll back functionality.

OneDrive is very similar to Google Drive in a lot of aspects. In particular it provides almost the same functionality. It makes data and data processing mechanisms available anytime anywhere. OneDrive provides simple, but yet powerful data sharing functionality based on generating item share links on a request. Such edit links are generated by an owner of the item, links are distributed by means of a special messaging mechanism. Users may communicate as they edit documents via a system of comments.

Similar to Dropbox and Google Drive, OneDrive provides local clients, toolboxes and an API that considerably improves user experience and acceptance of OneDrive services. Users may use:

- a number of OneDrive clients that allow to see personal OneDrive content as part of the local file system on any operating system.

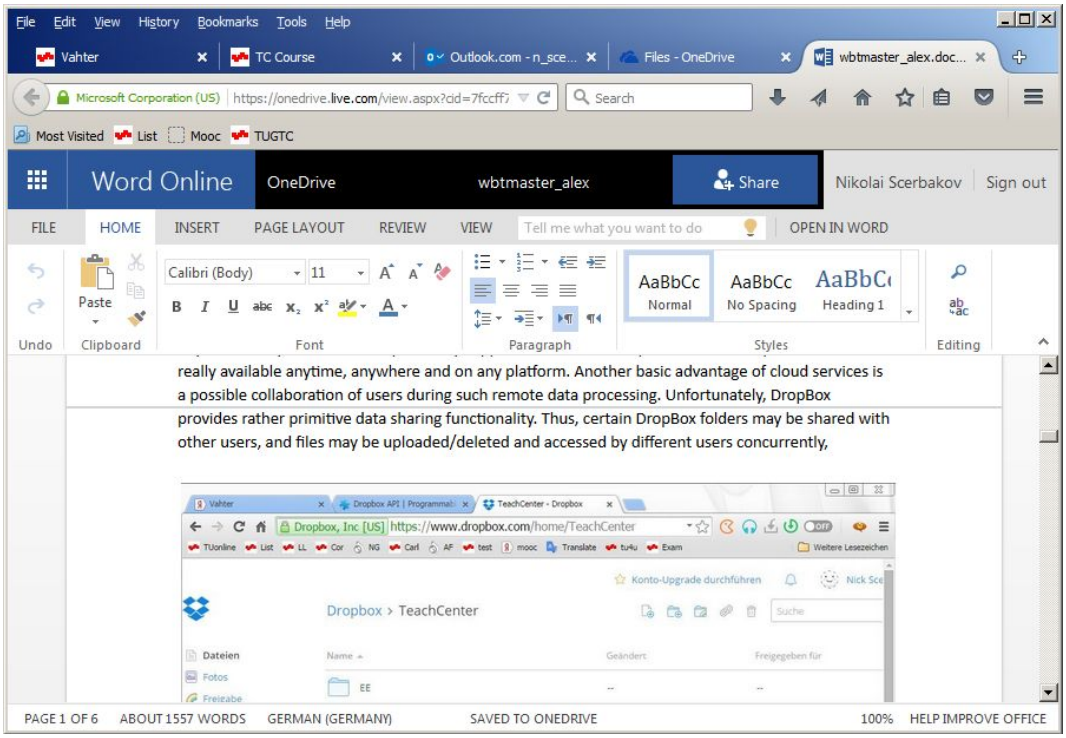

Figure 2.4: Remote editing of a Microsoft Word file in OneDrive.

- a special synchronization procedure that allows not only automatically download and upload of new, but also automatically download new versions of existing files as they are edited online.
- a OneDrive developer RESTful API that provides programmable access to the whole functionality of OneDrive such as creating items, uploading items, conversion formats, reading folder content, generating edit links, sending edit links to selected users, and so on.

The authentication mechanism and particular functions of the OneDrive API are described later in this thesis (see chapter 4).

2.5 Google Calendar

The Google Calendar is an Internet-based application that provides a calendar functionality. The cloud service operates with events that are attached to particular periods of time, and optionally to particular geographical coordinates. As the application provides Internet access to a calendar, as can be seen in figure 2.5, the calendar is available anytime and anywhere, and can be used concurrently by a number of users.

Google Calendar would be a rather conventional internet application, but it goes much further by providing:

- an interface and Add-ons for popular local calendar applications that allow to export and import from and into the remote Google Calendar.
- a synchronization procedure that allows to automatically download as well as download events as the calendar is edited.
- a REST developer API that provides programmable access to the whole functionality of Google calendar. This includes creating new events, editing existing events and deleting obsolete events.

2.6 Firefox Pocket

Firefox Pocket is an Internet-based bookmark management cloud service. The service operates with Internet bookmarks. Users may create, edit and

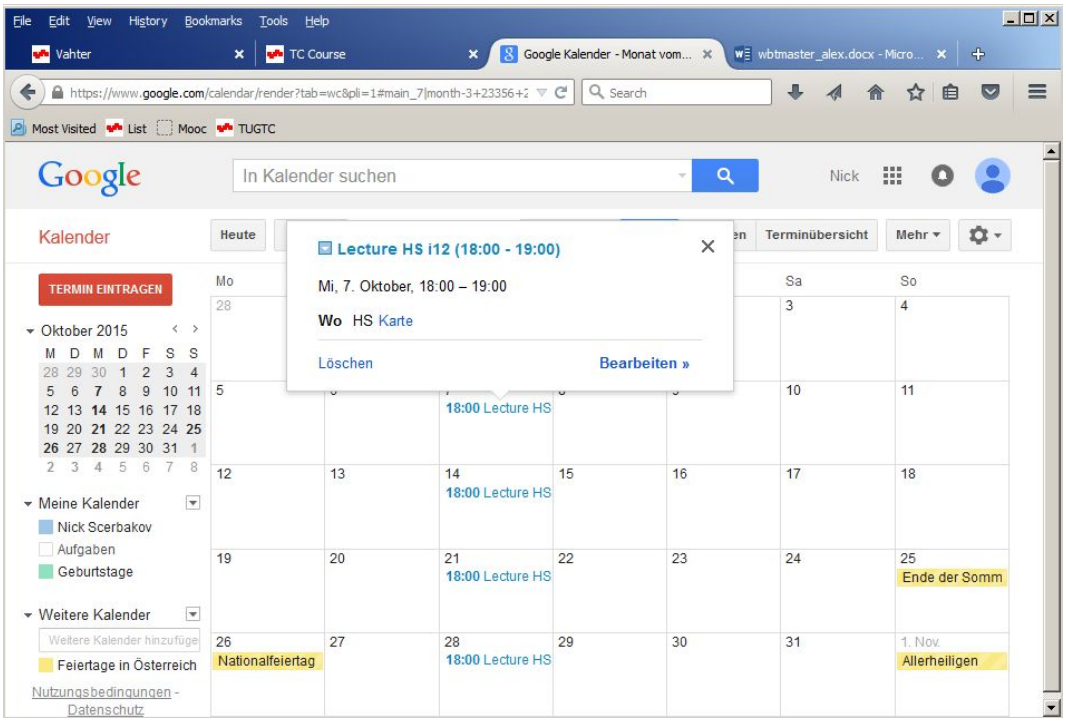

Figure 2.5: Remote editing of a Google Calendar.

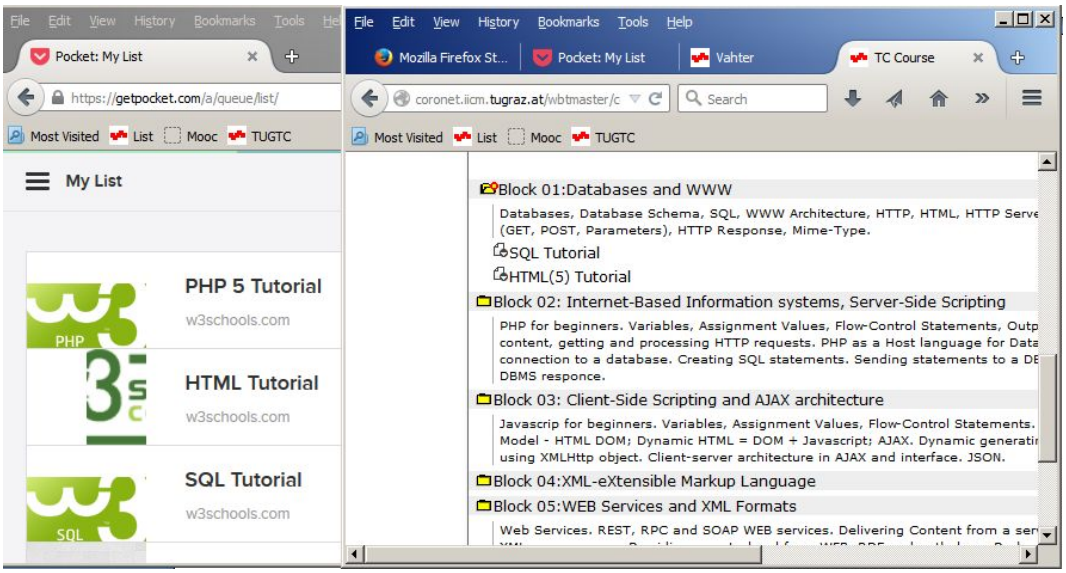

Figure 2.6: Synchronizing bookmarks in TC and Firefox Pocket

access bookmarks and web sites associated with the bookmarks. The main advantage of using Firefox Pocket is the automatic processing of bookmarks. Firefox Pocket strips away clutter, creates excerpts and thumb images, saves the info on internet pages in a clean, distraction-free view and lets users access them on the go through various Pocket clients. As the service provides Internet access to the bookmarks, the bookmarks become available anytime and anywhere, and can be used concurrently by a number of users.

Firefox Pocket provides a rich RESTful developer API that provides programmable access to the whole functionality of Firefox Pocket, as can be seen in figure 2.6.

3 Cloud services in LMS

In this chapter, a number of common LMS usage scenarios are introduced. The scenarios define data flows and knowledge sharing needed in LMS. For each of the scenarios a description as well as the ordinary solution are presented followed by a solution that was used in the TeachCenter using previously defined cloud services as well as a discussion of the advantages this solution offers.

3.1 Downloading training materials from TC

When discussing the download of materials from an LMS, three common scenarios come to mind:

1. A teacher uploads materials to the LMS. All materials are visualized with information on date of uploading and size of the uploaded file. Students may freely browse this repository and use information about the date of uploading and the size of the file in order to identify materials for downloading. No additional notifications on new materials are provided.
2. A teacher uploads materials to the LMS and it automatically sends emails to all users that are subscribed to notifications on updates. Users download new materials after receiving such emails.
3. A teacher uploads materials to the LMS. The LMS generates a Really Simple Syndication (RSS) feed where all the latest modifications are described. Students subscribe to one or more of such feeds and receive notifications from special RSS readers directly on their desktops.

The two main problems looking at the previous three scenarios are rather obvious. Students need to possibly download a large number of files by

hand and later on keep track of any changes made in these materials. There are quite a few solutions addressing these problems in LMS. The typical solution to the problem of downloading a huge amount of files by hand is giving the user the possibility to download selected and if needed all files with a single mouse click as a compressed archive, for example a .zip file. There are multiple solutions for the second problem concerning the updates of files. One solution is giving files and folders a date or another visual indication that changes have been done. Another attempt to solve this issue is a notification mechanism of any kind. As mentioned before, examples for such a notification mechanism are Email or RSS. Nevertheless the problem persists. Users need to download the materials on change, substitute older ones and organise them by themselves if they are kept on their hard drive. As a result, students can question the benefit of course materials, it can hurt their interest in the topic and lead to a bad learning outcome.

The above mentioned problems were solved in TeachCenter by using cloud services. Every user can grant TC access to their personal cloud sharing space. As a consequence, TC offers a one-click download of user selected course materials directly to their personal cloud sharing space, keeping the folder structure and saving the user a lot of time, as can be seen in figure 3.1. Therefore the problem of tedious and time consuming downloads of one file after another is solved. It should be especially noted that the download of materials may be done directly on the user's desktop, if a relevant cloud service client is installed on the user's computer.

Nevertheless, the problem with regular updates on course materials still persists. Cloud services can also help solving this issue. The user can grant TC access to the personal cloud sharing space and activate a so-called auto synchronization. As a result TC regularly checks if all relevant files exist on the personal cloud sharing space and compares their date of modification. If the files in TC are newer, they are automatically uploaded to the user's cloud sharing space. Therefore the user has all relevant course files, up to date and well organized in the personal cloud. The technical implementation will be described in more detail in chapter 4. The most suitable cloud services for implementation of these scenarios are Dropbox, Google Drive and Microsoft OneDrive.

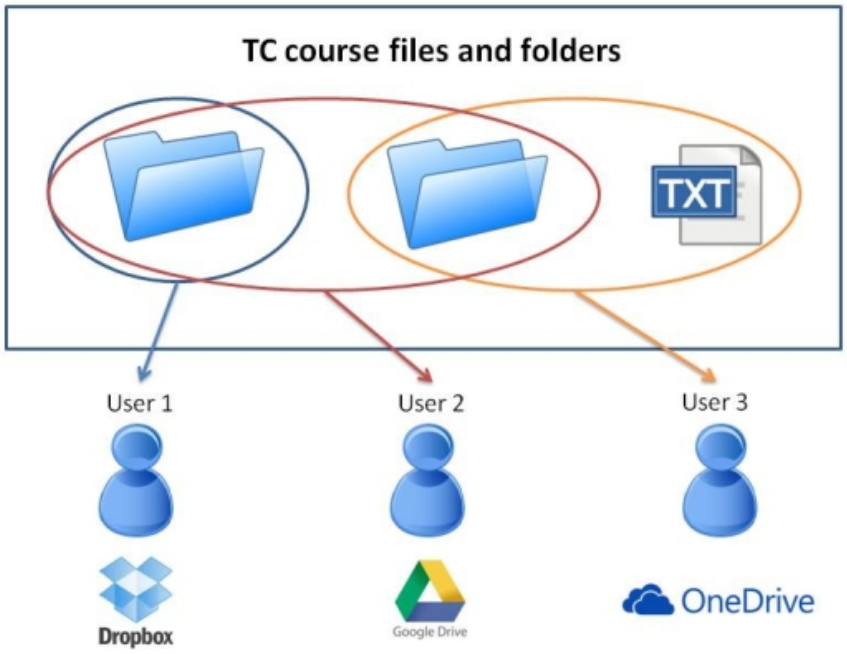

Figure 3.1: Download of selected course materials to personal cloud service repositories by different users

Figure 3.1 shows that users can download any selected course materials in their chosen personal cloud service repositories.

3.2 Uploading training materials to TC

Uploading course materials to an LMS can also be tedious. The main problem is that uploading a lot of files and their organisation into a clearly arranged folder structure is rather time consuming. A browser has to be used to carry out these tasks, and even the best designed user interface can not substitute a file explorer application. This is even more problematic in

an online course where a lot of files have to be uploaded and are updated on a regular basis.

Cloud sharing services such as Dropbox, Google Drive or Microsoft OneDrive are used in TC to help solving this issue. The idea is that a certain user, in this case the course teacher, selects a folder in their personal cloud service space and shares the content with a restricted group of users, the students of their course, as can be seen in figure 3.2. The students see the folders and files visualized in TC without any difference to normally uploaded files. Nevertheless the uploading task for the teacher becomes easier. Now, provided that a suitable client is installed, uploading a file, renaming it or moving it to a sub folder can be done on the users desktop. The changes are automatically carried out by TC. The technical idea behind this is similar to the auto synchronization for students, described in section 3.1 and will be described in detail in chapter 4. Simply speaking teachers grant TC access to their chosen cloud sharing service space. Subsequently, TC compares the cloud sharing service content of interest with the corresponding TC files and folders. If any changes are detected, TC applies them. Files and folders are added, removed or updated automatically.

3.3 Uploading shared assignments

A common scenario in LMS is students working as a group on an assignment and uploading the result to the LMS. The typical problem is keeping track of changes made by a group of users on one document. It can be rather time consuming for the students to merge a document modified by a number of people to include all changes before submitting it. In software development this problem is addressed with revision control system like Apache Subversion (SVN) or GIT. Although these solutions are very suitable for software projects, they can be too complicated for non-technicians or projects that just consist of a number of text files. Another problem that was already discussed in section 3.2 is the uploading of a large number of single files and folders, which can be very costly in terms of time.

Both problems are facilitated by the usage of cloud services in TC. Besides not being the primary purpose of cloud sharing services they can serve

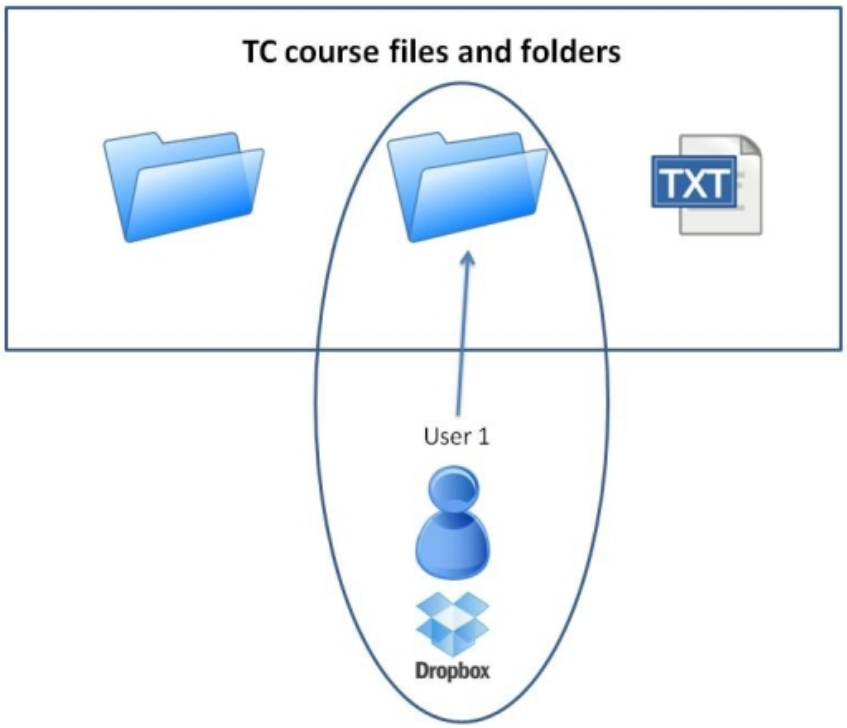

Figure 3.2: Upload of selected course materials from a personal cloud service to TC

well as a simple substitution for revision control systems. Users can form a working group and invite people to work together by granting access to a previously defined folder in the cloud sharing service. Subsequently, all the users having the granted rights can work together on the same files, all having them up to date. No extra implementation in the LMS is needed to use this possibility of cloud sharing services. As soon as the working process is done, typically there is a folder structure with a number of files in it that needs to be submitted to the LMS. In TC this can be done by a one-click upload where the user, after granting TC access to the cloud sharing service, simply selects the specific folder. The whole folder is then uploaded to TC.

In certain TC components uploads from cloud sharing services like Dropbox, Google Drive and Microsoft OneDrive are possible. These TC components are group lockers, project management rooms and student sharing files.

3.4 Collaborative authoring

The scenario of collaborative authoring is of use for any LMS course where text-based files need to be edited by a group of users. This editing needs to happen in real time and can happen simultaneously. Also mentioned in section 3.3, this scenario can be addressed by using revision control systems, although it is an inelegant solution.

In TC, collaborative authoring is done using fitting cloud services like Etherpad, Google Drive and MS OneDrive. A course teacher can choose a number of these cloud services to be used by the course learners. Subsequently, a group of learners can decide for a single cloud service which is being used by this user group. As a consequence the user group is able to work on a number of text-based documents residing on a cloud service server. Normally, the user group members are only provided with a link to the materials and edit the materials in their browser application. Nevertheless it is still possible to download the materials and edit them in a more convenient way.

3.5 Defining and sharing a course schedule

An important aspect of any e-learning course is organization or schedule. The training schedule provides teachers with an opportunity to deliver e-learning courses effectively. Practically, the course schedule is organized in a form of a course calendar. The course calendar keeps the e-learning course plan on track since it is supposed to notify learners of upcoming learning events and ensure that assignments and assessments are completed on time. Normally the course calendar is a component of an e-learning course available online. Teachers possess special tools to modify the calendar by adding, modifying and deleting events. Learners may access the course calendar any time by means of an Internet browser.

Such an online course calendar has a number of drawbacks:

- Syndication: Taking a university environment as an example, it can be observed that learners participate in a number of e-learning courses, each of the courses having a unique schedule that is provided by a separated online calendar. Working with multiple calendars can create some acceptance problems for learners and results in forgetting important meetings or events as well as missing deadlines. The solution may be a single personal calendar combining all the course calendars for a particular learner.
- Update notifications: One of the important duties of a teacher providing an online e-learning course is to keep the calendar entries constantly updated as a feedback to the course flow and current success of learners with previously defined tasks. Obviously, learners must be notified about such updates. A solution that is used by a number of modern LMS, is the ability to create e-mail notifications, which can be sent out to multiple recipients. This schema may have a negative effect though; learners tend to ignore multiple notifications, especially if one and the same event is modified a number of times. Learners must somehow remember changing events without spending additional efforts on modifying their personal calendar. The solution would be a special calendar synchronization mechanism that automatically synchronizes all the course calendars and a single personal calendar for a particular learner.

- Additional workload for teachers: The concept of online course calendars force teachers to work with a number of individual calendars. As a consequence, defining a course event without taking into account a personal calendar, may result in an appointment collision. Defining course events online by means of a browser requires some additional effort. The solution may be a special synchronization mechanism that allows teachers to synchronize their personal calendar and the course calendar. The synchronization works in such a way that modifications of a personal calendar can be automatically broadcasted to a course calendar after confirmation from the teacher.

The Google Calendar Web service was utilized in TeachCenter to solve all the problems above. Learners have got an authentication mechanism to set their Google accounts in TC and attach such accounts to their profiles. Figure 3.3 shows that learners may link all or some course calendars to their TC profiles. Therefore they have the possibility to automatically synchronize selected course calendars and a particular Google Calendar. Provided that the Google calendar is synchronized with their personal calendar, a required synchronization of a personal calendar with a number of course calendars is done. In figure 3.4 the result of linking a TC course calender with a personal Google calendar can be seen.

Teachers may also set their Google accounts to be used by TC. Only one Google account may be set as so-called course account. In this case, the Google calendar is synchronized with the course calendar and can be used as an ”authoring tool” for editing course calendars. Since the Google calendar application has very sophisticated online and offline editing facilities on different platforms including offline tools and mobile applications editing calenders is very easy to do.

3.6 Sharing bookmarks

The Internet is the main source of educational information nowadays and cannot be ignored if the discussion is about any kind of e-learning. In former times, learners simply searches the web for references or additional

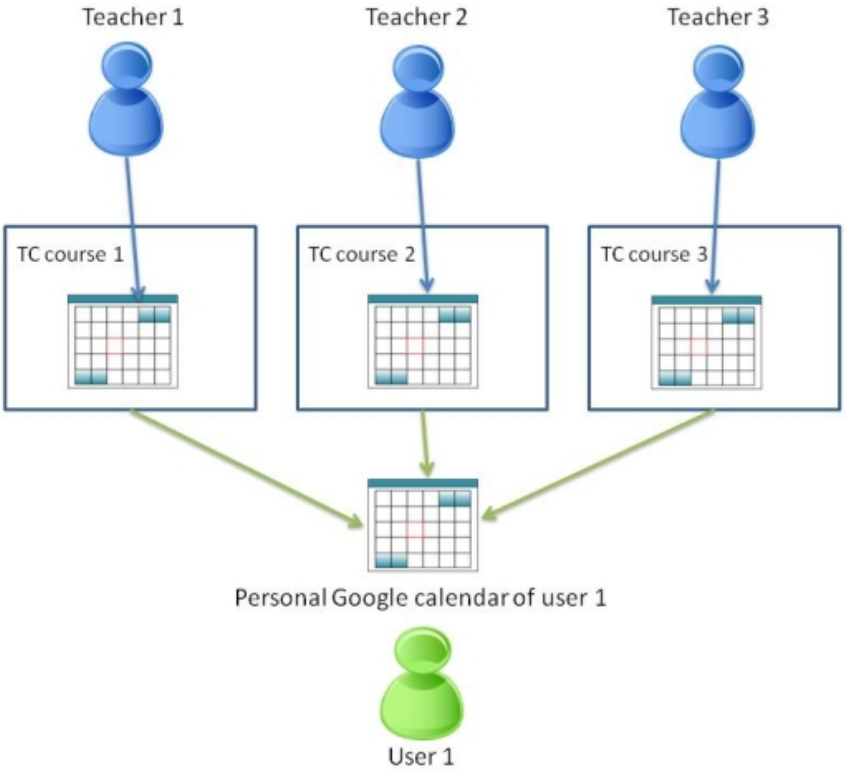

Figure 3.3: Synchronization of multiple TC course calendars with a personal Google Calendar

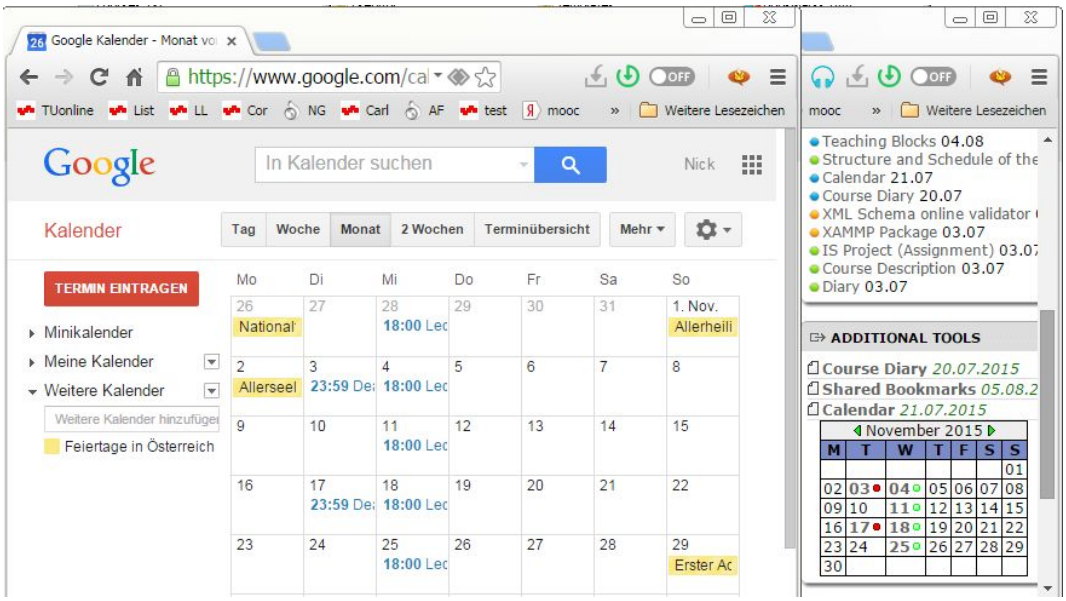

Figure 3.4: Synchronization of the TC course calendar and a personal Google Calendar

information about events or assignments defined by a teacher for a particular training course. Nowadays finding relevant information becomes increasingly tedious. Each Internet query results in thousands of hits, and simply looking through all the references results in a waste of time. Even worse, there is a strong tendency for increasing the amount of pointless or even wrong information available on the Web. A special component that allows teachers and learners the exchange of references to materials they find relevant, would be a big help for participants of an online e-learning course.

Such a shared collection of references must match certain requirements:

- Collaboration: The collection of references must be edited in a truly collaborative way, where all the participants have the possibility to provide references to relevant materials for others, and comment on references done by others.
- Update notifications: The references must be kept constantly updated. Users may include recent articles or informative sites that would be of interest to other learners. The learners must be somehow notified on

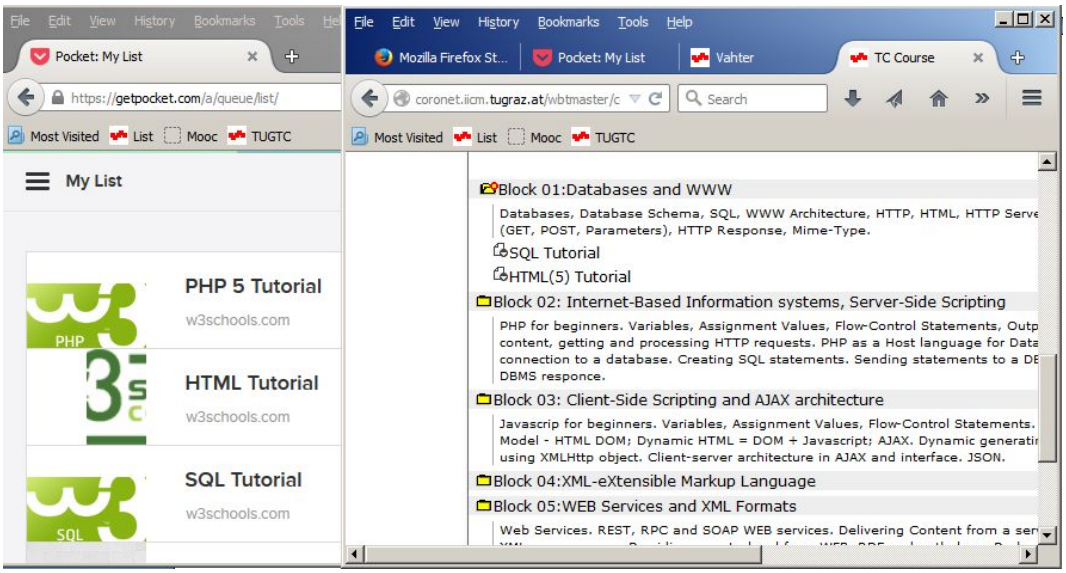

Figure 3.5: Synchronization of TC course shared bookmarks and a Firefox Pocket account

any modifications of the repository.

- Ease of link publishing: Publishing a new reference must be a ”single button click” action. A user should be provided with a tool that allows to publish a link as user accesses an interesting document with a browser.

The Firefox Pocket WEB service was utilized in TeachCenter to provide the functionality defined above.

Users may register their Pocket accounts in TC and synchronize them with one or a number of courses. As a course shared bookmarks repository is modified, the new references are automatically copied into synchronized pockets, and the users can see the links as a ”pocket” in context of their browsers, as can be seen in figure 3.5. The data flow may be organized in opposite direction as well. As users browse the Internet and see relevant documents, they may simply place a reference to the document in the pocket with one mouse click. The reference will be automatically provided with a thumb image and a short comment excerpted from the document. As the user accesses TeachCenter, the reference can be automatically imported with one mouse click.

4 Technical implementation

4.1 The OAuth 2.0 authentication and authorization mechanism

Most APIs nowadays use OAuth 2.0 for authentication and authorization. Therefore a look will be taken on the procedure to grant TC users access to certain cloud services using OAuth 2.0. The idea behind the OAuth procedure is that users can grant an application access to their data without the need of revealing their user credentials.

The simplified authorization sequence is shown in figure 4.1 and consists of the following steps that were implemented by using Groovy scripts residing on the TC server:

1. TC redirects the browser to a cloud service API endpoint, the redirect URL includes special query parameters to grant the needed access permissions.
2. The users credentials are entered and TC is authorized to use the cloud service.
3. The cloud service returns a user specific authorization code to a previously defined endpoint, in this case a Groovy script residing on TC server.
4. The script exchanges the authorization code for an access token and a refresh token, both tokens are stored in TC database.
5. The access token is used to make specific API calls.
6. If the token expires the refresh token is used to obtain a fresh access token.

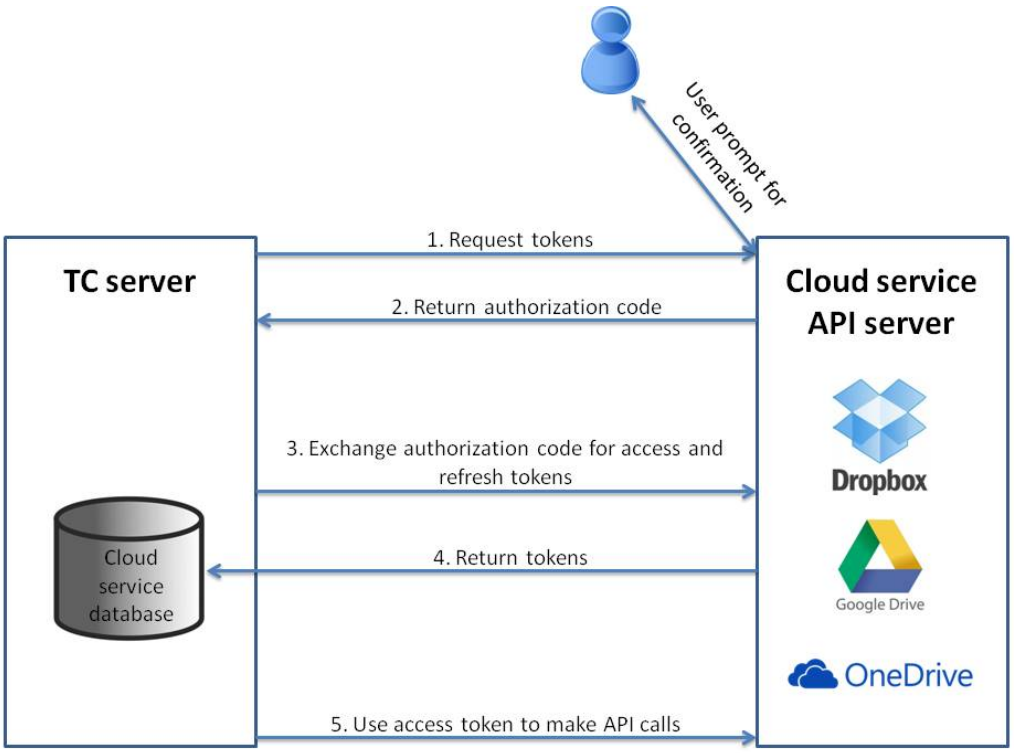

Figure 4.1: The OAuth 2.0 authorization procedure

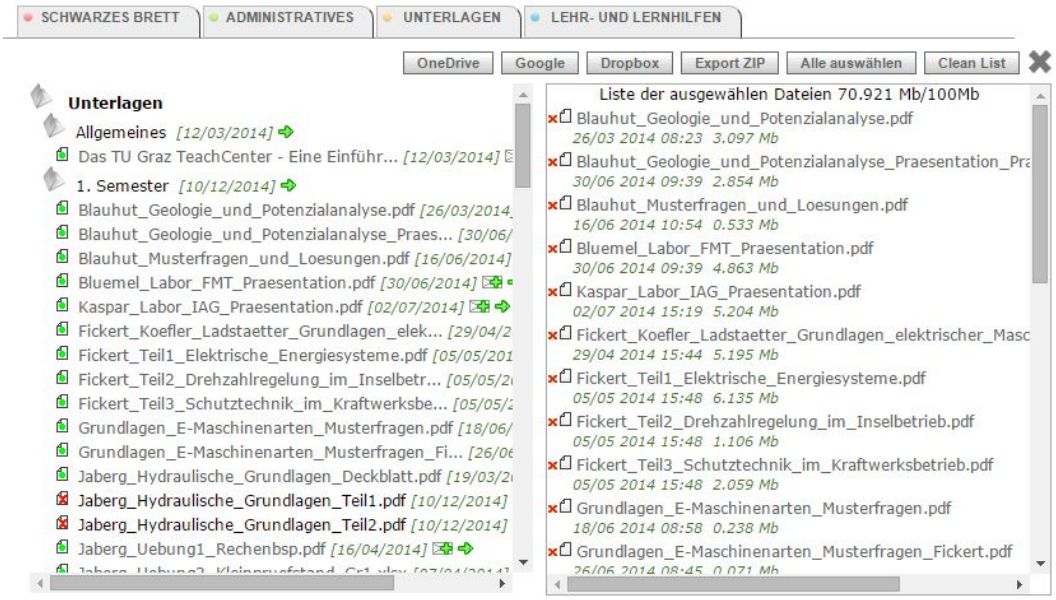

Figure 4.2: Download options of a TC course

4.2 Downloading from TC - Implementation

4.2.1 Functionality

As the add-on is activated, the user sees a whole list of files available for downloading from a particular course. Learners may select all files or some of them for downloading. In this case, all the selected files are visualized as a separated list, as can be seen in figure 4.2. The files can be added or removed from the list of selected files. There are four buttons that actually define functionality of the system: ”OneDrive”, ”Google”, ”Dropbox” and ”Export ZIP”. Functionality of ”Export ZIP” is as follows - the system simply builds an archive out of selected files and lets the user download the dynamically created archive.

As the user clicks the ”OneDrive” button, the system first checks whether the users OneDrive account is set to be used with TeachCenter. If the account is not set yet, the system initiates the OAuth 2.0 dialog to let the user log

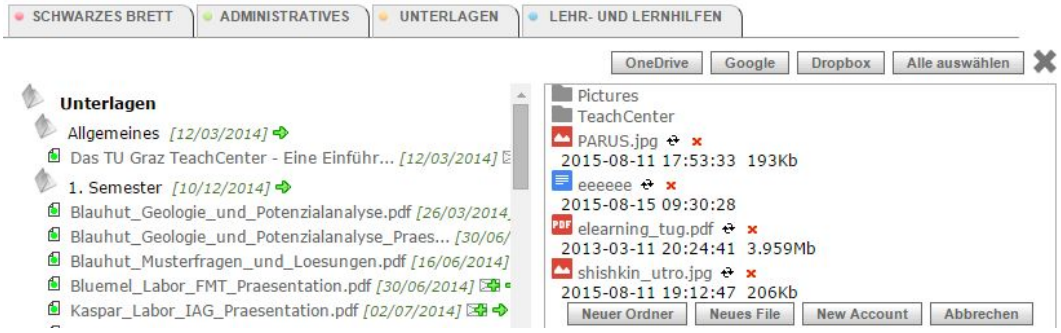

Figure 4.3: Transfer of TC files to Google Drive

in to a particular OneDrive account and confirm the permission to use API functions required by TC. If the account is set, the system simply uploads selected files to the Microsoft OneDrive where the files may be synchronized with other applications and become available in a convenient way. At any time, learners may visualize their OneDrives and have the possibility to preview files on OneDrive, create new folders and delete files and folders. The first time users download files to their OneDrive through TC, a folder named "TeachCenter" is created. All files are download into a sub-folder of this "TeachCenter" folder, having the same title as the TeachCenter course from which the files were copied.

Functionality of the Google Drive interface is very similar to the one of Microsoft OneDrive - files are selected for download, a Google Drive account is set for use with TeachCenter, and files are transferred to the user's Google Drive, as seen in figure 4.3.

The same functionality as for the two cloud services mentioned before is provided for Dropbox users via the Dropbox API. As before, files are selected and transferred to the users Dropbox with one mouse click. Choice between these three Cloud services - OneDrive, Google Drive or Dropbox - is up to the user, and is defined by personal preferences and local installations that make one or another service preferable for further usage.

4.2.2 Software architecture

The application is built on a base of AJAX principles. The whole application is a Dynamic HTML document that implements a number of JavaScript functions. The application communicates with the server by means of synchronous and asynchronous XMLHTTP requests. The source of functions for synchronous (serverSend(url)) and asynchronous communication (serverSendA(url,function)) can be seen in listing 4.1 and in listing 4.2.

Listing 4.1: Synchronous communication

```
{
    var xmlhttp;
    try {xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
        try {xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");}
        catch (E) {xmlhttp = false;}
    }
    if (!xmlhttp) {
        xmlhttp = new XMLHttpRequest();
    }

    lastTime++;
    var t = '';
    var x = getDate();
    var d = new Date();
    var i = d.getSeconds();
    x = x.substring(0,2) + x.substring(6,10) + i.toString() +
       lastTime;
    if(url.indexOf('?') != -1)t = '&token=' + globalToken + '&
       timestamp';
    else t = '?token=' + globalToken + '&timestamp';
    t += '=' + x;
    xmlhttp.open("GET", url + t,false);
    xmlhttp.setRequestHeader("checkURL", globalToken);
    try{xmlhttp.send(null);}
    catch(e) {x = '';}
    response = new String(xmlhttp.responseText);
    return(response);
}
```

Listing 4.2: Asynchronous communication

```
function serverSendA(url, functionToCall)
```

```
{
    var xmlhttpA;
    try {xmlhttpA = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
        try {xmlhttpA = new ActiveXObject("Microsoft.XMLHTTP")
            ;}
        catch (E) {xmlhttpA = false;}
    }
    if (!xmlhttpA) {
        xmlhttpA = new XMLHttpRequest();
    }

    lastTime++;
    var t = '';
    if(url.indexOf('?') != -1)t = '&timestamp';
    var x = getDate();
    var d = new Date();
    var i = d.getSeconds();
    x = x.substring(0,2) + x.substring(6,10) + i.toString() +
        lastTime;
    url = url + t + '=' + x;
    xmlhttpA.open("GET", url,true);
    xmlhttpA.setRequestHeader("checkURL", globalToken);
    xmlhttpA.onreadystatechange = function() {
        if (xmlhttpA.readyState == 4) {
            lastSearch = xmlhttpA.responseText;
            eval(functionToCall);
        }
    }
    xmlhttpA.send(null);
}
```

The application communicates with users by modifying the content of three "div" areas "topTools", "mainContent" and "downloadContent", as can be seen in listing 4.3.

Listing 4.3: Modifying of HTML div areas

```
<BODY onLoad="initIt()" BGCOLOR="#FFFFFF" MARGINWIDTH=10
   MARGINHEIGHT=10
        LEFTMARGIN=15 TOPMARGIN=5>
        <TABLE WIDTH=100% BORDER=0 CELLPADDING=0 CELLSPACING=0>
                <TR>
                        <TD colspan=2 align=Right WIDTH=100%
                            style="height: 30px;"><div
```

```
6                                           id="topTools"></div></
                                               TD>
7                   </TR>
8                   <TR>
9                           <TD valign=top><div id="mainContent"
10                                          style="height: 370px;
                                               width: 400px;
                                               overflow: auto;"></
                                               div></TD>
11                          <TD valign=top><div id="downloadContent
                                ">
12                                          style="height: 370px;
                                               width: 380px;
                                               overflow: auto;
                                               border: solid 1px #
                                               aaaaaa;">
13                                      </div></TD>
14                  </TR>
15          </TABLE>
16  </BODY>
```

Content of any of the areas seen in listing 4.3 is set by ordinary JavaScript using the statements, as can be seen in listing 4.4.

Listing 4.4: JavaScript content change

```
1  chosenId = document.getElementById("topTools");
2  if(chosenId){chosenId.innerHTML = "new HTML code";}
```

JavaScript functions are supposed to perform the final data processing on client-side, and provide a Graphical User Interface (GUI). Access to the TeachCenter database and communication to cloud services are carried out by a number of Groovy scripts residing on the TeachCenter server. The most important functions for the file downloading scenarios performed by the Groovy scripts are:

- Fetching the list of all files available from a particular TC course.
- Setting a particular cloud service user account to be used by TeachCenter.
- Upload a particular file to a cloud service.
- Fetching a folder content from a cloud service.
- Fetching a particular file from a cloud service.

Since communication to TC is outside of scope of this thesis, only modules providing an interface to the chosen cloud services are discussed.

4.2.3 Setting a particular cloud service user account

Setting a user account is done via OAuth 2.0, as described in section 4.1. The procedure starts with generating a particular URL where a user browser must be forwarded to confirm the permission for TC to use the service of choice. An example of this procedure, generating such "authorization" URL for Google Drive can be seen in listing 4.5.

Listing 4.5: OAuth2.0 step for Google Drive

```
function setGoogleProviderUP()
{
    responseString = openerX.serverSend0("/wbtmaster/facebook/
       google_0.groovy");
    responseArray = responseString.split('\%:\%');
    if(CurrentUserRole == 'student')return;
    url = 'http://accounts.google.com/o/oauth2/auth?';
    url += 'client_id=' + responseArray[0];
    url += '&redirect_uri=' + responseArray[2];
    url += '&response_type=code';
    url += '&approval_prompt=force';
    url += '&access_type=offline';
    url += '&scope=https://www.googleapis.com/auth/drive';
    WinPreview = open(url, "mapWindow","width=600, height=400,
       resizable=yes, status=no, toolbar=no, menubar=no");
    WinPreview.opener=self;
}
```

All the parameters of the authorization procedure shown in listing 4.5, such as client ID and redirect URI are stored on the server and fetched dynamically using the "google_0.groovy" procedure. As the user confirms permission to use the cloud service, a special access code is returned. The code must be exchanged for a so-called access token that is actually used to authenticate each API operation. A part of the source code is provided in listing 4.6.

Listing 4.6: Exchanging code for Google Drive access token

```
where = "https://www.googleapis.com/oauth2/v3/token";
```

```
URL obj = new URL(where);
HttpsURLConnection con = (HttpsURLConnection) obj.
    openConnection();
con.setRequestMethod("POST");
con.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
h = new getConnectionParameters();
l = h.getGoogle();
b = l.split("\%:\%").toList();
client_id = b[0];
client_secret = b[1];
redirect_uri = b[2];
urlParameters = "code=" + response_code + "&";
urlParameters += "client_id=" + client_id + "&";
urlParameters += "client_secret=" + client_secret + "&";
urlParameters += "redirect_uri=" + redirect_uri + "&";
urlParameters += "grant_type=authorization_code";

con.setDoOutput(true);
DataOutputStream wr = new DataOutputStream(con.getOutputStream
    ());
wr.writeBytes(urlParameters);
wr.flush();
wr.close();

int responseCode = con.getResponseCode();
if(responseCode<=200)
{
BufferedReader brin = new BufferedReader(new InputStreamReader
(con.getInputStream()));
String inputLine;
StringBuffer response = new StringBuffer();
while ((inputLine = brin.readLine()) != null) {
        response.append(inputLine);}
brin.close();
response_string = response.toString();
whole_token = getJson(response_string,"access_token");
refresh_token = getJson(response_string,"refresh_token");
print "response successful: " + response_string;
```

All the parameters of the application such as client ID, client secret and redirect URL are dynamically fetched from the server. After successfully completing the procedure seen in listing 4.6, an access token and refresh

token are saved to the user account on the TC server, the access token can be used for performing actions via the cloud services API. The refresh token is used to get a new access token as soon as the old access token is expired. It should be especially noted that the cloud services API normally returns a JSON file as a response. The JSON file must be parsed to extract data which is needed, the ”access token” and the ”refresh token” in this particular case, this is also done via Groovy script.

The same authentication procedure with small variations is used for each cloud sharing service - Google Drive, Dropbox and MS OneDrive. A procedure for generating an authentication URL for OneDrive can be seen in listing 4.7.

Listing 4.7: Exchanging code for OneDrive access token

```
function setOneDriveProviderUP()
{
    responseString = openerX.serverSend0("/wbtmaster/facebook/
       oneDrive_0.groovy");
    responseArray = responseString.split('\%:\%');
    url = 'https://login.live.com/oauth20_authorize.srf?';
    url += 'client_id=' + responseArray[0];
    url += '&response_type=code';
    url += '&scope=wl.signin wl.offline_access onedrive.
       readwrite';
    url += '&redirect_uri=' + encodeURI(responseArray[2]);
    WinPreview = open(url, "mapWindow","width=600, height=400,
       resizable=yes, status=no, toolbar=no, menubar=no");
    WinPreview.opener=self;
}
```

4.2.4 Transferring a file to a cloud service

This action is provided by all selected cloud service APIs. The actions is performed by posting the file by a POST HTTP request to a special API URL. This POST request must be authorized with a valid access token. The post request has a special ”multipart/form-data” format, and consists of two components: the first part having ”Content-Type” as ”application/json” contains meta information (title, mime-type, parent folder, and so on.) of the uploaded file, the second part contains the file itself as a binary stream.

A sample code for uploading files to Google Drive can be seen in listing 4.8.

Listing 4.8: Uploading a file to Google Drive

```
where = "https://www.googleapis.com/upload/drive/v2/files?
    uploadType=multipart";
URL obj = new URL(where);
HttpsURLConnection con = (HttpsURLConnection) obj.
    openConnection();
con.setRequestMethod("POST");
fileString =  "/wbtmaster/threads/" + room + "/" + baseDir +
    "/" + fid;
fX = new File(dir + fid);
map = URLConnection.getFileNameMap();
m = map.getContentTypeFor(fid);
con.setRequestProperty("Authorization", "Bearer " + whole_token
    );
con.setRequestProperty("Content-Type",'multipart/form-data;
    boundary="foo_bar_baz"');
try
{
        con.setDoOutput(true);
        DataOutputStream wrs = new DataOutputStream(con.
            getOutputStream());
        firstPart = '\n--foo_bar_baz' + '\n';
        firstPart += 'Content-Type: application/json; charset=
            UTF-8' + '\n';
        firstPart += '\n';
        firstPart += '{' + '\n';
        fidX = fid;
        firstPart += '"title": "' + fidX + '",' + '\n';
        if(folder != '*')
        {
                firstPart += '"mimeType": "' + m + '",' + '\n';
                firstPart += '"parents": [{"kind": "drive#
                    fileLink","id":"' + folder + '"}]' + '\n';
        }
        else firstPart += '"mimeType": "' + m + '"' + '\n';
        firstPart += '}' + '\n';
        firstPart += '--foo_bar_baz' + '\n';
        firstPart += 'Content-Type: ' + m + '\n\n';
        DataInputStream dos = new DataInputStream(new
            FileInputStream(dir + fid));
        byte[] bytes = new byte[2048];
```

```
32          int read_bytes = 0;
33          wrs.writeBytes(firstPart);
34          while(read_bytes != -1)
35          {read_bytes = dos.read(bytes, 0, 2048);
36                  if(read_bytes != -1)
37                          wrs.write(bytes, 0, read_bytes);}
38          firstPart = '\n' + '--foo_bar_baz--' + '\n';
39          wrs.writeBytes(firstPart);
40          wrs.flush();
41          wrs.close();}
42  catch(e){fileString = e.toString();}
43  int responseCode = con.getResponseCode();
44  if (responseCode < 300) {---Successful upload!---}
```

A similar uploading procedure with small variations is used for each cloud sharingservice - Google Drive, Dropbox and MS OneDrive. A sample procedure for uploading files to OneDrive can be seen in listing 4.9.

Listing 4.9: Uploading a file to MS OneDrive

```
1  if(folder != '*')where = "https://api.onedrive.com/v1.0/drive/
       items/" + folder + '/children';
2  else where = "https://api.onedrive.com/v1.0/drive/root/children
       ";
3  where += "/" + fid + "/content";
4  URL obj = new URL(where);
5  HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
6  con.setRequestMethod("PUT");
7  map = URLConnection.getFileNameMap();
8  m = map.getContentTypeFor(fid);
9  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
10 con.setRequestProperty("Content-Type",m);
11 con.setRequestProperty("Content-Length",ll.toString());
12 try
13 {
14          con.setDoOutput(true);
15          DataOutputStream wrs = new DataOutputStream(con.
               getOutputStream());
16          DataInputStream dos = new DataInputStream(new
               FileInputStream(dir + fid));
17          byte[] bytes = new byte[2048];
18          int read_bytes = 0;
19
```

```
20          while(read_bytes != -1)
21          {
22                  read_bytes = dos.read(bytes, 0, 2048);
23                  if(read_bytes != -1)
24                          wrs.write(bytes, 0, read_bytes);
25          }
26
27          wrs.flush();
28          wrs.close();
29  }
30  catch(e){uuu = e.toString();}
31  int responseCode = con.getResponseCode();
```

As seen in listing 4.9 the PUT request contains only the file as binary data in the body, all the file meta data, like parent folder and file title are defined by an API URL and the content type is defined by the request Header parameters.

4.2.5 Fetching a folder content from a cloud service

This action is provided by all selected cloud service APIs. The actions is performed by sending a GET request to a special API URL. The GET request must be authorized with a valid access token. The API returns the requested content in the form of a JSON text.

A sample code for fetching a folder content from Google Drive is provided in listing 4.10.

Listing 4.10: Fetching folder content from Google Drive

```
1  url = "https://www.googleapis.com/drive/v2/files?";
2  url += "fields=items(id%2CoriginalFilename%2CiconLink%2
       CmimeType%2CmodifiedDate";
3  url += "%2CfileSize%2Ckind%2Ctitle%2Cparents(id,isRoot))";
4  URL obj = new URL(url);
5  HttpURLConnection con = (HttpURLConnection) obj.openConnection
       ();
6  con.setRequestMethod("GET");
7  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
8  responseCode = con.getResponseCode();
```

```
9  if(responseCode<=200){
10         BufferedReader brin = new BufferedReader(
11         new InputStreamReader(con.getInputStream()));
12         String inputLine;
13         StringBuffer response = new StringBuffer();
14         while ((inputLine = brin.readLine()) != null) {response
               .append(inputLine);}
15         brin.close();
16         // Parse JSON response.toString();
17         .........
```

A similar procedure with small variations is used for each cloud sharing service - Google Drive, Dropbox and MS OneDrive. A sample procedure for fetching a folder content from OneDrive can be seen in listing 4.11.

Listing 4.11: Fetching folder content from MS OneDrive

```
1  url = "https://api.onedrive.com/v1.0/drive/root/children";
2  if(idX != '')url = "https://api.onedrive.com/v1.0/drive/items/"
       + idX + "/children";
3  URL obj = new URL(url);
4  HttpURLConnection con = (HttpURLConnection) obj.openConnection
       ();
5  con.setRequestMethod("GET");
6  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
7  responseCode = con.getResponseCode();
8  if(responseCode<=200){
9          BufferedReader brin = new BufferedReader(
10         new InputStreamReader(con.getInputStream()));
11         String inputLine;
12         StringBuffer response = new StringBuffer();
13         while ((inputLine = brin.readLine()) != null) {response
               .append(inputLine);}
14         brin.close();
15         JSONParser parser = new JSONParser();
16         JSONObject jsonObj = new JSONObject();
17         jsonObj = parser.parse(response.toString());
18         jsonList = jsonObj.value;
19         jsonList.each{.........}
```

4.2.6 Fetching a particular file from a cloud service

As it can be seen in listing 4.10 and in in listing 4.11, on a request to fetch content of a particular folder in the cloud, the API returns JSON text containing information on actual files. Each file has a download or content URL, which is an unique URL allowing to read the file as a binary string from the cloud service server provided that the GET request is authorized with a valid access token.

A sample code for fetching a file from Google Drive is provided in listing 4.12.

Listing 4.12: Fetching a file from Google Drive

```
1  url = "https://www.googleapis.com/drive/v2/files/" + fid + "?
       alt=media";
2  URL obj = new URL(url);
3  HttpURLConnection con = (HttpURLConnection) obj.openConnection
       ();
4  con.setRequestMethod("GET");
5  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
6  dir = context.getRealPath("/") + '/threads/' + room + '/' +
       baseDir + '/';
7  uX = fid + '.' + ext;
8  responseCode = con.getResponseCode();
9  if(responseCode<=200){
10         brin = con.getInputStream();
11         ByteArrayOutputStream baos = new ByteArrayOutputStream
               ();
12         DataOutputStream dos = new DataOutputStream(new
               FileOutputStream(dir + uX));
13         byte[] bytes = new byte[2048];
14         int read_bytes = 0;
15         while(read_bytes != -1){
16                 read_bytes = brin.read(bytes, 0, 2048);
17                 if(read_bytes != -1)
18                 dos.write(bytes, 0, read_bytes);}
19         dos.flush();
20         dos.close();
```

A similar procedure with small variations is used for each cloud sharing service - Google Drive, Dropbox and MS OneDrive.

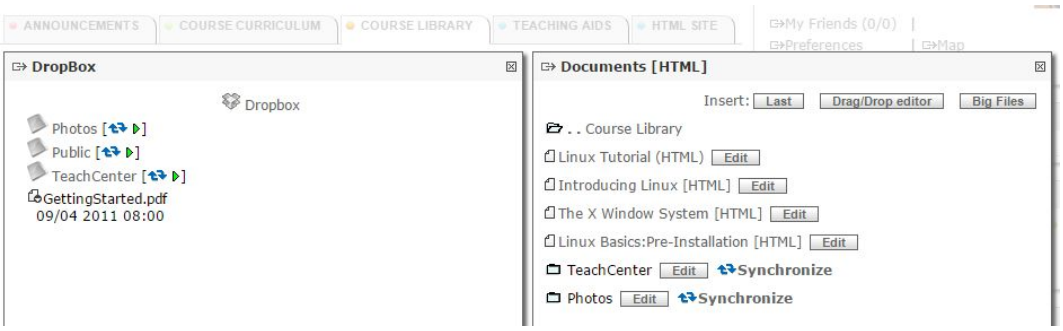

Figure 4.4: Synchronizing Dropbox folders with course library folders

4.3 Uploading to TC - Implementation

4.3.1 Functionality

The scenario is implemented as two independent components. The first component is part of core TC functionality. As it was already mentioned, TeachCenter supports a modular architecture. It is built out of a number of functional modules, and teachers may dynamically select a sub-set of modules that is used for a particular course. One of the core modules is the so-called course library. The course library is a structured repository of training materials available for downloading by learners. The course library consists of folders and files residing in the folders. In a standard scenario the course teacher creates folders and uploads files into this previously created folders. Additionally, the library editing software supports two further operations: the teacher may set a course cloud service account, and select a certain folder on this cloud service. The selected folder can be simply copied to the course as a normal course library folder. In addition the cloud folder may be synchronized with a library folder, as can be seen in figure 4.4.

If a folder is set as synchronized, information about the course cloud account and folder titles are stored in a special TC synchronization file. The teacher may also decide about a synchronization schedule, folders may be checked for discrepancy as any user accesses the course library folder, every hour, every day or once per month. It should be noted that files in such a

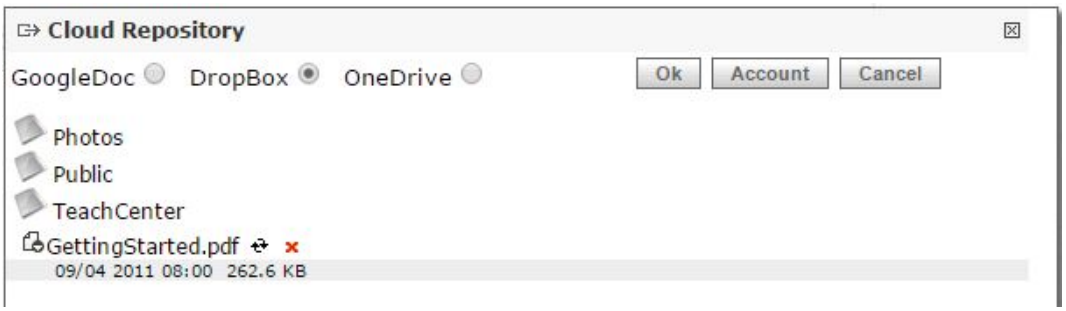

Figure 4.5: Using the "cloud repository" add-on in TC

synchronized folder are treated exactly in the same way as any other TC files. They can be downloaded using any available synchronization and massive downloading mechanisms, commented, activated at a certain time and so on. Moreover the content of such a synchronized folder is taken into account by all TC notification mechanisms, being RSS, e-mails, and so on. Simply stated, as a new file is uploaded into such a synchronized folder, all students are automatically notified using a previously selected notification mechanism.

Another mechanism of implementing the scenario of using cloud services in TC for uploading is also based on the modular architecture of TC. Besides the core TC modules such as "course announcements", "course curriculum", "course library", and so on, teachers may choose among dozens of additional modules. One such additional module is called "cloud repository". As soon as the module is activated, teachers may select one of the following cloud services - Dropbox, Google Drive or MS OneDrive, and set a course cloud service account. Setting a course cloud service account allows any learners to access the cloud service via a relevant API. Therefore, any course member gets access to a single cloud service account and may access as well as download files prepared by the teacher using that service editing tools, as can be seen in figure 4.5.

By using this method, learners always get access to the latest version of such remote files.

As a disadvantage of this method, it can be mentioned that files from the cloud repository are treated differently from other TeachCenter files. The

remote files cannot be commented or synchronized with learners accounts, as can be seen in section 4.2. There is also no mechanism for automatic notifying learners on latest updates of this "cloud repository" files. The functionality is provided for three cloud services - Dropbox, Google Drive and MS OneDrive. Choice between these three cloud services is up to the teacher of a particular course, and is defined by personal preference and local installations which make one or another service preferable for actual usage.

4.3.2 Software Architecture

The applications are built using different architectural solutions. Synchronization of TeachCenter folders with a cloud service is done by means of a special Groovy script which is called as a "Cron Job" in accordance with a predefined job schedule.

1. Read current content of the cloud service folder.
2. Check every file and sub-folder whether it is available in the TC folder.
3. Check time stamps for each item.
4. If there is a new or newer version of the file, it is downloaded to TC.
5. The process is repeated for each entry of a database file residing on TC server and containing references to all synchronized folders.
6. Read current content of relevant TC folder.
7. Check every file and sub-folder whether it is available on the cloud service, if not - delete the file.

A fragment of the DropBox synchronization script can be seen in listing 4.13.

Listing 4.13: Dropbox synchronization with TC folder

```
1 def readURLBinaryAuth(b_url,consumer,uXX,addD)
2 {
3         URL url;
4         try
5         {
6                 url = new URL(b_url);
7                 URLConnection connection = url.openConnection()
                      ;
```

```
8                   consumer.sign(connection);
9                   DataInputStream dis = new DataInputStream(
                       connection.getInputStream());
10                  ByteArrayOutputStream baos = new
                       ByteArrayOutputStream();
11                  DataOutputStream dos = new DataOutputStream(new
                        FileOutputStream(context.getRealPath("/") +
                        addD + uX));
12                  byte[] bytes = new byte[2048];
13                  int read_bytes = 0;
14                  while(read_bytes != -1)
15                  {
16                          read_bytes = dis.read(bytes, 0, 2048);
17                          if(read_bytes != -1)
18                                  dos.write(bytes, 0, read_bytes)
                                       ;
19                  }
20                  dos.flush();
21                  dos.close();
22          }
23          catch(e)
24          {
25                  writeFile(uX,e.toString());
26          }
27          return ext;
28  }
29  h = new getConnectionParameters();
30  ll = h.getDropBox();
31  a = ll.split("%:%").toList();
32  api = a[0];
33  URL_OAUTH_REQUEST_TOKEN = api + "oauth/request_token";
34  URL_OAUTH_ACCESS_TOKEN = api + "oauth/access_token";
35  URL_OAUTH_AUTHORIZE = a[3] + "authorize";
36  URL_TOKEN = a[2] + "token";
37  URL_ACCOUNT_INFO = api + "account/info";
38  URL_FILES_DROPBOX = a[0] + "files/dropbox/";
39  URL_FILES_SANDBOX = a[0] + "files/sandbox/";
40  URL_METADATA_DROPBOX = api + "metadata/dropbox/";
41  URL_METADATA_SANDBOX = api + "metadata/sandbox/";
42  OAUTH_SIGNATURE_METHOD = "HMAC-SHA1";
43  OAUTH_VERSION = a[1];
44  APP_KEY = a[4];
45  APP_SECRET = a[5];
46  fN= request.getParameter("fN")
```

```
47  if(!fN)fN = "hronoJob.txt";
48  l = readFile(fN);
49  A = l.split('%;%').toList();
50  r = '';
51  A.each()
52  {
53          ones = it.toString();
54          if(ones.contains('%:%'))
55          {
56                  a = ones.split('%:%').toList();
57                  room= a[0];
58                  folder= a[1];
59                  baseDir= a[2];
60                  r += 'Start=' + room + '=' + folder + '=' +
                        baseDir + '<br>';
61                  count = 0;
62                  rr = '';
63                  newFolder = folder;
64                  if(newFolder.substring(newFolder.length()-1) ==
                         '/')newFolder = newFolder.substring(0,
                        newFolder.length()-1);
65                  if(newFolder.contains('/'))newFolder =
                        newFolder.substring(newFolder.lastIndexOf
                        ('/')+1);
66                  passF = '';
67                  listF = '';
68                  if(baseDir != '*')
69                  {
70                          a = baseDir.split(':').toList();
71                          passF = a[0] + '/' + a[1] + '.lib';
72                          passM = false;
73                          fxx = new File(context.getRealPath("/")
                                + 'threads/' + passF)
74                          if(!fxx.exists())
75                          {
76                                  listF= '';
77                          }
78                          else listF = fxx.getText("ISO-8859-1")
79                          ndir1 = 'threads/' + a[0] + '/' + a[1].
                               substring(7);
80                  }
81                  else
82                  {
83                          passF = '';
```

```
84              baseDir += '/';
85              ndir1 = 'threads/server/' + room + '/'
                    + baseDir + newFolder;
86          }
87          ndir = context.getRealPath("/") + ndir1;
88          fxx_file = new File(ndir);
89          if(!fxx_file.exists())fxx_file.mkdir();
90
91          dir = context.getRealPath("/") + "threads/
                mainBlog/registration/";
92          ext = dir + room + '_';
93          OAuthProvider provider = null;
94          OAuthConsumer consumer = null;
95          String url = '';
96          def providerFile = new File(ext + "provider.xml
                ");
97          def consumerFile = new File(ext + "consumer.xml
                ")
98          if(providerFile.exists())
99          {
100             def xstream = new XStream()
101             providerFile.withInputStream { ins ->
                    provider = xstream.fromXML(ins) }
102             def xstream2 = new XStream()
103             consumerFile.withInputStream { ins ->
                    consumer = xstream2.fromXML(ins) }
104         }
105         x = readFile(room + "_dropBox.txt");
106         a = x.split("=").toList();
107         try
108         {
109             consumer.setTokenWithSecret(a[1], a[2])
                    ;
110         }
111         catch(e){}
112         url = "https://api.dropbox.com/1/metadata/
                dropbox/" + folder;
113         readURLBinaryAuth(url,consumer,"x.txt","threads
                /mainBlog/registration/");
114
115         // The remote folder content is read
116         ...
117         // Processing individual files
118         if(!check_file.exists())
```

```
119                 {
120                         // if the file does not exists, fetch
                               it
121                         readURLBinaryAuth(url,consumer,p,ndir1
                               + '/');
122                 }
```

The "cloud repository" module is built on a base of AJAX principles as a dynamic HTML document which implement a number of JavaScript functions. The application communicates with the server by means of synchronous and asynchronous XMLHTTP requests as mentioned in section 4.2.2. The module communicates to users by modifying the content of two "div" areas "mainTools" and "mainContent" as can be see in listing 4.14.

Listing 4.14: HTML of the div areas which are modified

```
<BODY onLoad="initCompleted()" bgcolor="#FFFFFF" MARGINWIDTH=0
        MARGINHEIGHT=0 LEFTMARGIN=0 RIGHTMARGIN=0 TOPMARGIN=0>
        <div id="mainTools"></div>
        <div id="mainContent" style="margin-top: 10px;"></div>
</BODY>
</HTML>
```

Content of any of these areas are set using statements which can be seen in listing 4.15.

Listing 4.15: JavaScript setting of content

```
o = document.getElementById("mainTools");
if(o){o.innerHTML = {new html code};}
```

JavaScript functions are supposed to perform the final data processing on client side, and provide a GUI. Access to the TeachCenter database and communication with chosen cloud services are carried out by a number of Groovy scripts residing on the TeachCenter server. The most important functions for the file uploading scenarios performed by these scripts are:

- Setting a particular cloud service user account to be used by TeachCenter.
- Fetching a folder content from a cloud service.
- Fetching a particular file from a cloud service.

Since communication to TC is outside the scope of this thesis, further discussion is only about modules providing interfaces to main cloud services.

4.3.3 Setting a particular cloud service user account

The authentication procedure is almost the same as was described in section 4.2.3. The only difference is that as TC gets a valid access token, it is not stored in the user profile, but in the course profile to make it available for any course the user is accessing ”on behalf” of the teacher.

4.3.4 Fetching a folders content from a cloud service

The procedure is almost the same as was described in section 4.2.5. The only difference is that in case of a ”personal” cloud service, the procedure additionally checks the current user name to make sure that the service is accessed by its owner. In case of the course cloud account, any user may access the service, and such check is omitted.

4.3.5 Fetching a particular file from a cloud service

The procedure is almost the same as was described in section 4.2.6. The difference is two-fold:

- As in the previous case any user may access the service, and therefore the check of current user name to be equal to the name of the service owner is omitted.
- In the previous scenario of downloading files from TC, files are read from a cloud service just to support a ”preview” function and are not permanently stored on TC. In this scenario, files are downloaded and saved on TC permanently to allow other learners to access the same files directly from TC.

4.4 Uploading shared assignments - Implementation

4.4.1 Functionality

The scenario is implemented as part of a number of assignment uploading component - ”student lockers”, ”project rooms” and ”shared files”. All these components allow to upload student assignments and thus make them available for evaluation by the teacher or other students.

”Student lockers” are a protected memory space on the server where students may place their assignments in a form of files. Lockers are allocated for groups of students working on the same assignment. Access to lockers is protected by a password. In order to work with the locker content, the password must be known by the student. Teachers get access to all lockers in their courses. ”Project rooms” are very similar to lockers except that the access is provided only for members of a previously defined user group, so no password mechanism is used. Files in the project rooms are divided into two kinds. So-called teacher files providing instructions for learners and so-called student files that may be uploaded by learners. ”Shared files” is a structured repository of files provided by learners. The shared files room consists of folders and files residing in the folders, these folders are not protected from access by other users. Any course participant may create folders and upload files.

All these components operate with assignments done by working groups. Cloud services are especially suitable for arranging a group authoring of training assignments. As soon as such an assignment is ready and should be submitted for evaluation, TeachCenter offers a special mechanism for uploading cloud service files to the TC server, as can be seen in figure 4.6.

Learners may set a particular cloud service account, and then select files residing on the cloud service. The selected files are simply copied to TC in the corresponding assignment uploading component. It should be noted that files uploaded in this way are treated exactly in the same way as any other student uploads, meaning they can be downloaded by teachers using any available export mechanisms, commented, evaluated and so on. The

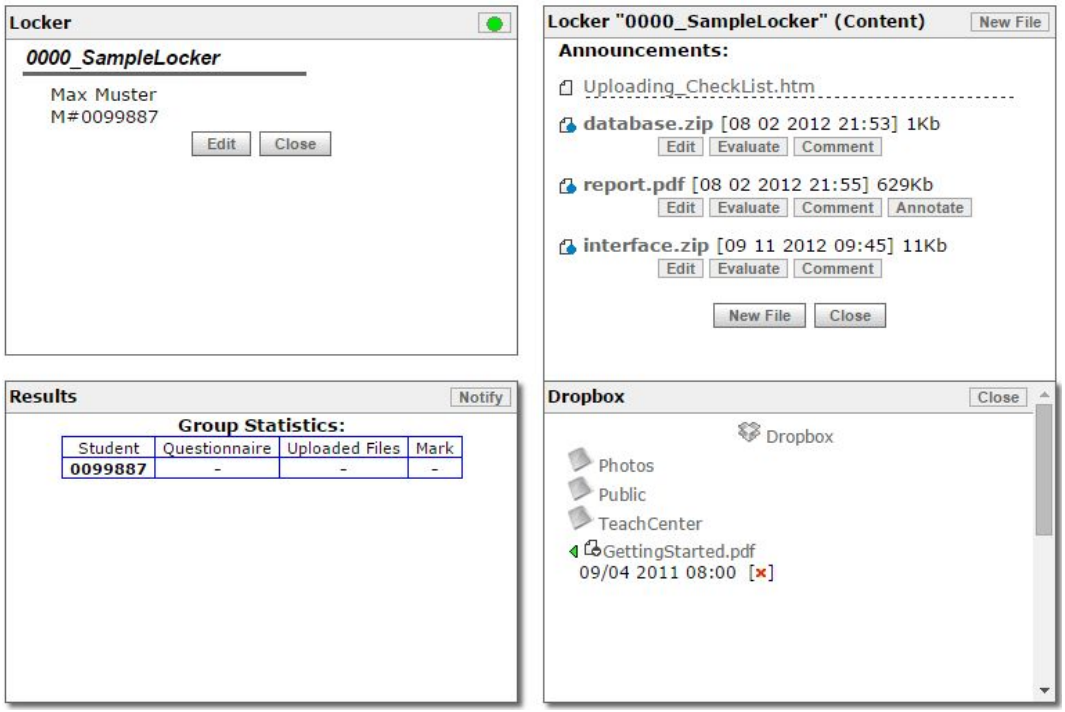

Figure 4.6: Uploading Dropbox files in an assignment locker

functionality is provided for three cloud services - Dropbox, Google Drive and MS OneDrive. Choice between these three cloud services is up to a student group and is defined by personal preferences and local installations that make one or another service preferable for actual usage.

4.4.2 Software architecture

The application is built on a base of AJAX principles as a dynamic HTML document that implements a number of JavaScript functions. It communicates to the server by means of synchronous and asynchronous XMLHTTP requests (see section 4.2).

JavaScript functions are supposed to perform the final data processing on client side, and provide a GUI. Access to the TeachCenter database and communication to cloud services are carried out by a number of Groovy scripts residing on the TeachCenter server. The most important functions for the uploading shared assignments scenario performed by Groovy scripts are:

- Setting a particular cloud service user account to be used by TeachCenter.
- Fetching a folder content from a cloud service.
- Fetching a particular file from a cloud service.

4.4.3 Setting a particular cloud service user account

The authentication procedure is almost the same as was described in section 4.2.3. The only difference is that as the TC gets a valid access token, it is not stored in the user profile or the course profile, but in the locker profile to make it available for any course user accessing the locker.

4.4.4 Fetching a folder content from a cloud service

The procedure is almost the same as was described in section 4.2.5. The only difference is that in this case the access is granted only to members of a working group or to users possessing an access key.

4.4.5 Fetching a particular file from a cloud service

The procedure is almost the same as was described in section 4.2.6.

4.5 Collaborative authoring- Implementation

4.5.1 Functionality

The scenario is implemented as a special TC add-on component called ”shared documents” as can be see in figure 4.7. This component supports the collaborative authoring of text documents, spreadsheets and presentations by a group of students. Such groups are called ”working groups”.

The ”shared documents” room consists of group lockers. Each locker is a protected memory space on the server where users may edit documents collaboratively in real-time. Thus students may write articles, press releases, to-do lists, etc., in collaboration with group members all working on the same document at the same time. Access to a locker is only provided for previously defined members of a working group as well as teachers of the particular TC course.

Cloud services are essentially used by TC to arrange collaborative authoring. Teachers may set a number of cloud accounts that can be used by learners. The system may work with Etherpad, Google Drive or MS OneDrive accounts.

Choice between one of the three previously defined cloud services is up to a student group and is defined by personal preferences and local installations that make one or another service preferable for actual usage. Learners may

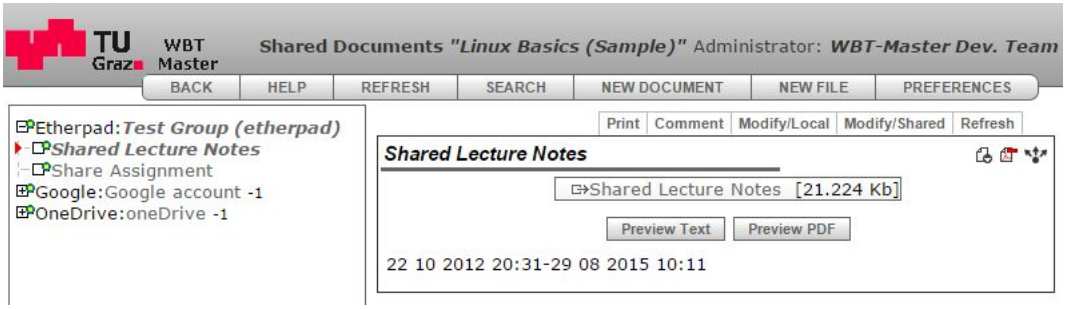

Figure 4.7: The "shared documents" room

create lockers and form a working group. As a working group is created, learners may select one of the previously defined cloud accounts to work with. Users may upload any local text files into the locker and synchronize the files with the cloud account. In this case, the file is copied to the cloud service and synchronized with the file on TC. As the file is synchronized, a special button "Modify/Shared" appears. This button allows to edit the file by means of the selected cloud service. As the remote file is modified, a new version may be downloaded to the locker as a synchronization procedure. The synchronization is carried out manually by pressing the "Refresh" button or when a locker is opened by any member of the working group. Furthermore synchronization may be done as a Cron Job.

As the collaborative authoring procedure is over, the shared files may be converted and downloaded in other formats such as PDF, HTML and TXT.

4.5.2 Software Architecture

The application is built as a dynamic HTML document that implements a number of JavaScript functions. It communicates to the server by means of synchronous and asynchronous XMLHTTP requests (see section 4.2).

JavaScript functions are supposed to perform a final data processing on a client-side, and provide a GUI. Access to the TeachCenter database and communication to cloud services are carried out by a number of Groovy scripts residing on the TeachCenter server. The most important functions for the collaborative authoring scenario performed by Groovy scripts are:

- Setting a particular cloud service account to be used by TeachCenter.
- Providing access to a selected cloud service to edit the shared documents.
- Uploading a file to a cloud service.
- Fetching/converting a particular file from a cloud service.

4.5.3 Setting a particular cloud service user account

The authentication procedure for using Groovy Drive and MS One Drive is almost the same as was described in section 4.2.3. Etherpad does not require to define user accounts, documents can be accessed and edited just by using a particular Pad ID. All API functions can be used if a special API key is provided.

4.5.4 Providing access to a selected cloud service to edit the shared documents

This operation is performed essentially differently for all of the three used cloud services:

Etherpad
This is the simplest case, all the documents shared via an Etherpad server have a special Pad ID. The document may be accessed and edited by anyone simply knowing this ID: https://etherpad.learninglab.tugraz.at/p/{pad ID}. Therefore, as the button "Edit/Shared" is clicked, the system uses this URL to load Etherpad into a new browser window or iFrame window, as can be seen in figure 4.8.

Microsoft OneDrive
The OneDrive API allows to dynamically generate an edit link. This link can be used further by any MS OneDrive account to edit a shared file. The link is generated by sending a POST JSON request to a special API end point, as can be seen in listing 4.16.

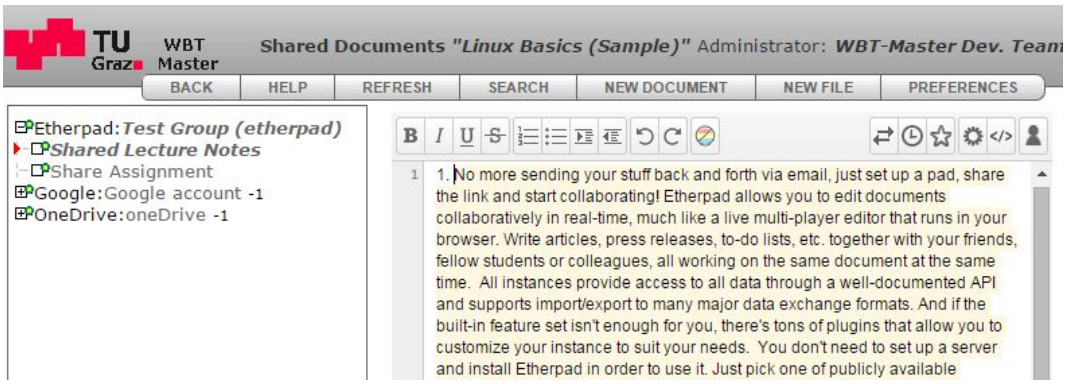

Figure 4.8: Collaborative Editing of a text document via Etherpad

Listing 4.16: Generating an edit link for MS OneDrive

```
1  firstPart = '';
2  firstPart += '{' + '\n';
3  firstPart += '"type": "edit"' + '\n';
4  firstPart += '}' + '\n';
5  where = "https://api.onedrive.com/v1.0/drive/items/" + file +
       "/action.createLink";
6  URL obj = new URL(where);
7  HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
8  con.setRequestMethod("POST");
9  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
10 con.setRequestProperty("Content-Type","application/json");
11 con.setRequestProperty("Content-Length",firstPart.length().
       toString());
12 try
13 {
14         con.setDoOutput(true);
15         DataOutputStream wrs = new DataOutputStream(con.
               getOutputStream());
16         wrs.writeBytes(firstPart);
17         wrs.flush();
18         wrs.close();
19 }
20 catch(e){print e.toString();}
21 int responseCode = con.getResponseCode();
22 StringBuffer response = new StringBuffer();
```

```
23 response_string = '';
24 BufferedReader brin00 = new BufferedReader(new
       InputStreamReader(con.getInputStream()));
25 String inputLine = '';
26 if(responseCode<=300)
27 {
28         while ((inputLine = brin.readLine()) != null)
29         {
30                 response_string += inputLine.toString();
31         }
32 }
```

Thus, as the button "Edit/Shared" is clicked, the application uses the script to generate such an edit link and load the link's content into a new browser window or iFrame window. Learners may need to login into their MS One Drive account to use this link for collaborative editing.

Google Drive
Google Drive is the most complex case of getting permission to edit a shared document out of the three used cloud sharing services. Each document on the Google Drive is associated with a number of users having different roles. The Google Drive API allows:

- To fetch info about all users associated with a certain document.
- Associate a new user with a particular role on a document.

Therefore, before accessing a shared document, the user must be associated with the document as can be seen in figure fig:setright by means of a special script.

A user is associated with a document by sending a HTTP POST JSON request to a special API end point. Before sending such a request, the script receives a new permission object for a user via their e-mail address. It is initiated by a user having access to the document, as seen in figure 4.9. A part of the source code can be seen in listing 4.17.

Listing 4.17: Setting right for a Google Drive document

```
1 eM = request.getParameter("emails");
2 if(!eM)eM = "tugtc@tugraz.at";
3 url0 = "https://www.googleapis.com/drive/v2/permissionIds/" +
      eM;
```

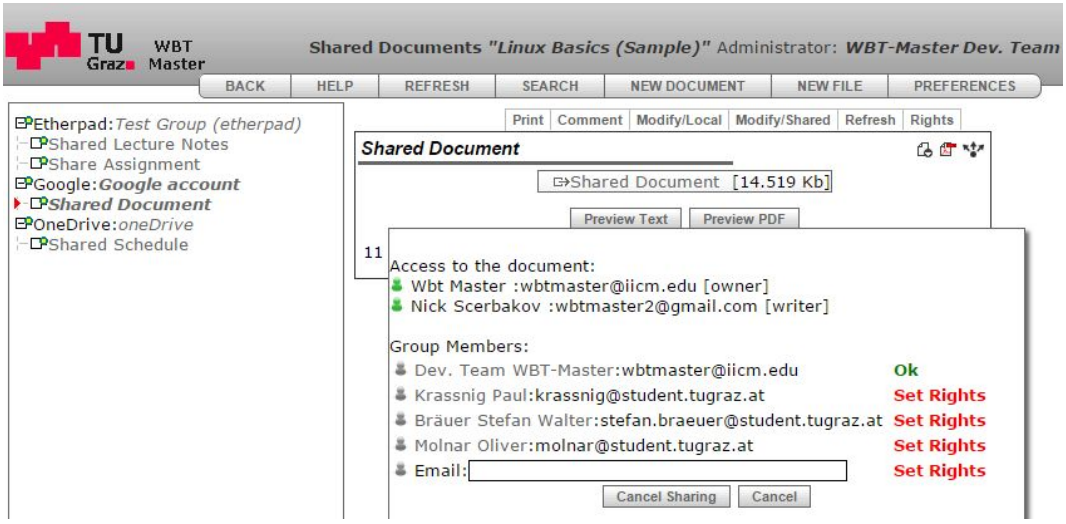

Figure 4.9: Setting rights to access a document in TC

```
4  URL obj0 = new URL(url0);
5  HttpURLConnection con0 = (HttpURLConnection) obj0.
       openConnection();
6  con0.setRequestMethod("GET");
7  con0.setRequestProperty("Authorization", "Bearer " +
       whole_token);
8  responseCode0 = con0.getResponseCode();
9  StringBuffer response0 = new StringBuffer();
10 if(responseCode0<=200)
11 {
12         BufferedReader brin0 = new BufferedReader(
13                         new InputStreamReader(con0.
                               getInputStream()));
14         String inputLine;
15         while ((inputLine = brin0.readLine()) != null) {
               response0.append(inputLine);}
16         brin0.close();
17 }
18 id = getJson(response0.toString(),"id");
19 firstPart = '';
20 firstPart += '{' + '\n';
21 firstPart += '"id": "' + id + '",' + '\n';
22 firstPart += '"role": "writer",' + '\n';
23 firstPart += '"type": "user"' + '\n';
```

```
24 firstPart += '}' + '\n';
25 where = "https://www.googleapis.com/drive/v2/files/" + fid + "/
       permissions";
26 URL obj = new URL(where);
27 HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
28 con.setRequestMethod("POST");
29 con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
30 con.setRequestProperty("Content-Type","application/json");
31 con.setRequestProperty("Content-Length",firstPart.length().
       toString());
32 try
33 {
34         con.setDoOutput(true);
35         DataOutputStream wrs = new DataOutputStream(con.
              getOutputStream());
36         wrs.writeBytes(firstPart);
37         wrs.flush();
38         wrs.close();
39 }
40 catch(e){print e.toString();}
41 int responseCode = con.getResponseCode();
```

After this procedure is done, the user may edit the document.

4.5.5 Uploading a file to a cloud service

The procedure is almost the same as was described in section 4.2.4. There are some special features though.

Uploading to Etherpad is essentially simplified, there is a special Etherpad client object that does almost all the necessary work, as can be seen in listing 4.18.

Listing 4.18: Uploading to Etherpad

```
1 String API_URL = "";
2 String API_TOKEN = "";
3 try{
4         h = new getConnectionParameters();
5         l = h.getEtherpad();
```

```
6           b = l.split("%:%").toList();
7           API_URL = b[0];
8           API_TOKEN = b[1];
9           }
10  catch(ee){}
11  ext = fileName.substring(fileName.lastIndexOf('.')+1);
12  ext = ext.toLowerCase();
13  EPLiteClient api = new EPLiteClient(API_URL,API_TOKEN);
14  try{
15          tt = readFile(fileName);
16          api.createPad(idX, tt)
17
18          r = 'oK-' + idX;
19          }
20  catch(e){r = e.toString();}
```

In case of Google Drive, a format conversion is needed. Google Drive works with a number of native formats:

- Documents: For creating text-based files.
- Spreadsheets: For processing data in a form of tables.
- Presentations: For creating sequences of slides.
- Drawings: For processing vector graphics illustrations as well as diagrams.

As a consequence, TC files must be converted into one of these formats before uploaded. For example a *.doc file must be uploaded as a "Google Document", a *.ppt file as a "Google Presentation", and so on.

4.5.6 Fetching and converting a particular file from a cloud service

The procedure of downloading files is almost the same as was described in section 4.2.5, a file content can be read from a particular URL as a stream. There are some special features that are imposed by necessity to convert files into different formats.

Conversion using Etherpad is simple because of using the EPLiteClient object as can be seen in listing 4.19.

Listing 4.19: Conversion using Etherpad

```
String API_URL = "";
String API_TOKEN = "";
try
{
        h = new getConnectionParameters();
        l = h.getEtherpad();
        b = l.split("%:%").toList();
        API_URL = b[0];
        API_TOKEN = b[1];
}
catch(ee){}
ext = fileName.substring(fileName.lastIndexOf('.')+1);
ext = ext.toLowerCase();
EPLiteClient api = new EPLiteClient(API_URL,API_TOKEN);
r = '';
try
{
        if(ext == 'txt')
        {
                HashMap pad = api.getText(idX);
                r = pad.toString();
                r = r.substring(r.indexOf('=')+1,r.length()-1);
                writeFile(fileName,r);
                r = 'oK-';
        }
        if(ext == 'html' || ext == 'htm')
        {
                HashMap pad = api.getHTML(idX);
                r = pad.toString();
                r = r.substring(r.indexOf('=')+1,r.length()-1);
                writeFile(fileName,r);
                r = 'oK-';
        }
        if(ext == 'pdf')
        {
                r = 'oK-';
                url = API_URL;
                url = url.substring(0,url.indexOf("/",9)+1);
                url = url + "p/" + idX + "/export/pdf";
                readURLBinary(url,fileName);
        }
}
catch(e){r = e.toString();}
```

```
44 print r;
```

Google API works a bit differently, it returns a number of links for one file. These different URLs can be used to download one and the same file in different formats. As soon as such links are known, a normal procedure for downloading binary streams can be used. The MS OneDrive API works similar to Google, it gives the link for downloading any MS Office file as a PDF file.

4.6 Defining and sharing a course schedule - Implementation

4.6.1 Functionality

The scenario is implemented as import and export facilities of the "calendar" TC component. The calendar component is used to define all time-related course events. The calendar can be seen as a list of events, each having a title, description, beginning and finishing time as well as an optional geographic location.

Teachers may define new events and edit or delete existing events. The course calendar is available for all learners as a part of the course site. Cloud services are used to simplify distribution of information about events by means of exporting events from a course calendar to the personal calendars of users as can be seen in figure 4.10 and synchronizing personal calendars with a course calendar. Moreover a Google account set by a teacher may be used to import events from the teachers account and synchronize a course account with a teacher account. Users set the Google account to be accessed by TC as a part of their user profiles, and the system provides export, import and synchronization functionality.

As learners see a course calendar, a button called "Import" is available. Import can be done in the format of an exchange file (iCalendar file) or directly to a Google cloud calendar. Users may export a whole calendar or individual events and optionally synchronize a number of calendars. In

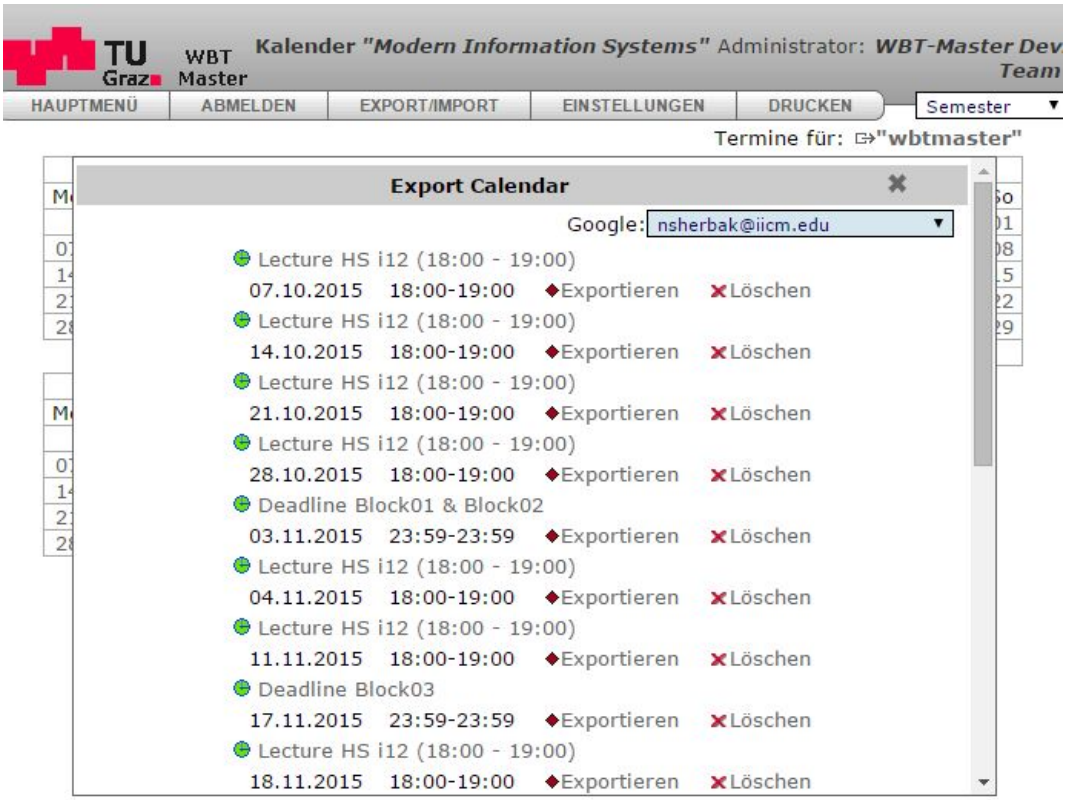

Figure 4.10: Exporting Events into a personal Google calendar

a similar way, a teacher may import events from his personal calendar or synchronize calendars in the sense that all new events added to a personal Google Calendar will be automatically added to the course calendar.

4.6.2 Software Architecture

The functionality is implemented using two different architectural solutions. Synchronization of personal user calendars with a course calendar is done by a Groovy script that is called as a "Cron Job" in accordance with a predefined job schedule.

The script performs the following actions:

- Read current content of the course calendar and content of the personal calendar.
- Check every event from the course calendar whether it is available in the personal calendar.
- Check time stamps for each event.
- If there is a new or newer version of the event in the course calendar, upload it into the personal calendar.

The synchronization of a teacher calendar is carried out in a similar way, but in opposite direction - events from the personal calendar are copied into the course calendar. Copying events is done by a number of JavaScript functions. The functions communicate to the server by synchronous and asynchronous XMLHTTP requests as described in section 4.2.2. The functions are supposed to perform the final data processing on client-side, and provide a GUI. Access to TeachCenter database and communication to Google Calendars are carried out by a number of Groovy scripts residing on the TeachCenter server. The most important functions for defining and sharing a course schedule scenario performed by Groovy scripts are:

- Setting a particular Google account to be used by TeachCenter.
- Downloading all the events from a Google Calendar.
- Uploading a particular event into a Google Calendar.

- Downloading a particular event and adding it to the course calendar. This operation simply takes info from a list of events and places a new event in the course calendar. Since this procedure is mainly TC internal, it will not be described further.

4.6.3 Setting a particular Google user account to be used by TeachCenter

The authentication procedure for using Groovy Calendar is almost the same as was described in section 4.2.3. The only difference is that the set of requested permissions include Google Calendar.

4.6.4 Downloading all the events from a Google Calendar

This operation is performed by sending a GET request to a special API endpoint. The end point refers to a particular Google Calendar and contains a minimum date for returning events. Parts of the source code can be seen in listing 4.20.

Listing 4.20: Getting events from Google Calender

```
1  url = 'https://www.googleapis.com/calendar/v3/calendars/' +
       calendarID + '/events?timeMin=' + timeMin;
2  URL obj = new URL(url);
3  HttpURLConnection con = (HttpURLConnection) obj.openConnection
       ();
4  con.setRequestMethod("GET");
5  con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
6  JSONParser parser=new JSONParser();
7  JSONObject jsonObj = new JSONObject();
8  responseCode = con.getResponseCode();
9  if(responseCode<=200)
10 {
11         BufferedReader brin = new BufferedReader(
12         new InputStreamReader(con.getInputStream()));
13         String inputLine;
14         StringBuffer response = new StringBuffer();
15
```

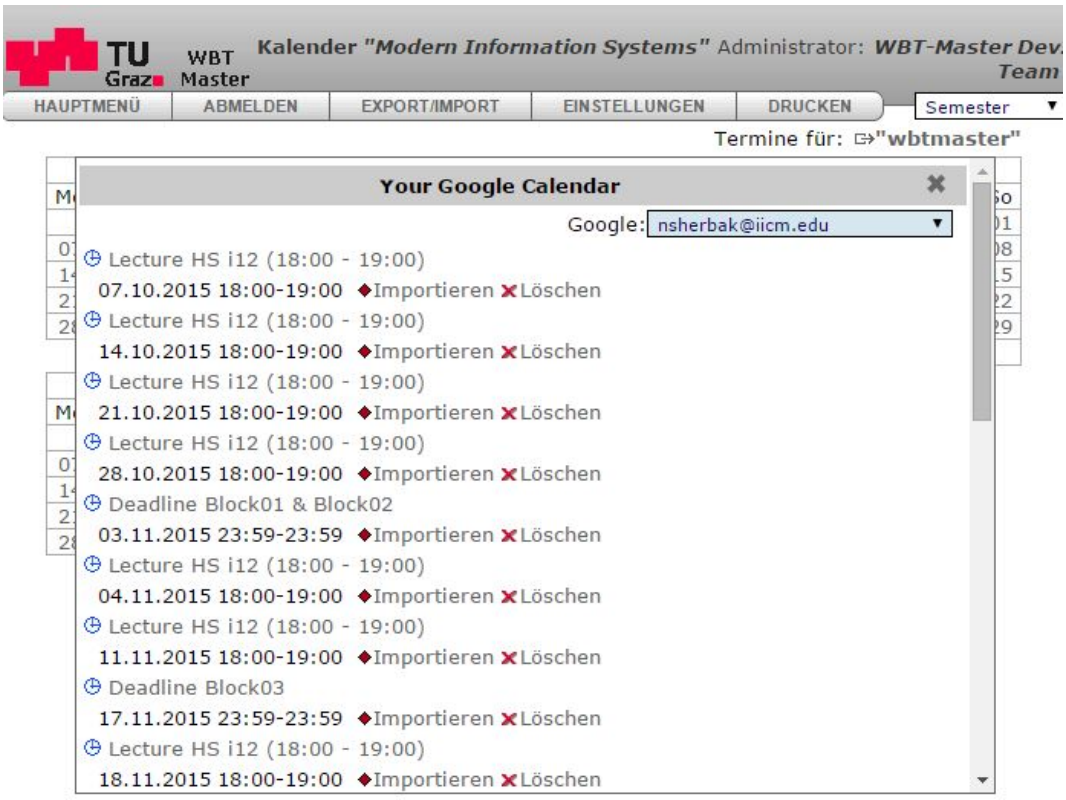

Figure 4.11: Visualizing a Google Calendar list for importing events

```
16          while ((inputLine = brin.readLine()) != null) {response
                .append(inputLine);}
17          brin.close();
18          jsonObj = parser.parse(= response.toString());
19          jsonList = jsonObj.items;
```

The result is returned in form of a JSON list. The list is visualized for users to import all or selected events as can be seen in figure 4.11.

4.6.5 Uploading a particular event into a Google calendar

This operation is performed by sending a POST JSON request to a special API end point. The JSON text contains all the parameters needed for a new event to be created on the cloud calendar. Parts of the source code can be seen in listing 4.21.

Listing 4.21: Uploading an event into a Google Calender

```
1  firstPart = '';
2  JSONObject json = new JSONObject();
3  json.put("summary", TitleOfEvent);
4  json.put("description", DescriptionOfEvent);
5  JSONObject json1 = new JSONObject();
6  JSONObject json2 = new JSONObject();
7  json1.put("dateTime",dateBegin);
8  json2.put("dateTime",dateEnd);
9  json.put("start", json1);
10 json.put("end", json2);
11 json.put("location",eventLocation);
12 firstPart = json.toJSONString();
13 ll = firstPart.length().toString();
14 where = "https://www.googleapis.com/calendar/v3/calendars/"+
       calendarID + "/events";
15 URL obj = new URL(where);
16 HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
17 con.setRequestMethod("POST");
18 con.setRequestProperty("Authorization", "Bearer " + whole_token
       );
19 con.setRequestProperty("Content-Type","application/json");
20 con.setRequestProperty("Content-Length",ll);
21 try
22 {
23         con.setDoOutput(true);
24         DataOutputStream wrs = new DataOutputStream(con.
               getOutputStream());
25         wrs.writeBytes(firstPart);
26         wrs.flush();
27         wrs.close();
28 }
29 catch(e){rr = e.toString();}
30 int responseCode = con.getResponseCode();
31 StringBuffer response00 = new StringBuffer();
```

```
32 response_string00 = '';
33 BufferedReader brin00 = new BufferedReader(new
       InputStreamReader(con.getInputStream()));
34 String inputLine00;
35 if(responseCode<=200)
36 {
37         while ((inputLine00 = brin00.readLine()) != null)
38         {
39                 response_string00 += inputLine00.toString();
40         }
41 }
```

4.7 Sharing bookmarks - Implementation

4.7.1 Functionality

The scenario is implemented as an import export facility of the "shared bookmarks" TC component. The "shared bookmarks" component is used to define a set of bookmarks to be used by all users. A particular bookmark can be:

- Organized into folders.
- Provided with a description.
- Commented by users.

The shared bookmarks are available for all learners as part of the course site. The cloud service Firefox Pocket is used here to simplify distribution of information about bookmarks by exporting bookmarks from a course repository into the personal Firefox Pocket as can be seen in figure 4.12, and by synchronizing personal lists of bookmarks with such a Firefox Pocket. In addition a Firefox Pocket account set by a teacher may be used to import bookmarks from the teacher's browser, or to synchronize course bookmarks with the teacher's Firefox Pocket. Users may set the Firefox Pocket account to be accessed by TC as a part of their user profiles. Subsequently, the system provides export and import as well as synchronization functionality.

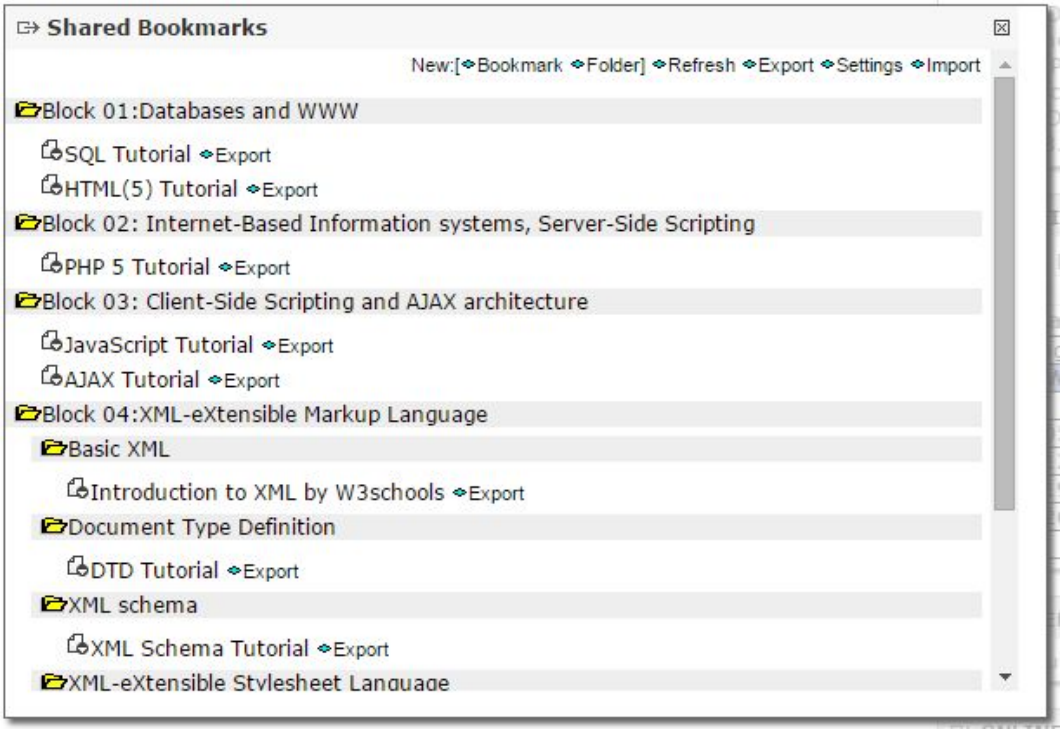

Figure 4.12: Exporting bookmarks into a personal Firefox Pocket

As learners access the shared bookmarks, a button named ”Export” is available. Export can be done in the format of a bookmarks exchange file, which basically is a HTML file, or directly to a Firefox Pocket. Users may export a whole bookmark repository or individual bookmarks, and optionally synchronize bookmarks.

In a similar way, a teacher may import bookmarks from a personal Firefox Pocket or synchronize bookmarks, in the sense that all new bookmarks added to the teacher's personal Firefox Pocket will be automatically added to the course bookmarks.

4.7.2 Software Architecture

The functionality is implemented using two different architectural solutions. Synchronization of a personal Firefox Pocket with course bookmarks is done by a Groovy script that is called as a ”Cron Job” in accordance with a predefined job schedule.

The script performs the following actions:

1. Read the content of the course bookmarks and the content of a personal Firefox Pocket.
2. Check every entry from course bookmarks whether it is available in the personal Firefox Pocket.
3. If there is a new bookmark in the list of course bookmarks, it is uploaded to the personal Firefox Pocket.

The synchronization of a teacher's personal Firefox Pocket is carried out in a similar way, but in opposite direction - entries from the personal Firefox Pocket are copied into the course list of bookmarks.

Actual copying of bookmarks is done by a number of JavaScript functions. The functions communicate to the server by synchronous and asynchronous XMLHTTP requests as described in section 4.2.2. The functions are supposed to perform the final data processing on client-side, and provide a GUI. Access to TeachCenter database and communication to Firefox Pockets are carried out by a number of Groovy scripts residing on the TeachCenter

server. The most important functions for the sharing bookmarks scenario performed by Groovy scripts are:

- Setting a particular Firefox Pocket account to be used by TeachCenter.
- Downloading bookmarks from a Firefox Pocket.
- Uploading a particular bookmark to Firefox Pocket.

4.7.3 Setting a particular Firefox Pocket account to be used by TeachCenter

The authentication procedure for using Firefox Pocket is almost the same as was described in section 4.2.3. Users are requested to access a Firefox Pocket using a special URL and confirm the permission to use the account by TeachCenter.

4.7.4 Downloading all bookmarks from Firefox Pocket

This operation is performed by sending a POST JSON request to a special API end point. The JSON text contains all the parameters needed to retrieve the list of bookmarks - consumer key, access token, and so on. Parts of the source code for this operation can be seen in listing 4.22.

Listing 4.22: Downloading bookmarks from Firefox Pocket

```
1  where = "https://getpocket.com/v3/get";
2  response_string = '';
3  response_string = '';
4  JSONObject json = new JSONObject();
5  json.put("consumer_key", consumer_key);
6  json.put("access_token",code);
7  json.put("count",100);
8  json.put("detailType","simple");
9  firstPart = json.toJSONString();
10 ll = firstPart.length().toString();
11 URL obj = new URL(where);
12 HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
13 con.setRequestMethod("POST");
```

```
14 con.setRequestProperty("Content-Type","application/json;
       charset=utf-8");
15 con.setRequestProperty("X-Accept","application/json");
16 con.setRequestProperty("Content-Length",ll);
17 try
18 {
19         con.setDoOutput(true);
20         DataOutputStream wrs = new DataOutputStream(con.
               getOutputStream());
21         wrs.writeBytes(firstPart);
22         wrs.flush();
23         wrs.close();
24 }
25 catch(e){print e.toString();}
```

The result is returned in the form of a JSON list. The list is visualized for users to choose which bookmarks to import as can be seen in figure 4.13.

4.7.5 Uploading a particular bookmark to Firefox Pocket

This operation is performed by sending a POST JSON request to a special API end point. The JSON text contains all the parameters of a new bookmark to be created on the cloud service. Parts of the source code for this operation can be seen in listing 4.23.

Listing 4.23: Uploading a bookmark to Firefox Pocket

```
1  where = "https://getpocket.com/v3/add";
2  response_string = '';
3  JSONObject json = new JSONObject();
4  json.put("consumer_key", consumer_key);
5  json.put("access_token",code);
6  json.put("url",urlX);
7  json.put("title",titX);
8  json.put("time",getCurrentUnixDate());
9  firstPart = json.toJSONString();
10 content_length = firstPart.length().toString();
11 URL obj = new URL(where);
12 HttpsURLConnection con = (HttpsURLConnection) obj.
       openConnection();
13 con.setRequestMethod("POST");
```

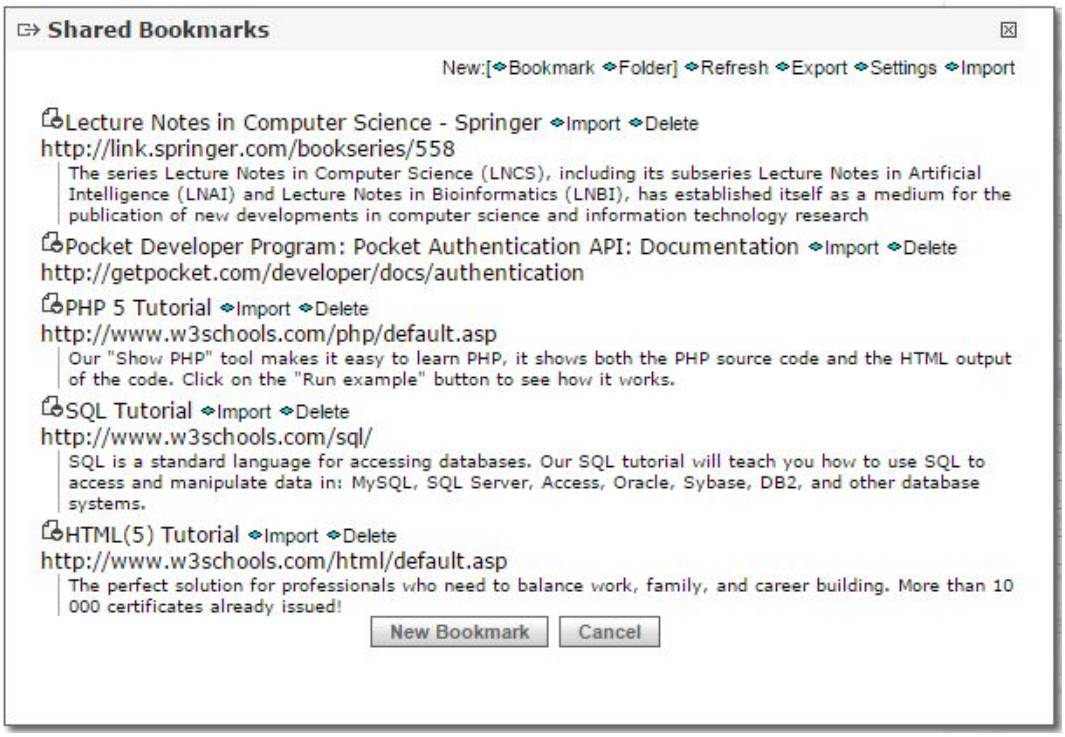

Figure 4.13: Choice of Firefox Pocket bookmarks to import into TC

```
14 con.setRequestProperty("Content-Type","application/json;
       charset=utf-8");
15 con.setRequestProperty("X-Accept","application/json");
16 con.setRequestProperty("Content-Length",content_length);
17 try
18 {
19         con.setDoOutput(true);
20         DataOutputStream wrs = new DataOutputStream(con.
               getOutputStream());
21         wrs.writeBytes(firstPart);
22         wrs.flush();
23         wrs.close();
24 }
25 catch(e){print e.toString();}
```

5 Discussion

Cloud services are getting increasingly popular in different areas of computer science. At the same time usage of cloud services in e-learning leaves much to be desired. In order to fill this gap a number of popular cloud services were integrated into a modern LMS as a result of this work. All of this cloud services were described in chapter 2. A number of scenarios for embedding of cloud services into the framework of e-learning were described in chapter 3. Implementations were described in detail in chapter 4. When talking about acceptance of these solutions by users it has to be taken into account that the implementation took a number of years. Historically, the first solution was downloading materials from TC to the personal Dropbox. This solution was implemented two years ago and more than 500 learners utilized this add-on to download files from TC. This number definitely indicates acceptance of this solution by learners. Approximately at the same time, a scenario for uploading materials from a personal Dropbox was implemented as well, here more modest figures can be seen. Only 20 teachers use this solution for providing materials to students. There is a simple explanation for this: first, the total number of teachers is much smaller than the number of students and second, teachers are more reluctant to creating accounts and use cloud services than students.

In addition about one year ago, both scenarios were implemented using the Google Drive cloud service. This service was also eagerly accepted by students and in less than one year about 160 accounts were created. Unfortunately in May 2015 Google made the decision to switch to a new authorization paradigm - OAuth 2.0 and stopped support for the old authorization procedure. Thus all user accounts that used Google Drive on TC had to be deleted and the authorization procedure for the cloud service had to be implemented. It had a negative effect on user experience. Many users

attempts to access their old accounts were monitored but new accounts were not created despite of the advice to do so.

The Interface to the Etherpad cloud service was also implemented simultaneously with the interface to Dropbox two years ago. The exact number of users using this add-on cannot precisely be measured, because Etherpad does not require user registration. Nevertheless it can be seen that usage of this cloud service is much less than the usage of Dropbox and Google Drive. It has also a simple explanation; the service is implemented only for collaborative authoring of textual documents and the number of users is determined by teachers who use this scenario of collaborative authoring in their courses. The number of such teachers is very low and log files only show five courses using this scenario.

Other scenarios and interfaces to MS OneDrive, Firefox Pocket and Google Calendar were implemented during 2015 and will be offered to actual users in winter term 2015/16. As a result actual figures of users cannot be provided. As previously provided figures of users using different scenarios are analysed, it can be seen that if using a certain scenario is up to each particular student as for example the download of materials into a user's cloud service, acceptance is rather high. A lot of students now possess accounts in one or two popular cloud services, and students are open for usage of external cloud services. If usage of a scenario is up to the teacher, for example uploading materials to TC or collaborative authoring of documents, acceptance is dropping. Teachers are much more reluctant in creating accounts in cloud services and in usage of such.

6 Outlook

In this chapter possible further developments as well as existing problems are described.

The user number of teachers using the cloud service based solutions leaves a big area for innovation. The number of teachers using these services needs to be increased. It can be done by:

- Providing well elaborated advertisement and documentation materials. Teachers must be easily provided with information about advantages that are provided as a result of usage of cloud services. At the same time usage of such cloud services should not require any additional cognitive overhead.
- Teachers must be provided with software wizards that in a dialogue mode explore teacher preferences and offer fitting solutions based on cloud services.
- Teachers must be widely notified using emails, news services, video clips and pictures about existence of such new features in TC.

Since the acceptance of comparatively new scenarios such as shared course calendars and shared bookmarks is unclear yet, monitoring the user activity with these components should be continued and final conclusions should be made on the basis of evaluation results.

Another issue which should be taken into account is the appearance of new cloud services useful for e-learning. All of these cloud services must be thoroughly investigated and offered to users to come to a final conclusion whether and how the services can be used. An entirely different aspect of using cloud services in e-learning should be mentioned. For example it is easy to see that most cloud services are simple repositories of data with tools for processing data online, any LMS falls in the same category, it is

also a repository of training materials with special tools for processing these materials. Sufficient difference between a modern cloud service and an LMS is that materials in the LMS can be processed only in online mode, while materials in cloud service can be processed by a number of application using the cloud services API.

Therefore development of a rich and secure API for a Learning Management System may provide a solid basis for developing of a wide range of applications working online and offline. For example a TC standalone application that may reside on a local computer and provide all necessary functionality of TeachCenter courses can be imagined, making announcements, uploading materials, downloading materials, uploading assignments, making questionnaires and so on. There is also a legal aspect of the usage of cloud services in e-learning, many teachers are reluctant in putting their materials onto external servers. Despite that from the legal perspective keeping individual materials on a local hard drive does not make any difference with keeping these materials on a cloud service. In our approach each user individually decides where to keep training materials after downloading or before uploading. Therefore the teachers may keep data on their hard drive or upload it to a server. There is no difference in keeping materials on a cloud drive and copying them to TC. The only result of using cloud service in this respect is the much easier procedure for uploading data to a production server, the same situation as with learner accounts. Learners decide themselves whether they want to download materials on a local drive or their cloud service space. As before, the only difference is the much easier access and processing of downloaded materials. Nevertheless to avoid any speculations about legal aspects of keeping materials on a remote server, cloud services that may be installed within a local area network of a particular university are of great interest. We can mention OneDrive, Etherpad, OwnCloud and many others. Cloud services residing within one LAN cannot create legal problems by definition.

7 Conclusion

The role and possible usage of cloud services in a modern LMS was thoroughly investigated in this thesis. An architecture of packages integrated in the LMS and reusing functionality of cloud services via a respectful Application Programming Interface was developed. A user interface solution providing seamless integration of cloud services in the natural functionality of a modern LMS was realized and evaluated with a number of users. Software based on modern software engineering principals, practically integrating functionality of cloud services into a particular LMS, was implemented. As soon as all the above mentioned solutions were embedded in a Learning Management System heavily used in real university environment, possible advantages and disadvantages of such solutions were investigated.

This thesis generally reports many years of practical work on populating a complex interface of a modern LMS with popular cloud services and results achieved during this work. As can be seen in chapter 5 not all implementations were equally accepted by users. Ultimate success was seen in the realization of two scenarios known as:

1. Downloading training materials from TC
2. Uploading training materials to TC

Cloud service based solutions for other scenarios were not used as often. Explanation for this was also provided in chapter 5. All software implementations presented in this thesis are currently an integral part of the TeachCenter, which is used at Graz University of Technology as the main e-learning solution.

Bibliography

Ally, Mohamed, Margarete Grimus, and Martin Ebner (2014). "Preparing teachers for a mobile world, to improve access to education." In: vol. 44. 1. Netherlands: Prospects, Springer, pp. 43–59. URL: http://link.springer.com/article/10.1007/s11125-014-9293-2 (cit. on p. 5).

Andrews, Keith, Frank Kappe, and Hermann Maurer (1996). "The Hyper-G network information system." In: Berlin, Heidelberg: J. UCS The Journal of Universal Computer Science, Springer, pp. 206–220. URL: http://link.springer.com/chapter/10.1007/978-3-642-80350-5_20#page-1 (cit. on p. 5).

Andrews, Keith, Frank Kappe, Hermann Maurer, and Klaus Schmaranz (1994). "On Second Generation Hypermedia Systems." In: J. UCS, pp. 127–135. URL: http://jucs.org/jucs_0_0/on_second_generation_hypermedia/Andrews_K.pdf (cit. on p. 5).

Augar, Naomi, Ruth Raitman, and Wanlei Zhou (2004). "Teaching and learning online with wikis." In: Perth, Australia: Beyond the comfort zone : proceedings of the 21st ASCILITE Conference, pp. 95–104. URL: http://dro.deakin.edu.au/eserv/DU:30005482/zhou-teachingandlearning-2004.pdf (cit. on p. 5).

Dietinger, Thomas and Hermann Maurer (2014). "GENTLE (GEneral Networked Training and Learning Environment)." In: (cit. on p. 5).

Downes, Stephen (2005). "e-Learning 2.0." In: *ACM e-Learn Magazine* 10. URL: http://elearnmag.acm.org/featured.cfm?aid=1104968 (cit. on p. 5).

Drago, Idilio et al. (2012). "Inside dropbox: understanding personal cloud storage services." In: Proceedings of the 2012 ACM conference on Internet measurement conference, pp. 481–494. URL: http://dl.acm.org/citation.cfm?id=2398827 (cit. on pp. 9, 13).

Ebner, Martin (2007). "E-Learning 2.0 = e-Learning 1.0 + Web 2.0." In: The Second International Conference on Availability, Reliability and Security. ISBN: 0-7695-2775-2 (cit. on p. 5).

Ebner, Martin (2013). "The influence of Twitter on the academic environment." In: IGI Global, pp. 491–498 (cit. on p. 5).

Ebner, Martin, Andreas Holzinger, et al. (2011). "EduPunks and Learning Management Systems–Conflict or Chance?" In: Berlin, Heidelberg: Hybrid Learning, Springer, pp. 224–238. URL: http://link.springer.com/chapter/10.1007/978-3-642-22763-9_21#page-1 (cit. on p. 6).

Ebner, Martin, Nikolai Scerbakov, and Hermann Maurer (2006). "New Features for eLearning in Higher Education for Civil Engineering." In: vol. 1. 1. Journal of Universal Science and Technology of Learning, pp. 93–106. URL: http://www.jucs.org/justl_0_0/new_features_for_elearning/justl_0_0_0093_0106_ebner.pdf (cit. on p. 5).

Ebner, Martin and Ulrich Waldner (2008). "New Features for eLearning in Higher Education for Civil Engineering." In: Vilnius: Journal of Universal Science and Technology of Learning, pp. 16–26. URL: http://lamp.tu-graz.ac.at/~i203/ebner/publication/08_vilnius.pdf (cit. on p. 5).

Etherpad of Graz University of Technology (2015). URL: https://etherpad.learninglab.tugraz.at/index.html (visited on 20/09/2015) (cit. on p. 18).

Evans, Chris (2008). "The effectiveness of m-learning in the form of podcast revision lectures in higher education." In: vol. 50. Computers & Education, pp. 491–498. URL: http://www.sciencedirect.com/science/article/pii/S0360131507001182 (cit. on p. 5).

Farmer, James and Anne Bartlett-Bragg (2005). "Blogs anywhere: High fidelity online communication." In: Proceedings of ASCILITE 2005: Balance, Fidelity, Mobility: maintaining the momentum? Pp. 197–203. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.2120&rep=rep1&type=pdf (cit. on p. 5).

Hao, Fang et al. (2009). "Enhancing dynamic cloud-based services using network virtualization." In: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, pp. 37–44. URL: http://dl.acm.org/citation.cfm?id=1592655 (cit. on p. 8).

Helic, Denis, Hermann Maurer, and Nikolai Scerbakov (2004). "Knowledge transfer processes in a modern WBT system." In: vol. 27. 3. Journal of

Network and Computer Applications, pp. 163–190. URL: http://www.sciencedirect.com/science/article/pii/S1084804503000559 (cit. on p. 5).

Hu, Wenjin, Tao Yang, and Jeanna N. Matthews (2010). "The good, the bad and the ugly of consumer cloud storage." In: vol. 44. 3. ACM SIGOPS Operating Systems Review, pp. 110–115. URL: http://dl.acm.org/citation.cfm?id=1842751 (cit. on p. 9).

Klamma, Ralf et al. (2007). "Social software for life-long learning." In: vol. 10. 3. Journal of Educational Technology & Society, pp. 72–83. URL: http://www.jstor.org/stable/jeductechsoci.10.3.72 (cit. on p. 5).

Lonn, Steven and Stephanie D. Teasley (2009). "Saving time or innovating practice: Investigating perceptions and uses of Learning Management Systems." In: vol. 53. 3. Netherlands: Prospects, Springer, pp. 686–694. URL: http://link.springer.com/article/10.1007/s11125-014-9293-2 (cit. on p. 1).

Maurer, Hermann (1996). *HyperWave: The Next Generation Web Solution.* Addison-Wesley Longman (cit. on p. 4).

Maurer, Hermann and Nikolai Scerbakov (1996). *Multimedia Authoring for Presentation and Education: The Official Guide to HM-Card.* Addison-Wesley (cit. on p. 5).

Mell, Peter and Timothy Grance (2011). "The NIST definition of cloud computing." In: URL: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf (cit. on pp. 8, 15).

Nishantha, G.G.D. et al. (2009). "CURRENT USAGE AND FUTURE TRENDS OF LEARNING MANAGEMENT SYSTEMS: A CASE STUDY IN ASIA PACIFIC UNIVERSITY." In: *INTED2009 Proceedings*. 3rd International Technology, Education and Development Conference. Valencia, Spain: IATED, pp. 948–958 (cit. on p. 4).

O'Reilly, Tim (2006). "Web 2.0: Stuck on a name or hooked on value?" In: *Dr. Dobbs Journal*, pp. 10–10 (cit. on p. 4).

Raitman, Ruth, Naomi Augar, and Wanlei Zhou (2005). "Employing wikis for online collaboration in the e-learning environment: Case study." In: Proceedings of the third international. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1488944 (cit. on p. 5).

Ramsay, Judith, Alessandro Barbesi, and Jenny Preece (1998). "A psychological investigation of long retrieval times on the World Wide Web."

In: vol. 10. 1. Interacting with computers, pp. 77–86. URL: http://iwc.oxfordjournals.org/content/10/1/77.short (cit. on p. 6).

Emerging Web Technologies in Higher Education: A Case of Incorporating Blogs, Podcasts and Social Bookmarks in a Web Programming Course based on Students' Learning Styles and Technology Preferences. (2009). Vol. 12. 4, pp. 98–109. URL: http://www.jstor.org/stable/jeductechsoci.12.4.98 (cit. on p. 3).

Scerbakov, Alexei, Martin Ebner, and Nikolai Scerbakov (2015). "Using Cloud Services in a Modern Learning Management System." In: *Journal of Computing and Information Technology*, pp. 75–86. URL: http://hrcak.srce.hr/file/199949 (cit. on p. 1).

Schaffert, Sandra and Martin Ebner (2010). "New Forms of and Tools for Cooperative Learning with Social Software in Higher Education." In: Computer-Assisted Teaching: New Developments. Nova Science Pub, pp. 151–165. URL: https://www.researchgate.net/profile/Martin_Ebner2/publication/257366452_New_Forms_of_and_Tools_for_Cooperative_Learning_with_Social_Software_in_Higher_Education/links/02e7e52f9e5805fe2d000000.pdf (cit. on p. 5).

Stantchev, Vladimir et al. (2014). "Learning management systems and cloud file hosting services: A study on students' acceptance." In: vol. 31. Computers in Human Behavior, pp. 612–619. URL: http://www.sciencedirect.com/science/article/pii/S0747563213002409 (cit. on p. 9).