Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Viban Terence Yuven

Harnessing The Power Of Web 2.0 For Providing A Rich User Experience In An eLearning Application

Fakultät Technik und Informatik
Studiendepartment Informations-und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Viban Terence Yuven

Harnessing The Power Of Web 2.0 For Providing A Rich User Experience In An eLearning Application

Master thesis based on the examination and study regulations for the
Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Thomas C. Schmidt
Second examiner: Prof. Dr. Hans-Jürgen Hotop

Date of delivery March 1, 2010

Viban Terence Yuven

Thema der Master Thesis

Ausnutzung der Stärken des Web 2.0. für eine verbesserte Benutzererlebnis in einer eLearning Anwendung

Stichworte

Ajax, annotation, Dojo, eLearning, Feedback, hyLOs, Google, Links, notes, tagging, Web 2.0, Wikipedia

Zusammenfassung

Die Lernprozesse sind aufgrund der Zunahme von neuen Technologien und Paradigmen der Programmierung im Web umfassender geworden. Diese Technologien bieten Möglichkeiten, um Inhalte mehrerer Quellen zu verwenden und neu zusammenzusetzen und dabei Inhalte in einer Form zu schaffen und zur Verfügung zu stellen, die wiederum von anderen genutzt und wiederaufbereitet werden können. Diese Master Thesis schafft eine eLearning Plattform, die ihren Nutzern Möglichkeiten zum offenen und flexiblen gemeinschaftlichen eLearning bereitet. Diese Plattform kann dazu dienen, Layers mit neuem Inhalt aus abgerufenem Inhalt aus dem Web zu erzeugen und hinzuzufügen. Der hervorgebrachte Inhalt kann mit anderen Nutzern und Gruppen geteilt werden. Ziel dieser eLearning Plattform wird es auch sein, eine sehr reichhaltige Nutzererfahrung zu schaffen, indem der Nutzer mit einer höchst interaktiven Bedieneroberfläche ausgestattet wird.

Viban Terence Yuven

Title of the paper

Harnessing The Power Of Web 2.0 For Providing A Rich User Experience In An eLearning Application

Keywords

Ajax, annotation, Dojo, eLearning, feedback, hyLOs, Google, links,notes, tagging, Web 2.0, Wikipedia

Abstract

The processes of learning have become broader due to the rise of new technologies and programming paradigms on the Web. These technologies offer possibilities for consuming and remixing content from multiple sources, while creating and providing content in a form that can be consumed and remixed by others. This master thesis builds an eLearning platform that offers possibilities for open and flexible collaborative eLearning, which can be used to create and add layers of new content to content retrieved from the Web. The content created can be shared with other users and groups. This eLearning platform will also aim at achieving a very rich user experience by supplying the user with a highly interactive user interface.

Contents

List of Figures

Listings

List of Abbreviations

A	
AJAX	Asynchronuos Javascript And XML
A	
API	Application Programming Interface
A	
ASP	Active Server Pages
C	
CSS	Cascading Style Sheets
D	
DB	Database
DBA	Database Admnistrator
DnD	Drag and Drop
DOM	Document Object Model
H	
HTML	HyperText Markup Language
hyLOs	Hypermedia eLearning Object Systems
I	
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
J	
JSF	Java Server Faces
JSP	Java Server Pages
JS	JavaScript

JSON	JavaScript Object Notation

L

LOM	Learning Object Metadata

M

MIT	Massachusetts Institute of Technology

O

OATS	Open Annotation and Tagging System
OCW	OpenCourseWare
OLTP	OnLine Transaction Proccessing

P

PHP	Personal Home Page (Hypertext Preprocessor)
PHP PDT	PHP Development Tools Project

S

SQL	Structured Query Language
SVG	Scalable Vector Graphics

X

XHTML	eXtensible HyperText Markup Language

U

UI	User Interface
URL	Uniform Resource Locator

Chapter 1

Introduction

The growth of the Internet as a global data communication system and the Web as one of its main services has led to a border-less world where distance and time are no longer an issue when it comes to eLearning. This globalization has led to an inter-networked community which allows massive transmission of text, images, sound etc. The Web now holds a huge amount of information and educational content, some of which is free and from trusted sources, e.g. some online course materials offered by most renowned universities under the name *OpenCourseWare (OCW)*[1] is free of charge and of high educational value. Another example of a major information source on the Web is Wikipedia, which is a free, collaborative, multilingual encyclopedia with about 13 million articles on different subjects. Search engines of which Google is the most widely used, can be used to sift through this information when information on a particular subject is required. However, given the amount of information available on the Web today, gathering and retaining information can sometimes be very tedious.

One other important issue that has affected eLearning in the past decade is the revolution in technology. One of such technologies which focus on services rather than software is Web 2.0. It has transformed the Web into an appropriate platform for development of highly interactive information systems and *groupware* applications. For example, a Web 2.0 site allows its users to interact with other users or to change website content, in contrast to non-interactive websites where users are limited to the passive viewing of information that is provided to them. Content can be found easily using keywords. This is usually possible due to the categorization of content by users adding tags, which are short, usually one-word descriptions to facilitate searching, without dependence on pre-made categories. Collections of tags created by many users within a single system may be referred to as *folksonomies*. The ability to create and update content leads to the collaborative work of many rather than just a few Web authors. In wikis, users may extend, undo and redo each other's work. In blogs, posts and the comments of individuals build up over time. These resources (wikis, blogs,

[1] MIT was one of the first to begin with OCW, http://ocw.mit.edu/OcwWeb/web/home/home/index.htm

forums, etc.) can be offered to the users easily to expand their learning environment and experience. Links can be used to connect information together into a meaningful information ecosystem using the model of the Web. In eLearning media of similar type and information from multiple sources can be combined into a single representation. With the aid of special programming APIs, frameworks and toolkits, services and applications can be built that are independent of the end users accessing methods. Such services would run on all major browsers and are searchable and accessible over the network.

However, extensive use of technology does not always lead to better ways of learning. The challenge is designing simple yet powerful, open and flexible systems that use Web 2.0 technologies to gather, retain and interact with content from the Web in collaboration with other users based on shared interest. Such simple systems will normally use the Web browser as a client because it offers the ability to update and maintain applications without distributing and installing software on potentially thousands of client computers. Moreover browsers support cross-platform compatibility.

1.1 Project Overview

In the course of this thesis, an eLearning application which supports the idea of a virtual learning environment is built. This learning environment gives its users possibilities to gather and interact with information from the Web. During this process a couple of tools are used by the user, which enables the system to categorize and retain the location of the content.

The system gives users the opportunity to collaborate within the context of the group or to work alone. Communication within the group is transparent and open, but this is up to the user to decide. Group work concentrates on two main issues; content and collaboration. In this application, group work does not automatically mean collaboration. Users might decide to create and use group content without actively collaborating with others.

The system design also allows layers of new content to be created and added to existing content.

The system uses the Web browser as a platform for hosting its powerful and highly interactive user interface (UI). The UI is designed to display a lot of information at any one time in ways that are intuitive to the user.

The design issues considered when building this project, result in an application which offers content gathering and interaction capabilities to its users, provides them with an interactive user interface and offers flexibility in the way in which they use the application and collaborate with each other.

1.2 Thesis Outline

The thesis is outlined in such a way that, it begins with a brief theoretical background on concepts, which are important to this project. It then moves on to discuss the following software development stages: analysis, design, specification, implementation and testing of the system. It concludes with a summary of what has been achieved and offers suggestions on how the system can be improved and expanded.

Chapters 2 gives a brief introduction on the meaning eLearning. It introduces and discusses the IEEE LOM elearning standard used in one of the main content sources for this project - hyLOs. hyLOs is briefly introduced. This chapter completes by discussing the impact of collaboration in interactive group elearning.

Web 2.0 and one of its main technologies, Ajax, are the subject of chapter 3. This chapter begins with a brief introduction to Web 2.0 and Ajax and then discusses techniques, patterns and technologies used in Ajax programming.

Chapter 4 gives a complete requirement analysis of the system. It begins by using a use case diagram of the system tools to show what is expected of the system. Activity diagrams are used to show and discuss how content retrieval and user-content interaction are to be realised. Requirements analysis for group communication and user interface are also carried out in this chapter.

Chapter 5 discusses the system design. A general system architecture is discussed. This is followed by the design of the front-end and the back-end of the application. This chapter also examines system design constraints.

In chapter 6, the system specification is discussed. The functionality offered by the application is discussed as services. This chapter concludes with the discussion of the database model.

The next chapter 7 lays out the implementation of the design proposed in chapter 5. It begins with a short introduction to the design environment, followed by a general software architecture used and concludes with an explanation of the core functionality of the system. Source code snippets are given whenever possible otherwise precise references are made to the CD attached to this report.

After implementation, the results of the usability test carried out are presented in chapter 8. Screen shots are used to demonstrate the functionality and UI of the application. The test results are evaluated for the following browsers; Mozilla Firefox, Safari and Internet Explorer and the results documented with respect to each browser.

The summary of work carried out in the application is given in chapter 9. This chapter also gives suggestions on how the system can be improved based on the test results and also on how the system can be expanded to offer more useful functionality to its users.

Chapter 2

eLearning Within Interactive Groups

2.1 eLearning and eLearning 2.0: Background

eLearning is one of the most important educational innovations of the past 3 decades. eLearning can best be understood in the broader context of using technology to meet society's needs for learning [6]. Following the success of the internet and the connection possibilities it offered, eLearning was over-hyped leading to unrealistic expectations in the late 90s [2],. At the time, the hype simply outpaced the existing technology.

Today, the term eLearning 2.0 is being used to described the eLearning trend that has emerged and evolved as a result of Web 2.0 technologies (see chapter 3), which has brought a major change in the way people use the Internet. This trend manifested itself in the field of learning in what is called "learner-centered" or constructivist learning. In his article "Constructivism and Discovery Learning" [3], the basic tenets of constructivism are identified as follows:

- Knowledge is constructed from and shaped by experience.
- Students must take an active role and assume responsibility for their learning.
- Learning is a collaborative process and students create their own meaning from obtaining multiple perspectives.
- Learning should occur in a realistic setting.
- Learners should choose their own path through content and activities.
- Content should be presented holistically, not broken into separate smaller tasks

[2] The State of eLearning: Looking at History with the Technology Hype Cycle by Kevin Kruse, http://www.e-learningguru.com/articles/hype1_1.htm

[3] Constructivism and Discovery Learning by Kevin Kruse, http://www.e-learningguru.com/articles/art3_6.htm

I disagree with the last point in the list above. In my opinion, users can better assimilate information when it is broken down into smaller units. These units are called eLearning objects and are discussed in detail in section 2.2.

2.2 ELearning Standards and IEEE Learning Object Metadata - LOM

Learning technology standards are important for the following reasons [4]:

- they can be used to mix and match content from many sources
- they can be used to develop interchangeable content that can be reused, assembled and disassembled quickly and easily
- they can be developed independent of vendor's proprietary learning technology
- they can lead to wise and risk adverse learning technology investments.

In the context of eLearning, terms which are central to many discussions and have led to some major standards include:

- learning objects
- learning object description via metadata
- learning object repositories

A working group was established by IEEE LTSC (Institute of Electrical and Electronics Engineers, Inc. Learning Technology Standards Committee) [5] a major standard body to work towards the standardization of learning objects. This working group defines learning objects as follows: "Learning Objects are any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning".

The second important aspect is metadata. Metadata literally means "data about data". The Learning Object Metadata (LOM) group [6] has developed and standardized LOM standard which focuses on the minimal set of attributes needed to allow Learning Objects to be managed, located, and evaluated. The eLearning objects from HyLOs, one of the main content sources used in this thesis, are built according to the IEEE LOM standard.

[4] Everything you ever wanted to know about learning standards but were afraid to ask by Wayne Hodgins and Marcia Conner., http://linezine.com/2.1/features/wheyewtkls.htm

[5] http://www.ieeeltsc.org:8080/Plone

[6] http://www.ieeeltsc.org:8080/Plone/working-group/learning-object-metadata-working-group-12/learning-object-metadata-lom-working-group-12

IEEE LOM is an open multi-part standard developed to define the structure of a metadata instance of an eLearning object.

This standard has the purpose to facilitate search, evaluation, acquisition and use of eLearning objects. This multi-part Standard also facilitates the sharing and exchange of learning objects, by enabling the development of catalogs and inventories while taking into account the diversity of cultural and lingual contexts in which the learning objects and their metadata are reused [10].

LOM has hierarchy of elements with nine main categories of meta-data. According to the LOM standard each metadata instance describes the relevant characteristics of the eLearning object to which it applies. These characteristics can be grouped into the following nine categories as described by the LOMv1.0 Base Schema [10];

- General: groups the general information that describes the learning object as a whole.
- Lifecycle: groups the features related to the history and current state of this learning object.
- Meta-Metadata: groups information about the metadata instance itself (rather than the learning object that the metadata instance describes).
- Technical: groups the technical requirements and technical characteristics of the learning object.
- Educational: groups the educational and pedagogic characteristics of the learning object.
- Rights: groups the intellectual property rights and conditions of use for the learning object.
- Relation: groups features that define the relationship between the learning object and other related learning objects.
- Annotation: provides comments on the educational use of the learning object and provides information on when and by whom the comments were created.
- Classification: describes this learning object in relation to a particular classification system. This category can also be used to provide certain types of extensions to the LOMv1.0 Base Schema, as any classification system can be referenced

The above categories are grouped into subcategories leading to a model which consist of a hierarchy of data elements. For the LOMv1.0 Base Schema, the simple data elements of leaf nodes, have values which are defined by their value space and data types. The data elements are described by the following characteristics [10];

- name: the name by which the data element is referenced.
- explanation: the definition of the data element.
- size: the number of values allowed:
- order: whether the order of the values is significant (applies only to leaf nodes, which are allowed to have values as opposed to aggregate nodes which do not have individual values).
- example: an illustrative example.
- value space: the set of allowed values for the data element – typically in the form of a vocabulary or a reference to another standard which in this case is ISO/IEC 11404:1996 Information technology Standard for Programming languages, their environments and system software interfaces and Language independent data types.
- datatype: indicates whether the values are LangString, DateTime, Duration, Vocabulary, CharacterString or Undefined.

Vocabularies are defined for some data elements. A vocabulary is a recommended list of appropriate values. Metadata that rely on such recommended list will have the highest rate of semantic interoperability according to the LOM standard. In some instances, the data elements may consist of list of values which may be ordered(order of items e.g. authors in a publication is significant), or unordered (list bears no meaning).

The above mentioned categories and data elements form the bases of the LOMv1.0 Base Schema.

IEEE LOM standard has the main advantage that it offers flexibility since any metadata element is optional and also because existing vocabularies can be extended to include values which are not in the list. A source content relevant in thesis that uses the IEEE LOM standard is the Hypermedia eLearning Object Systems (hyLOs).

2.3 Hypermedia eLearning Object Systems - HyLOs

HyLOs is an eLearning Content Management System that has been designed to provide full educational content. An eLearning CMS refers to a CMS that maintains and publishes its content on the Web in form of Web pages. It allows authoring, instructional composition and dynamic presentation of eLearning content.

HyLOs is built upon a more general Media Information repository (MIR) [9]. MIR is an open system supporting the standards XML, CORBA and JNDI. It provides general support of media data handling, authentication, user and connection handling. Its core is formed by

a media object database, implementing a duality of object oriented information model and relational structure. All data within the MIR data store is published in XML format. This leads to a separation of content from structural information, application logic and design elements. The MIR adaptive context linking environment (MIRaCLE) as used as the linking environment [9] in hyLOs. With the help of this linking extension, different hyperlink layers may be applied to the same content. These layers are defined by the content creator. Hyperlinks are generally made up of anchors and links which can be stored separately in a link base. Links concatenate anchors and identify sub-portions of content. Anchors can be expressed within XLINK statements by XPOINTER/XPATH Expressions [28]. Links denote the relationship between two or more anchors. Anchors have semantic descriptions which can be gotten from the LOM metadata.

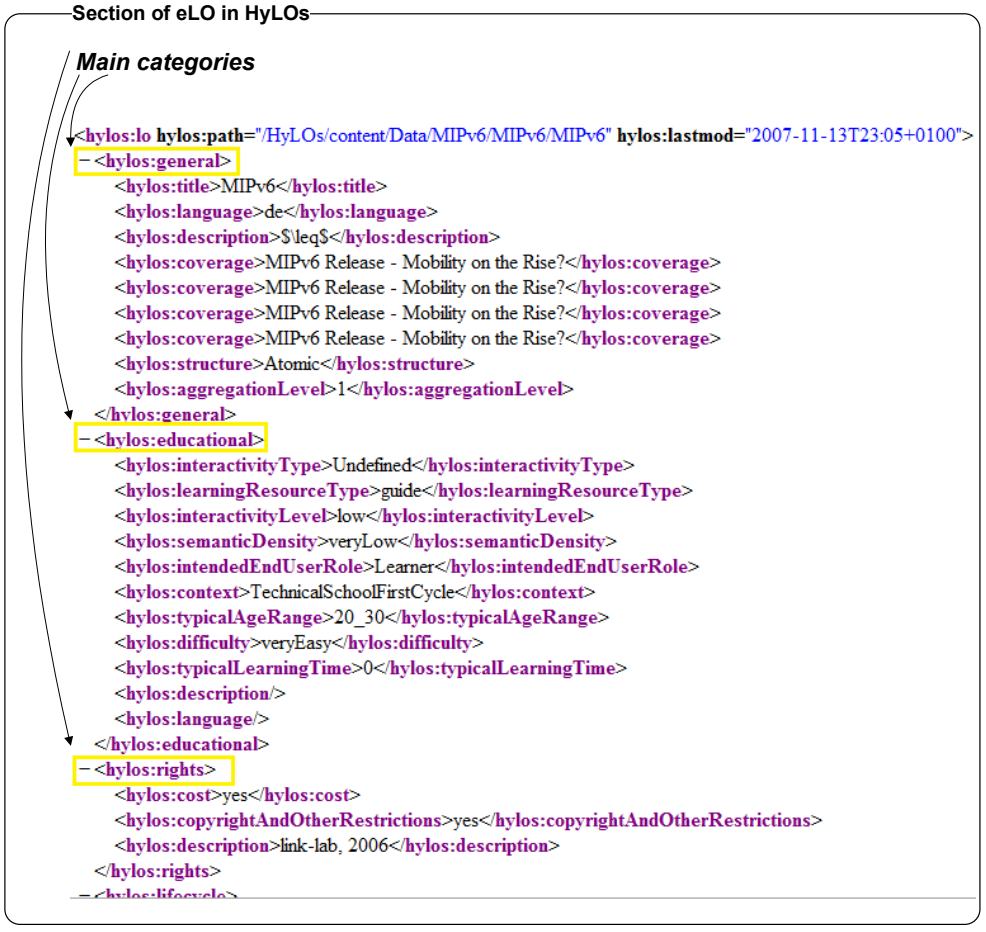


Figure 2.1: Section of eLO showing descriptive meta data.

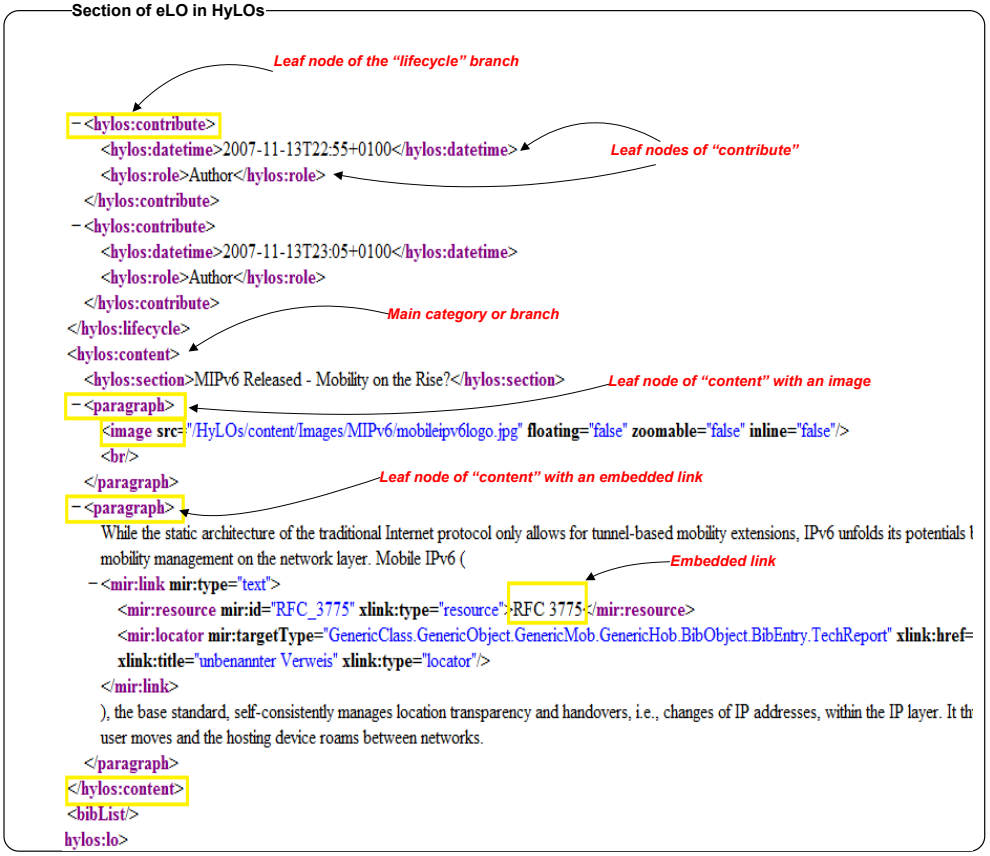


Figure 2.2: Section of eLO showing content entity within the paragraph tag with emdedded link.

HyLOs is based on a cellular *eLearning Object (eLO)* information model encapsulating meta data conforming to the LOM standard. The metadata and particular content entities such as titles, authors, keywords or information about courses, etc., are singled out as shown in figure 2.1. The separation of entities in this way provides content normalization and allows easy updating of content for authors and automatic generation of navigational overviews and updates. All other content units are organized in paragraphs (as in figure 2.2), which are collected to pages by means of external structures. This allows for easy re-use of content entities of paragraph dimensions by applying multiple structural references in a static or dynamic way. HylOs implements these concepts for practical implementation by using XML technologies and provides the consistent separation of content, structure, logic and design elements. For more information on the hyLOs eLearning CMS see the following references [9], [8],[21],[7]

2.4 Interactive Group eLearning

Learning in groups is a form of collaborative learning where users are responsible for their actions, including learning and respecting the abilities and contributions of their peers. Collaborative learning is an umbrella term for a variety of approaches in education that involve joint intellectual effort by students or students and teachers [24]. According to Wikipedia, collaborative learning refers to methodologies and environments in which learners engage in a common task in which each individual depends on and is accountable to each other. Collaborative learning is a learning method that uses social interaction as a means of knowledge building [13].

Advantages which can be gained from collaborative [24] eLearning include:

Group Diversity

Collaboration in the context of group work means multiple perspectives, backgrounds, learning styles, experiences, aspirations, etc. Content created in such cases can result from intensive discussions within the group, e.g. an application that lets group members give feedback in real time to the annotations of other group members. The content created will offer different perspectives on a subject and all group members can benefit from this.

Learners are active

Learners are actively taking part in the learning process. From the technological point of view, Web 2.0 techniques can be used to enhance the user experience making the user more involved within the group. The constructivist approach adopted in eLearning 2.0, section 2.1, can also act as a motivation factor since the user is in charge of his learning habits.

Learning is social

Due to the interaction with other learners, learners can acquire social skills and learn to treat each other with respect. The mutual exploration, feedback, sharing, etc. creates learners who seek to understand the group more and how to communicate better with the group as a whole and with the single individuals involved.

Goal-oriented

The purpose of collaboration is always to accomplish a particular task making group work goal-oriented.

Although collaborative learning is hardly a new idea, with the advent of Web 2.0 and other innovative ideas such as social networking, Really Simple Syndication (RSS) chapter 3, etc., it has evolved enormously. For example, using OpenCourseWare chapter 9, major learning institutions around the world place educational materials from their undergraduate- and graduate-level courses online for free. These courses are available to anyone with an Internet connection and can be access using RSS. Finding content on the Web is much easier because Web 2.0 technologies can use the tags or descriptive pieces of information attached to content by users to find related content. These technologies can also be used to build very interactive *User Interfaces*. Faster communication within the group and content retrieval can be achieved using web 2.0 techniques such as Ajax, section 3.3. Special tehniques can also be applied to give an illusion of extra speed during content retrieval from the Web. All these advantages can enhance the learning experience for the user.

To see how interactive group learning can affect the way we learn, two case studies of existing projects will be done in the next section. They will demonstrate the span of possibility which exist in this field as well as its usefulness.

To conclude this chapter, two examples of projects, which make use of collaborative eLearning will be mentioned here. They are not discussed because they are not used in this project. They are however, related to the work done here and references are supplied for further reading.

One is the Open Annotation and Tagging System (OATS) [23], which provides a collaborative tagging and an annotation interface by using a text highlighting metaphor. The second is Annotea[11], which can be used for collaborative document enhancement.

Chapter 3

Web 2.0 and Ajax

3.1 Introduction and Definition

The term Web 2.0 was coined by Dale Doherty[7] of O'Reilly Media in 2003 to describe the new trend which was noticeable on the Web at the time. This trend was seen in the way people and businesses were using the Web as a platform to create collaborative community-based sites such as blogs, social networking sites, wikis etc.

There has, however, been some discrepancy as to what Web 2.0 exactly means. There are those who argue that the concepts of Web 2.0 such as consuming and remixing data from multiple sources, delivery software as a continually updated service, creating rich user interfaces, etc. go beyond the page metaphor of Web 1.0 [17]. Others argue that Web 2.0 is nothing more than a late implementation of some of the core concepts proposed by Web 1.0[8].

In my opinion, Web 2.0 is a full implementation of Web 1.0 for the following reasons;

- Tim Berners-Lee's[9] book *Weaving the Web*[3] describes his original vision of the Web, as a collaborative workspace where everything was linked to everything in a single, global information space.
- ENQUIRE[10] an early software project written in the second half of 1980 by Tim Berners-Lee allowed pages to be linked together and edited.
- Web 2.0 technologies are based on Web 1.0 standards

Web 2.0 and Web 1.0 differ mainly in the ways they have been adopted and implemented.

[7] http://radar.oreilly.com/dale/

[8] Laningham (ed.), developerWorks Interviews, 22nd August, 2006. Transcript available at: http://www.ibm.com/developerworks/podcast/dwi/cm-int082206.txt

[9] Tim Berners-Lee - inventor of the Web, http://www.w3.org/People/Berners-Lee/

[10] Brief introduction about Enquire, http://en.wikipedia.org/wiki/Enquire

3.2 Impact of Web 2.0

The impact of the Web 2.0 is felt extensively on the Web today. A number of Web-based services and applications demonstrate the foundations of Web 2.0 concepts. Some of these applications are built entirely on user-generated content or harnessing collective intelligence [16] or the power of the crowd [2], while others rely on the way people interact and network. This has led to the following Web 2.0 concepts whose impact and implementations are thoroughly discussed here[5];

- User-Generated Content, e.g. wikis
- Collective Intelligence resulting from collaboration between users
- Social Networking
- Tagging
- Blogging
- Social Media
- Social Bookmarking
- Rich Internet Applications (RIA)
- Ajax

Ajax is the Web 2.0 technology that will be used to build the eLearning application in this thesis. This section, therefore, discusses Ajax in detail.

3.3 Ajax

The term Ajax was coined in February 2005 by Jesse James Garrett of Adaptive Path, in his online article entitled, "Ajax: A New Approach to Web Applications"[11]. In this article, he defines Ajax as a Web interaction technique or approach which involves transmitting only small amount of information to and from the server in order to give the user the most responsive experience possible. Ajax does this by adding a layer between the client and the server to manage communication between the two. He called this layer *Ajax Engine*.

The power of Ajax can be understood better if the Ajax programming model is compared to the traditional Web model as shown in 3.1, which is taken from Garrett's original article.

[11] Jesse James Garrett. Ajax: A new approach to web applications. Adaptive Path, http://www.adaptivepath.com/ideas/essays/archives/000385.php

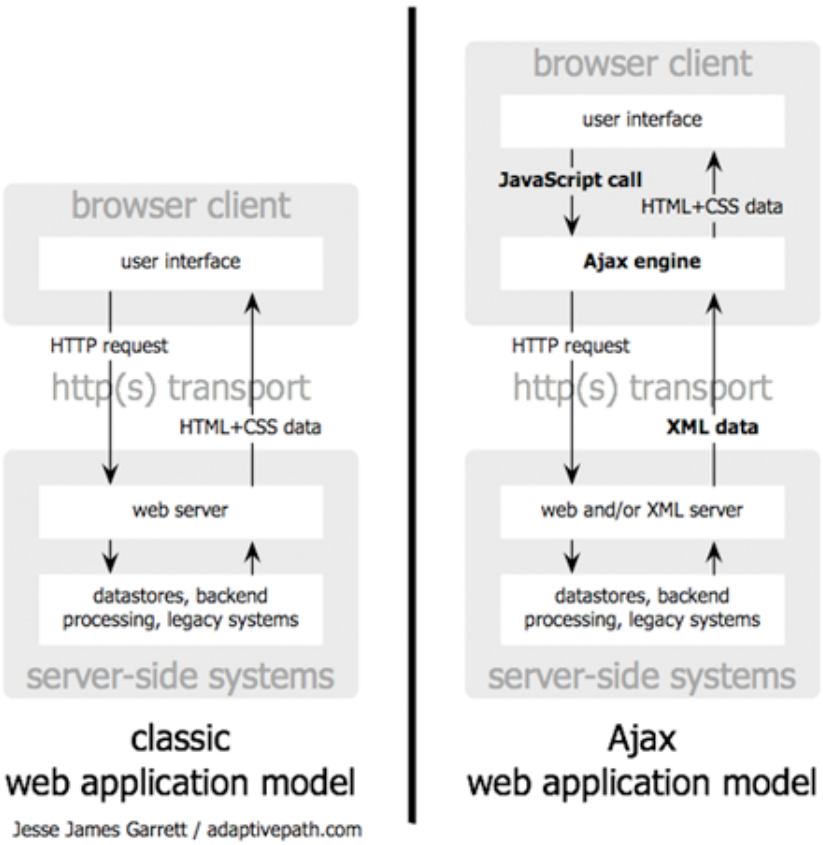


Figure 3.1: The traditional model for web applications (left) compared to the Ajax model (right).

Figure 3.1 shows the Ajax engine which is introduced in the Ajax model to eliminates the start-stop-start-stop nature of interaction on the Web. This Ajax engine is just a JavaScript object or function that is called whenever information needs to be requested from the server. With this model, the browser does not need to make a request directly to the server anymore. The browser initially loads the Ajax engine, which from then on is responsible for communication with the server and rendering pages from the server. Each link on the Web page, that would otherwise point directly to a resource on the server as is the case in the traditional Web model, now makes a call to the Ajax engine, which schedules and executes the request. The request is done asynchronously, meaning that code execution doesn't wait for a response before continuing. When the Ajax engine receives the server response, it parses the data and makes several changes to the user interface based on the information that has been provided by th server. Because this process involves transferring less information than the traditional Web model, user interface updates are faster.

3.3.1 Ajax Techniques

Hidden Frame Technique

HTML frames allow authors to present documents in multiple views, which may be independent windows or sub windows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced[12].

When using hidden frames, the idea is to create a frameset that has a hidden frame that is used for client-server communication. A frame can be hidden by setting its width or height to zero pixels or by setting its visibility to hidden in the style sheet effectively removing it from the display. Two types of frames used in HTML documents are a normal frame and the inline frame (iframe).

1. Hidden Frame

   The hidden frame always begins with a visible frame. This visible frame is an HTML page called the frameset document. It differs from the normal HTML page (without a frame) in that, instead of one head section and one body section, it has a head and a frameset in place of the body section. The frameset section of a document specifies the layout of views in the main user agent window. Then there is also a hidden frame which the user will naturally be unaware of. At some point, the user will perform an action that require data from the server of say fill a form that requires posting to the server, thereby making a call to the hidden frame. The hidden frame then makes a request to the server which depends on the action performed by the user. When the server returns with the response, a JavaScript function is then called which will transfer the contents of the return page for instance to the visible frame. This function is often given as the argument of the *onload* event of the hidden frame. Figure 3.2 is an adaptation from [15] and illustrates this process;

2. Hidden iframe

   Another form of behind-the-scenes client-server communication is communication using the iframe. The iframe stands for inline frame and is so called because an iframe can be integrated into a page not originally created as a frameset, making it much better suited to incremental addition of functionality. Apart from this difference they are basically frames and behave the same as frames. One of the main advantages of using iframes is that, they can be created on-the-fly using JavaScript and once created they can also be placed anywhere within the document thereby making them very suitable for adding Ajax functionality. An example of using hidden frames is in doing mashups, wwhereby a page can use iframes to combine data from several sources into a single integrated document.

[12] http://www.w3.org/TR/REC-html40/present/frames.html

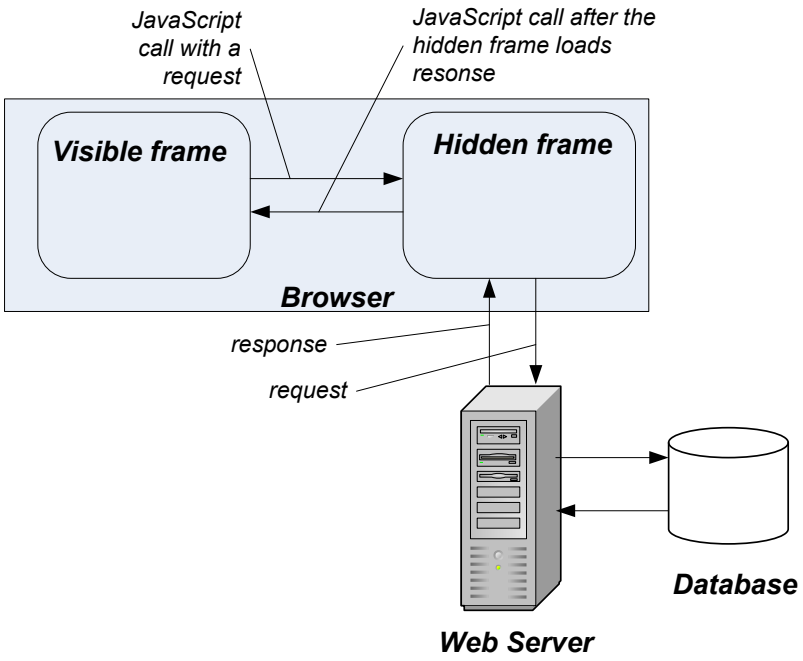


Figure 3.2: The hidden frame technique

Because iframes can be used and accessed in the same way as regular frames, they are ideal for Ajax communication.

Today Ajax applications are typically single url applications. This often raises the question of how to manage the browser history and bookmarking on the site since the page url never changes. One of the advantages for using hidden frames is that the browser history can be maintained and thus enable users to still use the Back and Forward buttons in the browser. The hidden frame keeps record of all request made through it. This may not be true for an Ajax application. These records can then be used to navigate through the history of the site using the hidden frame. For iframes, this is not necessarily true as browsers implement iframes differently. Whereas IE always stores the history of iframes, Firefox does so only if the iframe was defined using HTML (that is, not created dynamically using JavaScript). Safari never stores browser history for iframes, regardless of how they are included in the page [15].

One disadvantage of using hidden frames is that, they cannot make requests outside of the domain from which they are called. This is however a well known security

enforcement by browsers when it comes to JavaScript. Therefore JavaScript can only interact with frames that are from the same domain as itself.

Another disadvantage of hidden frames is that, there is very little information about the HTTP request being made by the hidden frame. If the frame fails to load the page, for instance, there is no way of knowing. If it is important to keep track of the HTTP request being made to the server, then another Ajax technique should be used. This technique which is very widely used today makes use of the XMLHttp Requests (XHR) which is the subject of the next section

XMLHttp Request

The XMLHttpRequest specification defines an API that provides scripted client functionality for transferring data between a client and a server [26].

The concept behind the XMLHttpRequest object was originally created by the developers of Outlook Web Access for Microsoft Exchange Server 2000[13]. An interface called IXMLHTTPRequest was developed and implemented into the second version of the MSXML library using this concept[14]. The second version of the MSXML library was shipped with Internet Explorer 5.0. This library used an XMLHTTP object to access the IXMLHTTPRequest interface. This access was done using ActiveX controls. ActiveX controls are reusable software components that perform a particular function or a set of functions in most Microsoft Windows products in a way that is indepenent of the programming language used to implement them.

The XMLHttp object was created to enable developers to initiate HTTP requests from anywhere in an application. Following its wide adoption, Mozilla duplicated the XMLHttp functionality for use in its browsers, such as Firefox. They created a native JavaScript object, XMLHttpRequest, which closely mimicked the behavior of Microsoft's XMLHttp object. Shortly thereafter, both the Safari (as of version 1.2) and Opera (version 7.6) browsers duplicated Mozilla's implementation. Microsoft even went back and created their own native XMLHttpRequest object for Internet Explorer 7. Today, all four browsers support a native XMLHttpRequest object, commonly referred to as ***XHR*** [15].

The different processes, methods and properties which have led to the success of the XMLHttp Request object will be briefly discussed in the following sections.

[13]"Article on the history of XMLHTTP by an original developer". http://www.alexhopmann.com/xmlhttp.htm.

[14]"Specification of the IXMLHTTPRequest interface from the Microsoft Developer Network". Msdn.microsoft.com. http://msdn.microsoft.com/en-us/library/ms759148(VS.85).aspx.

3.3.2 Ajax Patterns

Most of the patterns presented here have been used long before the advent of Ajax to enable intuitive client-server communication using JavaScript. When it comes to Ajax programming, it is important to look for the best way to initiate and continue to make request to the server. What the best pattern is will depend on the requirements of the system being developed. Some system will require that the user data already be available upon demand. This will give an illusion of increase in speed from the user's perspective. Other system will be designed such that user actions can be predicted and the needed data fetched before hand. In other systems, it will be more useful to periodically retrieved small chunks of data. No matter what system is being built, Ajax programming model offers fine granularity in controlling the communication between the client and the server to attain the desired behaviour.

Predictive Fetch

Predictive fetch or predictive download simply means anticipating likely user actions and pre-loading the required data. The start-stop nature of the traditional Web model was caused partly because of the fact that the user always has to click a link and then wait while the information is being fetched. From the user's perspective and following the nature of the Web today, this is no longer acceptable. Predictive fetch is therefore used when possible to fetch data that the use will need in the near future. When the user actually demands this data maybe by clicking a button, the data is displayed immediately. The key, once again, is the "logical-to-assume" criterion [15]. By anticipating and pre-loading information related to the user's most likely next steps, the application can be implemented to feel lighter and be more responsive

Submission Throttling

While predictive fetch is used to retrieve data from the server, an Ajax pattern required to submit data to the server is called submission throttling. Ideally a call could be made to the server on every keystroke and mouse movement, for example, like a desktop application, tooltips could come directly from the server as the user mouses around. But that's not practical due to bandwidth considerations, and possibly server constraints. The main problem with submitting data to the server in this manner is that, the server can quickly become overloaded. The intensity of this problem is increased by the fact that each message to the server has overheads such as packet headers and requires some processing at each stage of the browser-server round-trip. Submission throttling can solve this problem, by retaining data in a browser-based buffer and periodically uploading it to the server.

The main disadvantage of the submission throttling design pattern, is that it deliberately downgrades synchronization. This can make the system being developed vulnerable to integrity issues. Such a vulnerability is posed by spam bots which are malicious software written to harvest user names and passwords or submit useless content on forms. It is therefore important to consider these security issues when designing an application, choose the appropriate design pattern and take preventive measures to prevent security loopholes.

Periodic Refresh

Periodic Refresh design pattern describes the process whereby the browser periodically calls the server so as to refresh any volatile information. This approach, also called polling, requires the browser to keep track of when another request to the server should take place.

Another example is the XHTML Live Chat [15] at plasticshore.com. It uses periodic refresh to implement a chat room using simple web technologies. The chat room text is updated automatically every few seconds by checking the server for new information. If there is a new message, the page is updated to reflect it, thus creating a traditional chat room experience.

Although there are many different ways that period refresh can increase user experience, the basic purpose remains the same: to notify users of updated information

Multi-Stage Download

Multi-Stage Download is an Ajax pattern wherein only the most basic functionality is loaded into a page initially. Upon completion, the page then begins to download other components that should appear on the page. Basically content download is broken down into multiple stages, so that faster and more important content will arrive first. The extra functionality is loaded in the background and available when the user is ready.

Although nice, Multi-Stage Download does have a downside: the page must work in its simplest form for browsers that don't support Ajax technologies. This means that all the basic functionality must work without any additional downloads. The typical way of dealing with this problem is to provide *graceful degradation*, meaning that those browsers that support Ajax technologies will get the more extensive interface while other browsers get a simple, bare-bones interface.

[15] http://chat.plasticshore.com/index.html

3.4 Ajax Technologies

3.4.1 Client-side Scripting

Ajax isn't a technology. A group of different technologies, some of which have been used before Ajax, come together in different ways to give Ajax-based application their power. These technologies are;

- XHTML and CSS: standards-based content presentation languages
- DOM: dynamic display of and interaction with loaded content
- XML, XSLT and JSON: data interchange and manipulation
- XMLHttp: asynchronous data retrieval and communication with server
- JavaScript: Scripting language used to program an Ajax engine

The above listed client-side technologies are extensively discussed here [5] and [15]. With the growth of Ajax, many JavaScript frameworks have been developed to better handle client-server communication and minimise browser compatibility issues. The next section gives a brief introduction to JavaScript frameworks

JavaScript Frameworks

A JavaScript framework or toolkit is a library or API which offers pre-written JavaScript controls which allow for easier development of JavaScript-based applications, especially for AJAX and other web-centric technologies. There are several reasons to consider user a JavaScript framework when developing applications for the Web. Some of these are identified here[16] as;

- time: time is always a scarce resource and using a framework will considerable shorten development time
- pre-written controls: most of these frameworks provide functions to perform common task. Some of these functions have been written, tested and re-tested by many users over the years. Do not waste time trying to re-invent the wheel
- less code: less code has advantages of smaller file size, better maintenance and less development time
- more readable code: also aids in better maintenance and less development time

[16]Web Application Development Tips, Tricks and Techniques : Javascript frameworks overview, http://www.undisciplinedbytes.com/2009/10/javascript-frameworks-overview/

- web application execute faster: these frameworks are really fast in performing their tasks and most of them have been optimized for really fast execution.
- web application run in most modern browsers: in JavaScript development, different browsers have different ways of implementing the same things and handcrafted cross-browser development can be very time consuming and irritating. These frameworks have already taken care most cross browser issues.

The most widely used, open source, general-purpose frameworks as of today are: jQuery[17], Prototype/Script.aculo.us[18], Mootools[19], Yahoo UI[20], Ext[21] and Dojo Toolkit[22] [27]. Each of these frameworks are used by many thousands of developers, has an active community behind it, are well maintained and tested across different browsers, and supports commonly used JavaScript functionality such as in Ajax Web development, DOM traversal and manipulation (including CSS selectors), event handling, animation and various utility functions [27].

Dojo has been chosen as the framework of choice for the application built in this thesis, because of the following reasons;

- it has a powerful packaging system divided into three projects; Dojo, Dijit and Dojox.
- it handles many cross-browser incompatibility issues and memory leaks. In addition it normalizes the event system among many popular browsers. This is significant because Internet Explorer for instance does not implement the WC3 event model and it leaks memory [19]. As of now, the Dojo version (1.3) used in this project officially supports Internet Explorer(6.0+), Firefox (1.5+), Safari (3+), and Opera (9+) and Konqueror (3.5+). As for Opera and Konqueror only Dojo core components are supported.
- it has a very large UI component base which involves many UI controls and a graphics framework.
- it is organized into a hierarchy of functionality such that developers can focus on a particular needed area and can expand this area as their needs change.
- it is open source.

Dojo is made up of collection of static, client-side JavaScript scripts that are responsible for the above mentioned advantages. It includes the following;

[17] jQuery, http://jquery.com/
[18] prototype/script.aculo.us, http://prototypejs.org/, http://script.aculo.us/
[19] Mootools, http://mootools.net/
[20] YUI, http://developer.yahoo.com/yui/
[21] Ext, http://www.extjs.com/
[22] Dojo, http://dojotoolkit.org/

- A design and implementation that normalizes the browser, allowing the same source code to work in several browsers.
- Functions/libraries that abstract the sometimes inconvenient W3C DOM programming model into a convenient, efficient interface.
- Functions/libraries that fix several cross browser errors such as memory leaks.
- A library of a large set of HTML widgets
- A module system coupled with a build system that lets developers divide code into small, manageable chunks during development and later package the release system for optimal download performance— without any modifications to the source code. The build system even lets developers slice and dice Dojo itself in a way that's optimal for their project.
- Independent libraries (that is, you can load them on demand) that implement several other advanced capabilities

Dojo is currently divided into three projects [19]. These are;

1. Dojo: The foundation upon which everything else is built. Altogether, it includes about fifty JavaScript scripts and several other resources that handle browser normalization, JavaScript modularization, extensions to the core JavaScript library, extensions to the W3C DOM API (including a parsing and querying the DOM), remote scripting, Firebug Lite, drag and drop, a data store API localization and internationalization, and a few other miscellaneous functions.
2. Dijit: The Dojo widget framework and built-in widgets (about forty HTML user interface widgets).
3. Dojox: Dojo extensions. This includes everything from the grid widget to graphics libraries. It contains some very sophisticated and stable libraries that are currently deployed in real-world, for-profit systems as well as some completely experimental systems. Each library includes a ***readme*** that describes the project.

Each of these three projects resides in its own source code tree.

Through out this thesis, especially in the implementations and testing section, some advantages of Dojo and Dojo specific features that enable the application to function the way it does, will be pointed out.

3.4.2 Server-side Scripting

The Ajax model of programming is mostly about giving the client scripting ability.This is achieved by introducing the Ajax engine which is described in section 3.3. However to build

a complete system, server-side processing is mostly needed. The will mostly likely be no Ajax without a stable, responsive server waiting to send content to the Ajax engine. For this purpose, an application server is required. Most often, a database server is also required.

Common server side scripting languages that can be used in Ajax based applications include PHP, JSP, Ruby on Rails(RoR) and ASP.NET. PHP has been chosen for this project because it is fast, stable, secure, easy to use and open source. The other server side scripting languages have been mentioned here only for purposes of completion. In the book, ***Rich Internet Applications - Web Development for Programmers*** [5], these server side technologies are well explained. Examples are also given to support the discussion points.

The above mentioned book also describes two types of application servers which are suitable for Ajax backend development. These servers are Microsoft Internet Information Services (IIS) and Apache HTTP Server. The Apache HTTP Server has been chosen for this project.

Finally the database chosen for this project is MySQl database. This is a relational database which means that, data is written to the database in a logical representation which makes it access the data without any consideration of its physical structure.

Chapter 4

Requirement Analysis

The main aim of the project is to design an eLearning application that will give users the possibility to gather content from the Internet and then interact with this content in given ways to create new content. Given the vast amount of information on the Web today, such an application will give users the opportunity to systematically gather information from the Web that can be used in the future. It will have a very user-friendly and highly interactive interface (UI). It will be built using Web 2.0 techniques and Ajax concepts. Users can work alone or create groups and work together in these groups. Users will be given the possibility to share their content or content created by their groups with other users.

In this chapter a simple requirement analysis is given for the overall system using UML use case diagrams. UML activity diagrams are used to illustrate the detailed communication processes which let the user achieve the tasks of content retrieval and interaction.

4.1 System Tools

4.1.1 General Use Case

The system should provide the user with a set of tools to be used for specific task. These tasks include content manipulation, viewing and editing added content, group work and co-ordination and finally communication with other users. The use case diagram of figure 4.1 shows an overview of these tools.

Figure 4.1 gives a basic overview of the tools which will be available to the user. The user will be the main actor in this system. For now, no concept for user classification will be considered, i.e. all users are initially the same. However, the system can easily be extended to group users as administrators, guest, normal users, etc. The user interacts with this system initially by selected one of the main tools available. These tools include;

- viewing tools: used for viewing content added to the system

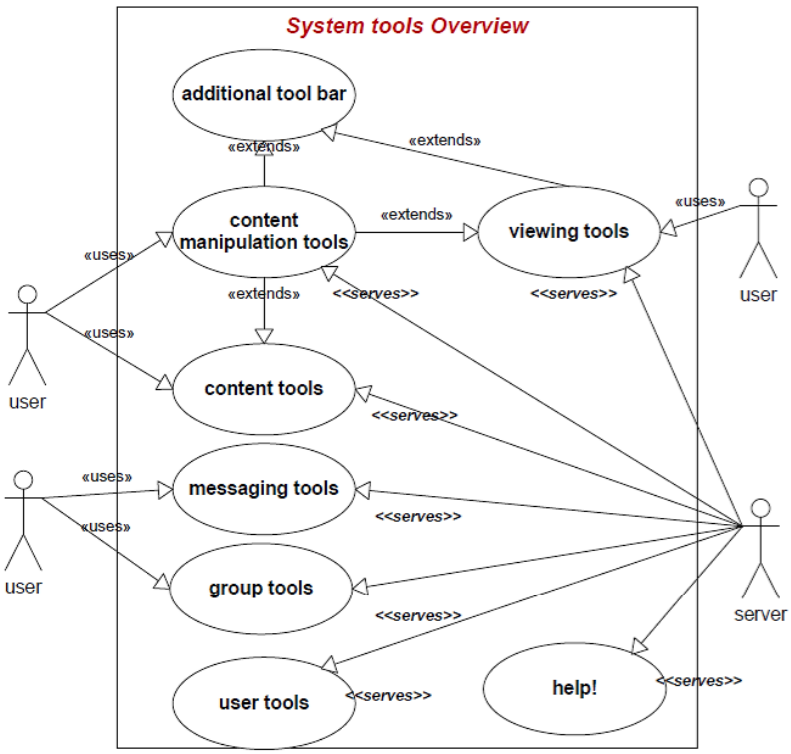


Figure 4.1: Use case showing system tools

.

- content tools: used for retrieving content
- group tools: used for creating and coordinating group work
- messaging tools: used for communicating with other users

The server is also an actor in the system because it interacts with the tools provided in a number of ways, but mostly it serves requested content. The server also interacts with a datastore to supply the information that is requested by the browser. This datastore will be described by the database model presented in section 6.8. The use case also shows an interaction between the viewing tools and the content manipulation tools.

Content manipulation tools can be used when the viewing tools have been used to show which words or phrases have already been highlighted in a document. When content has been added to a document, a highlighting abstraction is used to visualize this later. A user will choose a menu option, for instance, "show links", and the words or phrases with links attached to them is highlighted with a specific color. This holds for all form of added content (see chapter 6 which describes the application specification in detail).

When the user shows added content within a document, he should be able to use the content manipulation tools to further manipulate the content. In this case, the action of using the viewing tools to show the highlights in the document will be extended by using the content manipulation tools to manipulated content. The same principle holds for the content tools which will be used to retrieve content. Once the content is retrieved, the user can then call up the content manipulation tools and extend the functionality of the content tools by interacting with this content. Extended functionality in this case simply means that an action, e.g. clicking a button offers the possibility of clicking more buttons that accomplish more tasks.

The additional toolbar shown in figure 4.1, will always appear in the notification section of the header of the application (see figure 5.3) when additional functionality is required or when the user needs to show and hide content on some section of the UI. The options required for accomplishing such task will be created and placed on the additional toolbar. The relationship that exist between the viewing tools and the tools on the additional tool bar is similar to the relationship described between the content manipulation tools and the viewing tools. These tools of the additional tool bar are displayed dynamically based on the interaction of the user with the application user interface. Depending on what viewing tool has been selected, a section of the ***User Interface*** might change accordingly. The additional tool bar should be provided so that the user can hide, show or entirely remove this new section from the Document Object Model (DOM see section 3.4). This is discussed in detail in the system specification sections 6. What is important to note here is that the viewing tools extend the functionality supplied by the additional toolbar.

Although a similar relationship exist between the content manipulation tools and the additional toolbar, it is different in the sense, that the content manipulation tools reside on or use the additional toolbar. But they are not the only options offered by this additional toolbar. The additional toolbar as described above, also offers UI manipulation tools as an extension of the viewing tools. So the content manipulation tools also extend the additional toolbar to provide more functionality to its users.

Messaging tools should support communication between the users. The group tools should provide the users with a means to create, activate, join, leave or delete groups. The functionality provided by these tools will be discussed in detail in the specifications chapter 6.

4.1.2 Use Case For The Overview Tool

Overview tool is the tool which opens up most of the functionality within the system. For this reason, a detailed use case is provided in figure 4.2 to illustrate the possible functionality which the viewing tools can provide and how they affect other parts of the application.

Figure 4.2 shows an extension of the general use case of figure 4.1. As can be seen, the basic actors again are the users of the application and the server. It can be seen that

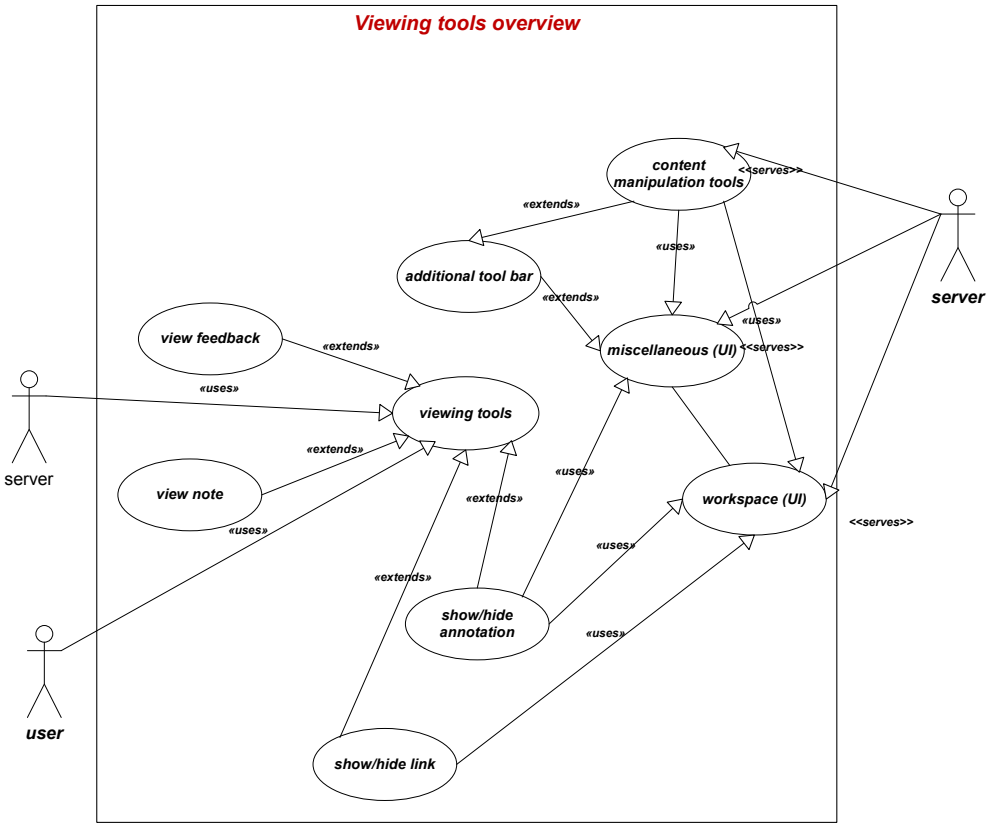


Figure 4.2: Use case showing viewing tools

.

the viewing tool is extended by four other tools which can be used to view added content. Links, annotations, annotation feedback and notes can be added to content. These tools should be used for viewing the added links, annotations, feedback to annotations and notes. The viewing tool no longer has a direct relationship to the content management tools and the additional toolbar. The relationship is now established through the new viewing options and the UI. This also holds for the relationship between the viewing tools and the server depicted in 4.1.

Whenever an added content is to be viewed, components of the UI are used to display or provide a visualization of the added content. These sections are the workspace and the miscellaneous section which will be discussed in detail in the UI design section 5.2.1. The additional toolbar then extends the miscellaneous section, such that the user can interact with this section and manipulate the DOM components here. Although this is not indicated

in the use case diagram, this could also be viewed as a two-sided relationship, because the additional toolbar can be either extended to provide DOM manipulation options depending on the user's interaction with the miscellaneous section.

The miscellaneous section also has a relation to the workspace (see section 5.2.1 for a detailed description of the UI). As of now it suffices to know, that the workspace is the container on the UI which will hold the content). The origin of this relationship will be eminent when the show/hide link or annotation tools are used. These tools will *use* the workspace as a platform on which to offer content interaction possibilities to the user. Depending on the user interactions with the workspace, the miscellaneous section will be changed as determined by the system. The activity diagrams of section 4.1.3 show the content interaction processes that will lead to interactions between different UI sections such as the interaction mentioned here.

4.1.3 Activity Diagram For User-content Interaction

The user-content interaction should have two starting point; when a user selects a word using the mouse or when the user uses the viewing tools to highlight words within the document that already have content added to them.

(a) Word Selection

User interaction with the content will follow the activity diagram of figure 4.3.

Selection of the word should load the additional toolbar with tools to enable the user interact with the content. Depending on the selected tool, an editor should be opened such that the user can give in the added content. If the added content is posted to the server successfully, it is written to the datastore. If the added content could not be written to the datastore, the user should be informed. Concretely, the application should be able to catch exceptions and give the user meaningful messages on the status of the request made to the back-end.

If the user is in the process of creating or editing content but suddenly changes his intentions and doesn't post to the server, then the system should just exit this functionality. The user might have selected a word for reasons unrelated to user-content interaction. The toolbar will be presented to the user anyway. This toolbar should be such that the user can simply ignore it and keep working or he can remove it if it is irritating for any reason.

(b) Viewing Tools

To understand this second method, it is important to know that users can add links or annotations to documents. The show/hide links or annotations viewing options should

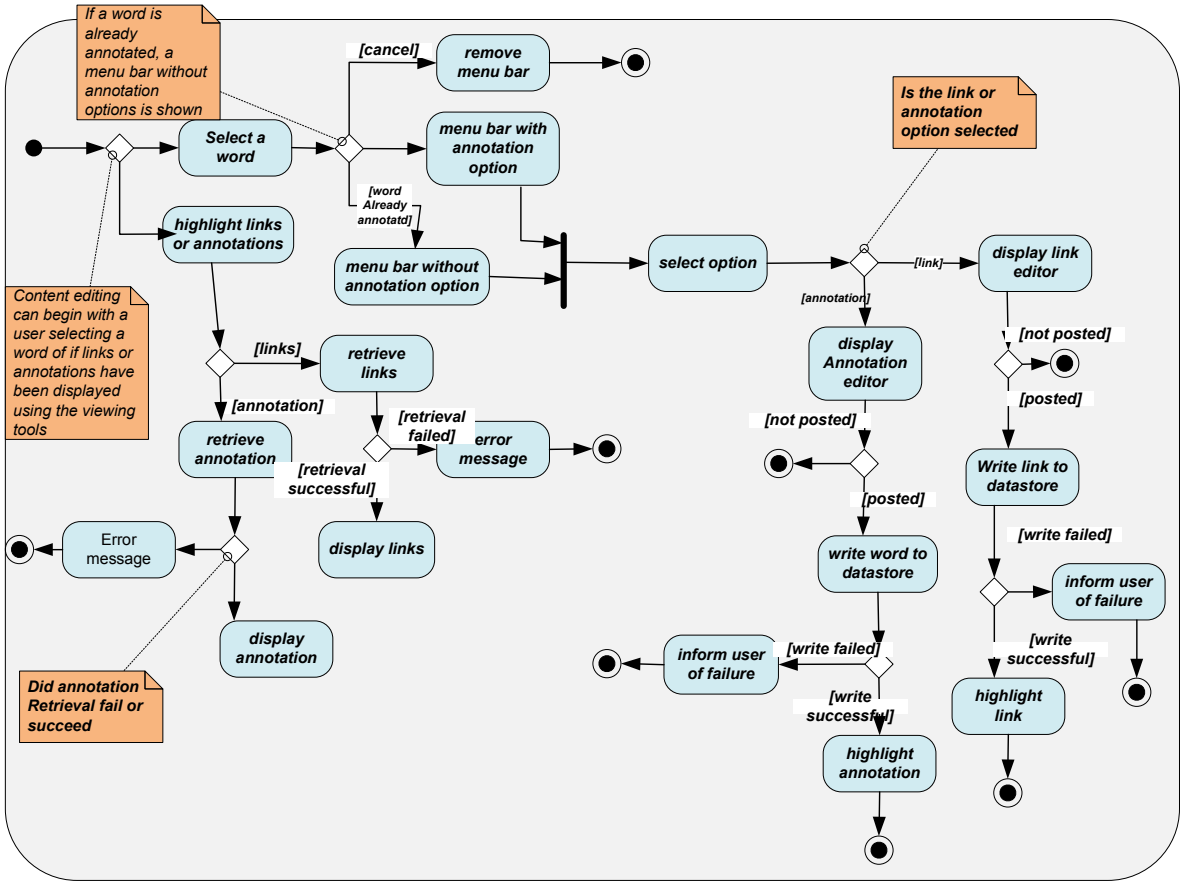


Figure 4.3: Activity diagram showing content manipulation by word selection

.

highlight the words or phrases in the document being viewed that have links or annotations added to them. Figure 4.4 shows the activities resulting from clicking any one of the highlighted words or phrases within the document. In this figure, notes as in "original note" means added annotation.

When a link is clicked, the links attached to that word or phrase should be displayed and this functionality exited.

When an annotation is clicked, the annotations should be displayed with options to interact with the content. Interaction with the content include the following;

- the user should be able to edit the annotation if he is the one who created it

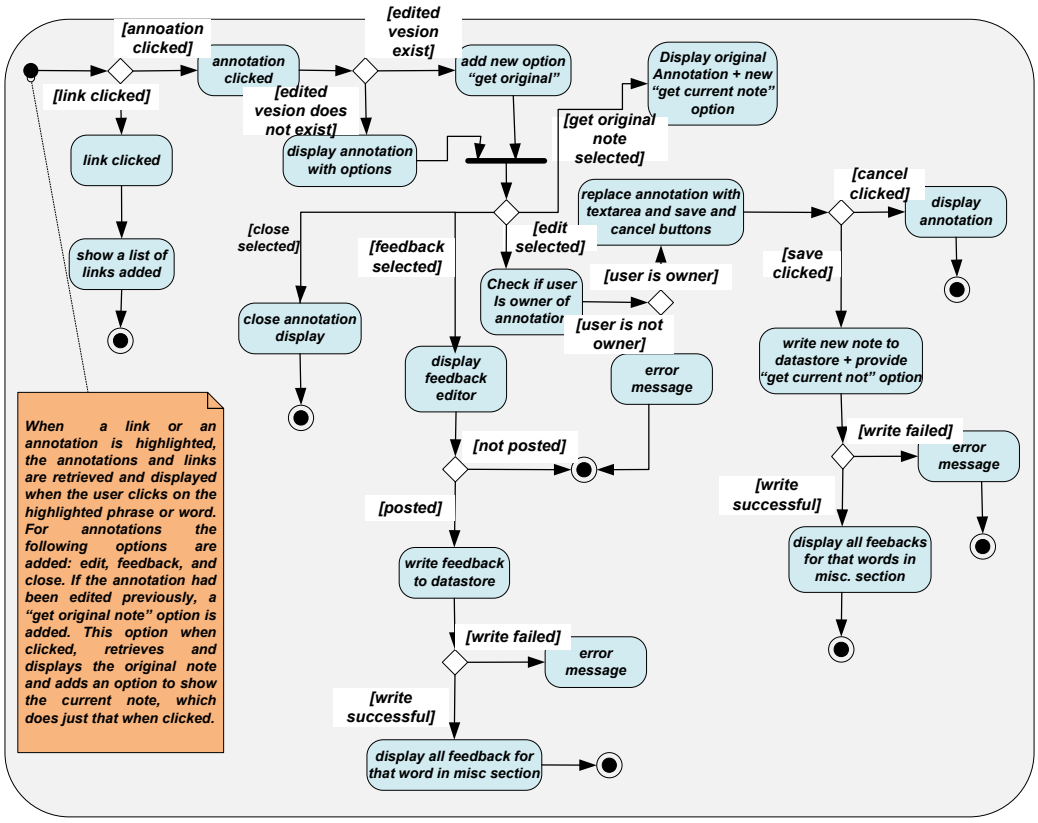


Figure 4.4: Activity diagram showing content manipulation by highlighting words within current document

.

- if the annotation has already been edited, the user should be able to read the original annotation
- the user should be able to give his opinion to an annotation in form of a feedback
- the user should be able to close the annotation display

Whenever an annotation is added or edited, it is written to the datastore. Other users can view this annotations if they are shared and give feedback. This feedback is also written to the database.

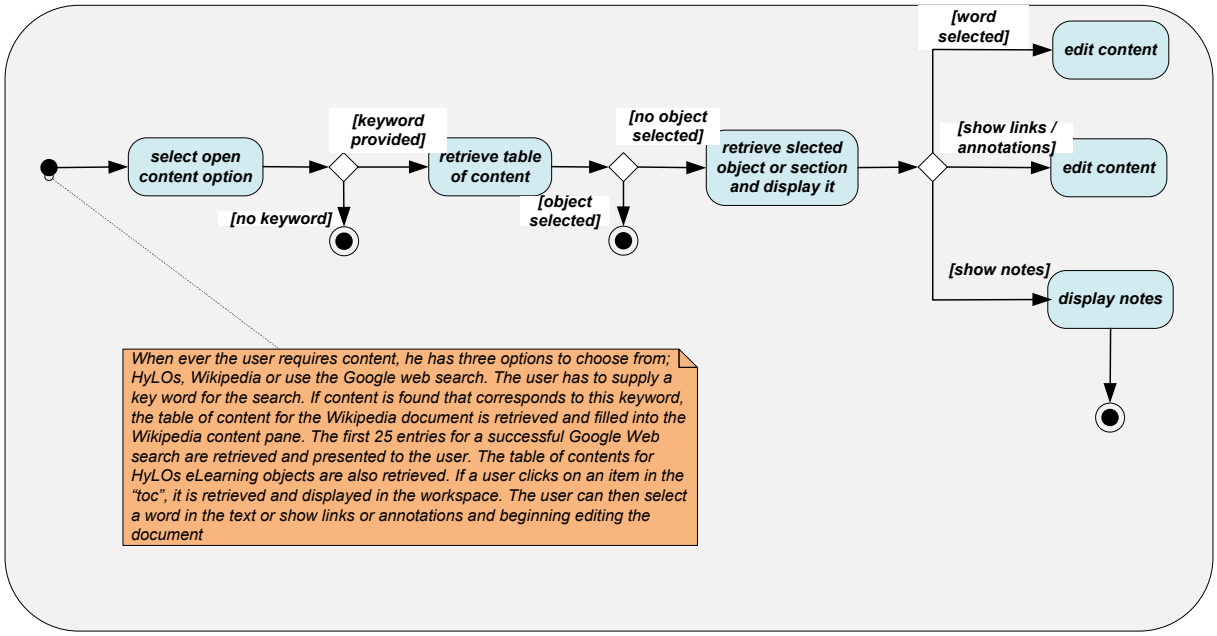


Figure 4.5: Activity diagram showing content retrieval

.

4.1.4 Activity diagram for content retrieval

All the above mentioned requirements can be met only if there is content to work with. Content retrieval is therefore very important. The system should give the user the ability to retrieve content and display it to the user for processing. The activity diagram of figure 4.5 shows the basics of retrieving content.

Using the content tools of figure 4.1, the user should be able to retrieve content by giving a keyword. The user should specify a content source when giving a keyword. The initial content sources should be hyLOs, Wikipedia and Google. For hyLOs and Wikipedia, the table of contents should be retrieved and entered into the appropriate UI container see figure

5.4. A container is simply a place holder in s section of the UI, that will hold the content. The first 25 results from the google search should also be entered into the appropriate container.

4.2 Group communication requirements

One of the purposes of this application is that it should allow users to come together in groups, open documents and extend them by adding meaningful content to these documents. Due to the different perspectives or contributions from different group members, the quality of the content might improve. The group functionality will initially be basic and should meet the following requirements;

- all tools available to single users should be available to groups
- group owns should be allowed to share with of hide their content from members not belonging to the group
- users may create and join as many groups as they want
- a group should be inactive if it is not being used, i.e. if there are no members of the group online who are using that group to create content.
- groups can only be activated by members of that group.
- no user should simultaneously be in two active groups .i.e no user should simultaneously take part in two discussion threads. An active group simply means that a member of the group has activated it by starting a discussion thread within that group when he is online.

These basic requirements can be extended in the future to give group owners more control over the group.

4.3 User Interface requirements

The success of any application depends on its acceptance. The acceptance in turn depends on its usability and usability is determined by the extend to which users can easily interact with the application. A good user interface should focus on the user's experience and interaction. The goal of user interface design is to make the interaction with content, transparent, intuitive and efficient. Good user interface design facilitates completing the task at hand without drawing unnecessary attention to itself.

The application built here will be a browser-based application. This is as opposed to the browser application paradigm of traditional web pages, where an application starts off with

an HTML page, and later on retrieves another page, which replaces the currently viewed page. Ajax can be used to make the pages more interactive, but the basic paradigm in use remains unchanged: get a page, perform some action, get another page.

Browser based applications on the other hand, use the browser as the UI Platform and takes the form of a single-page application. The user navigates to a URL that presents a complete application. The important thing to note here is that the browser is not just retrieving and displaying pages, but is acting as the UI platform which displays a complete GUI program. Using this approach, the UI should be designed to fulfill the following requirements;

- It should be user-centered. How can the interface help the user achieve a specific task effectively and easily?
- It should offer the possibility to always keep users informed about what is going on, through appropriate feedback within reasonable time. This is very important for interaction within a group.
- It should be consistent and should follow design conventions which are familiar from others systems. An example of such a convention are icons used for buttons
- It should make application navigation intuitive. The user should be able to go back and forth, or jump to specific sections of the application with ease.
- Information that is most likely to be used, should be accessible to the users at all times. Such information may include currently active users, content, group names etc.
- System tools most likely to be used, should also be available to users at all times. Such tools might include content retrieval tools, tools for viewing added content etc.

Chapter 5

Application Design

The discussions in this chapter will focus on how the application can be designed to fulfill the requirements discussed in the previous chapter. It starts by discussing the general system architecture which shows how the different components of the system work together and how they interact with each other. Since the system is designed to be user interactive, the interaction of the user with the application is very important. Therefore, this section will also discuss the design of the user interface in detail. In each step, possible design constraints will be examined and solutions discussed. This chapter will conclude with a discussion on the strengths and limitations of the application based on the concepts discussed.

5.1 System Architecture

The system as a whole will be made up of four main parts. These parts are as depicted in figure 5.1 and include;

- the browser
- the server
- the datastore
- the Internet

5.2 Application Front-end

The front-end of the system consist of the user and the browser. The system is designed such that the users can work alone or group themselves and work together in groups.

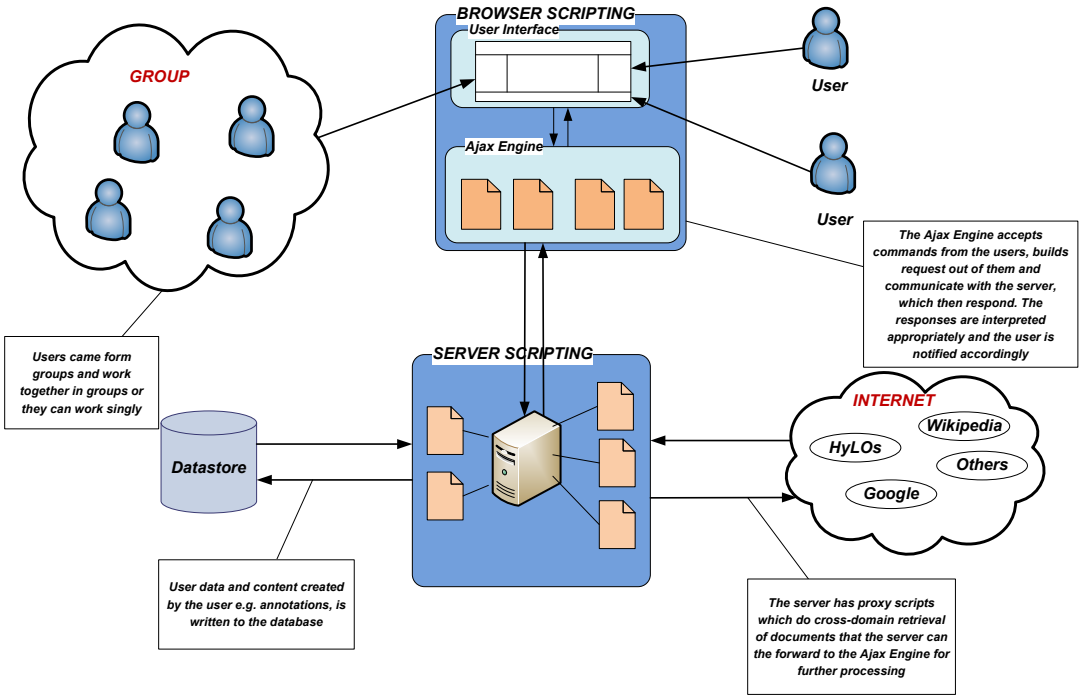


Figure 5.1: System Architecture of the eLearning application

.

The browser itself consists of two separate parts. In true nature of Web 2.0, the application is designed such that it can be used to edit Web documents. The browser plays an important role by providing the platform on which this Web editor runs. The users interact with this web editor via the ***User Interface (UI)***

5.2.1 User-Interface design

In order to meet the UI requirements discussed in section 4.3, the UI is divided into five main sections as shown in figure 5.2;

- header
- navigation
- miscellaneous
- status bar

- workspace

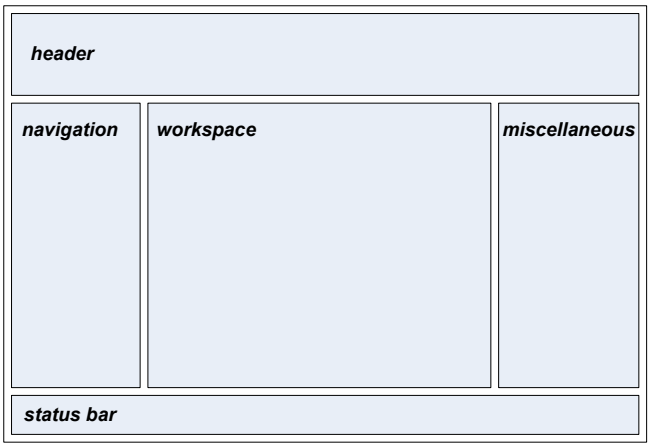


Figure 5.2: Basic User Interface layout template

.

1. Header

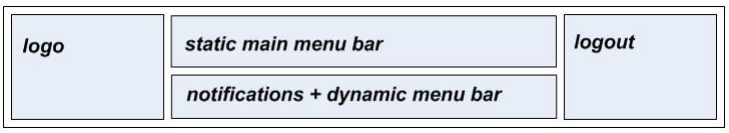


Figure 5.3: basic header layout

.

The header performs the following main task;

- provide the main menu bar for actions which the user can apply to the system
- provide a notifications section, which is used to notify the user on changes within the system. This notification section also provides an additional menu bar with additional functionality depending on the user's interaction with the application
- provide a way for the user to log out of the system.

To perform these tasks, the header is divided into four different sections as shown in figure 5.3. The section to the left holds the application logo. The middle section is further divided into two. The top section holds the main menu bar. The main menu bar will provide the user with tools for;

- retrieving and displaying content
- viewing content added to the system
- creating and managing group interaction
- sending messages to other users

The second half of the header makes up the notifications section. This section changes frequently depending on the users interaction with the system. At one point, it just shows a simple message informing the user of a change performed on some section of the GUI and at other times, it presents the user with a menu bar which provides the user with more functionality. This additional menu bar also adds or removes contents in response to actions being performed by the user.

The last section of the header holds an option which gives the user the possibility to log out of the systems.

2. Navigation

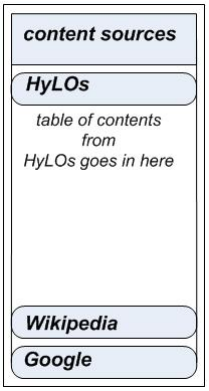


Figure 5.4: Basic navigation section layout

.

The navigation section holds the content sources. It is called the navigation section because the user is able to jump to any object he wishes by clicking on the navigational menus in this section. Figure 5.4 shows the navigation section.

The application initially offers content from three different sources: hyLOs CMS, Wikipedia and Google. This section is, therefore, divided into three different panes; hyLOs, Wikipedia and Google respectively. To open a document from Wikipedia, the user gives in a key word. If the document exists, the application retrieves the table of contents for that document and places it in the Wikipedia pane. The table of contents for the hyLOs elearning units are also retrieved and place in the hyLOs pane. For a

Google search, the description of the first 25 results are retrieved and placed in the Google pane. These are inserted here as links. The content in these three panes is available to the user at all times.

3. Miscellaneous

   The basic layout of the miscellaneous section will be as shown in figure 5.5.

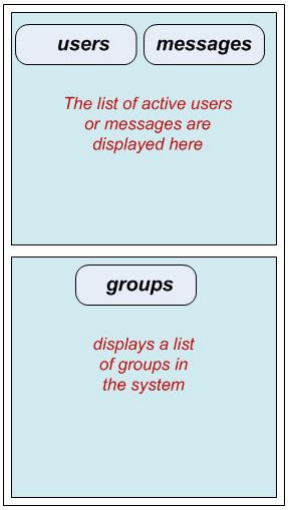


Figure 5.5: Basic layout of the miscellaneous section

.

Miscellaneous UI container represents one of the core abstractions used by this application and is made up of a couple of containers or container widgets which hold different types of data. Initially this section is divided into two main parts; the upper part is the user section which holds information pertaining to users and the lower section is the group section and holds group-related information. The user section is further divided into two main tabular containers. These containers hold list of active users and new messages from other users. These lists should be accessible at all times. The group section holds a list of available groups such that the user can choose from them. This list should be available at all times too.

This section holds containers for diverse content. Figure 5.6 shows how these sections will be changed to show different content depending on the users interaction with the application. These changes can be described as follows:

(a) the tab containers of the upper section are replaced by different tab containers; links and annotations. These containers hold a list of words within the the document currently viewed, that have links and annotations added to them. These two

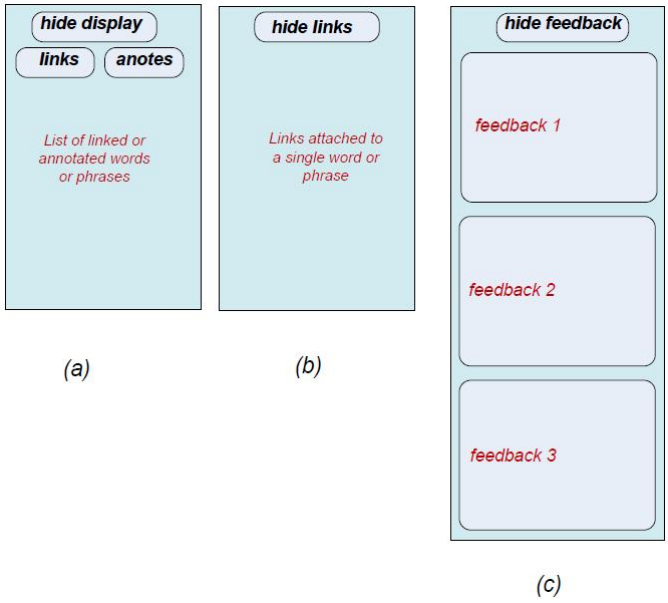


Figure 5.6: (a) tab containers which hold a list of linked and annotated phrases, (b) content pane which holds a lists of links attached to a particular phrase, (c) content pane which displays the annotations added to a particular word or phrase.

.

containers will be created using the ***viewing tools*** menu option of the main menu bar

(b) the upper section is replaced by a list of links attached to a single word or phrase within the document that is currently displayed. This section will be created when the user right clicks on a *linked* word within the document being displayed.

(c) the whole miscellaneous section is replaced by a list of feedbacks attached to a particular annotated word within the document currently being viewed. The transformation of this section as described will be triggered in two ways; by using the ***viewing tools*** from the top main menu and by giving feedback to an annotation.

4. status bar

Figure 5.7: Basic layout of the miscellaneous section

.

The next section which is displayed at the bottom of the page is the status bar. The idea is to use this bar to display general session information such as the user and the

group in which he is currently active in. But the most important reason however is to display the URL of the document which is being viewed. This is important because the page is a single-page URL and the URL displayed in the URL field of the browser is not be the true URL of the object currently being displayed. Figure 5.7 shows how these status bar is divided as described.

5. Workspace

   The rest of the space which is not occupied by any of the other UI sections is occupied by a widget container which is called the workspace. This container widget holds the elearning object currently being accessed in the case of hyLOs content or a document section in the case of Wikipedia or an HTML page in the case of a Google search. 5.2 shows how this section of the page will look like.

UI Design Constraints

The list of active users and groups must be available at all times. As discussed above, the miscellaneous section keeps changing depending on how the user is currently interacting with the program. This means that, in order to utilize the UI containers efficiently, layers of content are placed on top of each other.

Extra options to hide the newly created sections or to entirely remove them from the DOM are added. When these sections are hidden, the list of users and groups are re-displayed. If these sections are barely hidden, the additional toolbar is displayed with additional options to show the newly added sections if needed. In this way, all sections of the UI including the newly and dynamically created sections are available to the user at all times.

5.2.2 Ajax Engine

The component on the browser which makes all the Ajax functionality possible is the Ajax Engine. The Ajax Engine gives the browser scripting ability. The Ajax Engine is made up by a couple of scripts written in JavaScript. These scripts receive commands from the UI, build request and then communicate these requests to the server. When the server responds, the Ajax Engine then decodes the response, and interacts with the UI in a way that is determined by the requests and responses going to and coming from the server respectively.

Apart from communicating with the server, the Ajax engine also handles validation task which do not require any data from the server. An example of such a validation task that is handled by the Ajax Engine is password confirmation during sign up.

5.3 Application Back-end

The application back-end consists of the server and the datastore. Although the Internet is part of the system, it is not part of the application as it is external to the application.

The server has a couple of scripts which it uses to service its request. Some of these are proxy scripts that can grab XHTML documents from the Web. Depending on the command made by the user, the Ajax Engine builds the request and calls a particular script on the server to service this request. If content from the Internet is required, the script then grabs the document and delivers it to the Ajax Engine, which then renders it correctly on the UI. If the content requested had been created previously by the user, then it belongs to the system and resides in the datastore. The Ajax Engine then calls the appropriate script which retrieves the content. Whenever the user creates content by adding notes, annotations, etc, the Ajax Engine calls the appropriate script on the server, which then writes this content to the datastore.

As mentioned in the requirements section 4, the system offers the user the ability to interact with content in the following ways;

- add links
- add and edit annotations
- give feedback to annotations
- add, edit and delete notes

Flowchart diagrams will be used in this section to explain how the back-end is designed to handle the tasks mentioned above.

5.3.1 Handling Links

The flowchart of figure 5.8 shows the processes involved in handling links at the back-end. Whenever a link related request is received, the server examines the request and then decides what to do. The server expects one of the following requests;

- write request: write a new link to the datastore
- getlinks: retrieve linked words for a specific document which is uniquely identified by its URL
- linkentries: retrieve links for a particular linked word within a specific document which is uniquely identified by its URL

- morelinks: write a new link to the database for a word that has already been linked i.e already has one or more links added to it.

For all the above four request types, the system begins by checking if the link has been added within a group. If the group name has not been specified within the request, it is set to nothing (empty). This is done for two reasons;

- content is tied to a group, if that content was added during group work, otherwise it is tied to the user who added it.
- if the content was added within the group, the system has to inform all users currently online and working within that group, that new content has been added.

write request

When a *write* request is received, the URL of the document to which the link is added is examined to determine if the document is a Wikipedia document. If it is, the system checks to see if it has already made an entry for this document within the system. If it hasn't, it writes the document to the local database. See section at the end of this chapter for an explanation why writing the document to the database is deemed necessary.

Section 4.1.3 explained that the user can add a link by selecting a word or a phrase. This word or phrase which as of now will be referred to as the linked word is sent along with the request. The system checks to determine if this word is already a linked word that is, if one or more links already exist for this word. If the selected word is already a linked word, the system writes the link to the ***morelinks*** table in the database. See section 6.8 for a description of the database model. If not, the link is written to the links table. In this way, the system can easily fetch linked words for a particular document without duplicates.

The system then checks if the group name is empty or not. If it is not empty, the system writes the appropriate message into the ***group notify*** table. This table holds all messages which are meant for the users currently active within this group. It is always created when a group is activated and dropped when it is deactivated (see section 6.8). The message taht is written into the group notify table is predefined and describes the action that is currently being taken by group members, for example, adding a link, adding an annotation, writing feedback, etc. Finally the system then builds the JSON response to send to the client, which is basically a success or failure message.

getlinks request

When the *getlinks* request is received, the system retrieves linked words for a specific document, which is uniquely identified by the URL and sends them back to the client. During this process the system differentiates between linked words added by the user or group to which the user belongs and linked words added by other members. As mentioned in chapter 4, users can decide to share or hide their content from other users. The system first retrieves the links added by the user and then the shared links.

The system then builds a JSON response to send to the client, which indicates for each linked object, if it is shared or not.

linkentries request

The *linkentries* request is made up of a linked word whose entries are being requested and a URL to which this linked word belongs. The system then reads out the links from the database, builds a JSON response and returns the response to the user.

morelinks request

When the *morelinks* request is received, the system, does not need to determine in which table to write the request. This is because the morelinks request is sent if and only if a link is added to a linked word. The system then writes the link to the ***morelinks*** table. If the link is added within the context of a group, the appropriate message is written to the group notify table. The system finally builds the JSON response and returns it to the client.

For all the above request, a status message ***false*** is always sent as a JSON response , if the request cannot be serviced properly.

5.3.2 Handling Annotations

The processes involved with handling annotations are shown in figure 5.9. The back-end expects the following request, when handling annotations;

- write request: writes a new annotation to the datastore
- get annotations: retrieves annotated words for a specific document which is uniquely identified by its URL
- edit annotations: writes an edited annotation to the database
- get original annotation: retrieves the original annotation from the server.

write request

The *write* request determines if the annotation has been added within the context of the group. Just as with link handling, it checks to see if the URL points to a Wikipedia document. If it does, it determines if the document has already been written to the system and writes it if it hasn't. Next the annotation is then written to the ***annotations*** table of the database. If the annotation was added within the context of the group, an appropriate message is written to the group notify table. The message is predefined and says that an annotation has been added by a particular user to a particular phrase or word. Finally, the system builds the JSON response and returns it to the server.

get annotations request

When the *get annotations* request is received, the system checks to see if the user requesting the annotation entries is currently within an active group or not. The system then reads out the annotations and then the shared annotations. For each annotation read from the database, the system checks to determine if, the annotation has been edited or if it has feedback added to it. If it has been edited, then it has an original annotation attached to it. All this information is required by the client, when building and showing the annotation display. Finally, the JSON response is built and sent to the client. For each object within the response, the client will be able to determine whether or not an object has been edited and whether feedback exist for this object.

edit annotations request

The *edit annotations* request writes the edited annotation to the database. Since the system always shows the current annotation on the User Interface, the system separates the current annotations from the edited annotations on the back-end. The edited annotations are added to a different table. This is analogous to the links and more links table mentioned above. The system goes a step further and differentiates between the current annotation, the original annotation and edited annotations on the back-end. This is because although the user is always shown the current annotation, he is given the chance to get the original annotation or the edited annotations.

get original annotations request

The back-end receives the *get original* request if and only if an annotation has been edited. The original annotation is then retrieved and sent in the response. For every annotation, there will always be a single original annotation if it has been edited.

get edited annotations request

There can be many edited versions of an annotation. When the *get edited* request is received the system retrieves all these versions, builds a JSON response and sends it to the client.

The reason for storing original and edited versions of an annotation, is that they might have feedback added to them. If the original or an annotated version of an annotation is deleted, the feedback might lose its meaning.

5.3.3 Handling Notes

The processes involved with handling notes are shown in figure 5.9. The back-end expects the following request, when handling notes;

- write request: write a new note to the datastore
- edit note: update the database with the edited version of the note
- delete note: delete database entry for a note
- get all notes: retrieve the original annotation from the server.

write request

When the system receives the *write* request, it determines if the request has been added within the context of a group and, then writes the note to the database. If the note is added within the context of the group, it also adds an entry in the group notify table with an appropriate message. This message describes the action taken and the user taking the action. In this case, the action is simply "*adding a note*".

edit note request

The *edit note* request is received when a user edits a note. The field in the note table of the database identified by the key in the request is simply updated with the new note. The system does not keep track of old notes. If this editing is done during group activity, an appropriate message describing this editing process is written into the group notify table for the group.

delete note request

When a *delete note* request is sent, the system searches the note table of the database for the entry identified by the key or id in the request. This entry is then deleted from the note table. If this deletion process occurs during group activity, an appropriate message is written into the group notify table for the group.

all notes request

The *all note* request is made for a particular URL. The system then retrieves notes added by the user or his group for that URL. Just like with link and annotation, users may decide to share or hide their notes. The shared notes are also retrieved. The JSON response is constructed and sent to the client and indicates which notes belong to the user or his group and which notes are shared.

5.3.4 Handling Feedback

The processes involved with handling feedbacks are shown in figure 5.9. The back-end expects the following two requests, when handling notes;

- write request: write a feedback to the database
- get feedback: retrieve all feedback entries for a particular annotation

write request

The *write* request like all other write request, determines if the user is currently active in a group or not. The feedback is then written to the feedback table of the database. If the user who made the request is currently in an active group, a message is written into the group notify table, which says that the user in question has added a feedback. The system then reads all feedback added for that annotation and builds the JSON response to contain these feedback. The reason behind this, is to show the user not only his feedback, but all feedback for an annotated word. The user can compare his opinion to those of others.

get feedback request

The *get feedback* request is made when a user requests feedback for a particular word or when he adds feedback to the system. Feedback entries are read for a particular annotation which is uniquely identified by an annotation id encoded in the request. A JSON file is built with these feedback entries and sent as the response.

A response status of *false* is always returned if the response could not be serviced or if no feedback for the annotation in question exists.

5.3.5 Group Communication

The back-end design above mentions the ***group notify table*** for all the functionality the system is to offer. This table is created when a group is activated and holds messages destined for a particular group. The table name is built using the group name. Any script in the system that needs to write or read from this table, just needs to know the name of the group and then also build the name of the table. Figure 5.12 describes how the system handles writing and reading messages to and from this table.

The request received by the back-end depends on the activity each group member is currently undertaking. For example, if a user has selected a word and is adding a link to it, the response will indicate this. The system interprets this result and then, builds the table name and then writes a message to the table. The request indicates the user who is adding the link and the action (adding link) he is currently taking. The message written to the group notify table will contain this information. The same thing is done when a user is in the group and writing an annotation, feedback or note. The user request will also indicate the user and what he is doing.

The design for handling, links, annotations, notes and feedback all involve writing to the group annotation table when a write request is received. In this case, the message written to this table is always to indicate the completion of the task which the user was doing. So this group notify table keeps track of the user actions.

If the request received contains a ***read*** instead of an ***action***, the latest entry in the group notify table specified by the id in the read request is fetched. The request always has an id for the required message. This means the system keeps track of the ids for each member in the group, so that they get all messages. When reading the message the system makes sure that, it does not read the messages written to the group notify table on his behalf. For example, if a message is written to the group notify table saying that user A is writing an annotation, the message is written on behalf of user A. When message A sends a read request whose id corresponds to the id of the message written on his behalf, the back-end will simply ignore this request. It doesn't make sense to send a message to user A saying that user A is adding an annotation. But all other users in the group will be able to read this message.

Another important function of the back-end worth mentioning here is the messaging handling. Whenever the back-end receives a private message request, it determines if the request specifies a receiver, a sender, a subject and a message. A subject is optional. The system then determines if the receiver is currently online. If the receiver is online, the message is written to the pmessageson table (see section 6.8) else it is written to the pmessagesoff table. The receiver then reads his message from the pmessageson table if he is online.

If the user is offline, he is notified when he logs in. The messages from pmessagesoff are then retrieved and forwarded to the user identified as the receiver.

Each time a message is read from the pmessageson or pmessagesoff table, it is deleted

from the system. This is not a full- blown messaging system, just a simple way to let users send messages to each other. This can be very useful during group work. Users can also contact users who share the same interest as themselves.

5.4 General System Design Constraint

A general system design constraint is that the documents grabbed from the Internet should not be stored in the system datastore. This is due mainly to copyright infringement issues that might prohibit the use of content from its site in this manner. Secondly, there is no way to determine if content on a particular website really belongs to the owner of that web site. If my application grabs stolen content from the Web and presents it to my users, then I (as the developer) am an accessory to content theft, which is punishable by the law. Finally, if the content is available on the Web and is free for use and redistribution, storing it on my database will simply be a waste of storage space since I can get the content any time I want. Knowing the URL of the content is enough. A disadvantage is that if the URL of this content changes or if the content is taken off the Web, then my users cannot access it any more. This does not, however, mean that, the added content such as annotations, links, etc, becomes meaningless. Theoretically, the tags attached to these notes could make them useful when viewing other documents as well.

A solution to this problem is to store added content locally together with the URL of the document to which the content has been added. If the user grabs a document from the Web in the future and demands to view added content, the application then retrieves the added content identified by the URL of the document being viewed and sends this to the Ajax Engine which then displays it in ways defined by the application. This solution is viable only for content that is free for use and redistribution.

This solution brings with it a new and more complex problem. What if the original document to which content was added changes over time?. This means that at least some of the added content will become irrelevant if a few words are edited or changed. If the whole document is changed or its URL changed, then all the added content for the document becomes totally irrelevant.

There is no concrete solution for this problem but the application attempts to reduce this problem from occurring by providing limited editing power depending on the source of the content. Data from Google Web search will contain to a high degree only websites. The information on these sites is highly volatile. The application is therefore designed in such a way that the user cannot edit results from the google search. This functionality is suppossed to help the user gather information for future use. This solution can however be fine-tuned, such that the application can distinguish between HTML documents and documents of other formats such as pdf. The application could then allow editing of documents of formats other

than HTML on the assumption that, these will likely not change over time. This solution has not been included in this design.

Data from HyLOs is considered quite stable and as of now no solution has been included in this design for the case that the content changes.

Data from Wikipedia is semi-volatile as Wikipedia documents can be edited at any time. See chapter 9, on future work for a proposed solution on how to handle this issue for Wikipedia documents.

Figure 5.8: Handling links on the back-end

.

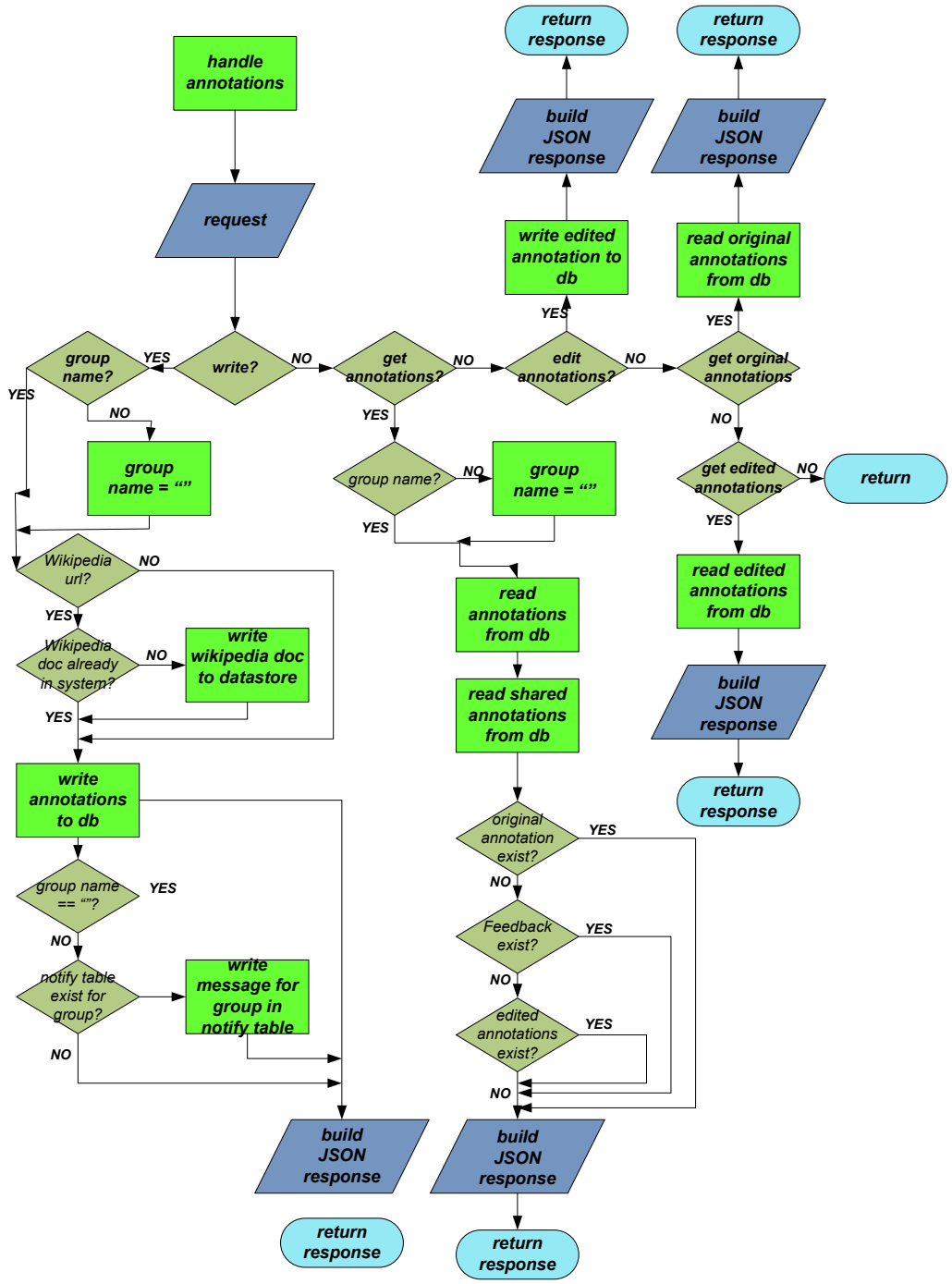


Figure 5.9: Handling annotations on the back-end

.

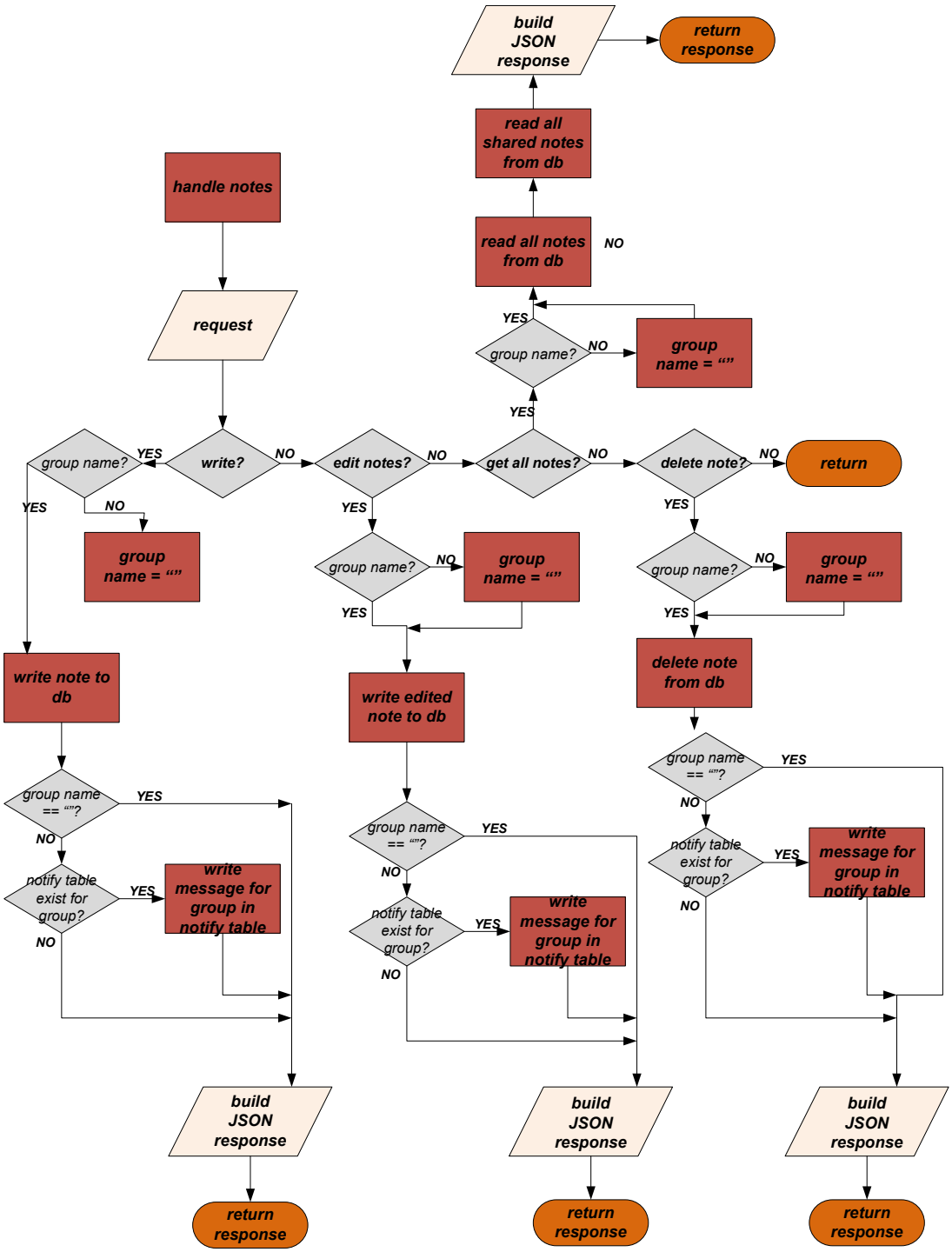


Figure 5.10: Handling notes on the back-end

.

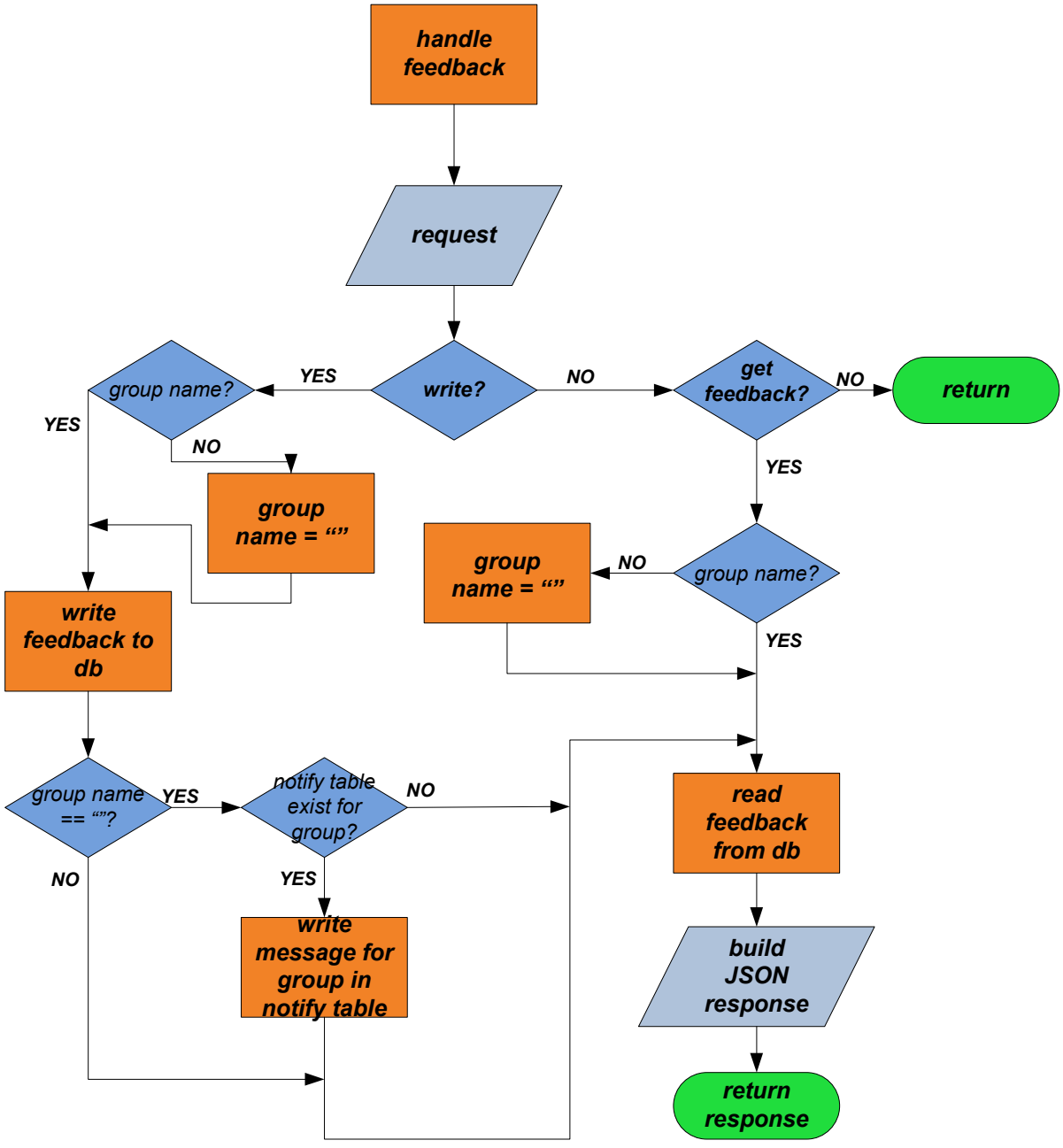


Figure 5.11: Handling notes on the back-end

.

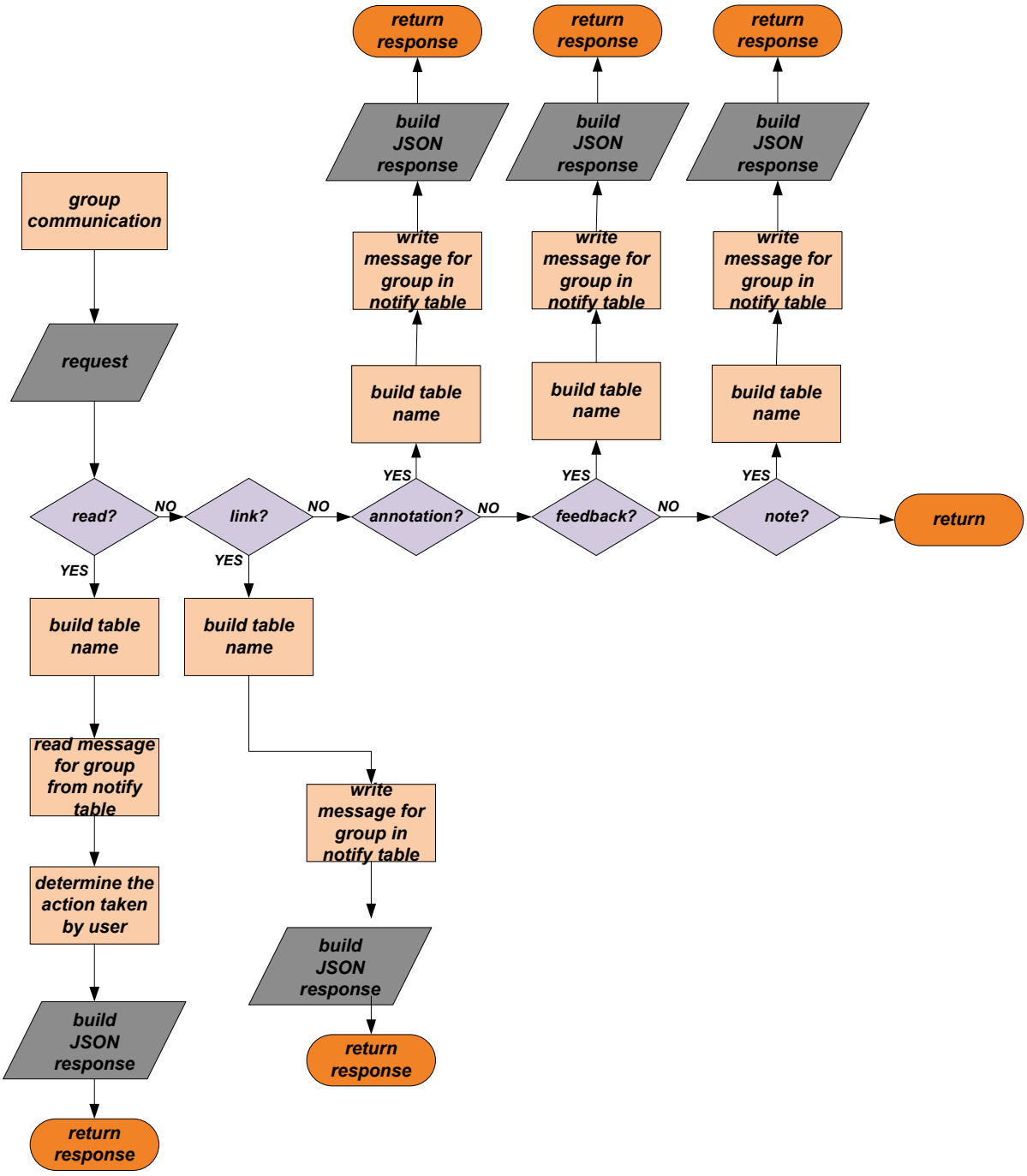


Figure 5.12: Handling group messages

.

Chapter 6

System Specification

This chapter gives a detailed specification of the functionality offered by this application. This specification will involve a discussion on the technological concepts applied and a database model for the data store. The system specification is done using a top-down approach which will begin with the UI. The events and actions set into place when the user interacts with the UI are discussed starting at the UI through to the Ajax Engine to the server and back. At each stage the extent of the implications of each action taken by the user is examined in detail.

The different ways in which a user can interact with the application will be classified under the following services;

- authentication service
- content retrieval service
- linking service
- annotation service
- note service
- message service
- group service

When the user initially enters the application, he immediately gets an overview of the system, who is using the system currently, what groups exist, where to find content and the tools to use for viewing and manipulating content, for coordinating group work and for sending messages to other users. Figure 6.1 shows how the UI initially looks like.

This UI is shown here again since it is the starting point of all functionality within the system and for reference purposes. But the authentication service will be briefly examined

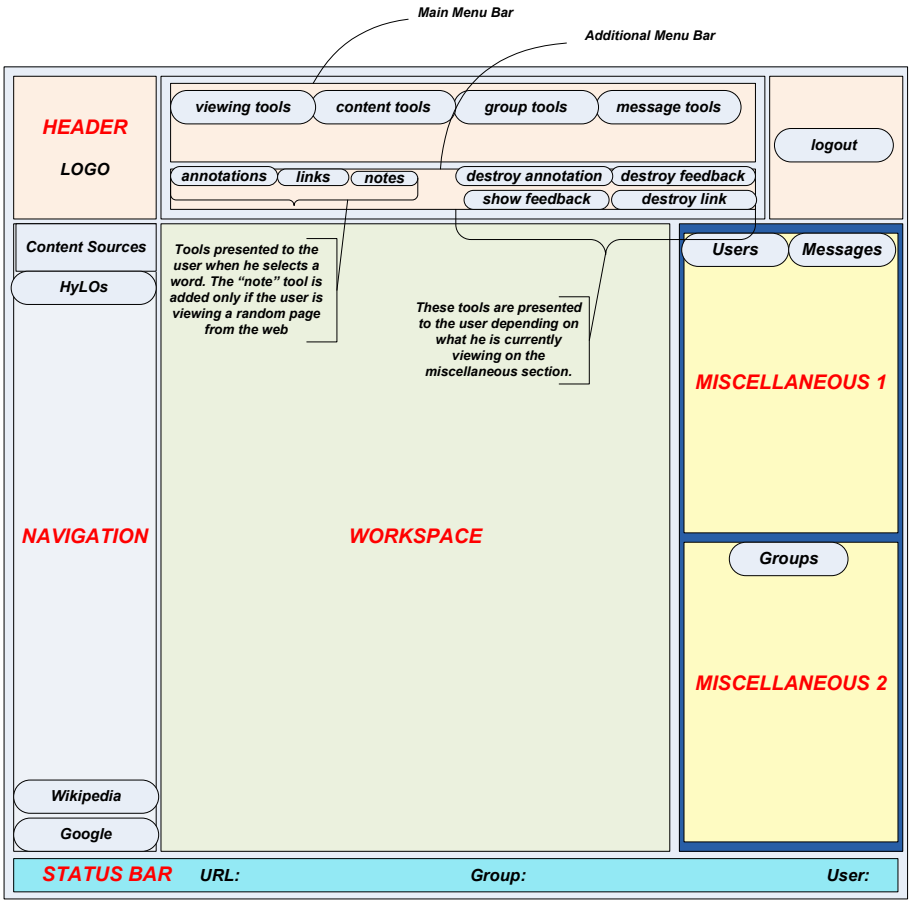


Figure 6.1: Overview of the UI

.

since it is a vital part of the system as a whole and it is implemented fully using Ajax live form concepts. These concepts allow the Ajax Engine to carry out validation tasks that do not require any information from the database.

6.1 Authentication Service

The authentication service is important because the system is designed to keep track of the activities of users. These activities include, what they are currently doing, what group they are in, if they are currently taking part in any active group work and the type of content they are adding to the system. During registration, a user can also give his field of interest as

well as set the status of his content to public or private. Public means he wishes to share his content with other users and private means hide from others.

The users will be required to sign up before they can use the application. During sign up, the users choose a username, a password as well as a field of interest. They are also required to give in their email address. The email address and password should respect a certain pattern and the username should not already be taken. The validation of the email address format and the password format is done by the Ajax Engine. These are processes which will otherwise be carried out on the server.

For the validation of the username, the Ajax Engine makes a request to the server to determine if a given username is allowed, i.e., not already being used by a different user. This validation also uses the Ajax patterns (submission throttling) discussed in sub section 3.3.2. When the user gives in the username and leaves the username field, the call to the server is made and the user is immediately informed if the username given is allowed of not. The same principle is also used during log in. When the user gives in the username, a call is made to the server which checks if this username is registered in the system. The user is immediately informed if the username is already taken.

The user is also required to confirm his password during sign up. The Ajax Engine does the comparison of the two passwords. The comparison is done incrementally and as the user types.

The users will identify each other via the username. The user also gives his Interest during sign up. This is done in the form of a list of the group names registered in the system which are presented to the user at sign up in form of a drop down menu. The user can either select from this list or enter something else in the field provided. This field of interest will be seen by other users when they move their mouse over the username of the user. They will then have the opportunity to invite the users to join their group if they share the same interest.

To summarize, the authentication service helps users identify each other and gives them the opportunity to know each other's interest without making contact. It also helps the system keep track of users activities and of the status of the content they created, i.e. if it is shared or hidden.

6.2 Content Retrieval Service

The application offers the possibility to begin with content from two main sources; hyLOs and Wikipedia. The system also offers the possibility to search the internet for documents using *Google Web Search*. The content tools of the UI are used to retrieve content. These tools are shown in figure 6.2

A user supplies a keyword and specifies the content source and the application then uses the keyword to retrieve the content from the specified source. The keyword is relevant

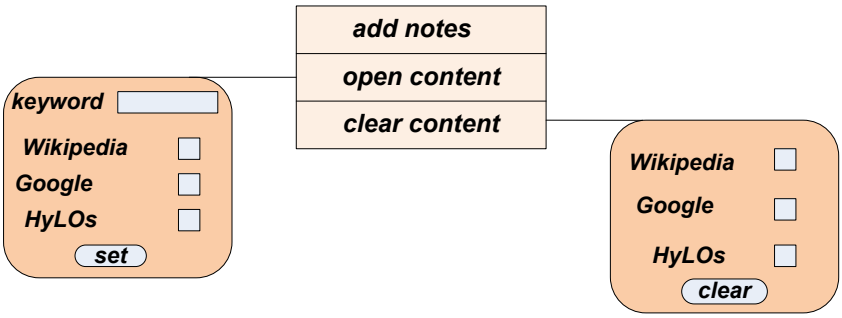


Figure 6.2: Content retrieval tools

.

only for Wikipedia and Google searches. If hyLOs is selected as a content source, the HAW course overview Page is retrieved and displayed to the user.

HyLOs Content

HyLOs as discussed in section 2.3 is an Learning Content Management System (LCMS) with the content standardized according to the IEEE LOM standard. This means that the data is packaged into eLearning objects. Each of these objects can be uniquely identified by their URL. This makes the content from hyLOs suitable for use in this application since the extra content added to the eLearning objects can be managed easily. This means that the added content, can be identified, written to a database, retrieved and displayed with ease by simply using its URL. Moreover, since the whole content is divided into smaller chunks, it makes the pagination easy. Working with smaller chunks of data is also easy and more comfortable from the users point of view. When the user requests content from hyLOs, the content overview page will be retrieved and displayed. The user can then start a learning unit and begin using the tools provided by the application to interact with the content.

The system retrieves and displays the XHTML page with all the formatting applied. This has the advantage that, the user will see the page as if he had opened it directly

on the hyLOs website. This is of importance for users who are familiar with hyLOs as they can start to work on the content immediately and will not need to spend time getting used to the presentation of the content, which will be the case if the formatting was changed. On the hyLOs website, the location of the table of contents is not immediately obvious to the user. When the hyLOs overview page its requested, the table of contents will be retrieved and displayed in the hyLOs pane in the navigation section. This will enable the user to jump between eLearning objects or between eLearning units. This table of contents will be accessible at all times.

Whenever the user is working on content from hyLOs, he will have the possibility to view content added by other members of his group if he is in a group or content added by himself if he is working alone. The added content that is uniquely identified by the URL of the eLearning object will then be retrieved and presented to the user for viewing or further processing.

Wikipedia Content

The style in which information is presented in Wikipedia also makes it suitable for use in this application. The information in Wikipedia is broken down into different sections that can be uniquely identified by their URL. This means that the content could be presented to the user in the same way as the hyLOs content. Initially when the user makes a request for a document from Wikipedia, the application checks if the document exists or not. If it doesn't exist, the user will be informed. If it does, the table of contents is retrieved along with the introduction of the article and presented to the user. The table of contents will be placed in Wikipedia pane in the navigation section.

Clicking on an item in the table of contents should make a request only for this specific item. The item will then be retrieved and displayed to the user with the formatting applied to it on Wikipedia. Given the fact that Wikipedia is being used by many people today, makes the advantage of presentation familiarity more important here. The user will not only be familiar with the style of presentation, but will be able to work on the data in smaller chunks.

Dividing the data from Wikipedia into smaller chunks is also related to the fact that the system should not store original content from content sources. Since documents are constantly being edited on Wikipedia, it cannot be guaranteed that the edited section of a Wikipedia document will still be the same if the user wants to view added content in the future. This is a general problem with Web content which has been discussed in the *design constraints* section 5.4. A partial solution to this problem, which is based on the division of the content into smaller sections will also be discussed in this section. See chapter 9, which explores this problematic a little bit further.

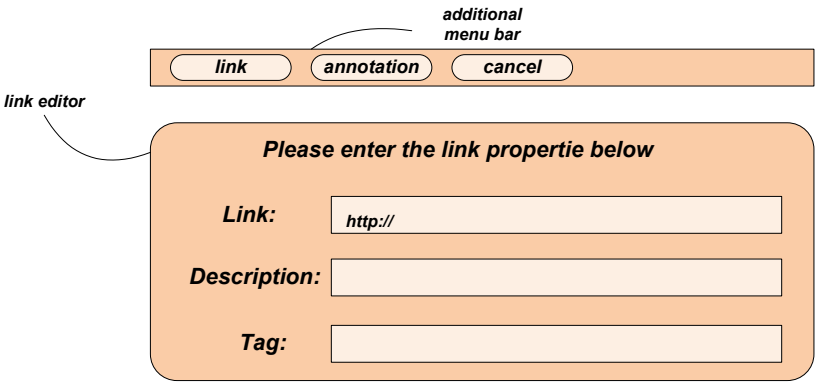
Google Web Search

The application offers users the ability to search the Internet for documents using the Google Web search. But as opposed to the hyLOs and Wikipedia content sources, this is different in the sense that, users will not be given the opportunity to add annotations to these sites. This feature is being included in this application to help users gather information from the web for use within the group or for future use. Users may, however, add notes to these Web pages. They may summarize the content of the Web page in this note such that when the user accesses the page in the future, he will determine its value just by looking at the notes

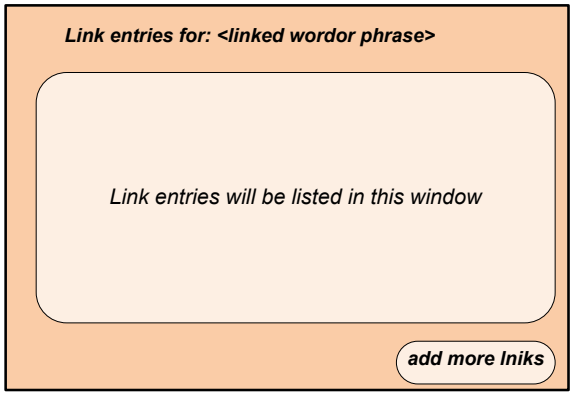
6.3 Linking Service

The users are allowed to add links to documents. When ever a user selects a word, the additional menu bar is shown in the notification section of the header. One of the options offered by this menu bar is ***link***. Clicking on this option opens the link editor shown in figure 6.3(a). The additional menu bar is also shown on the top section.

The user can give in the link URL, the description and a tag. The description is what the users will see when they view the link. The tag is added here, so that if search algorithms are written to sift through the data in the system, they can provide semantic meaning to the links. This functionality is, however, not implemented as of now. The users are allowed to add more than a one link to a single word or phrase. Figure 8.10 will be used to display the links within the document when a user clicks on the link. The option to add more links will be shown only if the user owns the link or belongs to a group that does. When the user right-clicks on the linked word or phrase, the link entries will be opened on the miscellaneous UI container. If the user navigates away from the current document, these link entries will still be available, such that the user can use them in another document. The user might want to see only the linked words within the document. Using the ***show added content*** option of the viewing tools opens a list on the miscellaneous section of the UI which shows all the linked words within the current document (see figure 5.6). If the user opens a different document in the workspace, this section remains open until the user explicitly closes it. The user can, however, hide or show this section using the additional options placed on the additional menu bar and on the miscellaneous UI section. If the user right-clicks on a linked word from the list, the linked URL is displayed. The idea is to let the user copy this URL and attach it to different documents if he wants. In this way, documents can be linked to each other. This can play a big role, when writing an algorithm to find content within the system. These links can then provide semantic relationships between different eLearning objects. The tags provided when adding links can offer the semantic property required to build these relationships. For a start, a simple relationship such as, document A is related



(a) Link Editor



(b) Link Display

Figure 6.3: Link editor and display

to document B can be constructed if these two links have the same tags. If the tags are accurate and meaningful, many such relationships can be constructed between the links leading to a semantic link network. This functionality is not implemented in this application. See sub section 9 for a future implementation suggestion that makes use of this idea.

6.4 Annotation Service

The annotation service offers the following functionality;

- adding annotations
- annotation feedback
- editing annotations

Adding Annotations

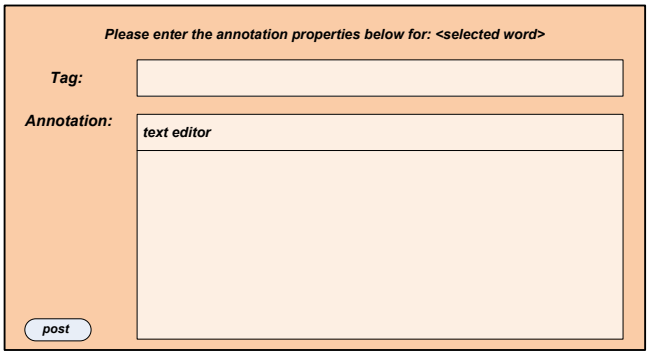


Figure 6.4: Annotation editor used to add links to the content

.

The annotation service also allows users to add content to the current document by selecting a word or phrase within the document. The annotation editor which is used for this purpose is shown in figure 6.4.

Just like with the link service, the ***show added content*** option of the viewing tools opens a list on the miscellaneous section of the UI which shows all annotated words within the current document. What differs here is the motivation. For a given document

or eLearning object, a word can be annotated only once for a given group. This means that for a given group, two different annotations are not supposed to exist for the same word within a document. The user might use the show added content option to check which words have already been annotated before using the annotation service. If the user, however, does not make use of this option and selects a word that already has an annotation, the application informs the user of this and offers him the opportunity to add a link or a feedback to that word. This also applies to users working alone.

Feedback Annotations

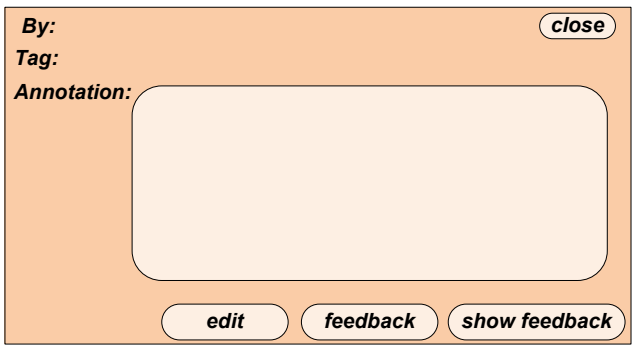


Figure 6.5: Displaying annotations within document

.

If a word has already been annotated by another group member, the user uses the ***show/hide annotation*** option from the viewing tools to show all annotated words within the document. The annotated words are highlighted. When a user clicks on an annotated word figure 6.5 pops up.

Selecting the show feedback option, retrieves all feedback for this particular word and displays them on the miscellaneous section (see figure 5.6). The Ajax Engine sends the request to the server which then retrieves the URL, using the keyword, document URL and group name or username in case the user is working alone to retrieve the feedback. When the Ajax Engine receives the response, it then creates a container to hold the feedback, hides the current content of the miscellaneous section and displays the feedback container with the most recent feedback at the top. This container shows when the feedback was added, who added it and the feedback itself. Feedback from the different users is placed one after another. This can be viewed as some sort of

"***document chatting***" where different opinions from different people working in the group are being collected as with regard to a particular issue. This enhances the document quantitatively and depending on the users within the group also qualitatively.

When the user clicks on the feedback option, the annotation window of figure 6.4 opens and the user enters his feedback. The Ajax Engine sends this feedback to the server which writes it to the database. If no errors occur, the server responds with a list of all feedbacks that have been added for that word or phrase. The Ajax Engine, then hides the current contents of the miscellaneous section of the UI, creates a container and stashes the feedback entries one after the other beginning with what has just been added by the user, and finally places this container in the miscellaneous section. In this way, the user is able to follow what is being written by other users.

When the container holding the feedback entries is displayed, it stays on the UI for as long as the user needs it. The user however is given the possibility to hide, show or remove this container at anytime. The container which holds the list of added container can be still be on the UI. These containers all stay there as long as they are needed by the user.

Editing annotations

The user who creates an annotation has the right to edit it. Figure 6.6 shows how the annotation display changes when the user is editing the annotation.

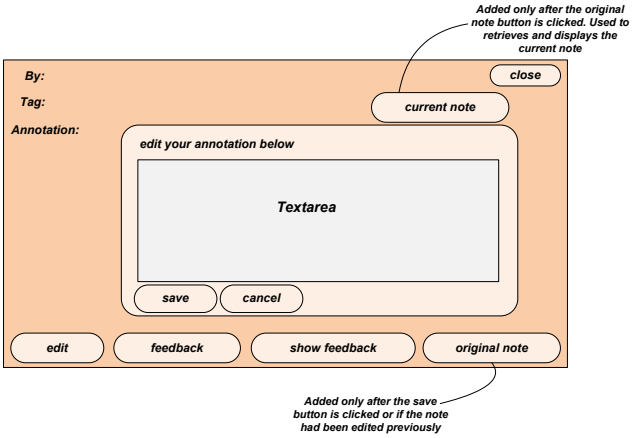


Figure 6.6: Editing an annotation

.

Editing an annotation uses an Ajax pattern of ***edit-in-place***. The annotation being edited is replaced with a text area containing the annotation entry and two buttons for

saving the edited text or for canceling the editing process. If the user clicks the save button after editing the annotation, the annotation is written to the annotations table on the datastore. The original annotation is not lost though. It is moved to a different table. The current annotation is then displayed and a new option "***original note***" is added to the display of the annotation. If the user clicks this button, the Ajax Engine retrieves and displays the original annotation. A new option "***current note*** " is added to the annotation display. This option retrieves the current annotation and displays it.

If the annotation is edited a second time, the system will save this annotation under edited annotations. The system differentiates between the original annotation, the current annotation and edited versions of the annotation. Since users give feedback to annotations, it is important that the original annotation and any edited versions of that annotation should not be lost when the annotation is edited or the feedbacks added as response to this annotation may lose meaning.

When the annotations are initially displayed, the "***original note***" will be displayed if the annotation had been previously edited. The server lets the Ajax Engine know if the annotation had been edited before.

Annotations and feedbacks are persisted within the system. They can be viewed and edited in the future. This means that the process of adding an annotation to content and giving feedback to this content, lets the users create a layer of new content on the already existing content.

6.5 Note Service

The users can add notes to elearning objects or documents. The idea of adding notes is just to help users summarize the contents of the document being viewed for instance. When this document is viewed in the future, the user will be able to judge its importance just by reading the notes attached to the document. This is just an example. Users are not restricted as to what they are allowed to write as notes.

There are two main ways of writing notes. One is by using the "***add note***" menu option (see figure 6.2). This opens up an editor similar to the annotation editor (see figure 6.4) and the user can then write his note. This method is available for use at all times. The second method which is available only when a web page from the Wikipedia search is opened within the application, makes use of the annotation editor itself.

Whenever a note is added to the Web page or elearning object, the note is sent to the server along with the document or Web page URL, user name and group name if the user is currently active in a group for writing to the datastore. The note is also displayed on the top right hand corner of the Page. An option is also added for deleting or editing the note.

The viewing tools also has an option for viewing notes added to the document. The notes are then retrieved and displayed in the misc UI container in a similar way as annotation feedback. Notes that have been shared are also shown. If the user owns the notes, extra options are added for deleting and editing the notes.

6.6 Message Service

Figure 6.7: Message Editor

.

The messaging service lets users communicate with each other as shown in figure 6.7. If users are online, they are notified immediately about a new mesage. They can then click on the message tab of the miscellaneous section (see figure 6.1 at the beginning of this chapter) to read the message. If they are offline, the message is written to the datastore and they are informed when they next login

Such a simple messaging system has two main advantages with respect to this application. Moving the mouse over a username displayed in the active users list shows the user's interest. A user might invite another user to join a group based on their shared interest or use the messaging system to communicate with other users with similar interest. This advantage is even more important if one of the users is offline. The drop down list with the names of registered users also shows the interest of the users. If a user is online, he can send a message to offline users who share similar interest as him. Such a message might be to arrange a meeting time to meet online and discuss or work on a document.

6.7 Group Service

The group tools of the main menu bar offer three options to the users:

- new group
- join group
- group options

The new group option just lets a user create a new group. As of now, anyone can create a group.

The join group option lets the user choose a group name and join the group. The user can also join a group by user the group tools on the main toolbar.

When a user logs in, a list of all groups within the system is presented on the lower half of the miscellaneous UI section (see figure 6.1 at the beginning of this chapter). When a user moves his mouse over each entry in the list, the name of the owner of the group is displayed. If a user right-clicks on an entry, a menu opens up. This menu is as shown in figure 8.12

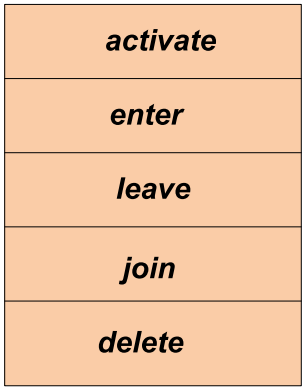


Figure 6.8: Group Menu

.

The activate option as the name says is used to activate the group. A group is active if any group members currently online are actively working in the group. When the list of available groups is initially displayed or updated, groups which are currently active are highlighted. When a group is activated, users can then click the *enter* option to take part in the group discussion. Only users who are members of a group can activate or enter a group. The user can leave the group any time by using the leave option. Using the back button of the browser when working within the group has the same effect as leaving the group. But in this

case, an alert warns the user that he will be thrown out of the group should he proceed. The join option lets the user become a member of a group. The user can delete the group if it belongs to him, i.e. if he created the group. In this case, all the content related to this group is ***released***. This means this data will be open for all to use since it is no longer tied to any group. It becomes system data. This means that content generated over time will not be lost even if the group does no longer exist.

6.8 Database Model

The database is an important part of the system because content created by the users needs to be stored somewhere. It is also required for authentication by the server.

Figure 6.9 shows the database model used by this system. This section briefly describes the database tables in relationship to the type of data they hold.

users and activeusers table

The users table holds the user's personal information. The most important fields in the users table are the userId, the fieldOfInterest, the username, password and the openstatus. The userId and the username uniquely identify the user. Within the system, the user will be known to other users via their username. The fieldOfInterest lets the users specify what their area of interest are. The user can, however, change this in the future by using the user options in the users tool. The openstatus field lets the users determine the status of their content. This field can be set to public or private. If it is set to public, then the user wishes to share his content with other users. If it is set to private, then the user wishes to hide his content from other users. This feature can also be changed from within the application using the user options.

links and morelinks tables

The links and morelinks tables are used to store all link related information. The linkid in the links table and the morelinkid in the morelinks table uniquely identify the links added. The links table is used to store all links added to a document. Each time the system attempts to add a link for a word that has already been linked, the link is added to the more links table. The link display window in the user interface also gives users the feature to add more links to a linked word or phrase. This links are added directly to the morelinks table.

Below is a brief explanation of the other fields or columns in both tabels;

- docurl: uniquely identifies the document to which the link has been added.

- seltext: holds the selected word or phrase.
- linkurl: holds the added link.
- description: holds the description of the added link. This is what will be shown to the users in the link viewing window.
- urltag: holds a tag which points to the added link.
- grpname: holds the name of the group to which the link belongs. This may be left empty in case the user is in the normal mode of the program, that is, working alone.
- owner: holds the username of the user who added the link.
- sessionId: holds a session id which is dynamically generated when a user logs in
- timestamp: used to monitor user activity
- userlevel: holds the user level. This level determines what rights the user has in the system.
- date: holds the date the link entry was made
- openstatus: determines the hide or share status of the added link. This is in turn determined by the hide and share status of the group if the link is added in the context of a group or by the openstatus of the user if the link is added in the normal mode of the application, that is, by a user working alone.

  The activeusers table shows which users are currently online and if they are actively participating in a group. The timestamp can be used to track user activity such that the user is thrown out of the system if he is not active for some time.

The linkid field in the morelinks table points to the linkid field in the links table, that is, it is a foreign key in the links table. In database language SQL (Structured Query Language), a foreign key within a table is always a primary key in another table.

Annotation, feedback, editednotes and originalnotes tables

The annotation table is used to hold annotations added by users. The table has the following columns;

- docurl: uniquely identifies the document to which the annotation has been added.
- seltext: holds the selected word or phrase.
- atag: holds a tag which points to the added annotation.
- anote: holds the annotation.

- grpname: holds the name of the group to which the annotation belongs. This may be left empty in case the user is in the normal mode of the program, that is, working alone.
- user: holds the username of the user who added the link.
- date: holds the date the link entry was made
- openstatus: determines the hide or share status of the added annotation just as in the link table.

The noteid, which is the primary key is used as the foreign key in three other tables. These tables are the originalnotes, editednotes and feedback tables. This means that all these three tables need to point back to the annotation table.

The feedback table is used to store annotation feedbacks. The noteid entry then identifies the annotation to which the feedback has been added. The *fbword* corresponds to the seltext column of the annotation table. The *fbtag* holds the tag word or phrase added by the user and points to this feedback. The column *fbnote* holds the feedback. During retrieval of the feedback all that is needed is the noteid of the annotation or the *fbid* of the feedback itself which uniquely identifies the feedback.

The originalnotes table is used to store the original annotation if the user edits his annotation. This table is used because the system gives the user the feature to view the original annotation that was added to the system, no matter how many times the annotation has been edited and also because the system always displays the current annotation. This table has an onoteid which uniquely identifies its entries. Knowing the noteid of the annotation is also enough in order to fetch its original annotation.

An annotation can be edited as many times as the user wants provided he is the owner of the annotation. All these edited versions of the annotation are stored in the editednotes table. A feature to view the edited versions of the annotations is also presented to the user. Apart from the current annotation, which is stored in the annotation table, or the original annotation, which is stored in the originalnotes table, all other versions of the same annotation are stored in the editednotes table and are uniquely identified by the noteid of the annotation they point to in the annotation table, or by the editid of the editednotes table.

notes table

The notes table holds the notes added by the users. Apart from the docurl, grpname, user, date and openstatus columns already explained above, it has the following three fields;

- note1id: holds unique id which can be used to reference this note.

- notetag: holds a tag which points to the added note.
- notetext: holds the added note.

availablegrps and activegrps tables

whenever a user creates a group, an entry is made in the availablegrps table for that group. The group is identified by a unique id grpId. The availablegrps has the following important columns;

- grpName: group name.
- members: members of the group.
- owner: owner of the group.
- date: when the group was created
- open: determines the content hide or share status for the group.

Whenever a group member activates the group, an entry is made in the activegrps table. This table holds the name of the person who activated the group. This information is supplied to each user when they join the group. This table also keeps track of the document the group is currently working on in the currenturl column. In this way, the correct URL is loaded when a new user joins the group discussion. This table also keeps track of the users currently taking part in the group discussion in the currentuser fields. The group is automatically deactivated if no users are currently using the application, that is, number of users in the currentuser field is zero.

pmessageson and pmessagesoff tables

These two tables hold private messages that users send to each other. The pmessageson table holds the messages for the users who are currently online. The client will constantly pool this table for new messages. The pmessagesoff table holds messages for users who are currently offline. The messages in this table will be read once at each time the user logs in.

Both tables specify the sender and receiver of the message, when the message was sent and the message itself. Whenever a message is read and sent to the user, it is removed from the table.

wikipedia table

The designed constraints described at the end of chapter 5, describes the data from Wikipedia as being partially volatile. A proposed solution is to store a section of the

Wikipedia document if content is added to it. If the user views this section of the Wikipedia document in the future, the current section and the store section are displayed side by side for comparison. The Wikipedia table lays the foundation for this solution by storing a section of the Wikipedia document if content is added to it. This is stored in the wikidoc column. The docurl points directly to the section of the Wikipedia document that has been edited. The noteid and the linkid points to the annotations and the links on the annotation and link table respectively that have been added to the Wikipedia document written to this table.

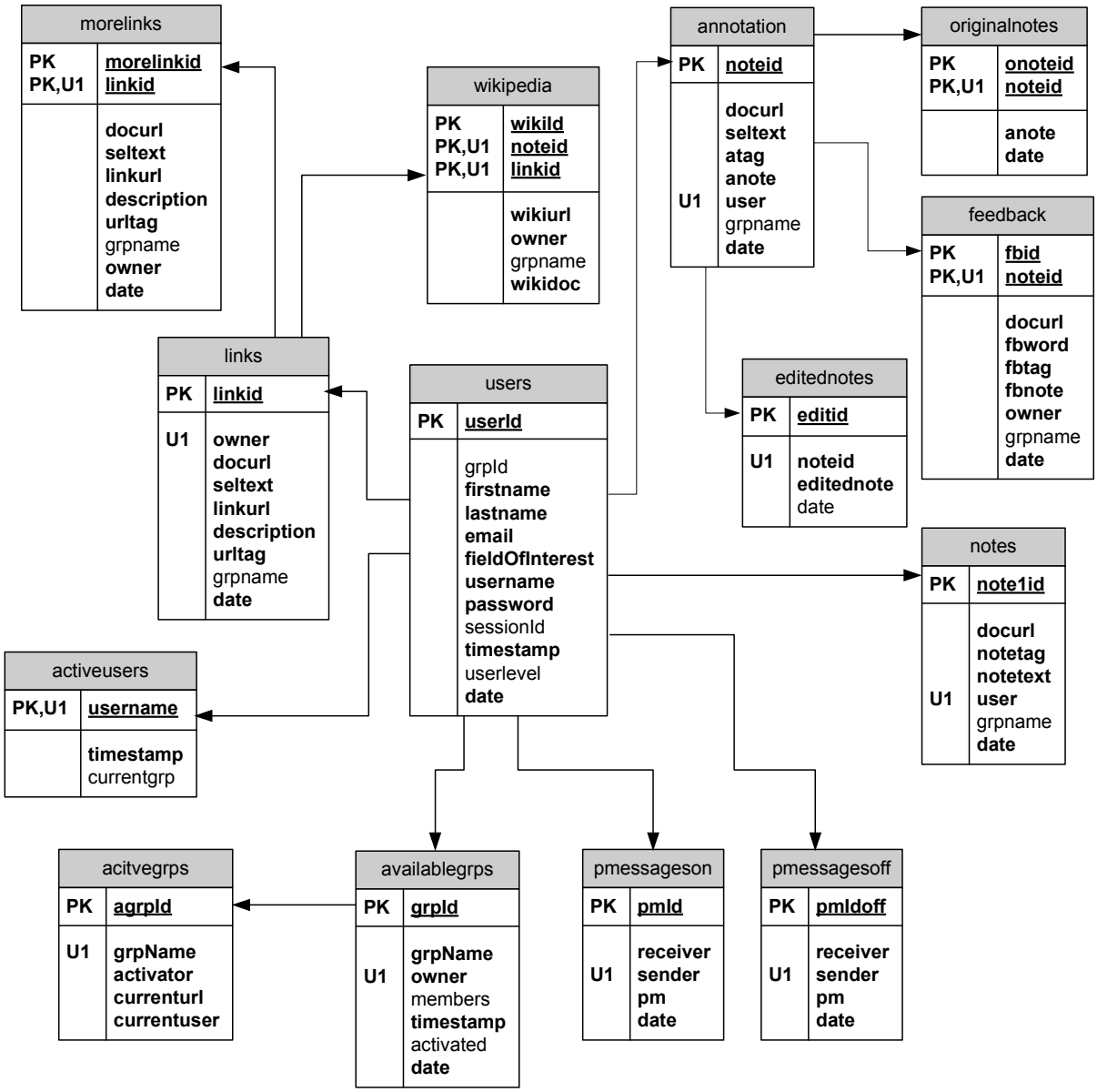


Figure 6.9: Database Model

.

Chapter 7

Application Implementation

The application implementation described in this chapter is based on the Ajax techniques and principles discussed in chapter 3 of this thesis. This chapter starts off with a brief introduction of the development environment and the tools used during the implementation of the concepts discussed in chapter 5. This is followed by a general discussion of the software architecture used for all functionality within the system. The implementation of the functionality is then discussed. This discussion is grouped into three different sections;

- content retrieval and display
- application tools : this includes tools for adding annotations, feedback annotations, links, notes and sending messages to other users.
- group tools: these include techniques implored to help coordinate consistent communication and content interaction within the group

Implementing the concepts drawn up for this project, involves writing a lot of code in JavaScript, PHP, CSS and SQL. Only a few snippets of written code will be shown in this section. A CD containing all the source code will be attached. This section will make references to modules on this CD.

7.1 Development Environment

The complete development environment is made up of PHP Eclipse IDE, the Zend Debugger and MySQL Workbench at the back-end. The front-end comprises of the Firebug Web development tool and its Dojo variant. Each of these tools and their uses during the implementation of this application are briefly discussed below.

This application was implemented using an PHP Eclipse Integrated Development Environment (IDE) called PHP PDT. The PDT project provides a PHP Development Tools framework for the Eclipse platform. This project encompasses all development components necessary to develop PHP and facilitate extensibility. It leverages the existing Web Tools Platform (WTP) and Dynamic Languages Toolkit (DLTK) in providing developers with PHP capabilities [23] .

PHP PDT Project was choosen for the implementation of this thesis for the following reasons;

- it is a free open source development tool
- it provides basic syntax coloring and code assist for Javascript, HTML, CSS and PHP
- it supports DOJO
- it provides HTML real-time error detection
- it supports PHP 4 and 5
- it has a built-in debugging client that operates the Zend debugging engine used for debugging the PHP code.

Visit this Website to view a list of all advantages offered by the PHP PDT Project [24].

The Zend Debugger is used to debug the back-end code written in PHP. The Zend Debugger is a full-featured php debugger engine. It is an interactive tool that allows is used to debug PHP scripts locally or remotely, from an IDE or from the console. As mentioned above, PDT has a built-in debugging client that operates the Zend debugging engine used for debugging the PHP code. This client was configured to communicate with the Zend debugging engine. The PHP binary included with the Zend Debugger client was also configured to use an external copy of the php_mysql.dll or php_mysqli.dll file to enable the debugger to work with MySQL statements. For more information on the Zend debugger visit this Website. [25]. For more information on debugging PHP in PHP PDT read the article on this site[26].

The debugging of the JavaScript code was done using the Firebug. Firebug is a Web development tool that allows the editing, debugging and monitoring of the CSS, HTML and JavaScript live on any Web page. The features and versions of Firebug can be found at this Web site [27]. The external *addon* for Firefox used in this project can be found here [28].

[23]PHP Development Tools Project http://www.eclipse.org/pdt/

[24]Advantages of PHP PDT Project http://www.zend.com/en/products/studio/comparison

[25]Using PDT : Installing the Zend Debugger http://www.thierryb.net/pdtwiki/index.php?title=Using_PDT_:_Installation_:_Installin

[26]Debugging PHP using Eclipse and PDT http://www.eclipse.org/pdt/articles/debugger/os-php-eclipse-pdt-debug-pdf.pdf

[27]Firebug sitehttp://getfirebug.com/

[28]Firebug addon for Firefox browser https://addons.mozilla.org/en-US/firefox/addon/1843

Dojo comes packaged with Firebug Lite, which includes the more useful features of Firebug. Adding the line in Listing 7.1 to the head of the main HTML page, adds the Dojo debug window to the browser being used.

```
1
2  <script type="text/javascript">
3  /*
4    Adding the Dojo debug console to the browser. The value of variables ↩
         can then be observed on this console by
5    using console.log(varable name);
6    */
7     var djConfig = "";
8     djConfig = {
9        isDebug : true
10     };
11
12 </script>
```

Listing 7.1: Activating the Firebug lite debugger window incorporated in Dojo

The database design and modeling was done using MySQL Workbench. MySQL Workbench is an integrated tools environment that enables a DBA, developer, or data architect to visually design, generate, and manage all types of databases including Web, OLTP (Online Transaction Proccessing), and data warehouse databases. It includes everything a data modeler needs for creating complex ER (Entity-Relationship) models, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort. Visit this site for more information [29]. To learn how to use MySQL Workbench, see the article on this site [30]

7.2 Software Architecture.

The application can be viewed as having three independent layers as shown in figure 7.1. These layers show the basic principle used when implementing this project.

The presentation layer is responsible for the user interface. The presentation layer is the single HTML page loaded during user registration or log in. It loads the UI components for the user to register or log in. Upon successful registration or log in, it loads the Dojo UI components to build the main user interface of the application.

The logical and communication layer acts as an interface between the UI and the data layer. The *Ajax Engine* written in JavaScript forms the core of this layer. It initially assist in

[29] MySQL Workbench http://www.mysql.com/products/workbench/

[30] MySQL Workbench Turtorial http://downloads.mysql.com/docs/workbench-en.pdf

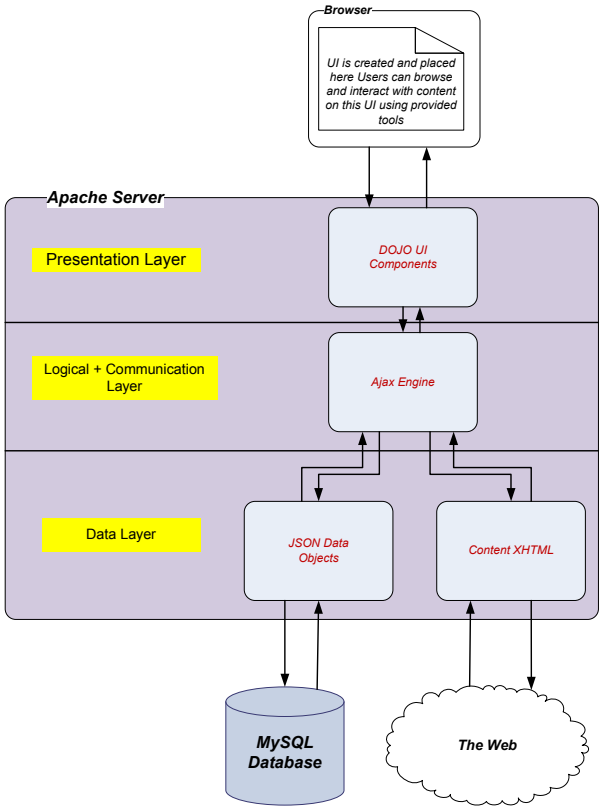


Figure 7.1: Use case showing system tools

.

user validation during authentication. When the user logs in and starts using the application, this layer determines how to respond to user input and modify the UI accordingly. This decision depend on who is currently logged in, what mode the application is running in and how the user is interacting with the application. Apart from these logical tasks, this layer also opens and coordinates communication with the data layer. This communication involves sending user data or created content to the server, sending requested data to the client or simply pooling the data store to see if a specific value has changed. How this layer is implemented to carry out these functions will be looked at in detail in this chapter.

The last layer is in charge of saving user data or user created content to the database as well as retrieving data external to the application or saved in the database. When reading data from the database, the raw data is converted to the JSON format. This has the advantage that the JSON objects which follow the format of JavaScript script objects can easily be

consumed and used by the application logic, which is also written in Javascript. The second component of the data layer is made up of the XHTML content, which is retrieved on demand.

All the functionality implemented in this application will follow the software architecture of figure 7.1.

7.3 Application Functionality

7.3.1 Getting Started: Dojo Toolkit and Application Module Paths

In order to the able to use the advantages offered by Dojo, the Dojo toolkit is initially loaded into the HTML head tag as shown in listing[31]. This is loaded along with the style sheets required by the Dojo UI components. Dojo defines three css themes; *tundra, Soria and nihilo*. In Dijit terminology (see chapter 3, section 3.4 for the meaning of Dijit), a theme is a set of fonts, colors, and sizing settings for components so they look good together. The tundra theme was choosen for this application. That is why the class of the body of the application is set as tundra.

Apart from the styles, the module paths are set in the main section of the HTML page. Initially Dojo calculates the application module path as the *dojo-module-path* i.e. the path where the *dojo.js* file is loaded from. If no other module paths are set, top modules will be loaded as siblings of the dojo.js. These module paths are set in the head section of the main XHTML page.

Tightly couple with this Dojo modularity pattern is the word *dojo.require*. Whenever a module is created, *dojo.provide* creates its module object. Another script which makes use of the created module can load it once by using the *dojo.require* function. The dojo.require is used in this way to load the modules that will be referenced within the page.

Dojo.require is used to load a module called dojo.parser. This module is required for all pages using declarative Dijit. Dijit widgets are created declaratively or programmatically. Declarative widgets are those that use nonstandard HTML attributes such as dojoType=. Setting the **parseOnLoad** attribute of the djConfig object to true in the head section of the main XHTML page, tells the dojo.parser module to parse all declaratively created widgets and create JavaScript objects out of them. Widget class methods can then be applied on these Javascript widget objects created by dojo.parser, in much the same way as style attributes are applied on pure DOM nodes.

After loading the Dojo toolkit with the required dojo modules and setting the desired theme and module paths, development of the main application begins.

[31] CD -> Source Code -> JavaScript -> client -> eui -> main-xhtml

7.3.2 User Interface

The main HTML page is initially divided into four main containers called ***divs***. There main container div holds two other divs. These are the *signupForm* div and the *loginForm* div. The other remaining two divs are the *statusReport* div and the *footer* div. The login div is always displayed when the application is loaded and the others blocked or hidden. The user can however click on the signUp button to get the registration form. In this case the other divs are hidden[32].

The div which initially holds the authentication widgets such as text boxes and buttons is destroyed upon successfully log in or sign up. The skeleton of the UI is created by calling JavaScript objects that divides the page into four main parts[33].

A JavaScript widget object *applCon*, which is the top-level container of the UI has five children containers which are themselves JavaScript widget objects. These children widget containers are defined in Dojo to occupy the top, left, right, center and bottom regions of the parent container. These children containers will in turn hold children widget objects. The children widget objects of the header are also defined to occupy the top (main menu bar), the left (logo), the center (notification bar) and the right (logout button). The navigator container holds a Dojo *accordion container* as its only immediate child. This accordion container in turn defines three *ContentPane* container widgets. In Dojo a ContentPane container widget is the lowest form of *placeholder* available, comparable to the div in XHTML. These three containers will hold the table of contents for the hyLOs eLearning units, Wikipedia document and the search results from the Google Web search. The workspace or the center container defines a single ContentPane widget which will hold the main content. The left container (misc) defines two Dojo *TabContainer* widgets. These are defined so that each occupies half of the container. The upper TabContainer holds two ContentPane containers, which hold the list of active users and new messages. The lower TabContainer has a single ContentPane widget as its child. This holds the list of groups available in the system.

7.3.3 Loading Content from the Web

When the application is initially loaded, the content overview Page for the content from hyLOs is loaded. After successful sign up, the application Ajax engine, sends two request to the back-end PHP proxy script. One has a request to load the hyLOs table of content[34]. while the other has the request to load the overview page which shows the different eLearning units [35].

The Dojo XHR GET object is used to make the Ajax request to the server. Although it is

[32]CD -> Source Code -> JavaScript -> client -> eui -> init.js

[33]"CD -> Source Code -> JavaScript -> client -> eui -> mainpage.js

[34]CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> loadHylosToc()

[35]CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> hylosOverviewPage()

similar to the XHTML GET request and functions in much the same way, it has been extended via attributes to perform functionality specific to Ajax programming. Such functionality include processing common types of response text (for example, JSON), ability to time out a request by setting the timeout within the request itself, ability to cancel a request, and robust and easy-to-use error recovery.

The request above calls a PHP script and hands it a query. This query contains the address of the script to load. The request also specifies that the response should be handled as text. When the response successfully returns, a callback function[36] is called and the response as well as input-output arguments (ioArgs) are passed to it. If the XHR fails, then the message given by the Error instance created as a result of the failure is sent to the debug console and the user given a meaningful error message.

Toaster Widget

The handleHylosLoadError() function called to display the error message to the user is showed in listing 7.2

```
1
2 /* Using the widget dojox.widget.Toaster to show error messages to ↩
     the user*/
3 function handleHylosLoadError(error){
4   dojo.publish("errors", [{
5     message : error ,
6      type : "error"
7   }]);
```

Listing 7.2: Publishing to a Topic in Dojo

Listing 7.2 shows a Dojo lightweight and non-intrusive alternative to alert(). It is called a dojox.widget.Toaster. This is going to be explained here in detail because it is going to be used extensively in this application to inform user of request ad error status and for group messaging during group work.

A Toaster widget is a box that pops up in the corner of the browser and then waits for a mouse click or simply fades out after a predefined time has expired. This widget makes use of *Topics*. Topics are based on Dojo's publish-subscribe event system. In this system, several functions register their interest in a "topic." Later, some process can publish a set of arguments about the topic. The action of "publishing" is accomplished by calling each subscriber with the given arguments. In the above example, the handleHylosLoadError() function publishes an error message it receives.This error

[36]CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> controlhyli()

message concerns the topic ***errors*** . This topic had been created and subscribed to by the toaster widget when the XHTML page was loaded as shown in listing 7.3. The publishing then calls this toaster widget and the error is diplayed.

```
1
2 <div dojoType="dojox.widget.Toaster" id="wiki"
3     positionDirection="tr-left" duration="10000"
4     messageTopic="errors"></div>
```

Listing 7.3: Creating and subscribing toaster widgets to topics

How long the toaster stays on the screen can be set using its duration attribute or the user can dismiss it manually by clicking on it.

Browser History and Dojo.back

Another important aspect worth discussing at this point is how the browser history is handled. This application being a single page application does not allow for the normal functionality of the browser back and forward buttons. So the application is designed to mimic the browsing functionality using the dojo.back function.

When using dojo.back, the idea is to manually change the DOM property window.location when content changes by setting the window.location.hash property. Editing this property has the effect of changing the URL in the address bar and adding an item to the browser history but does not reload the page since only the fragment identifier (i.e. the part of the path URL after the # character) is changed. With the browser history filled with distinct URLs, the back and forward buttons will cause navigation to these URLs by changing window.location. Dojo.back watches window.location and fires an event when it changes.

To get dojo.back to work properly, the Ajax Engine built a state object class [37], initialized the dojo.back() function, and passed the state objects to it at each content navigation. The module dojo.back is loaded with the XHTML page and the dojo.back.init() is called immediately after entering the body of the page to initialize dojo.back(). Dojo.back.setInitialState is called to set the overview page for hyLOs as the initial state object. This therefore represents the first page and first entry in the dojo.back history stack.

Each time a new document is loaded onto the workspace, the state object is called and the url passed to it is added onto the history stack of dojo.back. This is shown for the three content sources in listing 7.4; dojo.back.addToHistory(new hylosState(url));

[37] CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> hylosState

```
1  /* dojo.back.addToHistory takes a state object that represents the
       state of
2     that particular history item. This causes dojo.back to edit
3     the window.location.... as well as push the new state object on
          its
4     history stack */
5    dojo.back.addToHistory(new hylosState(url));
```

Listing 7.4: Calling the state object to add an item to the dojo.back history stack

The state object must include the property changeUrl, which holds the fragment identifier associated with a particular state as shown in listing 7.5.

```
1
2  /* if the url is from Wikipedia, the chnageUrl is manipulated so
       the URL fragment shown on the browser displays
3    the full URL of the loaded Wikipedia document. The urltrack is
       the URL passed onto the Back and Forward button
4    functions. So here the fragment an attempt is made to seperate
       the fragment URL from the URL loaded as a result of
5    forward/backward navigation by the user
6  */
7
8  var wikibaseurl = "http://en.wikipedia.org/wiki";
9
10 hylosState = function(url) {
11   url = String(url);
12   var bush;
13   bush = url.indexOf("eui/wikipedia");
14   if(bush != -1){
15      var parts = url.split("eui/wikipedia/");
16      this.changeUrl = wikibaseurl + "/" + parts[parts.length-1];
17      bush = "";
18      }else{
19      this.changeUrl = decodeURIComponent(url);
20    }
21     this.urltrack = url;
22 };
```

Listing 7.5: Defining the changeURL fragment to be shown a long with the base URL on the browser window

The state object includes the functions back and/or forward. Dojo.back fires these when back-button or forward-button navigation occurs.

To conclude the discussion on loading content, the back-end functions called during content loading will be discussed. The functions called to load the hyLOs content and the Wikipedia content are similar. The back-end functions needed to load the Wikipedia document will be discussed.

The PHP function[38] receives a query of the format "keyword#urlFragment" if the browser is Internet Explorer or "keyword/urlFragment#urlFragment" if the browser is Firefox. The keyword always identifies the Wikipedia document in question and section after the #, indicates what section of the wikipedia document to load. It then creates an instance of the class *wikipedia*[39] and call its' ***get_page*** function to retrieve the required page.

Whenever an object is loaded from hylos or Wikipedia, the dojo.query is used to return the Nodelist of all href tags found in the page as an array. The application then attaches an *onclick* event listener to this array as shown in listing 7.6.

```
1
2  dojo.query("a").forEach(function(node) {
3        dojo.connect(node, "click", function(e) {
4          dojo.stopEvent(e);
5          var url = node.href;
6          if(url.indexOf("hylos.cpt.haw-hamburg") < 0){
7              return
8            }else{
9              loadNewPage(url); // loads a new hyLOs document
10           }
11       });
12     });
```

Listing 7.6: Using dojo.query to get an array of all anchor tags within the loaded document

Whenever an element is clicked, an Ajax Engine function is called. It examines the URL of the clicked object using the *indexOf* JavaScript function and then decides if to make an Ajax call for the requested object or not. If it makes the call, it sets a callback function to load the response onto the workspace.

When the document is displayed, a div is created and placed onto the workspace. This div gets the URL of the document being viewed, *currentUrl*, set as its id. This is because the object URLs are unique. The id will be reference each time the contents of the workspace are to be manipulated in any way. This feature lays down the framework for content manipulation discussed in the next section. A reference to the current document is also stored in an object called *currentDoc*. What this is useful for, will be discussed in the next section.

[38] CD -> Source Code -> PHP -> server -> wikipedia -> getWikipediaPage.php

[39] CD -> Source Code -> PHP -> server -> wikipedia -> wikipedia.php

7.3.4 System tools and Content Manipulation.

When the application is successfully loaded a timer is created to listen to user selections. When a word is selected, it is picked up by the timer the next time is runs, i.e. if the word is still selected[40]. An object, ***controlMessage*** is used to set the selected word as a global variable because it is used by functions in other modules. The selection is processed only if it is not empty, i.e. a user selects and black section of the display.

The difficulty here is to find a balance between giving the user enough time to do his selection, and showing the additional tool bar. If the value of the timer is too small, it might catch only a portion of the selected word when it expires because the user will be in the process of selecting. If it is too large, the user might have to wait a long time before the additional toolbar is shown.

When the selection is detected, the Ajax Engine grabs the selected word or phrase and makes a quick call to the back-end to determine if the word has already been annotated[41]. This is because a user can annotate a word only once if working in the normal mode (alone), but he can edit the annotation. Also, there can exist only a single annotation for a group. Everything else is added as feedback to the annotation. When the server responds, the user is shown a toolbar with options to add the link if an annotation already exist[42] or a link and an annotation [43] if it doesn't. Whenever and annotation already exist, the Ajax Engine makes a list of annotated words for that document ans displays them to the user [44].

Whenever the user adds a link or an annotation, the add link and annotation options are removed from the additional menu bar. The next time the user selects a word, they will be created and displayed again when the timer picks up the selection. A cancel button is also provided. This button hides the additional menu bar when clicked.

Dojo UI widgets are used to build the link and annotation editors. The functions used to handle links and annotations are quite similar. This section will explain client-side based examples on how links are handled and server-side based examples on how annotations are handled. Feedback and annotations will be handled in a similar way. For complete code references, please see attached CD.

adding links

The link editor consist of three Dijit textbox widgets used to collect the link properties

[40] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> getSelectedText()

[41] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> isThisWordAlreadyAnnotated(selectedText)

[42] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> showJustLinkOptions()

[43] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> showToolbar()

[44] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> toggleannotation()

(url, description and tag) and a button widget. When the widget button is clicked, it calls a function which builds the query to send to the server side as [45].

An Ajax Engine function[46],[47], are then called to make the request to the server and set the callback function to handle the response. A control object **addlink** is defined as global and used to determine which function makes the request. If this object is set, it means the word is already a linked word and the appropriate function is called.

The application also checks to see if the document to which the link is being added is a Wikipedia document using a global variable **waikiki**. This is because the scripts called on the server side depend on the type and source of document being worked on. The widgets of the link editor are always destroyed after the link properties are collected in the way shown in listing 7.7. This is to avoid a JavaScript id exception if the link editor is opened again.

```
1
2 var diaglink = dijit.byId("link_diag");
3     diaglink.destroy();
```

Listing 7.7: Destroying the link editor

To get a reference to the widget, dijit.byId(id) is used. To get a reference to the DOM object for this widget, dojo.byId(id) is used. The function destroy() is called to completely remove the widget from the DOM. All the other children of this widget i.e. the textbox and buttons widgets must be destroyed in a similar way.

When a link is successfully added, the phrase that was selected by the user is highlighted[48] by setting its background color to yellow. This highlighting begins by replacing the document in the workspace with the document stored in the **currentDoc** object. Remember that this variable always holds a clean (not manipulated in anyway) version of the document in the workspace. This is done to avoid conflicts of any sort. Such a conflict could occur if the linked word also has an annotation attached to it which is also currently highlighted. This clean document is added onto the workspace by the function toggleBack() shown in listing 7.8

Highlighting the link adds a class ***linkEntry*** to the linked word. Dojo.query is again used to get a reference to this linked word. In this case, it does not make much sense to use dojo.query since it returns an array. But this same function will be used to display

[45]CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> getLinkProps()

[46]CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> postLinkProps(query) - adding link for the first time

[47]CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> postMoreLinks(query) - adding more links to a linked word

[48]CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> linkHandler(response, ioArgs)

all links added to a document. Dojo.query will then return an array of all linked words or phrases within the document. A ***click*** and an ***oncontentmenu*** event listeners are attached in a similar manner as in listing 7.6. When a word in this array is clicked or right clicked, a query is constructed and a function[49] is called to make an Ajax request for the links attached to that array entry or linked word. This function sets different callback functions depending on the type of click made by the user.

```
1 function toggleBack(){
2   var showndocdiv = dojo.byId(currentUrl); // get a reference to ↩
        the div on the workspace which holds the object
3   showndocdiv.innerHTML = currentDoc; //replace the innerHTML of ↩
        the above referenced div
4 }
5 </script>
```

Listing 7.8: Loading a clean version of the document being displayed in the workspace

When the user clicks on the word, the application builds a link display window within the document that shows the added links. This is done using the Dijit toolTipDialog window[50]. If the original link was added by the user or a group to which he belongs, the display window provides an option for the user to add more links. This is done by checking if the ***shared*** property of a JSON object received from the server is set to ***no***. If it is set to ***yes***, it means that the added link is shared and does not belong to the user or a group to which he belongs. Therefore, the user cannot add more links to this linked word. In the case where the user just added the link, it will always be set to no. In this way, the JSON objects are constructed to provide the Ajax Engine with the information required to make such real-time decisions as which options to present to the user.

If the user right clicks on the linked word, the retrieved links are opened on the upper section of the miscellaneous UI container. How this section is manipulated to display different types of information will be explained later in this chapter. The reason for doing this, is to have the links available in case they are needed even after a different object is loaded onto the workspace.

Adding Annotations

The annotation editor is similar in functionality to the link editor. It differs in appearance in that it uses the Dojo rich text editor widget[51] to collect the annotation information.

[49] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> getLinkEntries(query)

[50] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> getEntryLinksHandler(response, ioArgs)

[51] CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> showAnnotationEditor()

This widget offers HTML-backed editing with a handy toolbar and keyboard shortcuts. This toolbar is placed in a div where the user will type in his annotation. If the user formats the text after typing it, this widget formats the body of the text before it is sent to the server side.

This annotation editor is defined once and used for annotations, annotation feedback and notes. Javascript variables are used in much the same way as the *waikiki* variable for a Wikipedia document to determine which task the user wishes to accomplish. This application uses a partial color scheme to let the user know from the background color of the editor what task he is performing. This color scheme does not only apply to the annotation editor. For instance every visible feature of the UI that has to do with handling addition and display of links is set to yellow, for annotations the color aqua is used and so on and so forth. However, this color scheme is as of now, only partially implemented.

After the annotation properties have been filled in, a function is called which gets the values of these properties from the widget containers. This function then constructs queries and calls the Ajax Engine script to make the Ajax request to the server and set the callback functions to handle the response. On the server side, the appropriate script writes the annotation to the data store[52]. The script then checks if the document url belongs to a Wikipedia document. If it does, the system checks if a database entry already exist for this section of the Wikipedia document. The document is written to the database if an entry is not found. Finally, the script checks if the user is currently active in a group. If he is, the table name is then constructed from the group name and an entry made in the groups notify table. This entry simply says that the user in question added an annotation.

When the response returns, the callback function is called and it highlights the annotated word by setting its background color to aqua. It attaches the class ***annotationEntry*** to the annotated word. Dojo.query is also used here to build an array of all annotated words within a document. In this case, the array has just a single entry. An ***onclick*** event listener is attached to the array. When the user clicks on an annotated word, the annotation display opens up. This is built in a similar way as the link annotation. The only difference is that it always offers the user two options. The option to add a feedback and the option to edit the annotation

Annotations Options

The application lets the user view all annotations or links added to a document. These annotations and links are visualized as described above. For the links a single option is

[52] CD -> Source Code -> PHP -> server -> grpFxn -> handleAnnotations.php

added depending on whether the link is owned by the user or his group or shared. For annotations, the options added when a user clicks an annotated word are determined by the following considerations;

- has the annotation been edited?
- has the annotation been edited more than once?
- does the annotation has any feedback yet?
- is the annotation owned by the user or a group to which he belongs

The PHP side script helps the Ajax Engine answer these questions when building the annotation display window. For each JSON object sent to the client, three attributes, namely: ***originalnote***, ***feedbacknote*** and ***editednote*** are created. These attributes are assigned the id of the annotation if the annotation has an original annotation, a feedback, or an edited annotation. Having an original annotation means that the annotation has been edited. Having a feedback means the annotation has a feedback attached to it. Having editednote means that the annotation has been edited more than once. Of interest here is also the attribute ***shared*** which determines if the user owns the annotation object or not. Note that the server-side code for adding a link, feedback and note follow a similar pattern.

When the callback function receives the response, all it has to do is to check the values of the shared, originalnote, feedbacknote, and editednote for each JSON object and compare them to the id of the object in question. Wherever there is a match, a corresponding option is added to the annotation display window for that object. For instance, if the value of the originalnote object is equal to the id of a particular annotation object, an option to retrieve and display the original annotation is added to the annotation display. This is because the application always displays the current annotation.

Dynamically Manipulating the Additional Toolbar And The Misc UI Container

The user can click on the *view added content* option of the viewing tools. The follow processes are set into play

- hide the containers occupying the upper section of the misc UI container.
- create two TabContainer widgets and a toolbar and place them in the misc UI container.
- create an option to hide the newly created section and place it on the newly created toolbar of the misc Container
- show the additional toolbar if it is already created or create it it and place it below the main bar

- create and option to destroy (remove from DOM) the newly created containers on the misc container.
- call the Ajax Engine to create and make a request for all the annotated and linked words with the current document.

When the response is received, the callback function then loads lthe inked words into one of the newly created TabContainer widgets on the misc container[53]. The other TabContainer widget container is filled with annotated words[54].

This same pattern is followed when displaying feedback annotations for a particular annotation, link entries for a particular linked word and notes attached to the current document. The only difference being that for feedback and notes, the whole misc container is hidden and replaced by the new container to show the feedback or notes.

One thing common to all these interactions with the UI, is the fact that, the user can click the *hide* option to hide the newly created region and show the region that was previously there. The *show* option is then dynamically created and placed on the additional toolbar and the *destroy* option hidden. When the show option is clicked again, it is hidden and the destroy and hide options are shown along with the newly created container.

When dynamically manipulating the misc UI container and the additional toolbar[55], a reference to the upper section of the misc UI container is stored in an object called *tadArea*. This is then hidden. The new container is created and put in its place. The *destroy* option is created and place on the additional toolbar. The *hide* option is created. On click, it hides the newly created container and the destroy option and shows the misc container. It then checks if the *show* option exist. If it does it shows it, otherwise it creates it and places it on the additional toolbar. The separator widgets inserted into the additional toolbar are also hidden. When the *show* option is clicked, the whole process is reversed, i.e. the misc container and the show options are hidden, and the *destroy* option and newly created container are shown again.

Displaying Notes

When notes are immediately created, they are displayed differently as mentioned above. Instead of using the misc UI section, a div is created and placed on top of

[53] CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> showAddedLinks(response, ioArgs)

[54] CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> showAddedAnnotations(response, ioArgs)

[55] CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> toggleMoreLinks() builds the morelinks container on the misc UI container and dynamically hides and shows the misc container and the new container

the top-level container widget on the top right corner of browser window. This is possible by setting the *z index* of the new div to be greater than that of the underlying top-lvel container. The z-index property specifies the stack order of an element such that an element with greater stack order is always in front of an element with a lower stack order. This index is set in css as shown in listing 7.9.

```
1  /* css for setting the z-index of the div which holds the note*/
2  .notes {
3    width:260px;
4    height:150px;
5    border: 1px solid red;
6    padding: 10px;
7    background-color: orange;
8    z-index: 2;
9    position: absolute;
10   text-align: left;
11   overflow: auto;
12   top: 0px;
13   right: 0px;
```

Listing 7.9: CSS for setting the z-index of the div which holds the note

Options for deleting and editing the note are also added to the note div. When these buttons are clicked, they call an Ajax engine function and give it the id of the note to be manipulated. The Ajax Engine makes the request to the server and sets the callback. When the response is received, the callback uses a toaster widget to inform the user of the status of his request, i.e. successful or not successful.

Edit-in-place

The edit-in-place feature is used when editing annotations and notes. During edit.in-place, the innerHTML of the container displaying the content is grabbed and placed in an object. The container is then emptied. A textarea and two buttons for saving and canceling the editing process are created and placed in the container[56]. The previously contents of the container are then placed in the text area for the user to edit.

If the cancel button is clicked, the whole process is reversed. First the container is emptied again by setting its innerHTML to an empty string and the original contents are placed in the container. If the save button is clicked, an Ajax call is made to save the edited version of the content. The innerHTML is emptied to discard the textarea and buttons and replaced with the edited version of the content. For the annotation, the

[56]CD -> Source Code -> JavaScript -> client -> grp -> grpactions.js -> postNoteHandler(response, ioArgs)

original option (to get the original annotation) is created and placed in the annotation display. If the annotation had been edited before, the original button will already be available. So the application should not create it twice. This is not necessary for the notes since the note functionality does not keep track of old notes.

7.3.5 Group Interaction

In order to be able to work in a group, users have to create or join groups. Groups are created by using the group tools of the main menu. When creating the group, the user gives a name for the group and sets his content sharing status to public or private. The group content sharing options under group tools can be used to change this settings at anytime. The group name and settings are then communicated to the back-end by the Ajax Engine and the new group is stored in the system.

When the user requires to join a group, the application checks if the user is already a member of the group and then lets him join. After a user joins a group he can activate that group whenever he wants to start a discussion thread within the group.

Right clicking on the list of available groups on the lower section of the misc UI, pops up a menu with the following group options; join, activate, enter, leave and delete. This menu is implemented using Dijit widgets menu and menu Item. The menu widget stacks its menu items vertically. An ***onSelect*** object is used to get the name of the group when the user right clicks on the name of the group. The bindDomNode() function is called to add the menu to each item of the list of group names[57].

The program works in two modes. The normal mode is when the user works alone and is always signified by setting the ***NORMAL_MODE*** object to true. When the user is in the group ***GROUP_MODE*** is set and a variable ***activeGrpname*** set to hold the name of the activated group. All the functions implemented so far are also available during group work. Three things unique to group work are;

- users in an active group work on the same document
- users are always aware of what other users are doing.
- users can turn off synchronization to which deactivates the two features mentioned above, but still gives the user access to content available to his group.

All users in a group can work on the same document because whenever a group is activated, the Ajax Engine calls the server-side to make an entry in the *currentUrl* column of the table which holds the active groups. Whenever a new member enters the group, the Ajax Engine makes a request for the URL of the document the group is currently working on.

[57] CD -> Source Code -> JavaScript -> client -> eui -> init.js -> showGrpList(response, ioArgs)

The callback then examines the returned URL using the JavaScript *indexOf()* function and then decides whether to load a hyLOs document or a document from Wikipedia or a random XHTML page (which resulted from using the google search function). The callback sets a timeout. When the time out expires the current URL of the group is requested again, the old timeout cleared and a new one set. If the requested URL is equal to the URL retrieved after the previous time out no new page is loaded. The user who loads the new page, also sets a variable *iSentUrlupdate*, so that the new document will not be loaded again when the timeout expires[58]. .

Another important function called whenever a group is created is the *checkForNotifications()* functions. Remember that for group activity, whenever a user is performing an action, the Ajax Engine calls the server side and informs it of the action the group member is currently undertaking. This action is then written to the MySQL table which is dynamically created whenever the group is activated. This table is created by using the word notify and concatenating it with the name of the group. For example, if the group *Internet Technology* is activated, the table name will be *notifyinternettechnology*. In this way all functions that will write or read from this group can dynamically build the group name when they need to. The checkForNotifications() function, periodically pools this table to read out the actions being performed by other members. These actions are pre-coded as PHP constants in such a way that, they can be decoded when read to give the name of the user and the action he is performing and if possible, the action to carry out. Listing constant shows a small section of how these message constants are coded

```
1 ...
2 define("ANNOTATION2", ":and is adding an annotation to this phrase.");
3 define("ANNOTATION3", ":has NOTE_completed the annotation for:");
4 define("ACTIVATE", ":activated the group. This user is therefore
    responsible for this group.");
5 define("FB", ":is working on an annotation feedback for the phrase:");
6 ...
```

Listing 7.10: Messages pre-coded as PHP constanst

An example of how these pre-defined messages is written to the server is given in listing 7.11

```
1 function writeNotification($note, $seltext, $user, $grpname,
    $tablename){
2     if($note == "link"){
3       $sQuery = "INSERT INTO ".$tablename."
4       VALUES ('0','$seltext', '$user', '$grpname', '$user".LINK1."
          $seltext, ".LINK2."')";
```

[58] CD -> Source Code -> JavaScript -> client -> grp -> grp.js -> getUrlChangeHandle(response,ioArgs)

```
5     }else if($note == "annotation"){
6       $sQuery = "INSERT INTO ".$tablename."
7       VALUES ('0', '$seltext', '$user', '$grpname', '$user". ↩
            ANNOTATION1." $seltext, ".ANNOTATION2."')";
8     }else if($note == "activate"){
9       $sQuery = "INSERT INTO ".$tablename."
10      VALUES ('0', '$seltext', '$user', '$grpname', '$user".ACTIVATE." ↩
            ')";
11    ...
```

Listing 7.11: Writing to the notify group table using the precoded messages

Before returning a response to the checkForNotifications() function, the PHP server script interprets the read messages and builds the corresponding JSON objects that are returned as the response. This is shown in list 7.12.

```
1  ...
2  $actionid = $newnoti[0];
3  $actionheld = $newnoti[1];
4  $action = $newnoti[2];
5  ...
6  }else if(stristr($action,"NOTE_completed")){
7    $pieces = explode(":", $action);
8    $owner = $pieces[0];
9    $seltext = $pieces[sizeof($pieces)- 1];
10   if($owner!= $user){
11   $getnote = $database->getLastNote($seltext,$owner,$grpname);
12   if($getnote){
13     $json4 ='{"notes" : {';
14     $json4 .= '"note":[';
15     $json4 .= '{';
16     $json4 .= '"status" : "lastnote",';
17     ...
```

Listing 7.12: Interpreting the message read from the notify group table

The Ajax Engine constantly checks for new group notifications[59]. When the response is received, the *status* attribute of the received object is examined and the appropriate callback function called[60]. For example, if the status attribute has the value ***lastnote***, function will get the selected word out of the response it receives, search for it in the document on the workspace and highlight it to denote that an annotation was added to that word. All other messages which do not cause the DOM to be changed in any way, are displayed using the

[59]CD -> Source Code -> JavaScript -> client -> grp -> grp.js -> checkForNotifications()

[60]CD -> Source Code -> JavaScript -> client -> grp -> grp.js -> checkForNotificationsHandlerNote(response)

toaster widget (section 7.3.3). The first time the messages are read, the first message is always displayed. This message says which user activated the group. The id of the message to request next is set to the id of the current message in the notify group table.

When working in a group, some users might find the toaster messages which keep popping up irritating. An attempt to reduce this nuisance has been made by setting a duration time for these toaster widgets so that they disappear after a reasonable time. The fact that an object is loaded onto the workspace for all active group members when an active group member loads a different document, synchronizes group activity, but can cause problems for users who are still working on the current document. Finally, the UI of all members in the group can be manipulated when an active user adds a feedback. This can also be irritating to some users.

This application therefore offers a simple solution to this problem. The user can set his group synchronization status to on or off. Off will stop him from receiving toaster widget messages and URL change updates. This is done by simply clearing the timers which constantly pull the server for URL updates and group notification messages [61]. When the synchronization is turned on, checkForNotifications() and getUrlChange(activeGrpname) functions are called to start receiving group messages and URL updates.

7.3.6 Messaging tools

The messaging tool is included as a simple way of letting users communicate with each other. Two tables are created to hold user messages depending on whether a user is online or offline. When the user is online, he is immediately notified when he receives a new message. A timeout is set which pools the online table after the time out expires. It then sets a new timeout and this process keep repeating itself.

Whenever a user logs in, the offline messages table is read and the user notified if he has any messages. The messages are then retrieved and displayed to the user. The user is notified using the *yellow-fade* technique where a background color of a section of the UI is suddenly set as yellow to attract the attention of the user. As the yellow gradually fades, a message then appears to inform the user of the new messages[62].

[61] CD -> Source Code -> JavaScript -> client -> grp -> grp.js -> toggleGrpSync()

[62] CD -> Source Code -> JavaScript -> client -> contentHandlin -> hylosdata.js -> notificationMessages(Message)

Chapter 8

Test and Evaluation

Most of the functionality described in the design and specifications chapters 5 and 6 respectively, have been fully implemented. This chapter demonstrates some test scenarios of the whole system. Due to time constraints only user test have been carried out. No system profiling test were done. This section uses screen shots to show and explain what the browser displays to the user.

To carry out the test, the Apache HTTP Server was installed and configured in the mobi2 Server in room 580. A similar application as the one in this Thesis is being developed for the iPhone and also uses this server for testing. In order to configure separate domains to run under mobi2, DNS entries for this domains had to be made in the DNS server which was not possible due to time constraints. The server URL *http://mobi2.cpt.haw-hamburg.de* was used by the iPhone project. A virtual host was created on the apache server and a subdomain entry made with a private hosting service offered by InterNetWORX [63] to point to this virtual host on the mobi2 server. The sub domain address created to test this application is ***http://igelearning.mailquota.de/www_root/client/eui/main.html.***. The MySQL database used for storing user information and content was also installed and configured to run on the mobi2 server. The test were carried out on the windows PCs and the Mac PC in room 580. The application was tested for the following browsers;

- Internet Explorer
- Mozilla Firefox
- Safari

Figure 8.1: Registration form showing the field of interest and the content setting status

.

8.1 Setting User Interest and Content Sharing Status

During registration, users can choose their field of interest and set the status of the content they create to public(shared) or private(hidden). Figure 8.1 shows how the registration form looks like. Clicking on the login button on the registration form hides the registration form and shows the login form figure 8.2. Clicking on the Signup button of the login form will hide the login form and show the registration form.

The application offers user tools with an option user settings. User settings can be used to change the field of interest of the user and the content sharing status at anytime figure 8.3 and figure 8.4 respectively

Notice that when the mouse is moved over the users name, his field of interest is shown. Users can contact each other based on this interest.

[63] InterNetWORX Web Hosting http://www.internetworx.org/

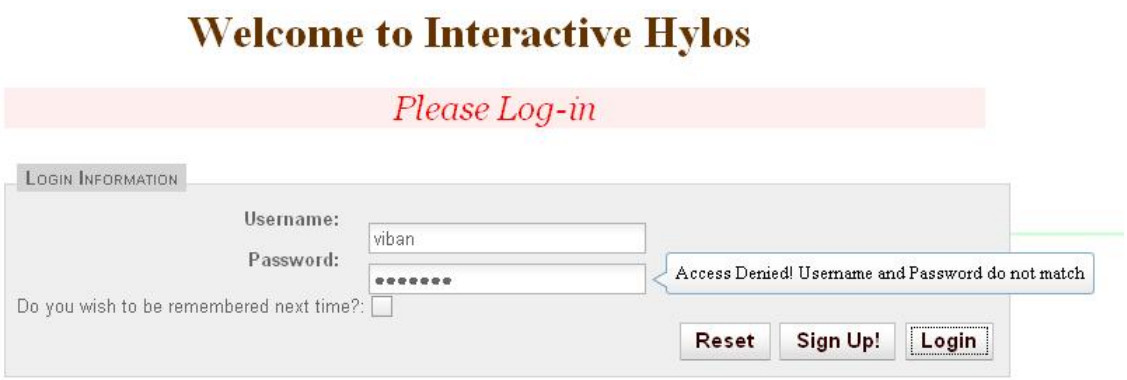


Figure 8.2: login section showing user authentication in progress

.

8.2 Adding, Displaying and Editing Content

8.2.1 Content Editors

A single content editor is used for adding annotations, feedback and notes as shown in figure 8.5. Only the title of the editor is changed. The partial colour scheme used in this project is also evident. Editors and displays used for links get the background colour yellow figure 8.5(d). For notes, orange is used (figure 8.5(c)) and olive green for Feedback figure 8.5(b). For annotations aqua is used. This is not evident from the background of the annotation editor 8.5(a). As said, the colour scheme is not fully developed yet.

Editing-in-place is done when editing annotations and notes. Figure 8.6 shows the annotation editing test carried out. On Internet Explorer, the textarea for the editing is inserted, but the text to be edited is missing. The save and cancel buttons are not also inserted in the div figure 8.6(a). Editing in place works well for Safari and Firefox, figures 8.6(b) and 8.6(c) respectively. However the textarea and button widgets are better placed within the div in the Safari browser. The button widgets are hidden in Firefox and the user has to use the vertical scrollbar to see them. Figure 8.6(d) shows how a note can be edited on Firefox immediately after creating it. Also notice the aqua color for the annotation editor background and the orange for the note editor background.

8.2.2 Annotation Display

Figure 8.7 shows how annotations are displayed within the document. Initially when a user adds an annotation, only the two options for editing and feedback are added to the annotation display figure 8.7(a). If the feedback already has an annotation attached to it, the displays is as in figure 8.7(b) with an extra option to get the feedback. If an annotation has been edited, it means an original annotation exist for the annotated word. If it has been edited more than

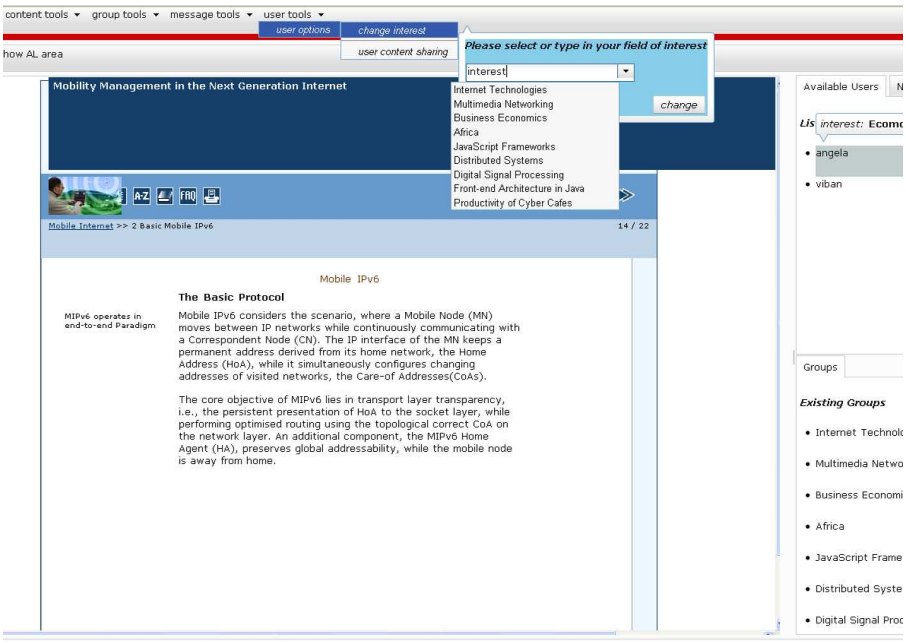


Figure 8.3: Changing the field of interest

.

once, then edited versions of the annotation exist. Figure 8.7(c) shows options added that can be used to retrieve original notes and edited notes. Clicking on the original notes button, gives the display showed in figure 8.7(d) which now adds an extra option to retrieve and display the current annotation. If the edited notes button is clicked the notes added to this annotation are retrieved and display on the right side of the page using the toaster widgets figure 8.7(f). These edited versions stay where they are until he user clicks on them, in which case they will gradually fade away. Figure 8.7(e) shows a special case of the display. In this case there are no options because this annotation has been shared by another user. This annotation is read-only for the user currently viewing it. If this annotation had feedback added to it or if it had been edited more than once, the current user will get the options to read the feedback, original annotation and edited versions of this annotation respectively. What the current user will not get are options to edit the annotation or add a feedback.

8.2.3 Link Display

The link display simply retrieves all links added to a particular word and list them one after the other. This happens when a user clicks on a link. Figure 8.8(b) shows how the link display window looks like. Notice the yellow background. It also has an option to add more links to the linked word. This option is available to the user if only he owns the link or belongs to a group that does. Compare this display to that on figure 8.8(a). Here the add more links

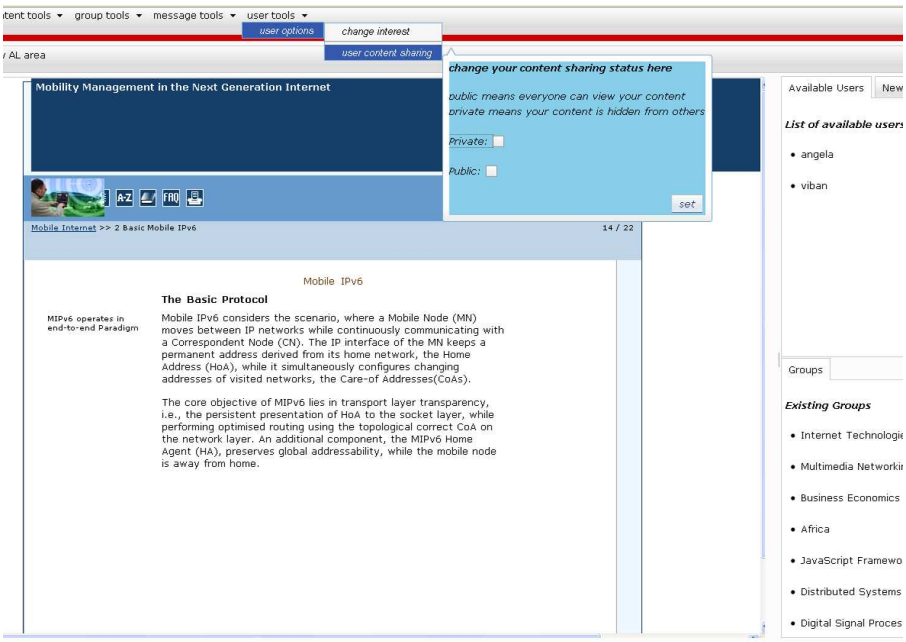


Figure 8.4: Setting the content sharing status for a particular user

.

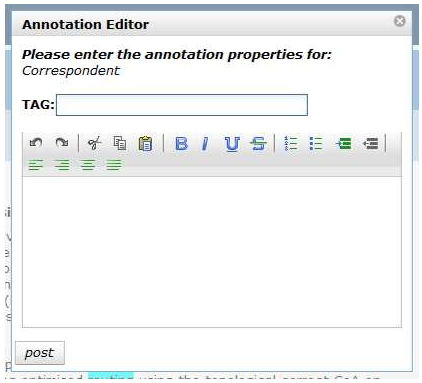
option is missing. Also notice that for both figures a list of all linked words in that document can be displayed on the UI window to the right. If the user moves his mouse over this word, he can already see who added the link and the description of the link. Figure 8.9(b) is added here for comparison. It shows a list of all annotated words in the current document.

A user can have access to links from a specific document, say *document A* while working on a different document, say *document B*. This can be done by right clicking on the linked word in document A whose links wish to be remembered. This opens up all the links the the misc UI section (right). The user can the load document B onto the workspace. This is depicted in figure 8.9(a). All the test for links documented in this section were carried out on the Safari browser.

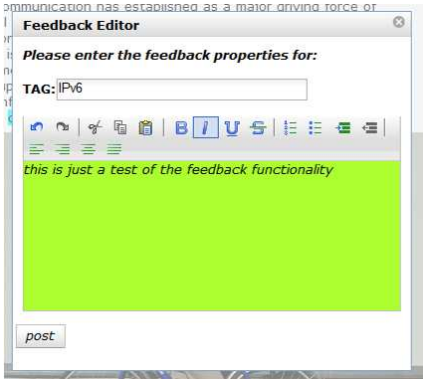
8.2.4 Displaying notes and feedback

User have two possibilities to view feedback for a particular word. They can use the *view feedback* option in the viewing tools or show the annotation display window and click *show feedback* option. In any case the misc container is changed to display the feedbacks for that annotated keyword.

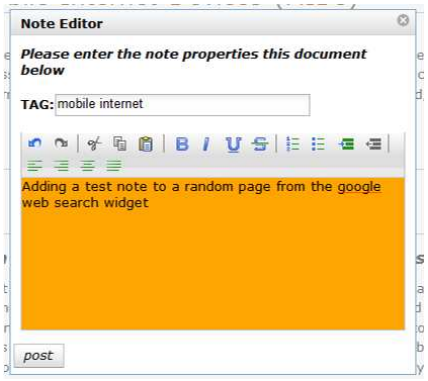
The notes are displayed in much the same way. To view notes, the user clicks on the *view note* option on the viewing tools. The notes are then shown on the misc container. If



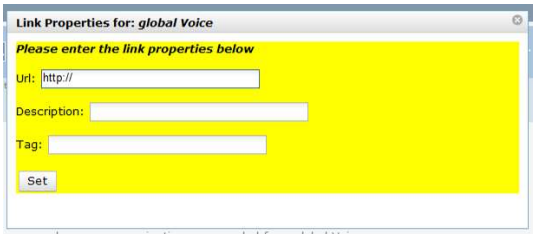
(a) Annotation Editor



(b) Feedback Editor

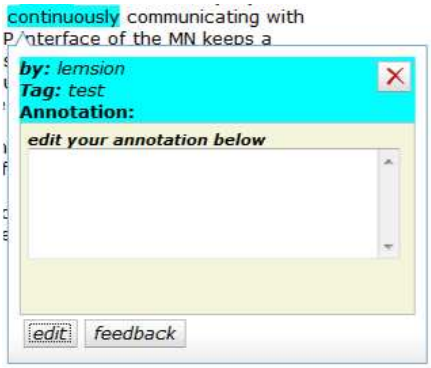


(c) Note Editor

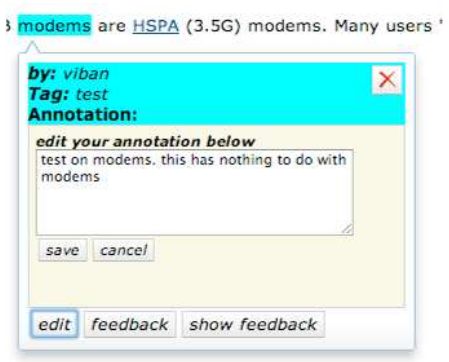


(d) Link Editor

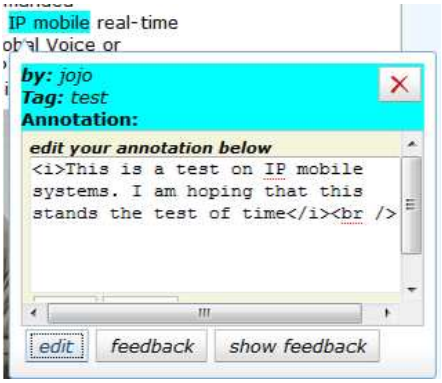
Figure 8.5: Application content editors



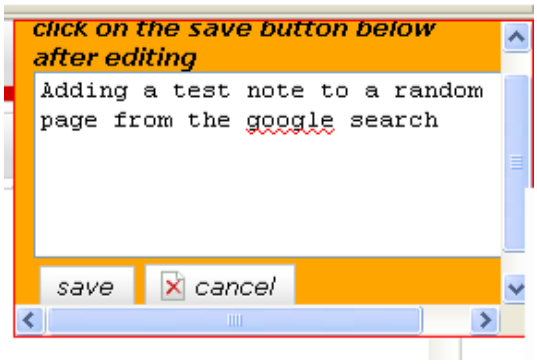
(a) editing annotation on IE



(b) editing annotation on Safari

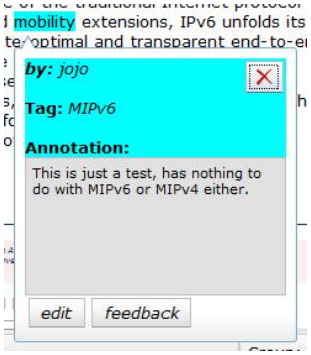


(c) editing annotation on Firefox

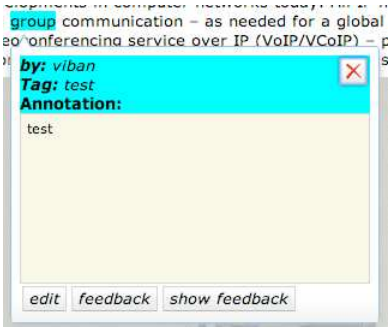


(d) Note Editor

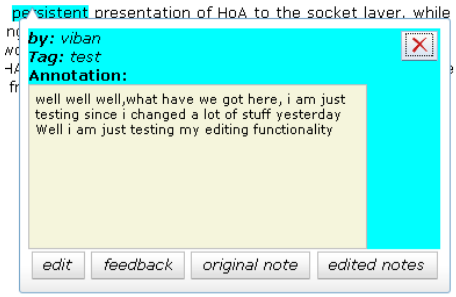
Figure 8.6: Editing-in-place



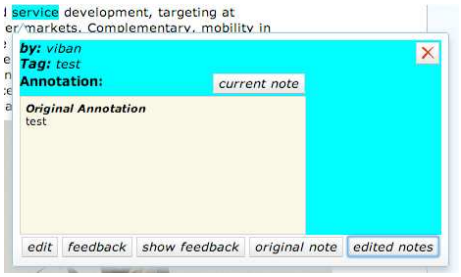
(a) initial annotation display



(b) annotation display when feedback has been added

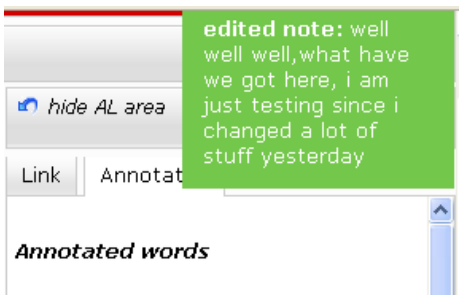


(c) annotation display when annotation has been edited



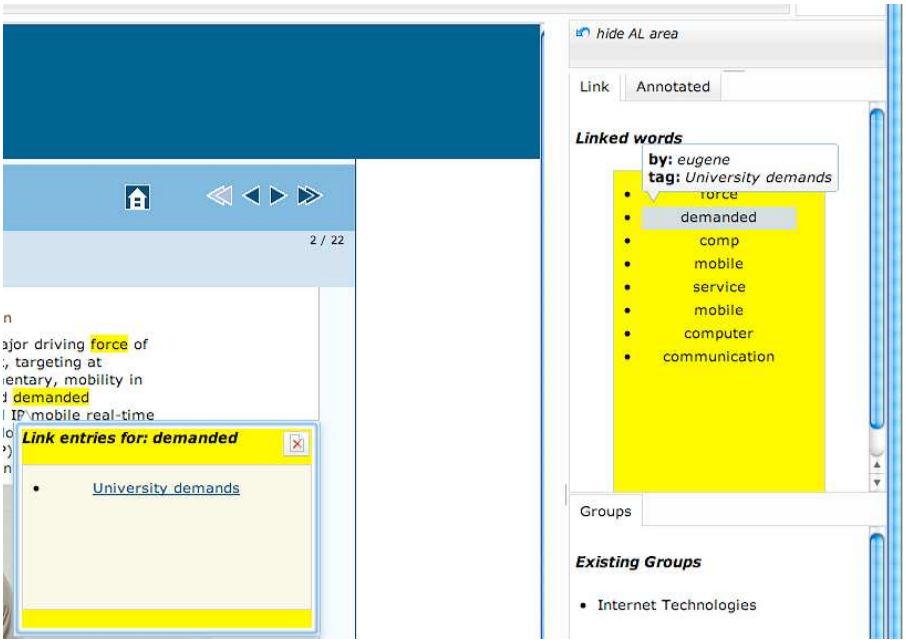
(d) annotation display showing original annotation

(e) annotation display for a shared annotation

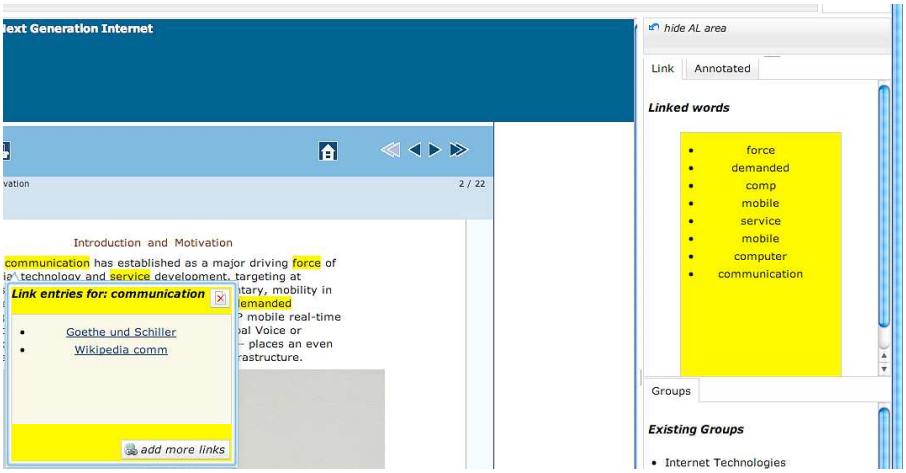


(f) edited annotation display

Figure 8.7: Displaying annotations

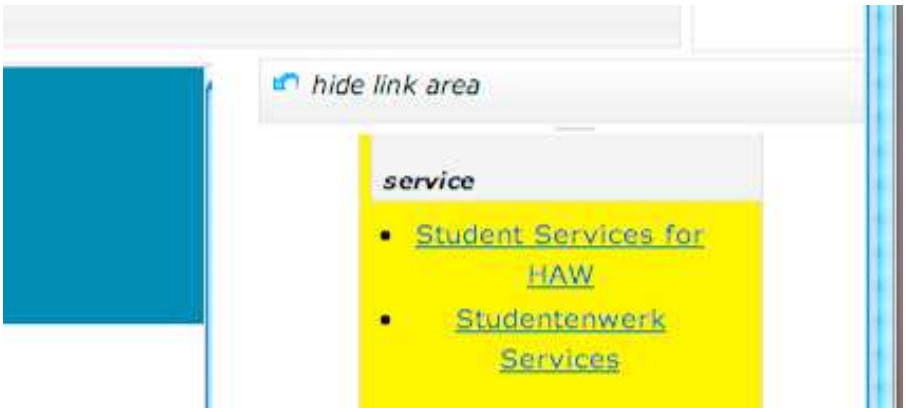


(a) link display for shared link

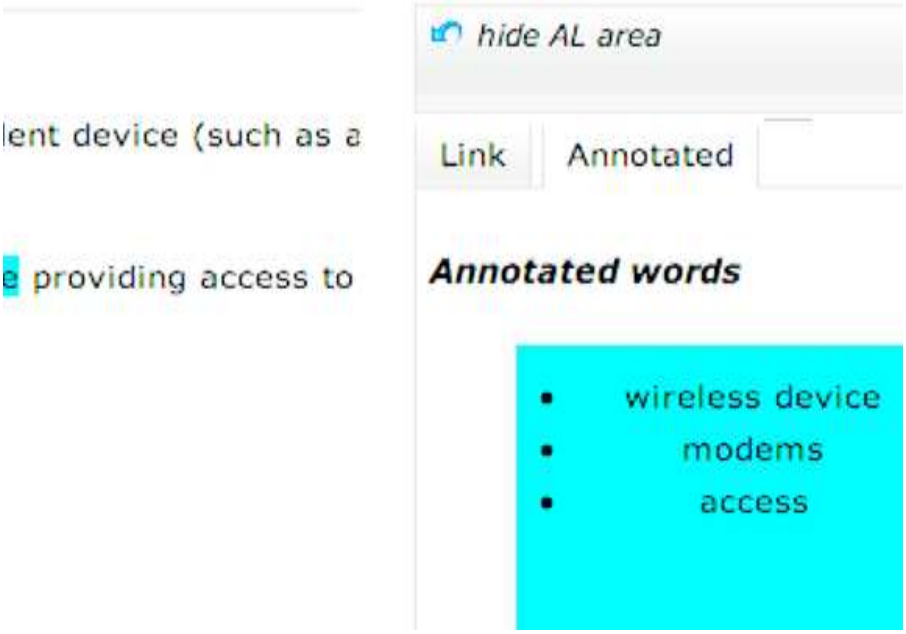


(b) link display for owned link

Figure 8.8: Displaying Links and showing linked words

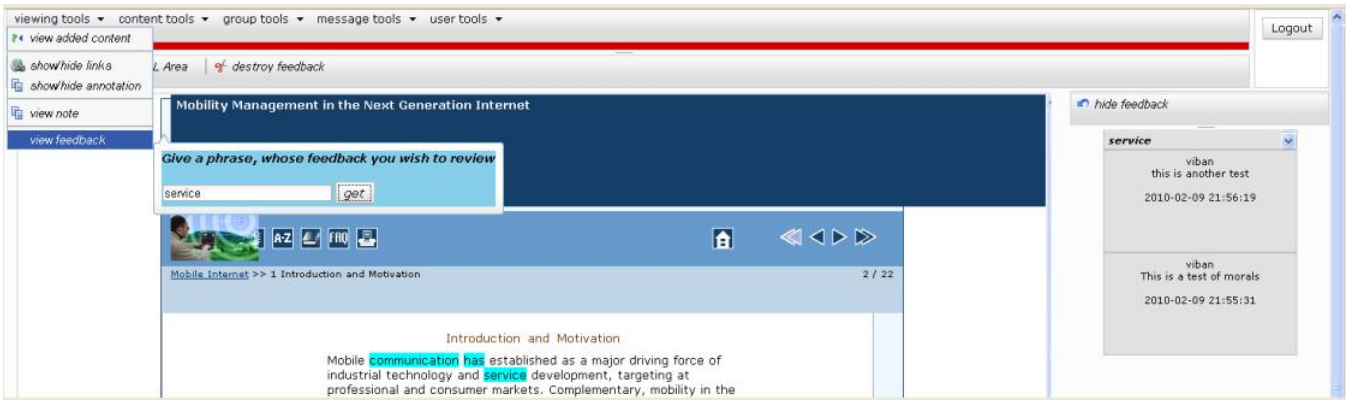


(a) remembering links on the misc UI container

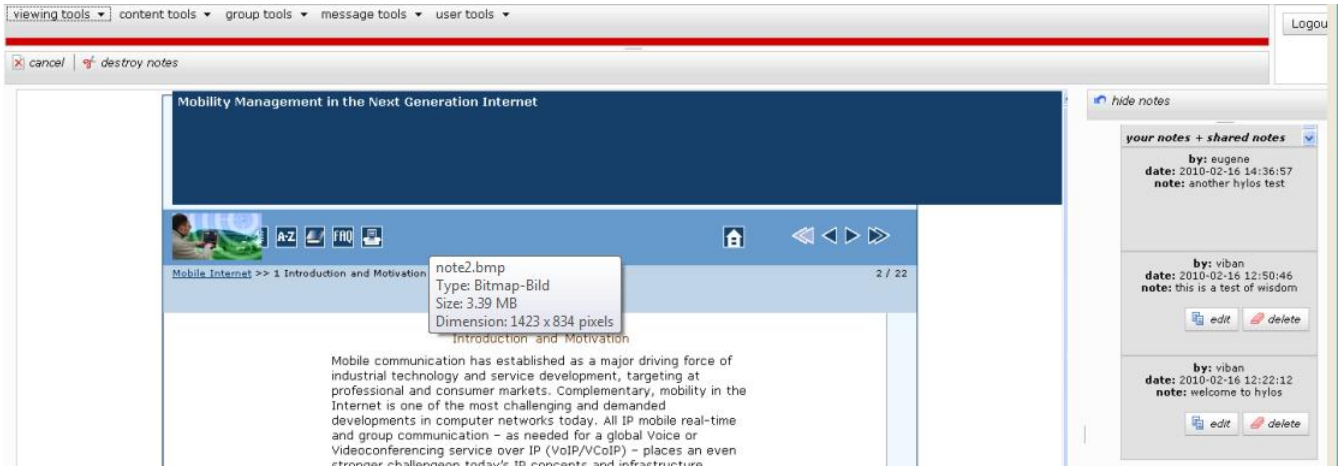


(b) showing annotated words on the misc UI container

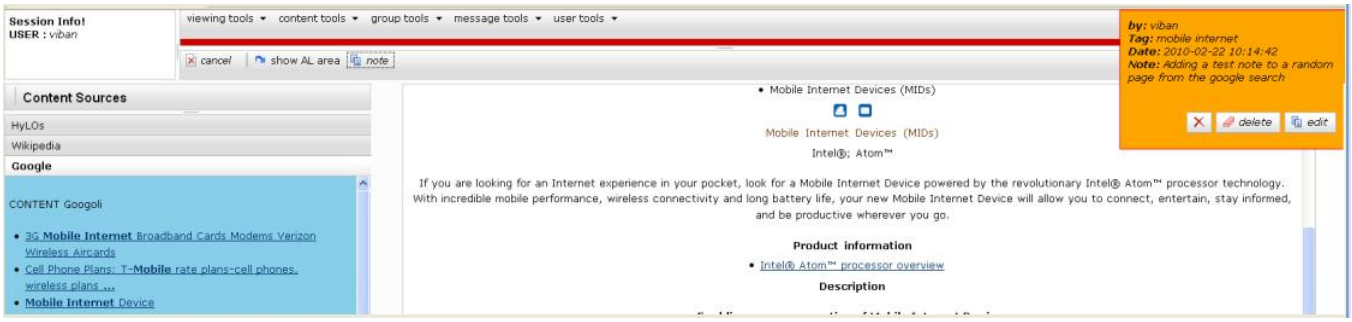
Figure 8.9: Showing Links and annotated words on the misc container



(a) retrieving and displaying annotation feedback



(b) retrieving and displaying notes



(c) displaying a note immediately after it is added

Figure 8.10: Displaying annotation feedback and notes

a note is owned by the user or a group to which he belongs, options to edit and delete the notes are added to the note when displaying it.

8.3 Group Work

This application has been designed to view group work as follows;

- the UI of a user can be manipulated indirectly by the actions of other group members
- the user knows what other group members are doing at all time.
- the user can decline to use the two features mentioned above but still be able to consume and create content for his group.

Any user can create a group by using the group creation editor shown in figure 8.11

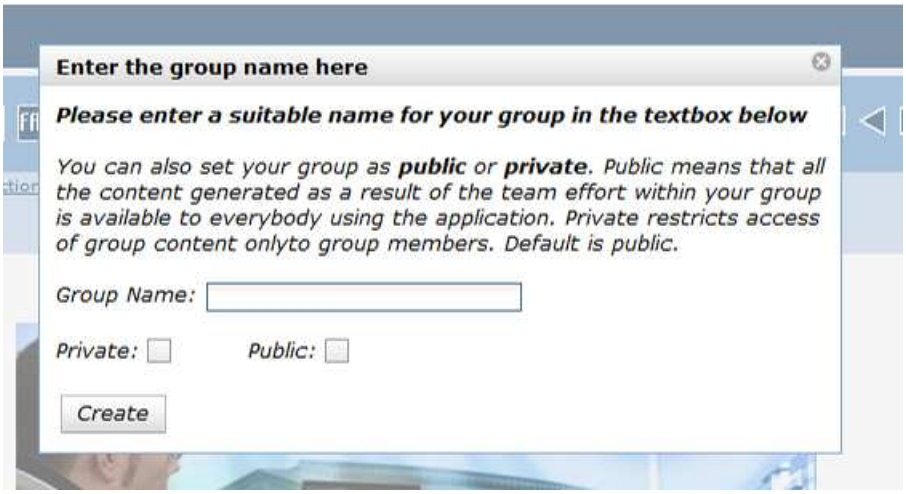


Figure 8.11: Creating a group

.

Once a group is creating, members can join, activate, enter, leave and delete groups by using the group menu shown in frgure 8.12.

For group messages the toaster widget (section 7.3.3) is used. When a user group is activated, figure 8.13(a) and a member enters the group, he gets the name of the user who activated the group, figure 8.13(b). The object the group is working on is loaded onto his

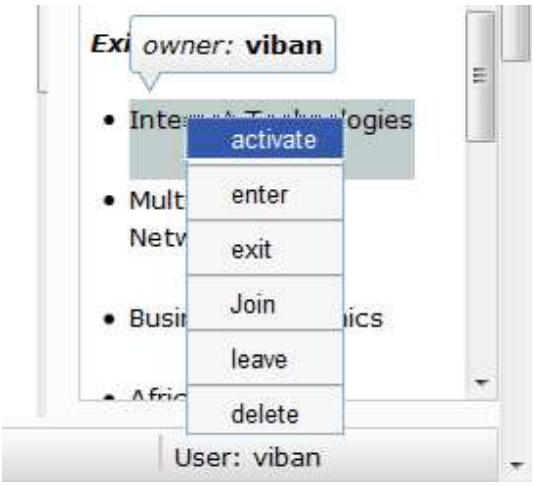


Figure 8.12: Group menu

.

workspace. The other users in the group are informed that a member has entered the active group figure 8.13(c). When a member leaves the group, the other members are notified as well, 8.13(d).

Figure 8.14 shows the processes which occur when a user is adding an another within the group context. When he selects the words and the annotation editor, the message of figure 8.14(a) is shown to him. All other active group members receive the message shown in figure 8.14(b). When the annotation is successfully written into the datastore, it is highlighted on the workspace of all active uses as shown in figure 8.14(c). If any of these users decide to add an annotation feedback for this annotation, all other users are informed with the message shown in figure 8.14(d). When the feedback is added, the misc UI container of all the users in the group is change to show the added feedback as described earlier in figure 8.10(a).

Whenever a new user loads a different object onto the workspace, other users are informed shortly before the new object is automatically loaded onto their own workspace as shown in figure 8.15

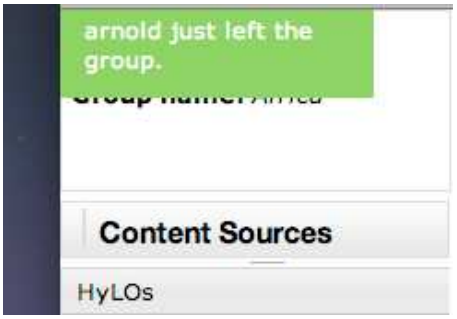
Users who wish to create content for or use content from the group without actively interacting with other users can set their synchronization status to off using the user options tool shown in figure 8.16.

The users can send messages to each other using the messaging tools as shown in figure 8.17. If a user is online, he is immediately notified using the yellow fad technique shown in figure 8.18. As the yellow gradually fades a message appears to inform the user of the new message. The user then clicks on the new messages tab on the upper section of the misc container to read his messages, fig 8.19.

(a) a group member activates the group

(b) user enters an active group

(c) other active members are informed of the arrival of the new user

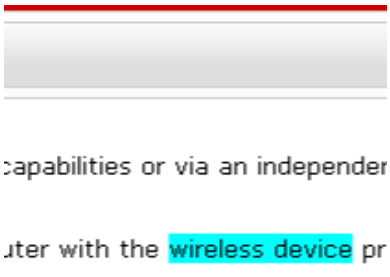


(d) showing annotated words on the misc UI container

Figure 8.13: Activating, entering and leaving a group

(a) an active group user is adding an annotation

(b) message sent to other users if an annotation is being added



(c) word automaticaly highlighted when annotation is added

(d) users informed that a certian user is adding a feedback

Figure 8.14: Activating, entering and leaving a group

(a) Wikipedia object has been loaded by an active group member

(b) An object from the Google search has been loaded by an active group member

Figure 8.15: Displaying Links and showing linked words

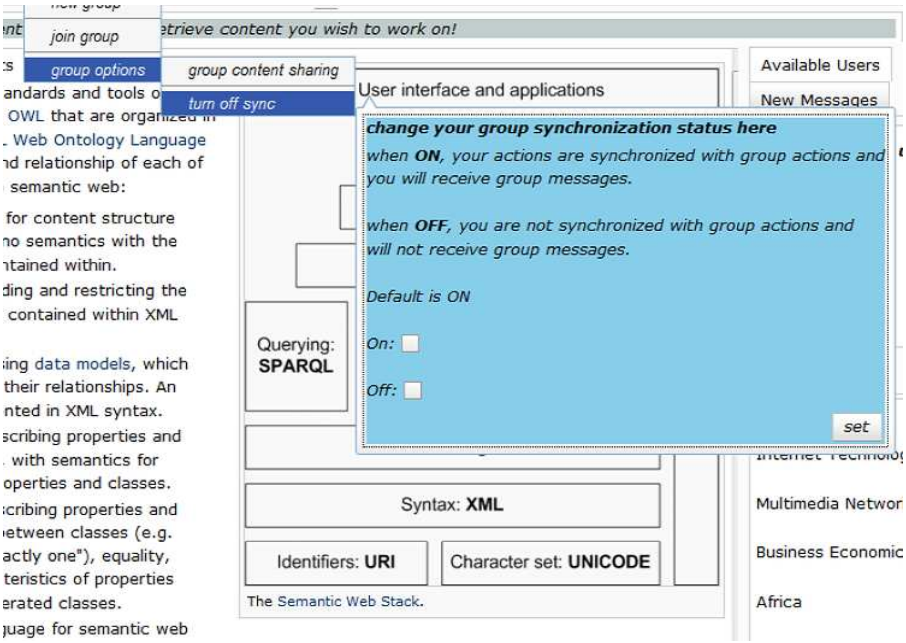


Figure 8.16: Setting group synchronization status

.

The usability test carried out and provided in this chapter, showed that the program worked as expected for Firefox and Safari. The look and feel of the program was much better for the Safari Web Browser.

Internet explorer worked only partially. The content retrieval and display worked well. Annotations and links could be added and viewed without any problems. The annotations and links display could also be displayed. Notes could also be added and viewed without problems. However, when a request involving the retrieval of data from the database is initiated such as retrieving all the feedback for an annotated word, IE throws a TypeError exception. These type of errors occur in IE if any of the values returned are null or undefined. This problem could be caused by errors in JSON file since JSON encoders were not used in this project and the encoding was done by hand. This problem could be solved by making sure that the value of objects is valid before attempting to use them. Due to time constraints, this errors have not been investigated further.

Dojo, the Javascript framework used in this project offers limited support for the opera browser as of now. So the application was not tested on Opera Web browser. No Unix-based browsers were tested.

Another issue which was noticed during the testing is CSS related. The CSS formating for hyLOs is not applied correctly when courses from hyLOs are loaded. The main CSS for hyLOs is also defined to affect the body of the document. This means that other sections of the UI, which are not suppossed to be affected by the hyLOs CSS are affected. To solve this

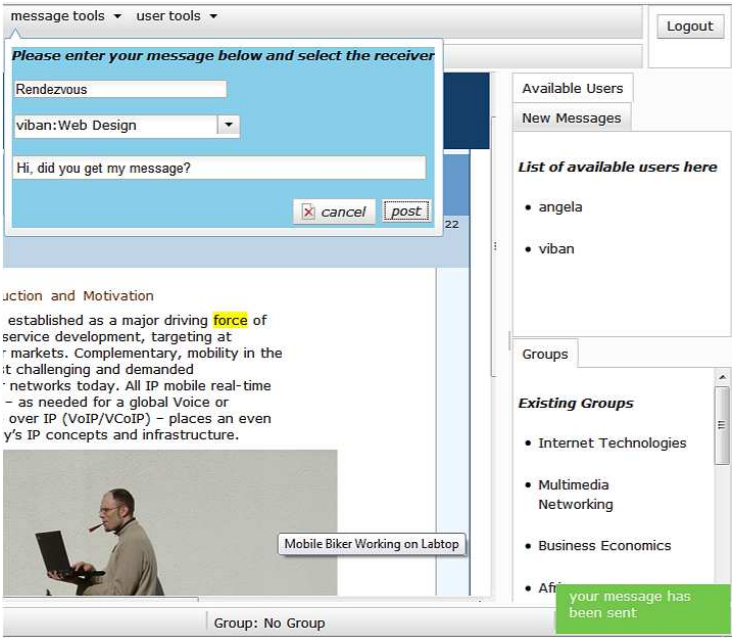


Figure 8.17: Sending a message

.

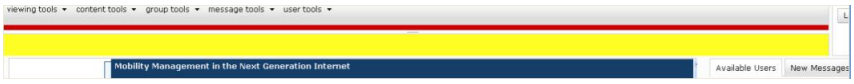


Figure 8.18: new message notification

.

problem, a CSS file for hyLOs was created for this application. The contents of the original hyLOs CSS file were copied into the newly created file. The formatting was modified to apply only to the workspace which will hold the hylos content. A JavaScript function was written to dynamically load this CSS file when hyLOs is loaded. Due to the fact the the dimensions of the workspace are smaller than the dimensions defined for the original hyLOs content, the CSS was not properly applied to the page.

The CSS from Wikipedia is also applied to the body of the XHTML document. A similar solution as above could not be implemented due to time constraints. When other XHTML Pages from the Google search are loaded their CSS if defined to format the body will also affect the body of the application. See chapter **??**, future work, for a proposed solution.

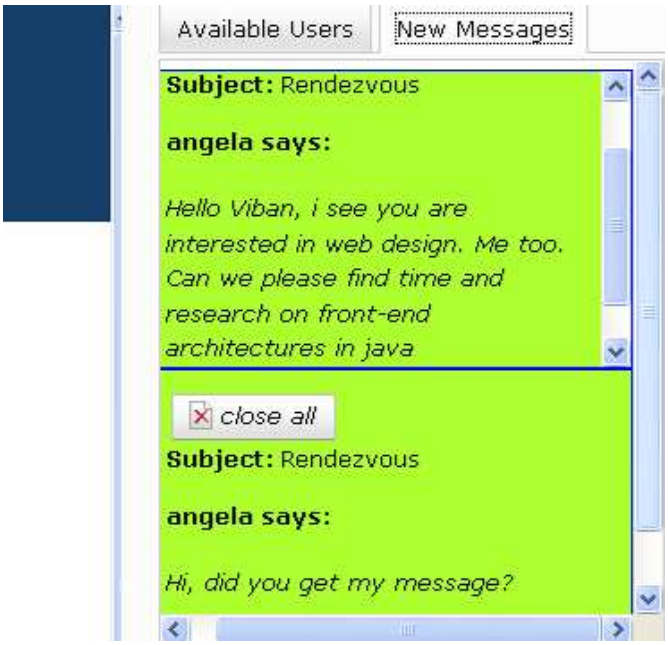


Figure 8.19: displaying messages

.

Chapter 9

Summary and future work

An elearning platform that enables users gather content from the Web and interact with it in given ways was built in this project. The three main content sources as of now are hyLOs (see section 2.3), Wikipedia and Google web search. The users can create their own content by adding annotations, notes and links to the content retrieved from the Web. Users can form groups and create this content together. This can be done by adding more links to linked words or giving feedback to annotations. In this way the content created has perspectives from different group members. This might improve the quality of the original document. This application was tested on three Web browsers: Mozilla Firefox, Safari and Internet Explorer. It worked well on Firefox and Safari. Internet Explorer threw *Typeerror* exceptions when data was retrieved from the data store. Due to time constraints, no attempt was made to resolve this problem.

The grouping functionality was tested for three users. This system has been designed such that the number of users who can actively take part in group work is limited only by the the hardware capabilities on the server on, which it runs. Theoretically, the Apache HTTP server used in this project can handle as many group users as there is available main memory. As more members keep joining the group and more request are made to the server, Apache will simply spawn more daemons or threads to handle these request, that is, until it runs out of main memory.

The design (5) of this eLearning platform allows enormous room for expansion. Like any other application in its infantry, this application also has its flaws, some of which became evident during the testing phase. A few suggestions and ideas are made here on how some of this problems can be solved and on how the application can be extended in the future and made more robust.

The system can be extended to offer more content sources to the user. A good example here is OpenCourseWare (OCW). OCW is an initiative which was pioneered in 2007 by Massachusetts Institute of Technology (MIT), whereby educational materials from its undergraduate- and graduate-level courses were placed online for free and made openly

available to anyone, anywhere. Ever since, most top universities around the world (Harvard Extension School [64], University of Notre Dame [65], United Nations University [66], etc.) have followed this trend. This has led to enormous amount of free educational material online. An OCW finder could be developed and integrate into the application developed here. This OCW finder could be based on a key word form the user. These courses can then be retrieved as *Really Simple Syndication* (RSS) feeds, which is a family of web feed formats used to publish frequently updated content such asblog entries, news headlines, audio, video, etc. The system could also create and manage a linking directory for OCW that have been found using the system. Another very important source of content could be the user himself. Users could be given the possibility to load content from their local file system onto the application. They can then work with group members on this content. They can decide to share this content with others of simply stored the modified version back onto the local file system. No concept has been developed for the *"more content"* suggestions made here. These are just ideas on how the application could let users profit from the vast amount of free educational material online, plus loaded content from the local file system can make the application more flexible.

As mentioned in section 5.4, one of the problems faced, when retrieving content from the Web is that the content is either volatile of semi-volatile. Wikipedia documents can be edited. The restrictions placed by the Wikipedia team on this editing feature makes them only semi-volatile. A two way solution to this problem is suggested here. Instead of grabbing the whole document from Wikipedia, only sections of the document will be grabbed. If any of these sections is edited, a copy of the edited section along with the added content is stored locally. If the user request this document in the future, the edited section that is store locally will be presented to the user together with the current section from Wikipedia for comparison. Tools should also be presented to the user to let him decide what to do if the document has been edited. Due to time constraints, only the first part of this solution has been considered here. Content keeps evolving and changing, which is a problem that cannot be solved within the scope of this thesis. Another future direction to consider would be to take the semantics of the added content into consideration when decided whether or not it has lost its value. The tags which users attach to added content or the description of links will play a big role if such a solution is considered. In this way, all added content which can no longer be linked to any document is considered ***stranded*** and can be reassigned to other content based on the meaning of the tags or link descriptions, i.e. its semantics. The assumption is that the tags and link descriptions are meaningful. This solution is not implemented here due to time constraints.

The content loaded into this application are mostly XHTML files. These files are formatted

[64] http://www.extension.harvard.edu/DistanceEd/

[65] http://ocw.nd.edu/courselist

[66] http://onlinelearning.unu.edu/en/

using CSS. For some of this content, e.g. from Wikipedia and hyLOs, the CSS is defined for the body of the document. This means the body of the application will be affected by the CSS styles of these sites when the content is loaded. The CSS styles of the content sources will affect each other depending on which file is loaded first, e.g. it was observed that the CSS from Wikipedia always affected the formatting of the hyLOs pages. A partial solution is to dynamically load and remove the CSS files depending on which content is loaded. For example, if a document from Wikipedia is loaded after a document from hyLOs has just been loaded, the system should automatically remove the CSS styles for hyLOs and load those for Wikipedia. If the user switches to the hyLOs content, the CSS styles for Wikipedia should be removed and those for hyLOs loaded. Each time a random page from the Google web search is loaded, the CSS style of both Wikipedia and hyLOs are removed. They should be dynamically reloaded when the page is closed. A script to dynamically load the CSS pages was written and used in this script. The hyLOs CSS styles were dynamically loaded after successful user registration or login to avoid the hyLOs CSS styles from affecting the CSS styles defined for the registration and login forms. Due to time constraints, the rest of the solution as described above could not be implemented. This will be a good starting point if the system was to be further developed.

The application design lets users tag all content they add to the system. The created content will be of no use if it cannot be used by other users, that is, if it is shared. With the help of tags, the application can be further developed to retrieve annotations and feedback for that annotation based on the tags added by the users who created that information. More useful here will be the links added to the system. The tags for links can be used to some sort of a semantic link network within the application. This means that, the application can automatically build relationships between documents based on the tags added. For example, if a link A is tagged IP addresses, link B IPv6 and link C MIPv6, then the relationships *B-isSubsetOf->A*, *C-isSubset->B*, *C-SubsetOf->A* can be built. If a user references the document pointed to by link C, documents pointed to by link A and B are suggested as parent documents. Such a semantic link network can be extended to build and interpret all sorts of complex relationships between links leading to automatic discovery of relevant content present within the system. Such a semantic link network will be based on the accuracy and richness of the tag supplied by the user. The application could also be extended to implement a linking directory based on the tags and the description of the added links. When a user requires information on a particular resource, the application based on its linking directory could make suggestions to the user about similar external resources. This can make the site more useful to users on the condition that the tags added are meaningful. The directory listing can be implemented to periodically remove dead links from the system.

The application that is developed constantly pools the server to determine if a new document has been loaded for the group and if a new group notification is available. For a small number of users, there are no noticeable latency issues. But as the number of users increase, latency will increase as well and the server might be eventually overloaded. Re-

verse Ajax is a *server push* low-latency method which can be used to minimize the amount of request made to the server. The technical term for this server push technique as used in web development is ***Comet*** [67] [68]. Comet allows the server to start answering the browser's request for information, and to continue answering on a schedule dictated by the server. This application could be implemented to use Comet on the server side. This will be advantages as the number of users increase because with Comet, there is no need to wait for the next time the browser connects. Information is simply sent to the client when it is available.

[67] Alex Russell's original post coining the term Comet http://alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/

[68] What Wikipedia says on Comet http://en.wikipedia.org/wiki/Comet_(programming)

Bibliography

[1] Terry Smith Alex Koohang, Liz Riley and Jeanne Schreurs. E-learning and constructivism: From theory to application. *Interdisciplinary Journal of E-Learning and Learning Objects*, 5:19, 20 October 2009.

[2] Paul Anderson. What is web 2.0? ideas, technologies and implications for education. *JISC Technology and Standards Watch,*, 6 December 2009, 2007.

[3] Tim Berners-Lee. *Weaving the Web*. tim, 2000.

[4] Cristian Darie. *Ajax and PHP - Building Responsive Web Applications*. Packt Publishing Ltd, 32 Lincoln Road, Olton, Birmingham, B27 6PA, UK, 2006. ISBN 1 - 904811 - 82 - 5.

[5] Paul J. Deitel and Harvey M. Deitel. *AJAX, Rich Internet Applications, and Web Development for Programmers*. Prentice Hall, January 2008.

[6] Wroclaw University of Economics Wroclaw Poland Eli B. Cohen, Informing Science Institute Santa Rosa CA USA Malgorzata Nycz. Learning objects and e-learning: an informing science perspective. *http://ijello.org/Volume2/v2p023-034Cohen32.pdf*, 8 October 2009:12, 2006.

[7] Michael Engelhardt, Arne Hildebrand, Alexander Lang, Thomas C. Schmidt, and Mathias Werlitz. A constructivist content exploration based on a hypermedia elearning object system. In Michael E. Auer and Ursula Auer, editors, *Proceedings of the International Conference Interactive Computer aided Learning ICL 2004. The Future of Learning*, 09 October 2009.

[8] Michael Engelhardt, Arne Hildebrand, Dagmar Lange, and Thomas C. Schmidt. Reasoning about elearning multimedia objects. In Jacco Van Ossenbruggen, Giorgos Stamou, Raphaël Troncy, and Vassilis Tzouvaras, editors, *Proc. of WWW 2006, Intern. Workshop on Semantic Web Annotations for Multimedia (SWAMM)*, May 09 October 2009.

[9] Michael Engelhardt, Arne Hildebrand, Thomas C. Schmidt, and Mathias Werlitz. The hypermedia elearning object system: Exploiting learning objects in a semantic educational web. In *Proceedings of the 19th Codata Conference*, Berlin, November 09 October 2009.

[10] IEEE. Draft standard for learning object metadata - ieee 1484.12.1. http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pd, July 10 October 2009.

[11] Marja-Riitta Koivunen. *Annotea and Semantic Web Supported Collaboration*. PhD thesis, Annotea project, 12 October 2009.

[12] link lab. Kursübersicht haw hamburg. http://hylos.cpt.haw-hamburg.de/index.xhtml, 20 September 2009.

[13] J. M. McInnerney and Tim S Roberts. Online collaborative learning: Have we overcome the obstacles? In *Proceedings of the International Conference on Computers in Education (IEEE)*, page 2, Central Queensland University Bundaberg, Queensland 4670, Australia 09 October 2009.

[14] Tom Negrino and Dori Smith. *Javascript & Ajax*. Peachpit Press, 1249 Eight Street, Berkeley, CA 94710, 2009. ISBN-13: 978 - 0 - 321 - 56408 - 5 - X.

[15] Joe Fawcett Nicholas C. Zakas, Jeremy McPeak. *Professional Ajax*. Wiley Publishing Inc., 10475 Crosspoint Boulevard, Indianapolis, IN 46256, 2006. ISBN: 0 - 471 - 77778 - 1.

[16] Tim O'Reilly. What is web 2.0 design patterns and business models for the next generation of software. O'Reilly website, 16 December 2009.

[17] O'Reilly Radar. Web 2.0: Compact definition?. Available online at: http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html, 6 December 2009.

[18] Peter Brusilovsky Rosta Farzan. Annotated. a social navigation and annotation service for web-based educational resources. In *Proceedings of E-Learning 2006 (Honolulu, HI, October 2006), AACE, 2753-2757*, 12 October 2009.

[19] Rawld Gill Craig Riecke Alex Russell. *Mastering DOJO JavaScript and Ajax Tools for Great Web Experiences*. The Pragmatic Bookshelf, 2008.

[20] Thomas Schmidt. General introduction to hylos. http://inet.cpt.haw-hamburg.de/projects/hylos/, 10 November 2009.

[21] Thomas C. Schmidt and Michael Engelhardt. Educational content management. In F. Garcia, J. Garcia, M. Lopez, R. Lopez, and E. Verdu, editors, *Educational Virtual Spaces in Practice*, pages 105–118. Ariel, Barcelona, 09 October 2009.

[22] Gordon McCalla Scott Bateman, Christopher Brooks and Peter Brusilovsky. Applying collaborative tagging to e-learning. Technical report, 10 December 2009.

[23] Rosta Farzan Scott Bateman Rosta, Scott Bateman, Peter Brusilovsky, and Gord Mccalla. Oats: The open annotation and tagging system. In *Proceedings of 12LOR 2006 (Montreal, Canada, November 2006)*, 12 October 2009.

[24] Barbara Leigh Smith and Jean T. MacGregor. What is collaborative learning? *National Center on Postsecondary Teaching, Learning, and Assessment at Pennsylvania State University.*, 9 October 2009.

[25] Jason Cranford Teague. *CSS, DHTML and AJAX*. Peachpit Press, 1249 Eight Street, Berkeley, CA 94710, 2007. ISBN 0 - 321 - 44325 - X.

[26] Anne van Kesteren. Xmlhttprequest. http://www.w3.org/TR/XMLHttpRequest/, 22 January 2010.

[27] Front-end software architect from Cogniance Vlad Agafonkin. Web 2.0 startups. choice of javascript framework. http://cogniance.com/uploads/white-papers/Web-2-0-Choice_Of_Javascript_Framework.pdf, 26 Janaury 2010.

[28] W3C. W3c xml pointer, xml base and xml linking. http://www.w3.org/XML/Linking, 11 November 2009.

Appendix A

CD-Contend

The following contents can be found on the included CD[69]

1. Master thesis in PDF-format and the LaTeXfiles
2. Source Code - JavaScript - client
    - clientValidation
    - contentHandlin
    - css
    - eui
    - grp
    - misc
3. Source Code - PHP - Server
    - authentication
    - grpFxn
    - hylos
    - misc
    - validation
    - wikipedia
    - google

[69] The CD can be viewed at the supervising examiner's office.

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that this Master Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, March 1, 2010 Viban Terence Yuven