Object Retrieval and Student Behavior Using Tags in a Learning Context

Christian Hochlin

Master's Thesis
Master of Science in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2011

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Learning objects are reusable digital resources used in education. They are expensive to create and it is hard to maintain a sufficient amount of metadata in them. This impedes the ability to locate learning objects for reuse.

Computer supported learning, and its branch called e-learning, aims to engage the students in their own education. This can be beneficially used to create metadata in learning objects, by engaging students to tag their own learning objects.

This thesis examines how students use a system designed for annotating learning objects, and discovers that students are not willing to tag. Interviews were conducted, which discover that a tagging system needs to be introduced as an integral part of the course to induce participation, as well as properly communicate the benefits for the students themselves. The used system, implemented in the Learning Object Repository DSpace, is evaluated to determine how well it is suited for implementation of social technologies like tagging in a production environment. The thesis also investigates how well the tags produced can be used to retrieve related objects by testing a new algorithm. Promising results were observed with a test set, but the algorithm could not be tested with real tags from this study, as the tag set was too small.

Acknowledgments

This Master's thesis is the final work produced during my study at the Faculty of Computer Science and Media Technology at Gjøvik University College.

I want to thank my supervisor Rune Hjelsvold for his guidance and input throughout the time spent working on this thesis. I would also like to thank my classmates and friends who kept me company, both off- and online, during what could have been a lonely time working on this thesis. Lastly, thanks to Jan A. Audestad, who made his learning objects available for my study, as well as the students who participated and especially those who agreed to be interviewed.

Christian Hochlin, 1. July 2011

Contents

List of Figures

List of Tables

1 Introduction

Increasing amounts of information today, including education, has been moved to the digital world. With the advent of The Internet, e-learning became more widespread. E-learning can be defined as *"all forms of electronic supported learning and teaching"*[1].

A branch of e-learning, Computer-Supported Collaborative Learning (CSCL) has also gained more attention. To some extent, it has emerged in reaction to previous attempts to use technology in education[2]. It has also been called e-learning 2.0, for its adoption of social concepts, akin to the colloquially named Web 2.0[3]. Much of the learning material has been made available online, available for students regardless of location. This has made distance education a viable alternative, with 63% of U.S. institutions reporting distance education as a critical part of their long term strategy[4]. As Web 2.0 changed the way we use the web, going from merely being spectators to actively take part as participants, collaborative learning regards education not as a one way-communication platform, where the teacher teaches and the students learn. A part of the philosophy is that knowledge is created in the community, by engaging the students[5, p. 37]. There are numerous ways the students can participate, e.g. commenting, rating and tagging learning objects. One problem is that few Learning Management Systems (LMS) facilitates interaction with the learning objects. A Learning Management System is a collection of tools to support learning. The tools are integrated in a common environment, with access to shared data, presented in a web-based interface. Examples of Learning Management Systems are Fronter[6] and It's learning[7], which are the two most used systems in Norway[8]. In addition to Learning Management Systems, Learning Object Repositories (LOR's) are used in education. They are used for storing learning objects and their metadata[9].

P. Ramsden states that "Access to the learner's perspective on the activities of teaching and learning is essential for understanding educational phenomena - and for improving education"[10, p. 35]. Albeit an old quote, it is a good summarization of the ideas behind the trends in e-learning today.

In this chapter, we will present the topics covered in this thesis as well as the problem description, our motivation for researching these topics and the research questions.

1.1 Topic

At the heart of all forms of E-learning are learning objects. Learning objects can be defined as

"any digital resource that can be reused to support learning"[11].

An important part of the research on learning objects focuses on the reusability of objects used for learning purposes[12], and a lot of research exists on how to ensure the reusability of learning objects. Many of the approaches rely on the learning objects having well formed and extensive metadata to accomplish this[13][14].

Metadata is generally defined as data about data[15], and is descriptive information about a resource. The "made in ..."-label on clothes, as well as the Dewey Decimal

System[16], used for classifying books in a library, are examples of metadata used in the real world. In the digital world, there are numerous standards specifying which metadata an object should contain, like Dublin Core[17] for generic resources and the ID3-format[18] for mp3-files. The most widespread standard for metadata in learning objects is the IEEE LOM-standard[19], which specifies a set of fields to fill out for each learning object. Metadata is used in many ways, including resource discovery, organization and for facilitating interoperability[20].

Most Learning Object Repositories use either Dublin Core or IEEE LOM as their metadata scheme. A number of software packages exist that can be used as the basis for an institution's repository, including Fedora[21] and DSpace[22].

Tagging is the act of assigning a personally determined keyword to a resource. It can be described as a special kind of metadata, as the tags are often used to describe the content of the resource, even though it differs from what we perceive as regular metadata by being less structured. An advantage of tagging is quicker retrieval of related content[23].

A folksonomy is a set of terms the users tag content with. A portmanteau of "folk" and "taxonomy", coined by Thomas Vander Wal, folksonomies allow people to connect items by "placing hooks", to provide meaning in their own understanding[24]. Despite its chaotic nature, with no inherent structure or relationship between tags, it is accessible to users and enables them to organize content without investing a significant amount of time or effort[25].

The main goals of the thesis are to investigate how students use a tagging system in a real-world scenario, and how the students' tags can be utilized to locate related resources.

1.2 Keywords

Tagging, Learning Object, Folksonomy, Metadata, DSpace

1.3 Problem Description

Learning objects are expensive to create[26], partly because creating them is a time-consuming process[27]. The practice of adding enough formal metadata—like dublin core-fields, or the IEEE LOM—is not widely adopted among professors, so reuse of learning objects is hard to achieve within a faculty or school. Lack of metadata in learning objects defeats some of the purpose, as an important part of the definition of learning objects is that they should be reusable. With distance education on the rise, there is generally less face to face communication between teachers and students, especially concerning distance students. Social media and communication with fellow students are a valuable asset, but this is not made easy by current Learning Management Systems, as they do not natively support "Web 2.0"-concepts like tagging.

1.4 Justification, Motivation and Benefits

As we have observed, a growing trend is to let students participate in the education and encourage them to be more involved in the courses. By utilizing this trend, students can be encouraged to participate by tagging their own learning objects.

The ability to tag learning objects may be valuable for both professors and students. For *motivational purposes*: As people may feel a stronger sense of belonging when they can be a part of—and not just observers of—the course material, and for *informational purposes*: How well received a particular learning object is among the majority of students. It could also give them the ability to organize content in a personal way[25], and discover related content. Professors could benefit by having a direct channel to gather feedback from students on their learning objects[28], and being able to discover related learning objects to use in the courses. By reviewing tags, they can get an indication of how well students understand the content of the learning objects, and what they feel is important.

Just as letting students tag their learning objects can solve the metadata problem, student created metadata can solve the problem of little reuse of learning objects. Student created metadata is relatively cheap to produce, as the task of producing it is distributed among a large number of users rather than maintained by a small group.

A method to facilitate easier reuse of the learning objects would save money and free up time and resources. This is a method that will provide additional metadata to learning objects without needing to spend extra time on their creation. The beneficiaries of this would include the schools currently struggling with lacking metadata in their learning objects, as well as students.

This thesis will assess how student tags can be used as metadata for retrieval, and how students themselves value the addition of tagging as an educational tool. To gather tags, the students will use a real system for annotating learning objects. Most research in this area investigates how student supplied tags are distributed, compared to another form of metadata. As far as we are aware, few studies focus on how the students themselves value the addition of tagging to learning objects. In [28], J. Fan did a preliminary study on the subject, but with no definite results.

To assess how students use a real system, a prototype of such a system needs to be created as tagging is not included in any Learning Management Systems or Learning Object Repositories as of now.

1.5 Research Questions

RQ1 **How can we integrate tags in a learning environment?**

This research question will try to understand how we can integrate tags into a learning environment in a way that will take advantage of the students' behavior.

RQ2 **How can student supplied tags be used to retrieve related learning objects?**

Tags can be used as complementary metadata[28], but will they work as well as regular metadata for retrieving related resources? Related resources means resources with subject matter that overlaps with what the original resource conveys. Technical aspects regarding the usage of tags as metadata used by retrieval algorithms will be answered in this question.

RQ3 **What are the challenges of retrieving related learning objects based on student tags?**

This question will be answered by outlining the challenges encountered while developing the system, the retrieval algorithm and during the user testing phase.

RQ4 **How can social technologies be implemented in DSpace?**

This question aims to answer how well suited the Learning Object Repository DSpace is to incorporate social technologies, such as tagging and rating, from a technical point of view. DSpace was chosen as the LOR of choice, as Gjøvik University College (GUC) is looking into putting it to use.

2 Related Work

2.1 Learning Objects

The definition of a learning object is the source of much disagreement. The IEEE's definition is ”any entity, digital or non-digital, that may be used for learning, education or training”[19]. This definition is considered by many to be too wide, as it can include everything that has ever existed. Wiley[11] tries to narrow the definition down, and states that a learning object is ”any digital resource that can be reused to support learning”. This is the definition adopted in this thesis. Others argue that this definition is also too wide. Parrish[29] says ”Instead of trying to define learning objects as entities or particular artifacts, it may be more useful to view learning objects as a process or strategy[. . .]. Construed this way, what is designated as an object is contingent on the learning object system being discussed, avoicing[sic] fruitless disagreement and confusion.”

The details of how to define a learning object can be discussed at length, but the majority of definitions agree that a learning object should be used to support learning, and be reusable.

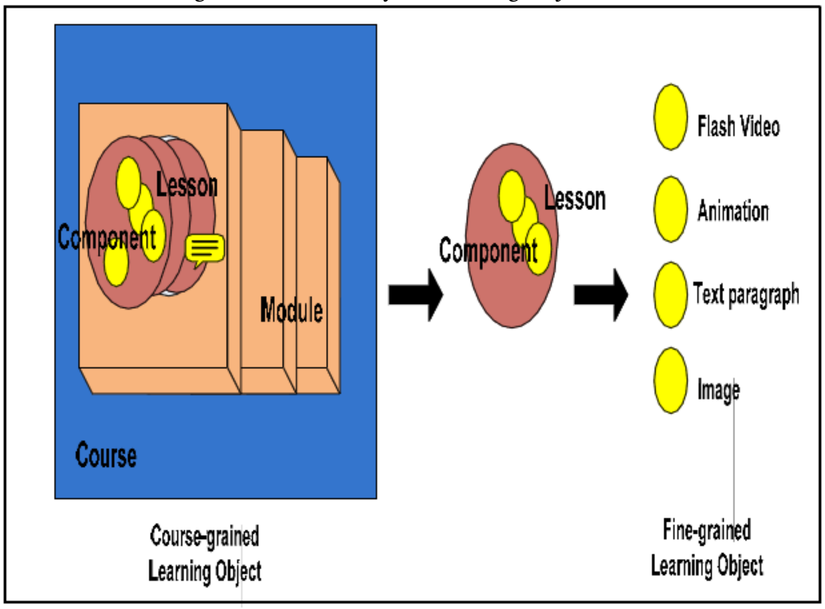
A popular metaphor to explain the use of learning objects is by comparing them with LEGO^{TM}-blocks. This succeeds in communicating the basic idea, namely that learning object are small pieces that can be assembled into different structures. Wiley argued in [11] that this metaphor is faulty, as some ideas that are true for LEGO are not necessarily true for learning objects. Any LEGO-block can be combined with any other LEGO-block and they can be assembled easily in any manner, which is not true for learning objects. He further proposed to compare learning objects to atoms, as they share a larger set of properties with learning objects. Primarily, not every atom can be combined with any other atom, and they can only be assembled in certain structures, depending on their own internal structure. This metaphor is much closer to how real learning objects behave.

Tags are used to organize and retrieve related content. Giving students access to multiple learning objects regarding the same topic, that may offer different interpretations, can turn out to be a major educational benefit[29]. This can also show the professors related objects that are reusable.

Several studies focus on the reuse of learning objects. Figure 1, adopted from [30], shows learning objects of different granularity. Granularity is by some recognized as the central component in how reusable a learning object is. Some state that reusability requires the LO to be in a fine-grain form, because raw media elements are often much easier to reuse than aggregate assemblies[30]. The metrics they used for measuring reuse in this study have been adapted from object oriented engineering techniques. They applied these metrics to fine-grained learning objects, and discovered that the reusability is generally high for fine-grained objects. They did not compare this to coarser learning objects in this study.

X. Ochoa and E. Duval in [12] measured the reuse of learning objects, based on the granularity of the objects. They used empirical data collected from three freely available

Figure 1: Granularity of Learning Objects[30]



sources, and their goal was to check if granularity was an important factor in the reusability of a learning object. They had access to usage statistics of the learning objects, such as reuse percentage. They found that even without proper facilitation, people reused about 20% of the learning objects, regardless of granularity. The most important discovery made was that objects with granularity immediately lower than the object being built was easier to reuse than objects of much lower granularity. For example, when building a course, it is easier to reuse whole lessons than individual images or text paragraphs. The Ochoa et al.-study has shown that it is possible to reuse objects, regardless of granularity.

To facilitate reuse, metadata is regarded as important. The IEEE have their own standard, the IEEE LOM, as mentioned in section 1.1. The creators of ARIADNE, a European Educational Content Management System, regard metadata as extremely important for the proper management of any resource[14].

2.2 Folksonomies

Mathes, in [25], examined user-generated metadata (tags) in Flickr and Del.icio.us. He identified limitations and advantages regarding folksonomies as a categorization tool. Ambiguity and lack of synonym control are the two major limitations identified. There is no way to systematically classify the terms, as different users use the terms differently. When no context is supplied one word can encompass numerous meanings and we have no way of knowing which meaning the tagger had in mind. Synonym control is also hard to do in a system with potential for large collections of keywords with different meanings. Apple would, for instance, be a synonym for Macintosh if the tagger meant the company, but not if he referred to the fruit.

Limpens, Gandon and Buffa in [31] identifies four issues regarding the use of folksonomies, which resemble the limitations identified by Mathes.

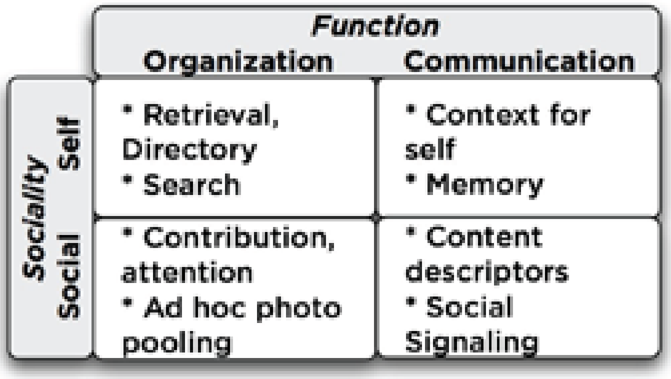
”(1) the ambiguity of tags, for one tag may refer to several concepts ; (2)

the variability of the spelling, for several tags may refer to the same concept; (3) the lack of explicit representations of the knowledge contained in folksonomies; (4) the difficulties to deal with tags from different languages."

One aspect folksonomies excel in is representing the users own vocabulary. Folksonomies can be seen as the digital equivalent of desire lines[32]. Desire lines are the foot-worn paths that sometimes appear in a landscape over time, that show where the pedestrians walk, rather than where the landscape architect planned for the pathways to be situated. In the same vein, folksonomies represent the users' needs, rather than the system designer's intentions. Furthermore, Mathes argued that the most important reasons why folksonomies work are the low entry cost for the users and that the context of use is not just personal organization but communication and sharing as well. Some of the areas of further research identified in this article deals with analyzing tags quantitatively, as well as a qualitative user behavior analysis. Mathes hypothesized that the distribution of tags will follow a power law scenario, where the frequently used tags are more likely to be used by others, due to higher visibility. Later studies have shown results that support this hypothesis[33].

M. Ames and M. Naaman have studied the incentives for tagging in Flickr, a web-based photo-sharing system, and ZoneTag, a cameraphone photo capture and annotation tool that uploads images to Flickr[34]. The main part of their study involved interviews with users of the two systems. They found several motivations for why people annotated their photos, which is shown in figure 2, from [34]. A majority of their participants were motivated to tag by organization for the general public, with self-organization and social communication in second place.

Figure 2: Tagging Motivations[34]



		Function	
		Organization	Communication
Sociality	Self	* Retrieval, Directory * Search	* Context for self * Memory
	Social	* Contribution, attention * Ad hoc photo pooling	* Content descriptors * Social Signaling

Bao et al.[35] observed two ways to benefit web searching using tags.

(1) Tags can be used as summaries of the web pages they are assigned to.

(2) The number of tags assigned to a web page indicates its popularity. This is also supported and elaborated on in a previous study by Xu et al.[36]

2.3 Tagging in a Learning Context

In [37], S. Bateman et al. compared students', experts' and text mined tags. They noticed a surprising correlation between student tags and the keywords created by text mining.

Only 50% of the expert tags were found in the student tags, but as much as 68% of text mined tags occurred in the body of student tags.

[23] investigated the role of social networks in computer science education. They observed that as learning has evolved from taking place in the physical world to computer-supported learning systems, substituting the social part of learning when dealing with digital learning is important. They looked at existing social software applications, and discussed the potential for employing these applications in education. They stated that an advantage of tagging is quicker retrieval of related content, and if users are from the same network and share similar interests—as students often do—it is easier and more reliable to apply recommendation algorithms.

J. Fan[28] studied how student tags can be utilized as metadata in learning objects. Findings include that student tags can be used as complementary metadata to keywords created by lecturers, that students tend to favor tags describing key aspects of the learning objects and that the tags fit into these metadata elements in Dublin Core and IEEE LOM:

> ”1. In Dublin Core, tags can be used to describe description, subject;
>
> 2. In IEEE LOM, tags can be used as 1.4 Description, 1.5 Keywords, 9 Classification;”[28]

2.4 Ranking Algorithms

There are numerous algorithms designed for ranking objects based on similarity. A typical process ranking algorithms use when queried are:

- Assign weights to terms in a collection of documents and the query
- Compare the weighted terms present in the document collection to the terms in the query
- Rank the results

This process is used by a number of algorithms, including the vector space scoring model.

2.4.1 Vector Space Scoring

The vector space model represents documents as vectors in a common vector space. This is the base for a host of information retrieval methods, including document clustering, classification and computing score based on queries.[38]

The idea is to assign a weight, w, to each term present in a document, $\mathtt{d}$, compute a score based on these weights, and use the score to determine how relevant $\mathtt{d}$ is to a given query, $\mathtt{q}$.

This is achieved by computing the *inverse document frequency* $\mathtt{idf}_t$ of a term. $\mathtt{N}$ represents the total number of documents in the collection.

$$\mathtt{idf}_t = \log \frac{\mathtt{N}}{\mathtt{df}_t}$$

The $\mathtt{idf}$ of a rare term will be high, and a common term will have a low $\mathtt{idf}$. This ensures that documents containing rare terms will be weighted higher when a query contains the rare term. The final weight of each term is computed by multiplying the inverse document frequency with the number of occurrences of the term in $\mathtt{d}$. Each document is then treated as a vector comprised of these weights.

2.4.2 HITS

The HITS-algorithm[39],developed by Jon Kleinberg, is used to classify pages into *Hubs* and *Authorities*. *Hubs* are documents which link to many high quality documents. *Authorities* are documents which are linked to by many high quality *Hubs*. When a user queries a search engine, *Authorities* are the truly relevant results for the given query, and the *Hubs* are used to identify the *Authorities* and point the user in the right direction. Every document is assigned two values, a hub weight, and an authority weight. The hub weight is increased if the document points to other documents with a high authority weight. The authority weight is increased if the document is pointed to by documents with a high hub weight.

The algorithm starts out with a set of documents with the highest occurrence of the search phrase, called R_Q, root query. These documents are typically not heavily interlinked, so the next step is to extend the set to include all documents linked to by any document in R_Q. All pages are now organized into a matrix, A_{ij}, and each entry is either set to one or zero, depending on if there is a link from i to j.

Figure 3: Simple Set Graph

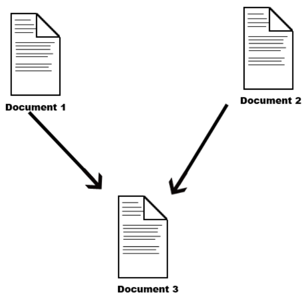


Figure 3 shows the graph to a simple set. Its corresponding matrix looks like this:

$$R_Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Now we have to assign a weight to each document. For this example, we set the initial hub weight vector to

$$u = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The easiest way to apply the weighting to the documents is to transpose the matrix and multiply the result with the weight vector.

$$R'_Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Multiplying the transposed matrix with the initial weight vector creates the authority weight vector, u.

$$v = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

We can then get the new hub weight by multiplying v with the original matrix R_Q.

$$u = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}$$

As this algorithm is developed primarily for use in search engines it must be modified to be useful in a tagging environment. Wu et al.[40] proposed a modified version of the algorithm to recommend documents based on tags. The algorithm works by starting from a small root of documents which includes the documents tagged with the keyword in question. Then the document set is expanded to include all documents that are associated with any tags contained in the root set. The documents are then weighted based on a set of metrics to determine relevance to the source tag.

2.5 Summary

A learning object is a digital resource that can be reused. Learning objects can be compared to atoms, as they cannot be freely put together, but have to be assembled in structures depending on their internal structure. This makes it important to be able to locate related objects, as a premise for reusing learning objects is that they are related to the goal of the new learning object. Having fine-grained learning objects in order to be able to reuse them has also been regarded as important, but some studies show that it is not necessarily a requirement.

Metadata is also regarded as important for managing resources. Folksonomies are a form of metadata created by the users. It is limited in some aspects to more controlled forms of metadata, but excel in others, notably the low accessibility barrier to participate, as well as the inherent social functions. Studies have shown that there is a need for social functions in digital education, and the utilization of tags in an educational context may improve the retrieval of related content.

2.6 Research Questions Revisited

This thesis will map the behavior of students using a prototype tagging system, to try to answer how the implementation process can be approached.

Retrieval of related learning objects is the first step for facilitating reuse. In addition, it may help students by supplying related learning objects for further study. We will investigate how student supplied tags can be used for this purpose, by employing an algorithm to relate learning objects and interviewing students about their perceived use of the system.

To answer research question 1, we will use the developed system to analyze the students' behavior while tagging. This, as well as in-depth interviews, will help us understand factors which are important when introducing tags as part of a learning environment.

Research question 2 will be answered by analyzing which tags students are using, and determine how these can be used efficiently in the creation and tuning of an algorithm designed to use tags to search for relevant literature.

Research question 3 addresses the challenges regarding using student tags as metadata for retrieval and will be answered by identifying issues raised while working, as well as concerns raised by students.

Research question 4 will answer how well suited DSpace is to be used as a development platform for social technologies and will be answered using our experiences developing the prototype tagging system used in the study.

Answers to these questions will help build a base for further research related to including social technologies in an educational environment.

3 System Description

The system created is a plug-in in the open source software package known as DSpace[41]. A plug-in which modifies or adds functionality to a DSpace installation is called an *Aspect*. To understand how and why design choices were made, some background knowledge on the design of DSpace is needed. The version of DSpace used in this thesis is version 1.6.2.

3.1 DSpace

DSpace is used as a management tool for digital content, often employed as a Learning Object Repository in educational facilities. Due to its open source nature, developers are able to extend on the basic functionality offered out of the box to fit their needs. The content is structured in a hierarchy, starting with *Communities* as the top level. Each *Community* contains one or more *Collections*. The objects are placed inside the *Collections*. Compared to the structure of an educational facility, a Community can be compared to the various faculties present in a school. As faculties contain a number of courses, Communities contain a number of Collections. Every item can resemble the learning objects associated with a course. This is one way to map content to closely resemble an existing system.

The classic user interface is called JSPUI, and is based on JSP-pages and servlets. A new approach has been included in later versions of DSpace, starting with version 1.5, which is called Manakin[42]. Manakin is based on Apache Cocoon[43], which is a framework relying heavily on pipelines, components and separation of concerns. These concepts will be addressed in later chapters. The development process when customizing these two interfaces are very different from each other. This thesis will focus on Manakin, as it is the newest and more robust of the two.

Manakin is built with modularity in mind, so all core parts are separated into different *Aspects*.

ArtifactBrowser

This aspect deals with every functionality regarding the navigation of the submitted items, i.e. browsing, searching and viewing.

Eperson

The Eperson-aspect is responsible for the user. Registration, login and logout are parts this aspect deals with.

Submission

The third core aspect deals with the submission of new items.

Administrative

The administrative-aspect manages all administrative tasks, e.g. letting only authorized users view an item.

General

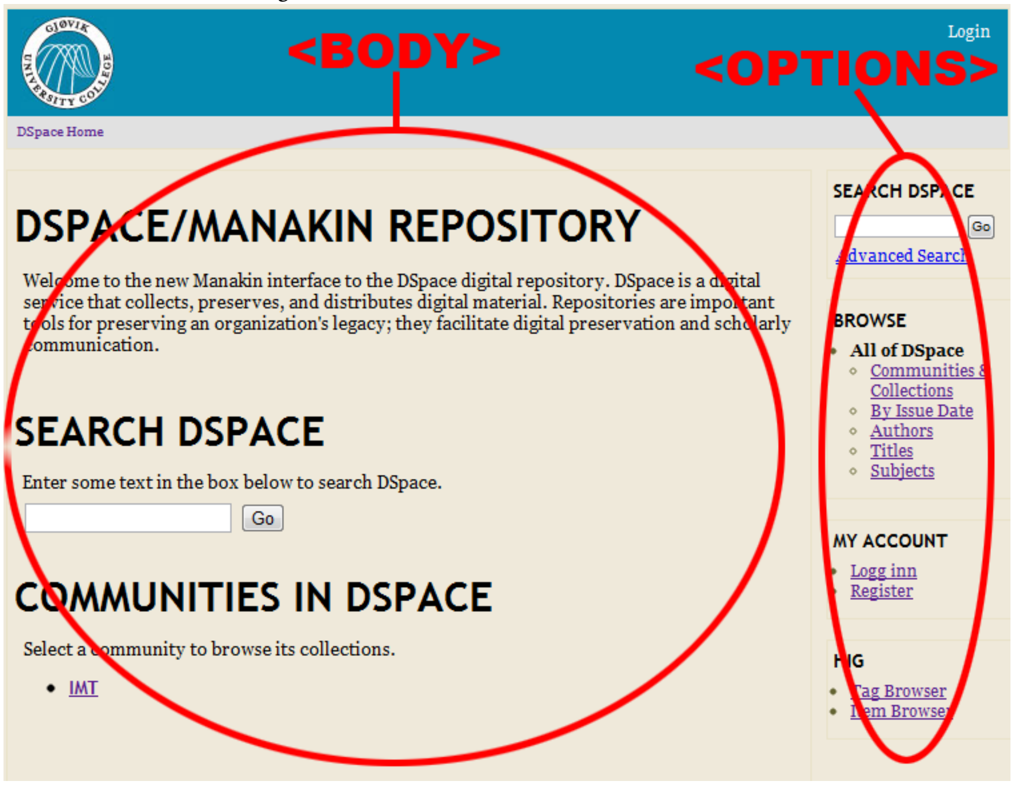
The last aspect is a collection of java-classes with general functionality, used by the other core aspects.

A custom aspect is implemented the same way as the core modules. This means that, conceptually, there is no difference between a user made aspect and the core functionality.

3.1.1 DRI

The DRI, Digital Repository Interface, document is an XML document that describes an abstract representation of a repository page, and contains no styling information. It has three main elements, <body>, <options> and <meta>. The contents of the <body>- and <options>-tag are the visible parts of the final html-page. The body-tag roughly translates to the content one can see in the editorial area, and can contain structural elements, such as lists and HTML forms. The options-tag are what constitutes the navigational menu on the right hand side of a stock install of Dspace, and contains structural elements. Figure 4 shows these two parts and their location on the page. If a user is

Figure 4: Visible Elements of the DRI Document



DSPACE/MANAKIN REPOSITORY

Welcome to the new Manakin interface to the DSpace digital repository. DSpace is a digital service that collects, preserves, and distributes digital material. Repositories are important tools for preserving an organization's legacy; they facilitate digital preservation and scholarly communication.

SEARCH DSPACE

Enter some text in the box below to search DSpace.

COMMUNITIES IN DSPACE

Select a community to browse its collections.

- IMT

interested in viewing an item, the information about the item must be visible. Thus, the aspect *Artifactbrowser* will be invoked, and add data about the item in the <body>-area.

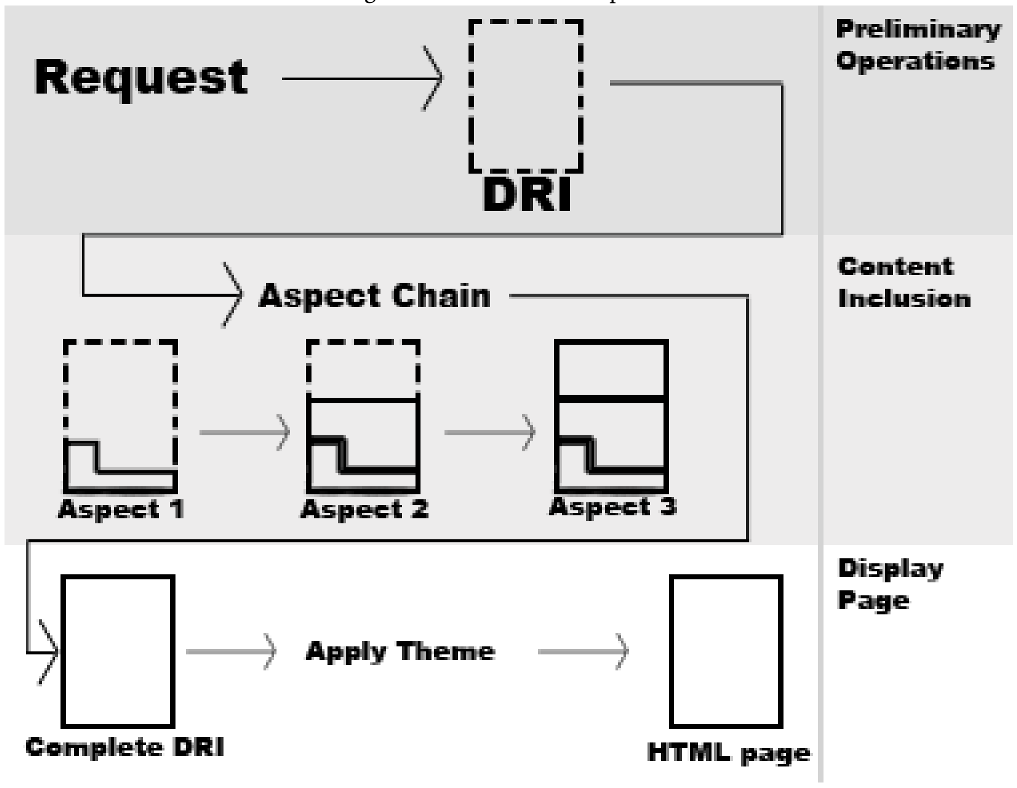
The meta-tag contains two parts, metadata connected to the current user, and the page's metadata.

The *Eperson*-aspect will add its data about the user in the <meta>-area. Other information added under the <meta>-tag is the port and name of the server running DSpace.

3.1.2 Pipeline

Figure 5 illustrates how DSpace manages the pipeline in Manakin.

Figure 5: The Manakin Pipeline



Links to DSpace objects have links that look like:

http://dspaceserver.com/handle/123456789/17

This is the request that is sent to DSpace. DSpace will then create an empty DRI document, and send it to the first aspect. The aspect itself will determine if it needs to add content to the DRI document. The aspect named *Eperson* deals with the user. In this example, it will add data about the user to the DRI document. The aspect that deals with items, *Artifactbrowser*, will add data about the item requested. Some aspects will not add data, as they may be unrelated to the current activity.

When every aspect have made their changes to the DRI document, a theme is applied to the completed DRI, which transforms it to HTML ready to be presented to the user.

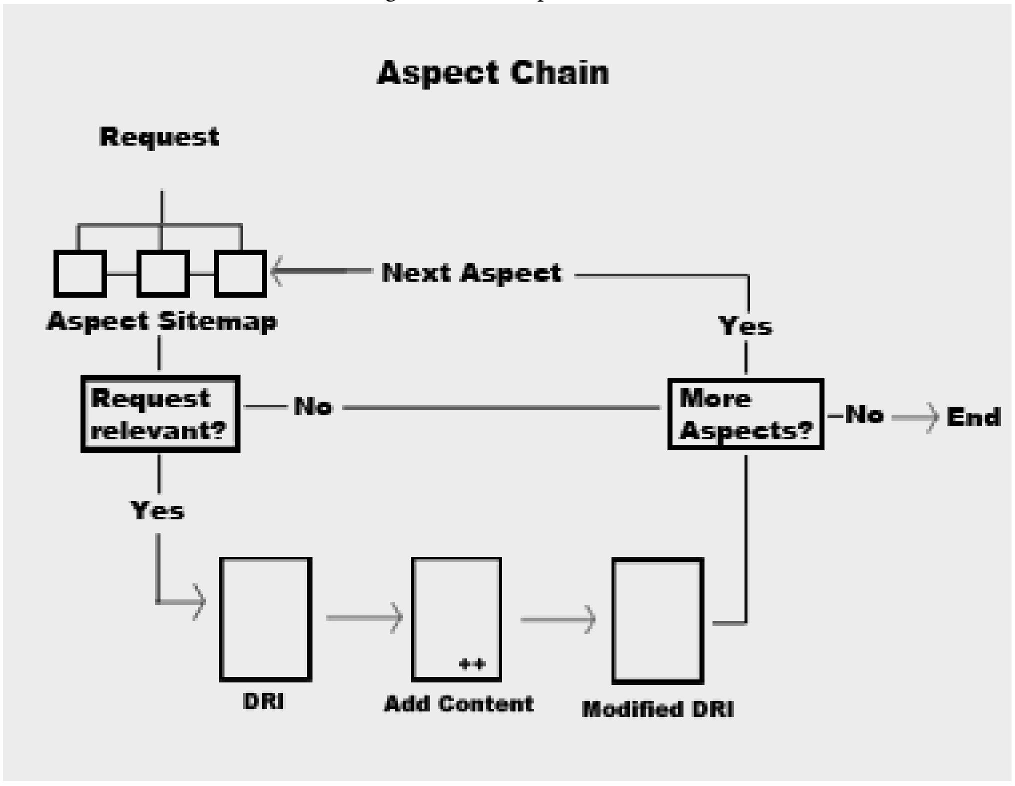
As mentioned above, Cocoon and, by extension Dspace, practices a strong sense of separation of concerns. This means that the content processing and presentation are handled by two very different parts, Aspects and Themes respectively.

3.1.3 Aspects

Aspects are used to change, add or remove content from the DRI document. A typical pipeline in DSpace starts with a request (a URL). A DRI document is then created, which gets passed to all active aspects in order.

The Aspects are chained together in an aspect chain, shown in Figure 6. Each time a user requests a page, the DRI document will be sent through the aspect chain, and each Aspect can make its own changes to the document. This assures that if an Aspect is removed no dead links will remain as all links are created on each page by the aspect itself. If an aspect introduces a new navigational menu entry, as well as a link to this new page, it is the aspect itself that handles both adding the new link to the menu and to the items. If the aspect is removed at a later date the aspect chain will skip the aspect and the links will never appear.

Figure 6: The Aspect Chain



Each aspect determines by itself if the request is relevant. If it is, the aspect is allowed to make its changes to the DRI document. This repeats itself for every aspect, where each subsequent aspect uses the DRI generated by the previous aspect. When all the aspects have made their changes, the DRI is passed to the Theme. If a user browses to the previously mentioned URL http://dspaceserver.com/handle/123456789/17, the sitemap of each aspect determines if this is a url the aspect needs to add content to. For an aspect that handles new submissions, this url will not be relevant, as it points to an already existing item.

3.1.4 The Sitemap

At the heart of both Cocoon and Manakin is a file called a sitemap. Every aspect uses its own sitemap which specifies how the aspect should behave. Sitemaps contain three major component pipelines: Generators, Transformers and Serializers.

Generators

Generators are typically the start of a pipeline, which generates the information stream that is to be used by other components, e.g. from an HTTP request or static XML file. When a generator is passed a DRI document, it generates an information stream which the transformers are able to manipulate.

Transformers

Transformers process, add or remove content from the information stream. This includes adding content to the pages the users browse. A *Transformer* is usually written like this:

```
<map:transform type="SomeTransformer"/>
```

Here, SomeTransformer references a Java-file which contains the code to be executed. Figure 7 shows a snippet of how such a Java file looks. The part shown is used to set the heading of a division to the same as the item, or its identifier if the name is omitted.

Figure 7: Sample Transformer Java Code

```
Division division = body.addDivision("item-view","primary");
String title = getItemTitle(item);
if (title != null)
    division.setHead(title);
else
    division.setHead(item.getHandle());
```

Serializers

Serializers serialize the content into HTML, XML or DRI for further processing by subsequent sitemaps.

The structure of a sitemap is shown in figure 8.

In addition to the three major components shown in figure 8, there are a number of other components that are important.

Matcher

A *Matcher* matches a request pattern to a resource. A request pattern is the url the user requests from the server. If a *Matcher* in a sitemap matches the request, the block inside the *Matcher* is executed. This is the regular way an aspect makes changes to a page, by putting a *Transformer* inside.

```
<map:match pattern="handle/*/*">
<!-- Matches the url http://siteroot/handle/*/* -->
        <map:transform type="SomeTransformer"/>
</map:match>
```

This particular *Matcher* will match a request sent by the user to view an item:

http://dspaceserver.com/handle/123456789/17

This is because * is used as a wildcard. Because this matches, the content inside the *Matcher* is executed.

Figure 8: Sitemap Structure

```
<map:sitemap>
        <map:components>
        <!-- Components that are to be used
        in the pipeline are specified here -->
        </map:components>

        <map:pipelines>
                <map:pipeline>
                <!-- Most aspects only need one pipeline -->
                  <map:generator />
                  <map:transformer />
                  <map:serializer />

                </map:pipeline>
        </map:pipelines>

</map:sitemap>
```

Action

Unlike *Transformers*, *Actions* do not modify the information stream generated from a DRI document in any way, but can be used to modify runtime parameters and process logic that does not need to be displayed. Logging can be done using *Actions*, as displaying data is not needed when logging user activity. *Actions* are usually paired with a *Matcher*.

```
<map:match pattern="handle/*/*">
   <map:act type="SomeAction" />
   <map:transform type="SomeTransformer"/>
</map:match>
```

Much like the *Transformer*, the *Action* is also usually a reference to a Java file with the appropriate code.

In Cocoon the components of the sitemap treat SAX-events[44], but in Manakin they are used to manipulate the DRI.

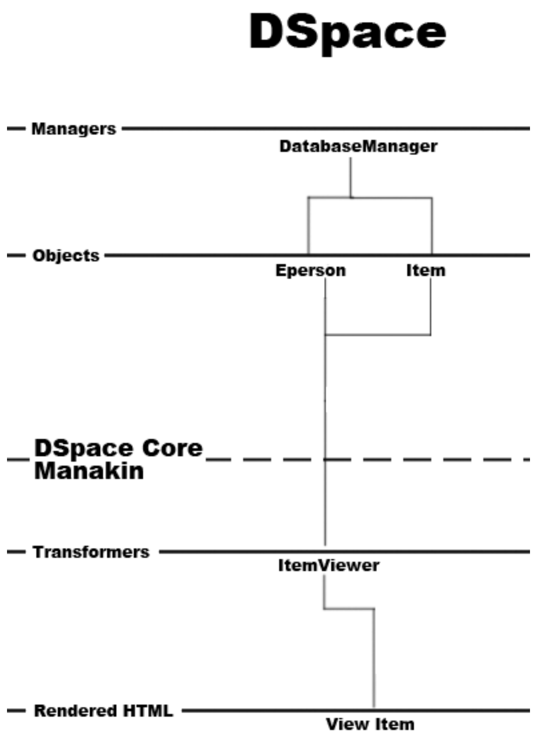
3.1.5 Architecture

DSpace is made up of objects and managers, with which the aforementioned components—transformers, matchers and actions—interact with. Figure 9 shows the relationship between these components.

The resulting HTML of a page which is used for viewing items in a Collection uses a transformer to add information. This transformer is called ItemViewer and handles every task related to showing this particular page. Both of these steps are specific to Manakin. The transformer creates objects based on the needed parts, in this case EPerson and Item. There are also objects for e.g. Communities and Collections. These objects are using a database manager to fetch information about itself. This manager provides an abstraction layer between the database and the objects attempting to access it. There are managers for most tasks, for example AccountManager and AuthorizeManager, which manages user accounts and authorizing users for access to items respectively.

The DSpace components are divided into three layers, each managing a separate set

Figure 9: DSpace Relationship Chart



of tasks. Figure 10, from [41] shows these layers and their components.

The storage layer

This is responsible for the physical storage of objects and metadata. This includes the database manager, named DatabaseManager.

The business logic layer

This manages most of the logic in DSpace. AuthorizeManager and AccountManager are included in the business logic layer.

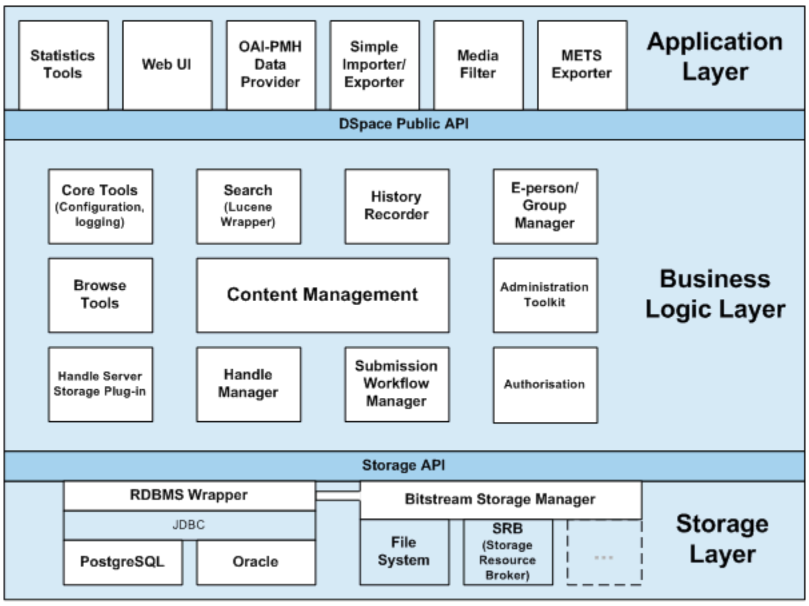
The application layer

This is comprised of components whose task it is to handle communication with the outside world. The web user interface is a part of the application layer.

3.1.6 Themes

Themes are used to modify the look and feel of a Dspace installation. A theme contains a sitemap, a folder for static content, like images, supporting files, e.g. CSS-stylesheets and Javascript-files, as well as XSL-files for transforming content. A Theme's task can be split into four steps. When a theme receives the DRI-document from the aspect chain, it 1) adds metadata concerning the theme, e.g. its name and location. 2) transforms it to XHTML, usually using a common DRI-to-XHTML.xsl-file supplied by DSpace. 3) localizes the page by inserting the correct language strings. When a transformer needs to output text on the screen, instead of hardcoding the strings into the Java-code they are referenced from an XML-document which can contain the same strings in a number

Figure 10: DSpace Architecture[41]



of languages. This step selects the correct strings to insert in place of the references generated in the DRI. 4) sends the complete page to the user's browser. A useful feature of themes is that different themes can be applied to different parts of the repository. Every *Community* or *Collection* can have their own theme applied, and this cascades downwards so individual items in that particular *Collection* will have the same theme. Different themes can also use reuse elements from other themes. This is called compound themes, where one theme is the main theme and several sub-themes use the main theme's resources, and override them as needed.

4 System Creation

The system created in this thesis has two parts. The first part is the implementation of the tagging- and rating interface, and the second is the development of the algorithm to relate objects based on the tags applied by students using the first part.

4.1 Tagging and Rating

This part adds the ability for users to tag objects present in DSpace, as well as rating them with a number between one and five.

The system was made as a DSpace aspect for Manakin. The aspect is composed of a sitemap which controls what the aspect does, as well as a number of Java-classes which contain the code. It adds a tagging- and rating interface to every item entry, as well as the possibility of browsing tags and items with a particular tag. The relevant actions a user can carry out are also logged, to be accessible for reviewing and analyzing.

4.1.1 Requirements

Some requirements had to be fulfilled when creating this system. A number of elements that needed to be considered for this system to be usable were identified:

- Tagging
  - The tagging interface
  - Anonymous tagging
  - Public/private tagging
  - Administration of tags
- Rating
  - Anonymous rating
  - Rating range
- Logging
  - Important events

Participation

An important facet of the aspect is how to get students to participate. We hypothesize that the participation rate is governed in part by the accessibility threshold. This threshold is based on two parts, which can be defined as follows:

Access The extent to which a system is removed from the most direct route to the content.

Usage The amount of work a user must do from having discovered the system to a tag is applied.

As DSpace is not in use at GUC during this study, users will have to access the system through an indirect route. All learning objects are currently available only through Fronter, so the most direct route by which we can reach the students is links on the course's Fronter-page. *Access* is now relevant, as the route students have to use to access the system is not the most direct route. Students will have to deviate from their regular workflow to be able to tag the lectures. This is mainly a problem during this thesis. An institution using DSpace as their primary method of storing learning objects will already have this as their most direct route to the content. Methods for minimizing the impact of *Access* is discussed in Chapter 5.

Usage encompasses two parts. The effort a user must put in when first discovering the site, to his being able to use the site's full functionality, i.e. the registration process. The second part is the effort expended continuously when tagging. The second part is important to simplify. A cumbersome registration process—although not desirable—can be forgiven, as it is a one-time effort and need not be repeated. If the process of tagging objects is perceived as more trouble than it is worth, participation rates will most likely drop.

Usage can be directly influenced in the design process, and decisions related to this is discussed in the relevant sections in this chapter.

LDAP Integration

LDAP is used by GUC to let professors and students log in to all services provided by the school with a single user name and password. Integrating DSpace with LDAP aids the project by providing a number of benefits. It further closes the gap between this experimental system and the regular services, making it feel more as a part of GUC. It also helps by removing the registration process in DSpace, as students already have an account.

DSpace supports LDAP integration, so integrating them posed no significant problems. As most of DSpace, the log in-method is modular as well, so the process of integrating LDAP into DSpace consisted of replacing the regular "username and password"-module with a pre-existing LDAP-module, and specifying the parameters of the school's LDAP server in a config-file.

Tagging

When creating the tagging system, some choices regarding the implementation were made.

The tagging interface

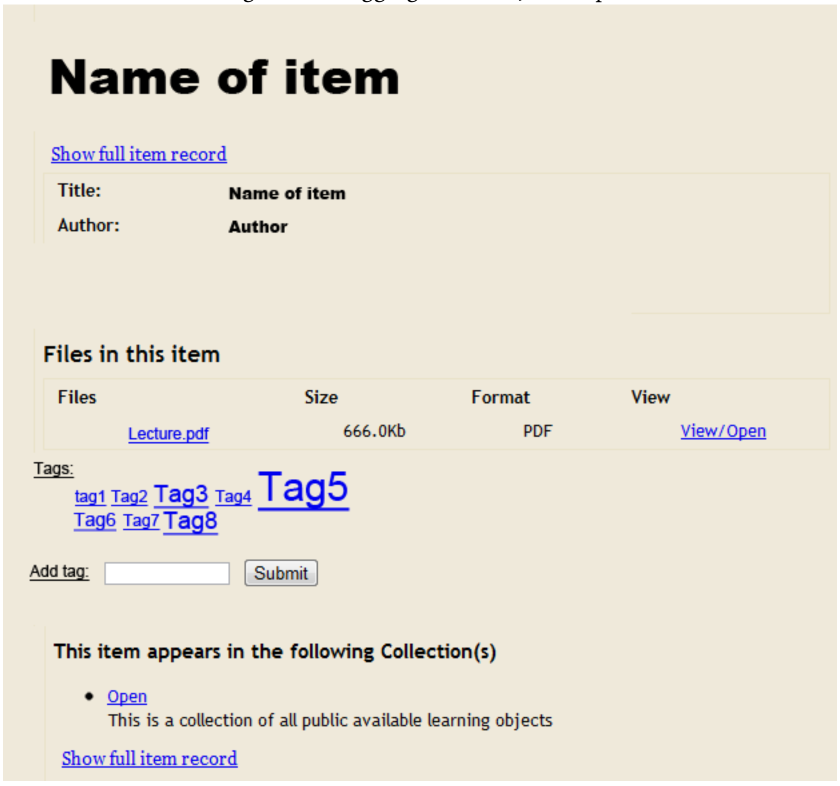
This is the most important part, as it is the main interface the users will interact with.

Figure 11 shows an early design. This design included positioning the tag-part closer to the link to the item, as well as using a tag cloud to visualize the popularity of certain tags in an efficient and user-friendly manner.

Unfortunately, due to limitations in DSpace, we were forced to rethink the design. As mentioned in Section 3.1, DSpace relies heavily on separation of concerns. This means that the presentation part is left to the theme, and there are few ways to influence how text is represented on the aspect level. Tag Clouds had to be abandoned because of the difficulty in changing the font size of single words in DSpace.

Additionally, since the aspects add data in succession, there are no ways to insert the tag section between two already existing sections, where both sections are created by a

Figure 11: Tagging Interface, Concept



Name of item

Show full item record

Title:	Name of item
Author:	Author

Files in this item

Files	Size	Format	View
Lecture.pdf	666.0Kb	PDF	View/Open

Tags:
tag1 Tag2 Tag3 Tag4 Tag5
Tag6 Tag7 Tag8

Add tag: Submit

This item appears in the following Collection(s)

- Open
  This is a collection of all public available learning objects

Show full item record

single aspect.

Figure 12 shows the final appearance of the tagging interface. Tag Clouds have been removed and replaced by a usage counter in parentheses.

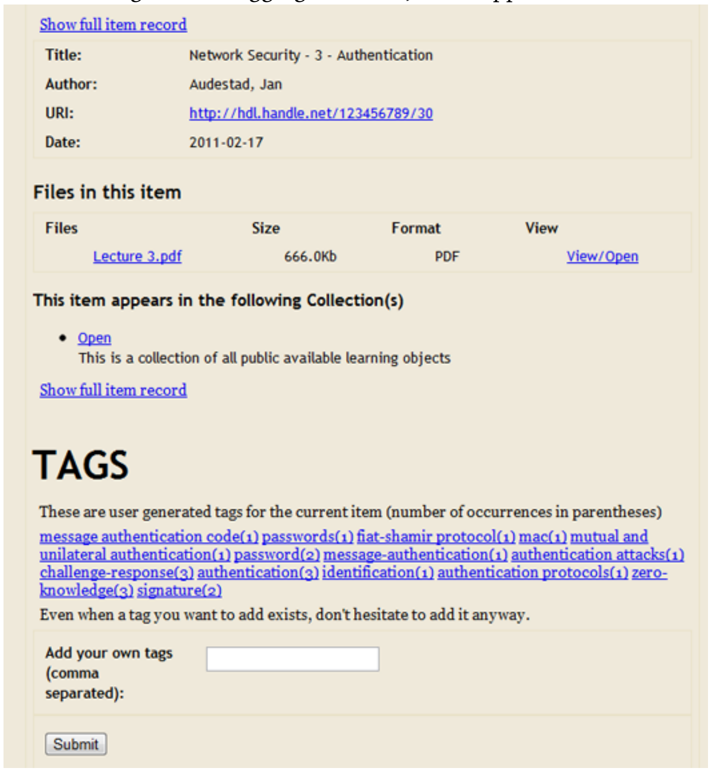
The main issues are visibility and position. The tagging interface needs to be visible, but it should not dominate the view. We especially need it to be visible during the course of the experiment as we need users to tag items. Putting the tagging-division above everything else was discarded quickly, as it is not the most important part of an item. Putting it on top would make it seem more important than the actual contents of the item. We decided to position it below the content DSpace adds by itself. To compensate for the position change, the font size of the header was increased. Its visibility is consequently increased, and thus its perceived importance.

Anonymous tagging

Allowing anonymous tagging brings some problems and some benefits with it. The problems can be summed up in one sentence: Lack of control. We will have a harder time controlling the behavior of the users, as we cannot restrict each user to only tag an item once with the same tag. This means that a malicious user can ”spam” the system by applying the same tag multiple times. The user will also have lack of control, as there is no way the user can delete a tag he applied while anonymous.

The pros of anonymous tagging is that the threshold to participate is much lower when one does not have to log in. This is important for this study, as lowering the threshold may cause a higher rate of participation. This reduces the impact of *Usage*,

Figure 12: Tagging Interface, Final Appearance



Show full item record

Title:	Network Security - 3 - Authentication
Author:	Audestad, Jan
URI:	http://hdl.handle.net/123456789/30
Date:	2011-02-17

Files in this item

Files	Size	Format	View
Lecture 3.pdf	666.0Kb	PDF	View/Open

This item appears in the following Collection(s)

- Open
  This is a collection of all public available learning objects

Show full item record

TAGS

These are user generated tags for the current item (number of occurrences in parentheses)

message authentication code(1) passwords(1) fiat-shamir protocol(1) mac(1) mutual and unilateral authentication(1) password(2) message-authentication(1) authentication attacks(1) challenge-response(3) authentication(3) identification(1) authentication protocols(1) zero-knowledge(3) signature(2)

Even when a tag you want to add exists, don't hesitate to add it anyway.

Add your own tags (comma separated):

Submit

one of the parts governing participation mentioned in 4.1.1, to its bare minimum. Anonymous tagging makes the system usable at once when users first discover it, allowing them to skip both the registration process and the login.

Public/private tagging

Another issue was to decide how accessible the tags should be. Should everyone see their own tags only, or should all tags be visible publicly? We chose to let all tags be public, to allow tagging for both social and personal reasons, according to the tagging motivations described in Figure 2 in Chapter 2.

Tag administration

Administrators should be able to review and delete unwanted tags. This is especially important when dealing with anonymous tagging. Every user should also be able to delete their own tags. Deleted tags should also be preserved, as they can provide insight into the behavior of students.

Rating

The idea behind providing the user the ability to rate is that the content creators can receive immediate feedback on how well received the object is. This kind of continuous feedback during a semester can be valuable by making the professor able to assess with greater precision how their class responds to the teaching material. There are some considerations to keep in mind when introducing rating in the system:

Publicly available average rating

Choosing to make the average rating public will have some implications. Students can be influenced by watching the average rating when they are rating the objects for themselves. On the other hand, the rating exists partly to give students an indication of how his fellow students perceive the quality of the object. Since this thesis is interested in how students behave in a real world situation, we chose to make the average rating publicly available.

Anonymous rating

As with tagging, anonymous rating has pros and cons, which is very similar to the pros and cons of anonymous tagging. A user can rate an object numerous times, to increase or decrease the average rating. It will on the other hand lower the threshold to rate, and may allow the users to rate the objects more truthfully, when they know it is anonymous.

Rating range

Selecting too large a range will end up overwhelming the user, which hurts the participation rate, while too narrow a range will hinder diversity in the ratings, and will ultimately have a negative effect on the data available for analysis. We settled on a range closely mirroring the grading range in school, from one to five, resembling the five pass grades in higher education, A-E. The grades were converted to numerics, as we decided to make the average rating publicly available. Average ratings can have decimals, which is hard to represent with letters. Rating an object with a letter and seeing a numeric average would not feel consistent.

Logging

The cornerstone of the system is its ability to log how it is used.

Important events

The most significant decision regarding logging was which events should be deemed important enough to log. Every action associated with the system the user performs should be logged, but some may be possible to infer from other data. We decided to not log when users log in and out, as this can be approximated by analyzing their movements, e.g. entering items and tags, tagging items and rating them.

These actions were decided to be logged:

- A user visits an item's page
- A user tags an item
- A user deletes his tag
- A user clicks on a tag, bringing up a list of all items tagged with said tag
- A user rates an item

Additionally, every action has a timestamp, to determine when the action occurred.

4.1.2 Implementation

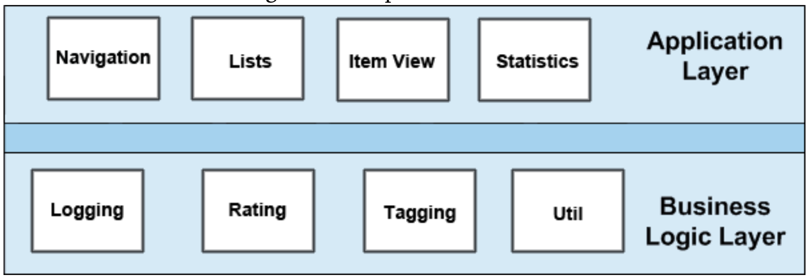
The Dspace Aspect was developed based on the requirements outlined in this chapter.

Architecture

The architecture of the aspect was designed to follow the convention of core DSpace components. They are divided into the application layer, which handles the presentation of data, and the business logic layer, which handles all logic. DSpace employs a set of components called managers to act as a wrapper for tasks associated with a common topic. The AuthorizeManager handles all tasks connected with authorizing checks in DSpace. DSpace also uses objects representing entities, like a person or an item in a collection.

The created aspect's code is structured in the same manner, shown in Figure 13. The storage layer is not represented, as the aspect uses the existing Database manager in DSpace for all database transactions.

Figure 13: Aspect Architecture



The Aspect contains a sitemap to control the behavior, as well as a number of Java-classes. The Java-classes can be divided into five groups.

Transformers

These classes are responsible for how the components are displayed. This includes adding the tagging- and rating interfaces discussed above to an item's page, new navigational choices to the sidebar and additional pages which enables the user to browse aspect-specific pages.

Transformers are primarily used by the application layer.

Actions

These are responsible for functionality behind the scenes. Logging user activity is the most used action, and happens every time a user enters an item or a tag. Adding and deletion of tags are also actions. When a user adds or deletes a tag, they are redirected to a pseudo-URL, which starts an action and redirects them back to the previous page. This simulates an Ajax-approach. Using Ajax in the tagging process was considered, but decided against. Ajax relies on a fully rendered HTML-page to work, because it needs to manipulate the DOM in some way to show results (e.g. remove a tag). In Manakin, the HTML is only created at theme level—we work with DRI before that—so to implement Ajax would be on a per-theme-basis and would require too much time consuming work at this point in time.

Tag Clouds were removed because of the same issue. Without heavily modifying the core of DSpace there are no ways to change the presentation of an arbitrary word using the DSpace API. This means that the most used words cannot be styled on an aspect level.

Actions are in use at the business logic layer.

Objects

Objects are classes possible to instantiate as an object representing a specific entity. The aspect includes objects representing a tag and a log entry. Objects are used in the business logic layer.

Managers

Managers are classes containing all methods that handles functionality for a specific area. Managers in the aspect include a tag manager and a rating manager. Managers are used in the business logic layer.

Utility classes

Utility classes are used to provide functionality outside of the scope of the regular managers. Utility classes are used in the business logic layer.

4.1.3 Business Logic Layer

Tagging

Table 1 shows the Java classes associated with the tagging functionality of the business logic layer.

Tagging	
Class Name	Type
DeleteTag.java	Action
AddTags.java	Action
Tag.java	Object
TagManager.java	Manager

Table 1: Aspect Tagging Components

The tagging is implemented with actions to add and delete tags, plus a tag manager to manage the Tag-objects fetched from the Database.

Methods

The tag manager implements a public method to add tags from a CSV-string, and a number of methods to fetch Tag-objects based on different criteria, including fetching all tags from an item, fetching a tag based on its ID and fetching all tags applied by a specific person. The tag manager utilizes core DSpace components, including EPerson and Item for fetching tags based on users and items, as well as extensive use of the database manager to communicate with the database.

The tag object includes a method to populate the object based on an ID, and a method to retrieve a list of all items the tag has been applied to.

The actions are using TagManager to add or delete tags.

Tag administration

A logged in user has some advantages compared to an anonymous user. An authenticated user can delete applied tags and see extended statistics about their own tagging behavior. Administrators should be able to delete every tag, to prevent abuse of the system. DSpace includes a way to easily check if a user is an administrator, via the AuthorizeManager-class in DSpace. The method deleteTag in TagManager uses AuthorizeManager to check whether the user is an administrator in DSpace before deleting the tag.

Logging

Table 2 shows the Java classes associated with the logging functionality of the business logic layer. Logging is the action which handles users viewing tags and items. It instantiates TagLog, which is the manager class used for all logging.

Logging	
Class Name	Type
Logging.java	Action
LogItem.java	Object
TagLog.java	Manager

Table 2: Aspect Logging Components

Methods

TagLog has one general purpose method to log events, called logEvent, which uses a numeric constant for each event. This makes it easy to include additional events to log without modifying the database. It also has methods to return every log entry—in the form of a LogItem-object—for each action. Some methods for extracting statistics are also included.

Anonymous tagging

To prevent spamming, a timestamp and IP address is included on every action. This means that numerous consecutive actions from the same IP within a short time frame can be tracked and removed.

Logging solution

The events are saved in a database table. There are both advantages and disadvantages to this approach. The overhead cost of logging to a database are large compared to text file logging. On the other hand, the flexibility offered by using SQL on the log largely outweighs the initial cost, especially on a relatively small scale project as this, with few users. SQL-queries are a fast and efficient way to extract only the useful data for a particular task.

DSpace ships with a logging utility, called Apache log4j. The benefits of using the already well-documented and widely used log4j is that less work has to be done up front, as it already is complete and functional. The logs produced may also be reusable by other applications as the format used is well-known and parsable by other software. As mentioned above, we need the logging to be routed to the database. Alas, configuring log4j to log to the database would force all logging, including much data which is unneeded, to be inserted in the database. This would cause a lot of clutter and unnecessary data inserted in the database, so a custom logging class was made to circumvent this issue.

Rating

Table 3 shows the Java classes associated with the rating functionality of the business logic layer.

As the tagging, the rating is implemented as an action to add or change the rating which uses the RatingManager to manage the ratings. There is no rate-object, as a rating

Rating	
Class Name	Type
Rate.java	Action
RatingManager.java	Manager

Table 3: Aspect Rating Components

is only a value, with no additional information.

Methods

RatingManager has a method to add a rating to an item, and methods for retrieving the average rating from an item as well as the rating from a single person.

Utility

Table 4 shows the Java classes associated with the utility functionality of the business logic layer.

Utility	
Class Name	Type
HiGHandleUtil.java	Utility

Table 4: Aspect Utility Components

The HiGHandleUtil-class is a variation of the handleUtil-class shipped with DSpace which fetches items, communities and collections based on the URL or the handle of the object. As the original handleUtil-class can only retrieve objects from the core of DSpace, an extension was needed to retrieve Tags based on the URL.

4.1.4 Application Layer

Navigation

Table 5 shows the Java classes associated with the navigation functionality of the application layer.

Navigation	
Class Name	Type
Navigation.java	Transformer

Table 5: Aspect Navigation Components

The transformer Navigation adds some new navigational choices to the sidebar using the transformer-API in DSpace. The new navigational choices are:

(1) A tag browser, where the user can browse all currently applied tags, and discover which items are tagged with the tag.

(2) An item browser, as a convenient shortcut to browse all items in DSpace, regardless of which Collection it belongs to.

(3) A logged in user gets access to a list of his own tags.

Lists

Table 6 shows the Java classes associated with the List functionality of the application layer.

Lists	
Class Name	Type
BrowseOwnTags.java	Transformer
BrowseSingleTag.java	Transformer
BrowseTags.java	Transformer
Overview.java	Transformer

Table 6: Aspect List Components

These classes are represented as separate pages, which are invoked by the Sitemap depending on the URL passed to it. Their function is to provide lists for the users.

BrowseOwnTags Creates a list of all tags applied by the user. Menu item only visible when logged in.

BrowseTags Browse a list of all tags applied across all items in DSpace.

BrowseSingleTag Creates a list of all items tagged with the chosen tag. This is not a navigational choice selectable from the sidebar, but are invoked when a user clicks on a tag either inside an item, or from the list of all tags.

Overview Browse a list of all items in DSpace. This is a convenience method during this study, when the number of documents is small.

Item View

Table 7 shows the Java classes associated with the Item View functionality of the application layer.

Item View	
Class Name	Type
ItemViewer.java	Transformer

Table 7: Aspect Item View Components

ItemViewer is the transformer which adds the tagging and rating interface to every item. This creates the interface seen in Figure 12.

Statistics

Table 8 shows the Java classes associated with the Statistics functionality of the application layer.

Statistics is a page not accessible to the general public, with statistics from the log presented in CSV-format to make it easy to import into spreadsheets or other software. Only some statistics are presented on this page, as much information is just as convenient to extract directly from the database with simple queries.

Item View	
Class Name	Type
Statistics.java	Transformer

Table 8: Aspect Statistics Components

Component Relationship

Figure 14 shows how the created aspect fits into the DSpace architecture. The parts above the dotted line are independent of Manakin, and can theoretically be used by the JSPUI too. The parts below the dotted line are Manakin-specific. Not every part of the aspect has been included in this figure, but the most significant parts are present. The lower part is the completed HTML the user sees. When viewing an item, both Manakin's own ItemViewer-transformer is used, as well as the custom ItemViewer, which adds tagging and rating functionality. This is determined by the sitemaps in each Aspect, using Matchers to match the URL. The regular ItemViewer creates items which accesses the DatabaseManager. The custom ItemViewer accesses the TagManager by creating Tag-objects, which in turn uses the DatabaseManager to fetch information about the tags, and returns them as Tag-objects.

When a user is viewing a list of all tags, no pre-existing transformers are used, as that particular page did not exist before the new aspect was introduced. Only the BrowseTags-transformer is used.

4.2 Retrieval Algorithm

The algorithm is based on a modified version of the HITS-algorithm, as mentioned in section 2.4.2. The modified HITS-algorithm identifies relevant documents when filtering by a tag. As this thesis is investigating how to retrieve relevant documents based on other documents, this algorithm had to be extended. The second part uses the results from the modified HITS to do this.

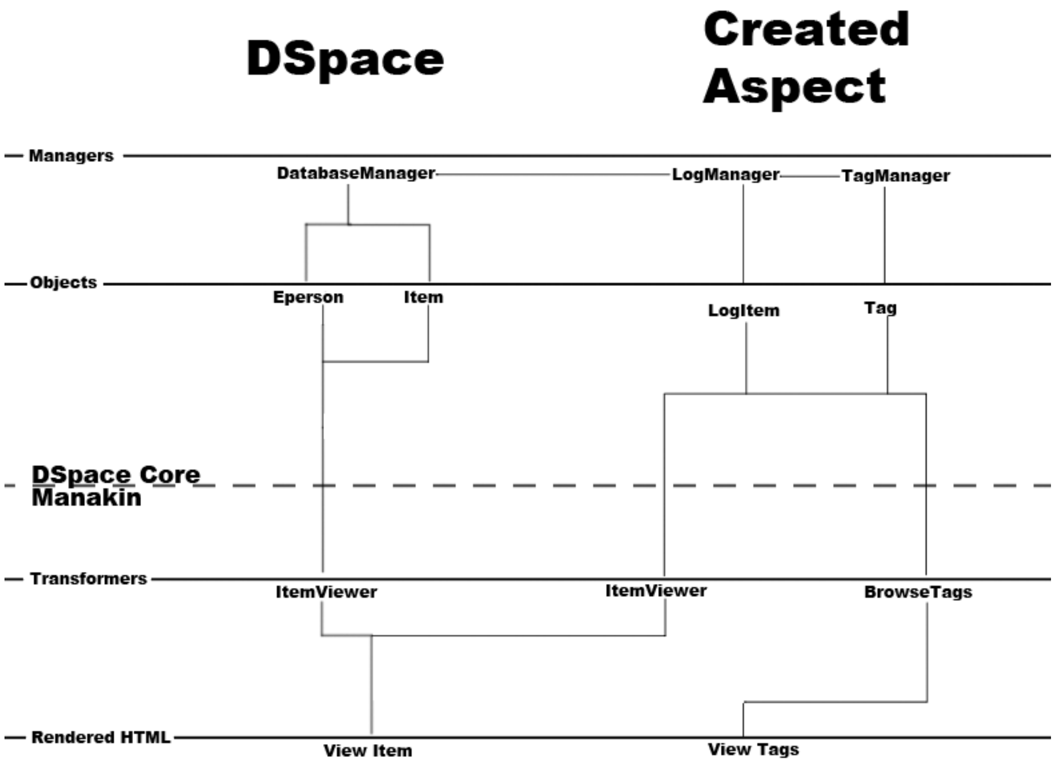
4.2.1 Modified HITS-Algorithm

The algorithm proposed by Wu et al. had to be modified further to better fit this thesis' intention. As anonymous tagging was allowed during the course of this research, we had to remove the user-aspect of the algorithm, thus reinventing the way hubs are used. Wu et al. qualifies experts (users to trust), based on their hub score. We are dealing with students that may be anonymous, so we used tags with high hub score to increase the weight of documents tagged with these "valuable" tags. This de-emphasizes the users and places more weight on the actual tags. We hypothesize that this will shift focus from who tagged it, to how it was tagged. "Valuable" tags are tags which are reused on a lot of documents. These tags have been deemed important by the users, and will thus be considered as more valuable than tags which are used less.

This was developed simultaneously with the students' tagging of their learning objects, and as a result, the number of documents tagged were low during the development of this algorithm. We were using a test set to test the algorithm.

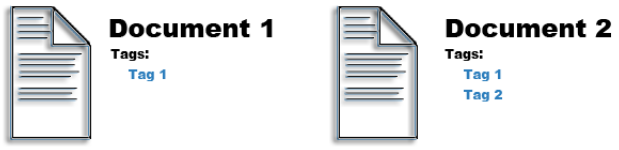
The algorithm starts with a Root Document (R_d). It then creates a set of documents (T), which are all documents tagged with the same tags as R_d. It then expands, to include

Figure 14: Aspect Components in DSpace Architecture



all documents tagged with the tags present in T, creating T_{ext}. A matrix A_{ij} is created based on T_{ext} and all tags. The matrix indicates if there is a link from node i to node j. All documents and tags are represented on both axes, because the matrix needs to be of equal dimensions for transposing purposes.

Figure 15: A Simple Set

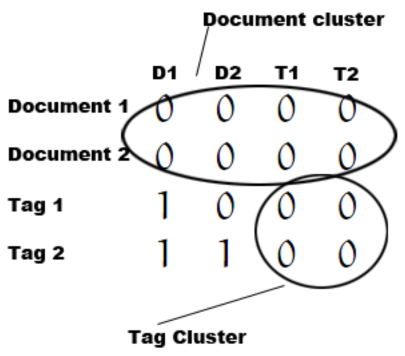


An example would be to apply the algorithm to a simple set containing two tags and two documents, shown in figure 15. The first document has one tag, and the second has two tags, with one common tag among them.

The matrix for the simple set is shown in figure 16. Take notice of the tag cluster and document cluster. These parts will always be zero. Because a tag cannot be used to tag another tag, the tags in the tag cluster will have no links to each other. Similarly, documents have no links going from them, as only tags can ”link” to documents.

The documents have no outgoing links, so the rows of all documents are always zero. The two last rows are tags, and, in this example, the first tag is used on document 1,

Figure 16: The Matrix for the Simple Set



and the second tag is used in both document 1 and 2. Because tags only have outgoing links, and documents only have incoming links, they are ordered naturally into hubs and authorities respectively.

We will then compute the authority weight in the same manner as the original HITS algorithm, explained in Section 2.4.2. This is done by transposing the matrix T_{ext}, then multiplying with a weight vector. This creates v, which shows the relevance of the documents in the set. Furthermore, calculating the new hub weight is done by multiplying the original matrix T_{ext} with v:

u shows which tags are deemed important. This can be used as a new weight, to further emphasize the documents containing the important tags. This example demonstrates how the modified HITS ranks authorities and hubs. Note that documents are always zero in the hub-list u, and tags are always zero in the authority-list v.

As this algorithm only applies to tags, we have developed a second part to retrieve documents based on other documents.

4.3 Extended Algorithm

When a user chooses to find related documents when browsing a particular document, we need to run the algorithm mentioned above on all tags applied to the document, and apply a similarity measurement to identify the highest ranked documents, based on their cumulative authority weight.

A challenge we encountered is how to assign weight to the tags correctly. Some tags describe the central concepts of a document better than others. Consider a document on computer security. This document has two tags: ”Malware” and ”Virus”. The first tag is addressing a broader concept of the document, while the latter is an example of a tag addressing a narrower part of the content. This introduces the concept of granularity, as mentioned in section 2.1, only this time it is the granularity of the tags used which is in focus. An important part of locating relevant resources is identifying other documents that address the same central concepts. By using tags to do this, we need to attempt to classify tags based on granularity, and give more weight to the tags of coarse granularity.

Developing such a system is outside of the scope of this thesis, but is further discussed in section 7.5.

The completed algorithm’s pseudocode is shown in figure 17.

Figure 17: Algorithm Pseudocode

```
Matrix authorityWeight;
For(Tag t in Document )
{
        documentCollection = Find all documents tagged with t;
        tagCollection += Find all tags used on documentCollection;

        Matrix matrix = createMatrix(documentCollection, tagCollection);
        Matrix matrixT = matrix.transpose();
        for(i iterations)
        {
                authorityWeight += MatrixT*hubWeight;
                hubWeight = matrix*authorityWeight;
        }

}
```

The resulting matrix is an n-by-1 matrix, with the values representing relative relevance to the parent document. The higher valued documents are more relevant. The last step is to apply a cutoff for values under a certain score.

Modifying the weight matrix changes which documents will get a higher weight. Modifying the weight of the tags contained in the Root Document to be greater than the remaining documents in T_{ext} will skew the results in the original batch of documents' favor, as they are assumed to be more relevant.

A cutoff value will remove all documents with less than a given score. This cuts down on the number of less relevant documents that are returned.

5 Experiment Planning

This study needed the participation of a group of students. Initially, we had planned on using two classes to participate in the project. Due to unforeseen difficulties, the students were not available when needed. A third class, following a course in network security on the master's level, joined the project.

The system that was created was made available for the students at `http://matuku.hig.no`. All learning objects related to the course were added as entries. Three types of learning objects were used in this course: Chapters from a book in pdf-format, audio recordings of each lecture, as well as photographs of the blackboard. These learning objects were added to DSpace, following this naming convention:

Course - Lecture no. - Type of Learning object

Type is either the name of the lecture, the name of the lecture with "(audio)" appended or "Blackboard" for the blackboard images.

5.1 Informing Students

The project was presented to the class during the second lecture of the course. The class consisted of 29 students, where a part of the group were following the course as distance students. To inform the distance students of the experiment, a web site (located at `http://www.stud.hig.no/~091213/`) was created with information on what the experiment entailed, which was spread by an update to the fronter room and sent by mail to all the participants in the course. With help from a student in the course, information about the website and experiment was spread via a student created Skype-group. Both the content of the web page, and the sent e-mail can be found in appendix A.

The day before the presentation all test data and logs were wiped, so as not to interfere with and get mixed up with the real data.

5.2 Interviews

During the course of the experiment, it became apparent that user participation was lower than expected. This could result in difficulties when extracting meaningful information from the data, as the data set was small.

This meant an approach based on quantitative data had to be extended by the inclusion of interviews with the participants, which would enable us to harvest qualitative data.

Some questions were prepared, which served primarily as guidelines for the conversation. The questions were meant to direct the interviewee in a certain direction, and follow-up questions were asked to extract additional information when necessary.

The set of base questions asked is as follows:

When did you usually tag?

The goal of this question is to understand the process people go through when choosing a time for tagging. We know when they tagged by going through the logs, but not why they chose that specific time.

Why did you choose the tags you chose?

The goal of this question is to try to understand what factors are present when students are choosing tags for course material. We know of, from the study conducted by M. Ames and M. Naaman mentioned in Section 2.2, users' incentives for tagging photos. This question will hopefully show if the incentives for students tagging their learning objects mirror those of other, currently existing, systems.

How much time did you spend tagging?

This question should reveal some information about the users' behavior while tagging.

Any reason you only tagged lecture notes, and not blackboard images and audio?

This question concerns the fact that no one tagged blackboard images and audio. We want to find out why the students were hesitant about tagging these objects.

Any comments about your experiences using tags in this way?

This question will give the interviewee room to voice concerns or ideas he might have gotten while using this system.

Any comments about your experiences working with the system (DSpace)?

Similar to the above question, this question should give us feedback on the technical aspects of the solution used.

Would you choose to use such a system if it was included as a standard part of the courses?

While the previous questions may make the interviewee think about the system in a "perfect world", this question is designed to make the interviewee contemplate how he would have made the system appealing to use, based on it being used in a real life setting.

These interviews were conducted during the middle of May, after the last lecture in the course, but before the exam, with four participants. All interviews were recorded, and later transcribed.

5.3 Retrieval Algorithm

5.3.1 Test Set

As the amount of tags applied by the students in this study were relatively small, a substitute set was utilized to get satisfying results when testing the algorithm. The set used to test is the ImageCLEF dataset[45], available from the imageCLEF website[46]. This is a large collection of still images, complete with annotations. The collection is primarily used in image retrieval research, but made available free of charge to the general public, without copyright restrictions. All images contained in figures regarding the retrieval algorithm are taken from this set. The content of the images range from pictures of buildings, through nature photographs, to photographs of people. Figure 18 shows the variation of motifs across the collection.

About 1400 images were used as a calibration set, roughly 7% of the complete set. The annotations are a full text description of the contents of each image. Every image is described in the same manner.

Most common stop words were removed from the annotations, although no stemming was done.

Figure 18: Composite Image Showing Variety of Motifs

5.3.2 Comparing with a Plain Algorithm

To assess its usability, we compared it with a plain algorithm based on Vector Space Scoring, mentioned in section 2.4.1. Each document is assigned a vector of weights based on the tags they contain in a similar manner to how the original vector space scoring uses terms. By viewing each document, as well as the source document as such a vector, we can compute the distance between the source document and each additional document using cosine similarity measurement. The source document is viewed as the query vector $\overrightarrow{Q}$. This distance indicates how closely related the document is to the original document.

The weight of each tag is larger if a larger number of people have used the tag on the same document. The tag is assigned lower weight if it is common among a large number of the documents, by multiplying the tag frequency, `tf` (originally term frequency) with the inverse document frequency, `idf`.

6 Results and Analysis

This thesis has data associated with the students and their behavior when using a tagging system for learning purposes to analyze. This is, as mentioned in Section 4.1.3, stored in a database. This is a powerful tool which makes us able to extract highly relevant data in an efficient manner.

The second part to analyze is the performance of the created algorithm based in part on HITS. The algorithm's precision was measured by a human user comparing the results with the results gathered from a simple retrieval algorithm.

We will also identify challenges encountered when using DSpace to implement social technologies.

6.1 Tagging Behavior

The tagging activity was generally low throughout the experiment. With a few exceptions, all tags described the actual content of each object. Popular tags include ”asymmetric”, ”kerberos” and ”authentication”, all of which are topics covered in the objects they were applied to.

Other non-descriptive tags that were used were ”basic lecture” and ”introduction”. It is worth noting that some students attempted an SQL-injection. This is discussed further in section 6.1.3.

6.1.1 Data

All numbers presented have been cleaned up by removing logged crawlers and our own movements on the site.

People

Table 9 shows statistics harvested from the log about user activity. Of the 29 students participating, a maximum of eight participants have tagged objects. There are eight unique IP addresses that have been used to tag learning objects. Two of these IP addresses belongs to the same person. Both IP addresses have only been used to tag by a logged in user. One of the IP addresses can also be disregarded as a legitimate user, as the only tag applied by this IP is the SQL-injection attempt mentioned above. An action is defined as

# of students	29
# of unique IPs having accessed DSpace	58
# of unique IPs having tagged	8
# of logged in users having tagged	3
Suspected number of active users	6
Number of IPs with a single visit	22
IPs with more than 7 actions	14

Table 9: Logged User Statistics

one of the following:

Delete a tag	Access an object
Access a tag	Rate an object

Objects

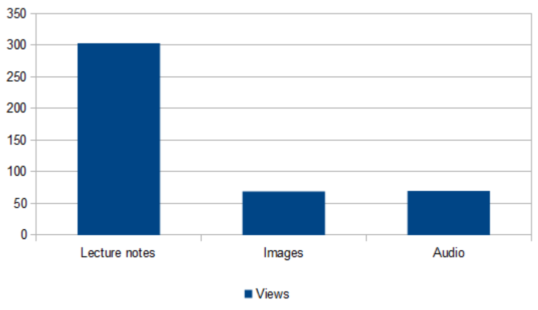
Table 10 shows statistics about objects available in the system. Even though the set of learning objects contained lecture notes, audio recordings and blackboard pictures, only the lecture notes were tagged by students, with 14.2 being the average number of tags applied to each lecture note. The blackboard images from Lecture 8 and 10 was not made available to us until after the analysis was finished, and as such is not represented in the set of learning objects.

# of Learning Objects	28
# of Lecture notes	10
# of Audio recordings	10
# of Blackboard pictures	8
Unique tags applied	119
Tags applied overall	142
Average tags applied, Lecture notes	14.2
Average tags applied, Audio	0
Average tags applied, Blackboard	0

Table 10: Object Statistics

As we can see from the Table 10, 23 of the tags are reused, but only three tags are reused on other documents ("mac", "identification" and "authentication"). The rest are tags applied by other users on the same documents. This small reuse of tags across documents are investigated, through the in-depth interviews conducted, in Section 6.2. Some tags are reused, but not recognized as such, due to differences in spelling. Two tags, written 'diffie-hellmann protocol" and "diffie hellmann protocol" respectively, is obviously referencing the same concept, but as folksonomies have no structure between tags and no synonym control, these tags are not recognized as identical by the system. Figure 19

Figure 19: Aggregate views Divided by Category



shows the views by category, where the lecture notes, audio recordings and blackboard images are the categories. The difference between these categories is that lecture notes were available before the lecture, while recordings and images were uploaded after the lectures, usually one day later. The lecture notes had a much higher amount of views than images and audio combined.

6.1.2 Behavior of a Non-Tagger

By studying the actions the users took in a single session, we are able to outline their behavior.

Most of the non-tagger IP addresses were one-time visitors. Often the visitors were checking out the lecture notes for the next lecture, or a previous audio recording. Some users went through every lecture, possibly to save them locally, which would explain why they never returned. One non-tagger did rate some of the learning objects. When users entered a tag, they always returned to the previous object.

Most users that accessed and stayed on the site for an extended amount of time were acting like casual browsers, seemingly with no clear goal in mind.

6.1.3 Behavior of a Tagger

Not surprisingly, a majority of the most active IP addresses are the same IP addresses that tagged learning objects. The difference between non-taggers and taggers lie in that the latter group actually tagged the learning objects. The browsing patterns of those tagging closely resemble how non-taggers behave, browsing different items and going back and forth between them.

The logs show three different kinds of behavior when tagging.

The first is a systematic approach, where the tagger clearly has assembled a list of tags prior to accessing the site, because he utilizes the ability to apply multiple tags at once, using a comma separated list. A typical usage scenario is to access DSpace, locate the relevant lecture, spend a couple of minutes filling out the tagging form, and then rating the item.

The next user approach is one who most likely tags while he is reading. His tags are applied one at a time, with a space of everything from a couple of minutes to half an hour between each tag.

The last user type is accessing an item, applying some tags, and then browsing to other items before coming back and applying more tags.

Those who logged in usually did so before they started tagging. If a logged in user made a typographical error, he deleted the erroneous tag, and applied a correct version.

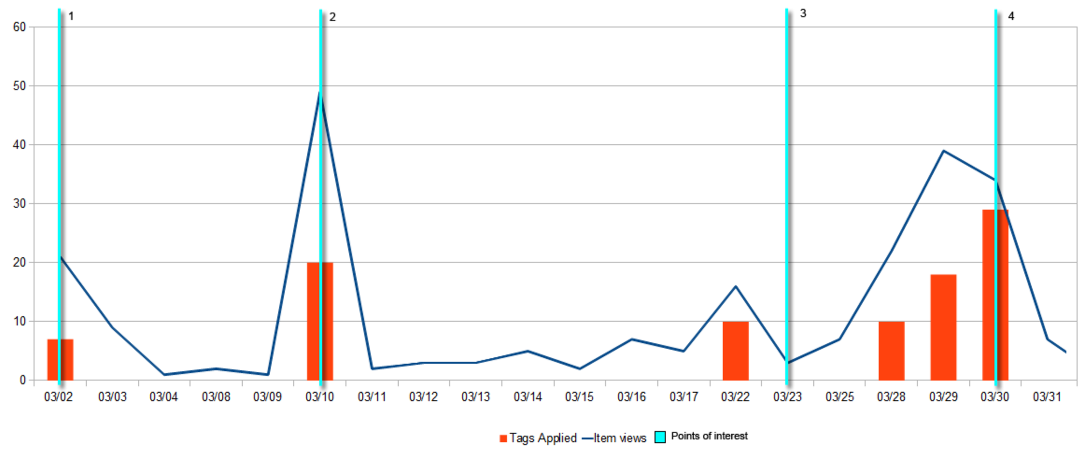
As mentioned above, some students attempted an SQL-injection. This was attempted by applying the tag ”\';select * from *”. The injection in itself was not destructive, but if successful could be used to create harmful injections. Taking into consideration that the course is teaching network security, one can conclude that this is most likely an attempt with no malicious intent. Since the created aspect is using the database manager supplied with DSpace, such attempts are pre-empted.

6.1.4 Comparison of Views and Tagging Activity

As touched upon in Chapter 4, the participation rate is governed partly by how accessible the system is.

We were interested in figuring out if students did not tag as much because the system was too inconvenient to locate compared to accessing the learning objects directly from Fronter, or if people would not tag for other reasons. By comparing the activity in the system with the frequency of tagging, we can get an indication of this. Figure 20 shows a temporal comparison of number of views and number of tags during the month of March. The thin vertical bars are points of interest, where activity spikes can be found. POI1 is the date we presented the system for the students, and the date of a lecture. Unsurpris-

Figure 20: Item Views Versus Tags Applied



ingly, the activity spiked the same day and fell rapidly when the weekend approached.

The second POI is the day the reminder e-mail was sent. Both tagging activity and views increased, largely due to the mail reaching the distance students too, whereas the presentation was only heard by the students on campus.

The lectures had an intermission during this and the next week, so there are no activity on Wednesday 9th and 16th (the day the lectures takes place).

In POI3, we see a shift in behavior. All visits and tagging are done the day before the lecture. This is expected behavior, and indicates that students tag while reading the material.

The increased activity the 28th to 30th is interesting. The students following the course had a delivery deadline in another course just prior to that date, which may indicate that when the students have less work on their mind, secondary activities get more attention.

Even though the class only consists of 29 students, the activity spike in POI2 show almost 50 views. This is due to the fact that the view values are aggregates of the views all documents received on a particular day. This means that a user viewing two items is counted twice.

Figure 21: Site Accesses Filtered by IP

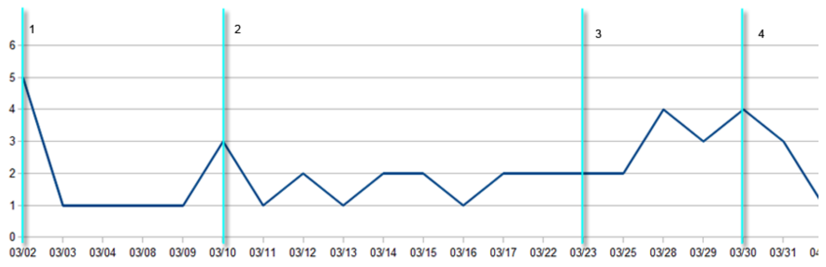
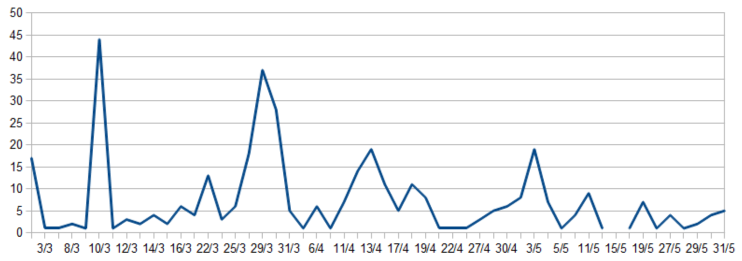


Figure 21 shows the item views from the same period of time filtered by IP. IPs are only counted once every day, so this is a much closer approximation to unique visitors each day. This figure evens out the large spikes shown in Figure 20. The large spikes serves as an indication that users are browsing through multiple items when first accessing DSpace.

The visits to DSpace decreased as the exam period approached, as shown in Figure 22.

Figure 22: Site Accesses, March–May



6.1.5 Rating

The rating was not much used. Only 15 ratings were registered, where half of the ratings were made by a single person. This data is insufficient to draw any conclusions regarding rating behavior.

6.1.6 Indications Based on the Data

The small amount of data collected in this study may influence the precision of the results. Any conclusions drawn from this can only be indications.

The usage patterns show some indications of possible use in further studies and development of a tagging system.

As we can see from Figure 21, the actual usage of DSpace is low, with a maximum of five unique visitors in a day. This is an indication that the problem of low participation is related to the accessibility of DSpace, as opposed to coming from problems with the actual system. This is a subject explored in greater depth in Section 6.2.

The activity spike the 28th, probably caused by a dip in workload in other areas, paired with a spike in activity after the e-mail was sent, may indicate that the few students who chose to participate were not properly invested in tagging their learning objects, as tagging only happened when other activities were done or the students were reminded of the system's existence. This is also supported by the steady decrease in activity as the exams approached. In other words, tagging was not of high priority for the students. Reasons for this is elaborated on in Section 6.2.

The fact that none of the blackboard images and audio files were tagged is an indication of the different needs for on-campus students and distance students. The lecture notes were available prior to the lectures taking place. Images and audio were uploaded at a later date, and is of little use to the students who have already attended the lecture. Distance students could have more use for these types of learning objects, but we cannot know if they would tag them, as none of the distance students participated. We do know that the on-campus students did not tag or use the images and recordings as much, as shown by the number of views for each category in Figure 19, Section 6.1.1.

Knowing there are different approaches to tagging, this can help us create a tagging interface which caters to everyone. We need a way to add tags in batches, while simultaneously supporting single tag adding, as different students add tags in different ways.

The type of tags students are using are generally content descriptors. This type of metadata fits into the Subject-element in Dublin Core, General:Keyword-element in IEEE LOM, and to a lesser extent Description in Dublin Core, and General:Description in the IEEE LOM, corresponding to the results found by J. Fan's study, mentioned in Chapter 2. Knowledge that tags tend to fit into these categories can help in the development of an algorithm, as the type of metadata is known. Features from existing algorithms using this kind of metadata can be incorporated into an educational content retrieval algorithm using tags, for more efficient retrieval.

6.2 Interviews

As the participation rate during the study were low, perceived indications from the study were explored further using information gathered in qualitative interviews conducted with students from the class.

When did you usually tag?

The interviewees were unanimous regarding this question. Everyone tagged straight after having read the documents.

Why did you choose the tags you chose?

The four different motivations for tagging, mentioned in Section 2.2, were organization and communication for ourselves and others. It seems like most people use tags as an extension of their study habits, and mostly for their own benefit. Unlike the study by Ames and Naaman, in which they concluded that most tags are applied for social motivations, tagging in our study focused more on the topic and the benefit the tagger himself can reap by tagging. One student actually did not like that he could see other people's tags before he finished tagging his own, as he did not want to be influenced by other students' choices. Other liked seeing existing tags, and took it as confirmation that they weren't missing much of the key points when their tags coincided with the rest. Everyone handled each document as a separate entity, and did not think about reusing tags across multiple documents. This may come from the fact that most of the lecture notes' themes did not overlap much.

How much time did you spend tagging?

No one spent a long time contemplating which tags to use. The average time spent was about five minutes per learning object. The difference was in how long it took students to apply them. Some tagged on-the-fly while reading, while others compiled a list of tags after having finished reading the document.

Any reason you only tagged lecture notes, and not blackboard images and audio?

All of the students interviewed were on-campus students. No one had used the blackboard images and sound clips yet, as they felt that kind of learning objects were tailored towards the distance students. Most told us that they listened to the audio as part of the preparation for the exam.

Any comments about your experiences using tags in this way?

Some found it helpful, as it served as an extra tool for them to use while studying. It worked especially well for parts with little space devoted to them, parts that were still important. Tagging these concepts helped emphasize their importance, and thus make them easier to remember. Others were more ambivalent, as they didn't understand how tagging would benefit them.

Getting used to tagging lectures on the bachelor level was brought up as a way of priming students to utilize tagging. Introducing a new concept as late as the master's level, and make it work successfully, is more difficult than introducing it to first year students. The change in study routines was a hurdle to overcome, as the students were used to study in their own manner.

Any comments about your experiences working with the system (DSpace)?

There were two major concerns related to the DSpace aspect. It would have been more effective if it was integrated with Fronter, their usual environment. As it is now, it is harder to reach, as it is a separate system. Those who tagged most had bookmarked the site, and used that bookmark to access DSpace. Others accessed DSpace from Fronter. Most agreed that it would be beneficial for the project to have the tagging more accessible.

None of the interviewees had any trouble using the system, but concerns regarding non-technologically minded users were voiced. The system uses comma separated values to enter multiple tags at once, which —for technical users who are used to working with CSV— pose no usability problem. For other students it may represent a hurdle compared to a more graphic approach, with icons for adding additional tags.

A suggestion made was the inclusion of the ability to tag parts of a learning object, instead of having to apply every tag to the whole learning object.

Would you choose to use such a system if it was included as a standard part of the courses?

Everyone agreed that if tagging was implemented for the entirety of courses, they would need to understand the benefits this brings to consider actively participating. If the benefits were not properly communicated, they would see no reason to continue tagging. They likened it to the course evaluations—which are accessible for every course—but few they knew bothered completing, because there was no perceived reward for doing it.

A possible reason for the low participation was brought up during one of the interviews. A lot of students did not work on the course on a regular basis, but resorted to start reading the course material close to the exam.

6.2.1 Indications Based on the Interviews

Some reasons for the low participation observed earlier were brought up during the interviews. Unfamiliarity with tagging as a concept in an educational setting, and students not working consistently through the year were cited as reasons participation was low. Another concern was the lack of integration with familiar tools already in use.

This seems to agree with the indications observed in Section 6.1.6, that the participation does not stem from DSpace itself, but rather caused by the accessibility to the system, and student motivation.

The interviews confirms the assumption that students view blackboard images and audio recordings as tools primarily used by distance students, although some on-campus students use them as repetition material before the exam.

Suggestions for improving the tagging interface were made by the students. A different way to enter more than one tag was suggested, using a button to add more tag fields. This could be implemented, but using comma separated values or other efficient forms of batch tag input should not be removed, as our data shows that students are using it. A more interesting suggestion was the ability to tag parts of a learning object. This is explored further in Section 7.5.

The most important information gathered from the interviews was that none of the students had any reason to continue tagging if this became a standard part of the courses. Getting students to tag is dependent on the students realizing its usefulness to them. The prototype system developed in this study lacks functionality to make it feel usable for the students, as they cannot utilize their tags to accomplish any tasks. Students should be able to understand how they can benefit using such a system quickly.

6.3 Retrieval Algorithms

The retrieval algorithm tested is a modified version of the HITS-algorithm. For testing, we compared its performance to the performance of a simple vector space algorithm.

6.3.1 Assigning Weights

The algorithms were initially tested without tweaking the initial weighting. With a default weighting matrix comprised of only ones the HITS can display some peculiar behavior, as exemplified in Figure 23. The source image had 923 as its ID and, as displayed below, did not score highest on a test designed to find images with similar keywords as itself. Please note that the score is assigned on a per query basis and can only be used to

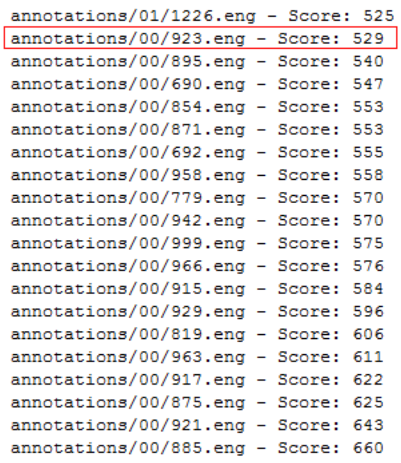

```
annotations/01/1226.eng - Score: 525
annotations/00/923.eng - Score: 529
annotations/00/895.eng - Score: 540
annotations/00/690.eng - Score: 547
annotations/00/854.eng - Score: 553
annotations/00/871.eng - Score: 553
annotations/00/692.eng - Score: 555
annotations/00/958.eng - Score: 558
annotations/00/779.eng - Score: 570
annotations/00/942.eng - Score: 570
annotations/00/999.eng - Score: 575
annotations/00/966.eng - Score: 576
annotations/00/915.eng - Score: 584
annotations/00/929.eng - Score: 596
annotations/00/819.eng - Score: 606
annotations/00/963.eng - Score: 611
annotations/00/917.eng - Score: 622
annotations/00/875.eng - Score: 625
annotations/00/921.eng - Score: 643
annotations/00/885.eng - Score: 660
```

Figure 23: Peculiar Behavior, HITS

rank documents internally in a query. The score document 923 received in this query can be a very low score in another query, where the values may peak at 2000.

This is due to the way HITS handles score. There are two parts which differ in importance. First we have the source document whose tags should be given the highest importance. Next is the documents in the extended set. These documents share some common tags with the source document. When a plain weight vector is utilized, both these parts are given equal importance by HITS. This means that the *number* of tags a document has will have a much greater impact than desired on the final ranking, as its score will increase equally for every tag applied to it when no specific weighting is applied.

A correct weighting is an important part of retrieving the correct data. We experimented with adding ”dynamic weighting” which divides the weighting into two groups corresponding to tags from the source document and other tags respectively. Overall, the results improved when this type of weighting was introduced. The source image, mentioned above with ID 923, is shown in figure 24. Without dynamic weighting the returned images were a mix of relevant and not relevant images. Figure 25 shows some of the results without dynamic weighting. All these results received a higher score than the source image, which scored as the 19th most relevant image.

Figure 24: Source Image 1

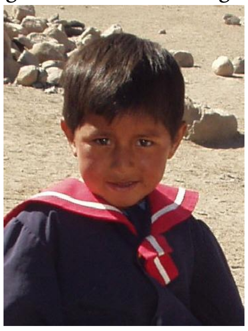
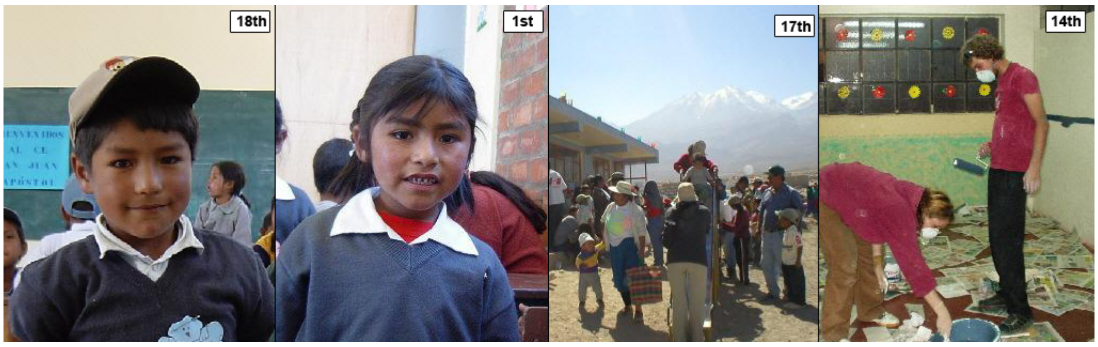


Figure 25: Results Without Dynamic Weighting



When dynamic weighting was applied the top results were more relevant. The picture of people painting a room was pushed down from one of the top 15 to below the 30 most relevant, situating itself as the 33rd most relevant image. Even so, this is a curious incident as a lot of images with a lower score would be deemed more relevant by a human observer. When analyzing which tags are applied to the picture of painters compared to a more relevant but lower scoring image we understand how this happens.

The picture of workers painting a room has 37 tags, whereas the image of the girl in Figure 26 only has 24. Using the initial weighting, all tags are given some weight.

Figure 26: Low Scoring Relevant Image

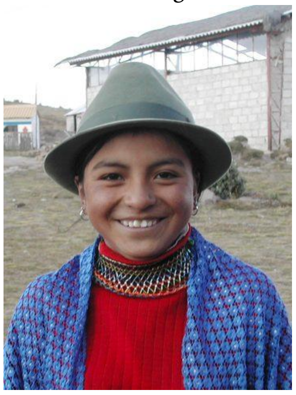


Table 11 shows a list of all tags each image has in common with the source image in Figure 24. Even though the image of a girl has more relevant tags, this is not taken into

Painters (37 tags total)			Girl (24 tags total)		
wearing	blue	red	godchild	dark-skinned	dark-haired
white	arequipa	peru	wearing	red	blue
			background		

Table 11: Tags Applied

account as all tags belonging to the source image are equally weighted in this example. The presence of many tags can outweigh an image with fewer tags.

6.3.2 Iterations

To diminish this factor, we can utilize the iterative nature of the HITS-algorithm. When the HITS computes a new hub weight, we can use that as the new weighting. This new hub weight will typically assign high weight to tags which are frequently used, and less weight to tags found in only a few documents. Not many iterations are necessary as the importance of each tag is determined early. The difference between the weight of two arbitrary tags shows signs of converging during the second iteration. One iteration is deemed sufficient to produce a refined weighting scheme, as the extra precision gained from additional iterations is minimal compared to the performance hit on the system. Table 12 shows how the weight values of two tags converge in subsequent iterations. After four iterations the difference converges at 1.6, and subsequent iterations will not change this as long as the weights are normalized.

Iteration	0		1		2		4	
	Tag 1	Tag 2	Tag 1	Tag 2	Tag 1	Tag 2	Tag 1	Tag 2
Weight	2	2	16	11	27	18	32	20
Difference ($\frac{w_1}{w_2}$)	1		1.45		1.5		1.6	

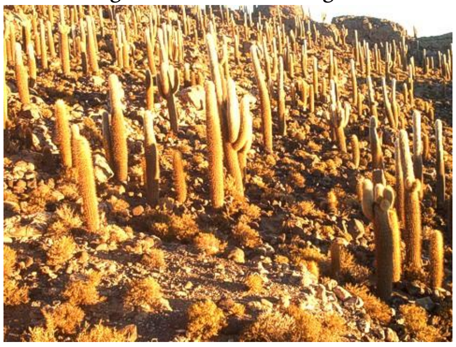
Table 12: Converging Values

Using this approach, an image with many relevant tags can still score higher than the source image if the source image has few tags. This is because the extra set of tags carry more weight when the source image has few tags to base the retrieval on.

6.3.3 Comparison

To assess the performance of the modified HITS-algorithm, we compared its results to a plain Vector Space Algorithm. Figure 27 shows the source image we used.

Figure 27: Source Image 2



This image had a collection of tags with most of the stop words eliminated. The list of tags is as follows:

isla	del	pescado	salar	de
uyuni	cactuses	rocky	hill	dry
grass	during	sunset	bolivia	

At first glance, a weakness in using a ”single word”-keyword approach is apparent. Some of these tags would be a better descriptor if they were merged with their neighbors.

The vector space algorithm’s top hits are shown in Figure 28. The modified HITS’ top

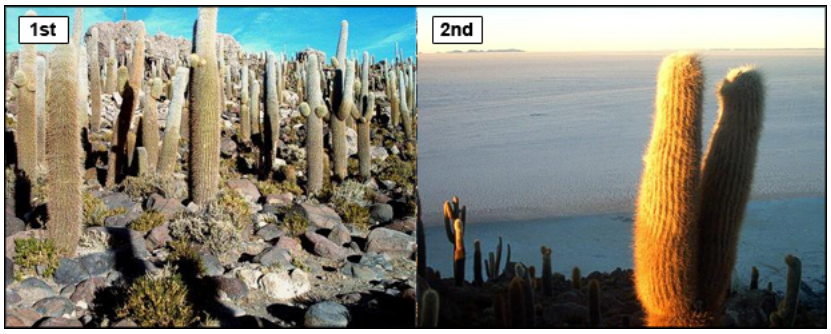


Figure 28: Top Hits, Vector Space

hits are in Figure 29.

These results are without any iterations to refine the weight-matrix of the HITS.

The algorithms do not agree on which images are most relevant. All four images are represented in the top ten of both algorithms, but they do not agree on the order of relevance.

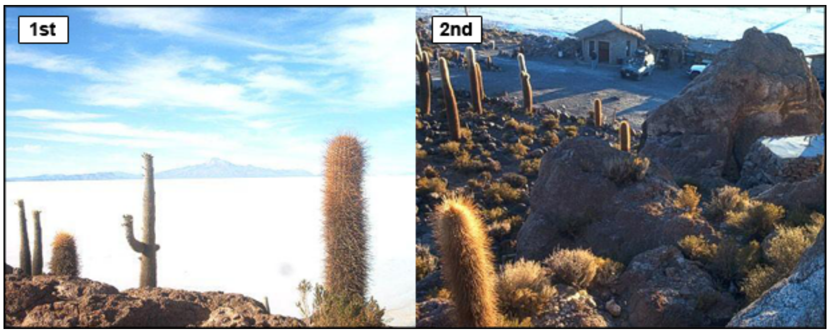


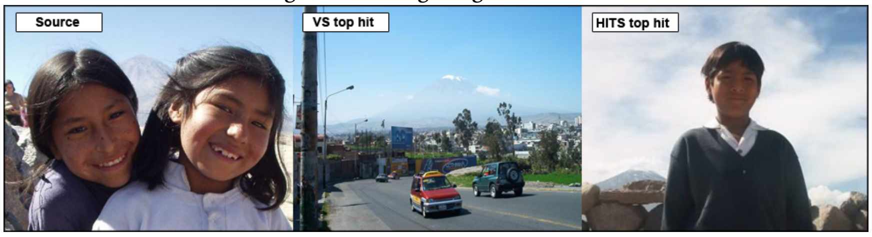
Figure 29: Top Hits, HITS

To understand why this happens, we need to outline the difference between how Vector Space and HITS handles the tags. Vector Space only considers the tags present in the source document. It assigns less weight to common words and more weight if tags are assigned multiple times to the document. HITS, on the other hand, considers all tags present in the extended set but as of now does not put more weight on tags which are present multiple times in a document. This is no problem in this study, as the used set rarely uses the same tag twice on a document. As stated above, dynamic weighting puts more weight on the tags present in the source document but considers the other tags as well. Because of this we observe a trend in the documents returned by HITS. They are consistently tagged with a larger amount of tags than the documents returned by Vector Space. The images returned as top results by HITS have 17 and 15 tags respectively, while the top hits of Vector Space have 8 and 13 tags respectively.

Vector Space has drastically lower execution time on its queries compared to HITS. The average execution time is 1000 ms for Vector Space, and 10,000 ms for HITS. This may be influenced by the fact that the HITS algorithm is not optimized at all. The HITS does more processing on the collection when queried, so a fully optimized HITS would be slower compared to a fully optimized Vector Space algorithm, although not by as much as the present implementation.

On some occasions, the Vector Space fails. Figure 30 shows a girl at the left. The tags assigned to this image include ”volcano”, which is not a very important part of the image. The tag only has 13 occurrences across the entire set, so the Vector Space algorithm assigns great importance to it and shows the image following image in the figure as a good match.

Figure 30: Weighting Differences



6.3.4 Testing with the Set of Student Tags

None of the algorithms worked when testing the student tags because of the small overlap of tags between the documents and the small size of the tag set. Only three tags were used in more than one document. Both algorithms generate a set of documents by using tags in common with the source document. When no other documents have tags in common with the source document, the basis for relating documents is non-existent. This means that this study is unable to test the algorithm performance with a set of user generated tags.

6.3.5 Summary

The modified HITS works differently from the Vector Space algorithm. Vector Space will only consider tags present in the source document. HITS assigns weight to tags in related documents as well. When the weight is refined with multiple iterations, HITS assigns higher weight to popular tags which is the opposite of how Vector Space handles popular tags. When handling user generated tags instead of keywords this may be a good thing, as indicated by the studies done by Bao et al. and Xu et al., referenced in Chapter 2.

Both algorithms return relevant results. Both algorithms often have the same images in their top ten, but in different order. This is due to the differences in behavior observed. The HITS algorithm is not optimized so the execution time is too slow to be of immediate practical use.

For relating documents to a source document, the Vector Space algorithm outperforms the HITS algorithm in execution time because of its simplicity. The full potential of HITS is best shown when the starting point is a single tag or a small collection of tags. Vector Space will only return a list of documents the tag is assigned to while HITS produces a more refined weighting scheme based on several factors, including correlation among tags. If a tag tends to be used alongside the source tag this will be reflected in the weighting scheme of HITS when it is queried with a single tag.

A way to distinguish relevant tags in related documents from the irrelevant tags would be a valued addition for both algorithms. This is discussed in more depth in Section 7.5.

6.4 Social Technologies in DSpace

When creating the DSpace Aspect, some challenges to using DSpace with social technologies were identified. In this section, we will evaluate the recognized advantages and disadvantages of developing for DSpace, and how they pertain to the inclusion of social technologies.

6.4.1 Aspect Integration with DSpace

Aspects are designed to be integrated efficiently in DSpace. This is evidenced by how consistent an additional aspect is with the core system. Every aspect has access to the same functionality the core parts use, as opposed to only a limited subset of functions. This makes the creation of Aspects flexible, as well as easy to utilize already existing parts, most notably the database manager.

6.4.2 Database

When adding tagging and similar technologies additional information needs to be stored in DSpace. The most convenient location is in the database, as it is easy to utilize the database manager in DSpace to retrieve information. This makes it unnecessary to create custom wrappers for each database implementation used at different institutions, as the DSpace database manager handles it. The inconvenient part is the difficulty in adding additional tables to the DSpace database. There is no functionality provided by DSpace when activating an Aspect to create a set of tables. This breaks with the philosophy in DSpace that an Aspect should not be seen when deactivated. The only way to add tables is by creating them either directly in the database or through the database manager in DSpace with a custom SQL query. This impedes the reuse of an Aspect in different institutions, as the tables cannot be created automatically when the Aspect is activated. They need some form of user interaction besides activating the Aspect.

6.4.3 Ajax

Ajax is a trademark technology of the social web. Asynchronous calls usually require a fully built DOM. The DOM in DSpace is only built at the theme level. A fully functional social technology package built on Ajax in DSpace would require both a custom theme and an aspect. This means that each institution interested in using such a package would need a person skilled in technologies such as XSL and Javascript to modify their institutions theme to include the Ajax-functionality required by the Aspect. This is due to the separation of concerns inherited from Cocoon. This also limits other presentational options, e.g. the separate styling of arbitrary words which is the basis for tag clouds. This must also be solved by using a combination of themes and aspects.

There is limited support for Ajax in the API, which seems to support asynchronous submission of forms only. The documentation is lacking in this regard as the only mention of Ajax is one method which enables it for forms, without elaborating on how it is used.

6.4.4 Logging

The extensive logging employed in this study is unnecessary for a production environment. DSpace ships with a useful logging utility which is adequate for normal logging duties, as well as a set of classes to process the logs.

7 Conclusion and Future Work

This thesis has analyzed how students in a real life scenario respond to the introduction of tagging. This was done to investigate if student generated tags are feasible tools for retrieving learning objects related to each other. The modified version of the HITS algorithm, first proposed by Wu et al., is employed as the chosen algorithm to locate related objects. The algorithm is also compared to a plain algorithm to measure its effectiveness.

The conclusions are presented under the pertinent research questions below.

7.1 How can we Integrate Tags in a Learning Environment?

This research question tries to understand how we can integrate tags into a learning environment in a way that will take advantage of the students' behavior.

The first issue is the students' behavior when studying. For some students, tagging is merely an extension of their usual routine while studying. For other students, however, to adopt tagging poses a significant structural change to their study routines. To integrate tagging successfully for this type of student an early introduction to the concept, before they start developing their study routines, may prove effective.

Integrating the tagging system with the learning management systems currently in use at an institution will make it easier and faster for students to use, as they will not need to learn a different system. The goal is to put the tagging as close as possible to where students usually find their learning objects. If the system used for tagging is significantly different or harder to reach, fewer students are likely to use it. Combining tagging with a personalization approach could potentially create a more tailored learning experience for the students.

Our proposal on how to efficiently integrate tagging in a learning environment is to introduce tagging early, preferably to first year students. Integration with systems currently in use will help make students feel familiar when using the system, as well as an easy-to-use user interface, which also is powerful enough to facilitate different kinds of user input, e.g. adding single tags, batch tags (CSV or other methods), and ways for less technically inclined to add multiple tags using the user interface. Personalization could help highlight relevant learning objects for users, which makes them more likely to tag them. For example highlighting audio recordings and blackboard images for distance students, but not for on-campus students.

For student participation, the most important part is to create the system in a way that highlights their own benefits of using the system. Possible uses for tagging should be easily understandable for students, as motivation for use was the hardest part for students participating in the study to understand.

7.2 How can Student Supplied Tags be used to Retrieve Related Learning Objects?

Students are usually using academically centered tags, which describe the content. Not many tags are used for organizational purposes. This tendency to use academically centered tags, can make locating topically related objects easier, as most tags describe the content. Our study is small, and this disposition towards such tags may prove to be false in a larger study, although J. Fan's study seems to agree with our findings.

As mentioned by S. Bateman et al., student tags in their study had a surprisingly high correlation with text mined keywords. The original HITS algorithm uses automatically generated keywords. This makes us feel confident that the modified HITS algorithm will work well by using student tags to locate relevant objects.

Tags such as ”basic lecture” can also be helpful, as they are not possible to generate with text mining. They may contribute by locating not only objects with the same subject matter, but also objects of the equivalent complexity.

The HITS algorithm can be used as a retrieval mechanism, but is too slow to be used efficiently at the moment. Its behavior differs from a plain Vector Space algorithm, so using the algorithm for a different purpose than Vector Space could prove to be effective. HITS is more effective than Vector Space when retrieving objects based on a single tag, while they are comparable when the origin is a document.

7.3 What are the Challenges of Retrieving Related Learning Objects based on Student Tags?

Some challenges were identified during the work on this thesis, which can be summed up as either technical in nature, or related to general challenges.

7.3.1 Technical Challenges

As mentioned in Section 2.2, there are challenges related to tags. Especially that tags can be homographs and synonyms. There is no disambiguation at work in the created system, which should be implemented. We discovered some tags which clearly referenced the same concept, but differed in spelling. This caused these tags to be treated as different concepts in the system. A relevant document may be lost due to such differences in spelling. One approach is to collapse similar tag terms into a unified term representing the specific concept.

Integrating the tagging system in a learning management system currently in use can be regarded as a challenge as there are a large number of learning management systems in use at various institutions. Some are easier than others to modify.

Granularity can be a challenge when the learning object contains a number of different topics. Tagging will then link the whole document to that tag, even though that specific tag only applies to a small part of the learning object. A way to determine how narrow or broad the scope of a tag is, or a way to link a tag to a part of a learning object would be a helpful feature. This was brought up during the student interviews, where one suggestion was to be able to tag parts of a learning object.

7.3.2 General Challenges

How much related material that exists is a valid concern. A retrieval algorithm is dependent on a diverse selection of learning objects, and a tag set with tags overlapping on multiple learning objects. Only one course was included in the study, so the amount of related material was insufficient to correctly determine the scope of this challenge.

The most important challenge is how to motivate students to tag. We have shown that there is an indication that students need to know exactly how they can benefit from using the system to concern themselves with it.

7.4 How can Social Technologies be Implemented in DSpace?

This question aims to answer how well suited the Learning Object Repository DSpace is to incorporate social technologies, such as tagging and rating, from a technical point of view.

DSpace is very well suited for extension. It includes a powerful API, giving developers much freedom when developing Aspects and Themes. DSpace is also open-source, which further extends the customization possibilities. Due to this, it is possible to implement social technologies in DSpace very well.

Knowledge of Java, XML and familiarity with relational database design and SQL are required for developing Aspects. All classes present in an aspect is written in Java. Sitemaps are an integral part of both Aspects and Themes and is based on XML, and every Aspect that needs additional information stored needs to use databases. Themes require the developer to be familiar with XML, XSL, Javascript, HTML and CSS. XML for the Sitemap and DRI, and XSL for being able to transform the DRI to HTML.

The developer of such a system must have good command of various technologies. A complete addition of social technologies in DSpace, though, requires effort beyond developing a simple Aspect or Theme. The aim would be to develop it as a system which could be distributed to a number of institutions interested in adding social technologies in their education and enabling them to install it with relative ease.

The limitations in DSpace makes it hard to create such a system as a complete package and allowing institutions to install it with little to no knowledge of the mentioned technologies. The limited methods of creating extra tables present in the DSpace API necessitates manual creation of the needed tables.

If Ajax is desired, a custom theme is required. The theme should be introduced alongside the existing theme of the institution's DSpace installation. This can be solved using compound themes, where one theme uses another theme's resources. Some manual customization of the theme is still needed.

The need for a Theme to utilize Ajax may disappear in subsequent versions of DSpace, as we have seen indications of initial support for Ajax-integration on the aspect level in the current DSpace version.

Dspace Aspects and Themes is usually easy to install, which is an advantage. The inclusion of more complex functionality makes it harder to distribute as an easily installed stand-alone package, but is still a viable solution if an institution desires to include social technologies in their education.

7.5 Future Work

This section outlines topics that need more work in future studies.

7.5.1 Student Reactions

A study on what factors are important for students to get them to utilize tagging in their education. This can include topics such as:

- Are distance students and on-campus students motivated by different reasons from each other?
- Can tighter integration with existing Learning Management Systems aid participation?

A study on how tags are received by first year students should be done. It is, by itself, a rather small study, but could be integrated as part of a larger study.

How do students take advantage of other students' tags in a tagging scenario? This can be important to understand when developing a system designed for students.

If given the option, would students use private tags—tags only they can see—differently than public tags? Private tags may be easier to use as organizational tags than public tags are.

7.5.2 Technical Studies

Further research on the granularity of tags in learning objects may prove interesting. Research on different ways to classify tags based on their granularity and relationship with other tags. Proposed ways include:

- Mapping the relationship between tags into an ontology.
- Meta-tagging, which can be described as tagging tags.
- Synonym control.

Test if methods such as these would improve the locating of relevant objects.

Further studies on how effective HITS is on retrieving related resources. Indications show it works as well as a plain algorithm, although in a different way, but an optimized version is not tested on a real set of data yet. Further investigations on how this difference can be utilized in a beneficial manner is an interesting topic.

Completing a working version of the tagging system for DSpace and investigating how cumbersome the distribution process is made by Tag Clouds, Ajax and other visual indicators dependent on both themes and aspects is an approach to further facilitate social technologies in DSpace.

Bibliography

[1] Tavangarian, D., Leypold, M. E., Nölting, K., Röser, M., & Voigt, D. December 2004. Is e-Learning the Solution for Individual Learning? *Electronic Journal of e-Learning*, 2(2).

[2] Stahl, G., Koschmann, T., & Suthers, D. *Computer-supported collaborative learning: An Historical Perspective*, 409–426. Cambridge University Press, Cambridge, UK, 2006.

[3] Karrer, T. 2007. Understanding E-Learning 2.0.

[4] Allen, I. E. & Seaman, J. 2010. Class Differences - Online Education in the United States. `http://sloanconsortium.org/publications/survey/pdf/class_differences.pdf`. [Online; accessed 11-December-2010].

[5] Stahl, G. 2004. Building Collaborative Knowing. In *What We Know About CSCL*, Dillenbourg, P., Strijbos, J.-W., Kirschner, P., & Martens, R., eds, volume 3 of *Computer-Supported Collaborative Learning*, chapter 3, 53–85. Springer Netherlands, Dordrecht.

[6] Fronter AS. 2011. Fronter - Platform. `http://com.fronter.info/mnu1.shtml`. [Online; accessed 12-June-2011].

[7] itslearning. 2011. Learning Platform - itslearning. `http://www.itslearning.eu/`. [Online; accessed 12-June-2011].

[8] Utdanningsdirektoratet. Digitale læringsplattformer – en mulig katalysator for digital kompetanse i grunnopplæringen. `http://www.udir.no/upload/Rapporter/LMS.pdf`. [Online; accessed 15-December-2010] - In Norwegian.

[9] Neven, F. & Duval, E. 2002. Reusable learning objects: a survey of lom-based repositories. In *Proceedings of the tenth ACM international conference on Multimedia*, MULTIMEDIA '02, 291–294, New York, NY, USA. ACM.

[10] Ramsden, P. 1988. *Improving Learning: New Perspectives*. Nichols Pub Co.

[11] Wiley, D. A. 2002. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. *The Instructional Use of Learning Objects*.

[12] Ochoa, X. & Duval, E. 2008. Measuring Learning Object Reuse. In *Times of Convergence. Technologies Across Learning Contexts*, Dillenbourg, P. & Specht, M., eds, volume 5192 of *Lecture Notes in Computer Science*, chapter 36, 322–325. Springer Berlin / Heidelberg, Berlin, Heidelberg.

[13] Skaar, L. A., Heiberg, T., & Kongsli, V. 2003. Reuse learning objects through LOM and XML. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, 78–79, New York, NY, USA. ACM.

[14] Duval, E., Forte, E., Cardinaels, K., Verhoeven, B., Van Durm, R., Hendrikx, K., Forte, M. W., Ebel, N., Macowicz, M., Warkentyne, K., & Haenni, F. May 2001. The Ariadne knowledge pool system. *Commun. ACM*, 44(5), 72–78.

[15] Bargmeyer, B. E. & Gillman, D. W. 2000. Metadata Standards and Metadata Registries: An Overview. In *International Conference on Establishment Surveys II*.

[16] Introduction to Dewey Decimal Classification. `http://www.oclc.org/dewey/versions/ddc22print/intro.pdf`. [Online; accessed 12-December-2010].

[17] Weibel, S. L. & Koch, T. The dublin core metadata initiative: Mission, current activities, and future directions. Technical report, 2000.

[18] M. Nilsson. 2000. id3v2.4.0-structure - ID3.org. `http://www.id3.org/id3v2.4.0-structure`. [Online; accessed 12-June-2011].

[19] Institute of Electrical and Electronics Engineers (IEEE). Draft Standard for Learning Object Metadata. Technical report, 2002.

[20] Press, N. 2004. *Understanding Metadata*. National Information Standards Organization Press.

[21] Payette, S. & Lagoze, C. 1998. Flexible and extensible digital object and repository architecture (fedora). In *Research and Advanced Technology for Digital Libraries*, volume 1513 of *Lecture Notes in Computer Science*, 517–517. Springer Berlin Heidelberg.

[22] Smith, M., Barton, M., Bass, M., Branschofsky, M., McClellan, G., Stuve, D., Tansley, R., & Walker, J. H. 2003. Dspace: An open source dynamic digital repository.

[23] Liccardi, I., Ounnas, A., Pau, R., Massey, E., Kinnunen, P., Lewthwaite, S., Midy, M. A., & Sarkar, C. December 2007. The role of social networks in students' learning experiences. *SIGCSE Bull.*, 39, 224–237.

[24] Wal, T. V. 2007. Folksonomy coinage and definition. `http://www.vanderwal.net/folksonomy.html`. [Online; accessed 12-December-2010].

[25] Mathes, A. 2004. Folksonomies - cooperative classification and communication through shared metadata.

[26] Boyle, T. 2003. Design principles for authoring dynamic, reusable learning objects. *Australian Journal of Educational Technology*, 19, 46–58.

[27] Wilhelm, P. & Wilde, R. February 2005. Developing a University Course for Online Delivery Based on Learning Objects: From Ideals to Compromises. *Open Learning*, 20(1), 65–81.

[28] Fan, J. Utilizing Students' Inputs to Create and Manage Learning Object Metadata in Educational System. Master's thesis, Gjøvik University College, Norway, 2010.

[29] Parrish, P. March 2004. The trouble with learning objects. *Educational Technology Research and Development*, 52(1), 49–67.

[30] Noor, S. F., Yusof, N., & Hashim, S. Z. November 2009. A Metrics Suite for Measuring Reusability of Learning Objects. 961–963.

[31] Limpens, F., Gandon, F., & Buffa, M. September 2008. Bridging ontologies and folksonomies to leverage knowledge sharing on the social web: A brief survey. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, 13–18. IEEE.

[32] Merholz, P. 2004. Metadata for the Masses. `http://www.adaptivepath.com/publications/essays/archives/000361.php`. [Online; accessed 12-June-2011].

[33] Hotho, A., Jäschke, R., Schmitz, C., & Stumme, G. *Information Retrieval in Folksonomies: Search and Ranking*, volume 4011 of *Lecture Notes in Computer Science*, chapter Chapter 31, 411–426–426. Springer-Verlag, Berlin/Heidelberg, 2006.

[34] Ames, M. & Naaman, M. 2007. Why we tag: motivations for annotation in mobile and online media. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, 971–980, New York, NY, USA. ACM.

[35] Bao, S., Xue, G., Wu, X., Yu, Y., Fei, B., & Su, Z. 2007. Optimizing web search using social annotations. In *Proceedings of the 16th international conference on World Wide Web*, 501–510, New York, NY, USA. ACM.

[36] Xu, Z., Fu, Y., Mao, J., & Su, D. 2006. Towards the semantic web: Collaborative tag suggestions. In *Proceedings of Collaborative Web Tagging Workshop at 15th International World Wide Web Conference*.

[37] Bateman, S., Brooks, C., Mccalla, G., & Brusilovsky, P. 2007. Applying Collaborative Tagging to E-Learning. In *Proceedings of the 16th International World Wide Web Conference (WWW2007)*.

[38] Manning, C. D., Raghavan, P., & Schtze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[39] Kleinberg, J. M. September 1999. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 604–632.

[40] Wu, H., Zubair, M., & Maly, K. 2006. Harvesting social knowledge from folksonomies. In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, HYPERTEXT '06, 111–114, New York, NY, USA. ACM.

[41] The DuraSpace Foundation. 2010. DSpace Manual. `http://www.dspace.org/1_6_0Documentation/DSpace-Manual.pdf`. [Online; accessed 22-February-2011].

[42] The DuraSpace Foundation. 2010. DSpace Manual. `https://wiki.duraspace.org/display/DSPACE/Manakin`. [Online; accessed 22-February-2011].

[43] The Apache Software Foundation. 2008. Overview of Apache Cocoon. `http://cocoon.apache.org/2.1/overview.html`. [Online; accessed 22-February-2011].

[44] The Apache Software Foundation. 2008. Apache Cocoon Pipelines. `http://cocoon.apache.org/3.0/reference/html/pipelines.html`. [Online; accessed 22-February-2011].

[45] Grubinger, Michael, C. P. D. M. H. & Thomas, D. 2006. The IAPR Benchmark: A New Evaluation Resource for Visual Information Systems. In *International Conference on Language Resources and Evaluation*.

[46] ImageCLEF. 2010. ImageCLEF Dataset. `http://www.imageclef.org/photodata`. [Online; accessed 6-April-2011].

A Data

A.1 Student Available Texts

This is a collection of the texts students received as information about the project.

A.1.1 The Web Site

Permanently available at `http://hochlin.com/skole/gjovik/imt4901`

The text on the page is as follows:

My name is Christian Hochlin and I'm currently doing my master's thesis in media technology. My thesis is about how to utilize social media in education. This includes researching how tags and ratings can be use to improve education, both for professors and students.

What I need you to do

The lecture slides are made available in a LOR (Learning Object Repository) called Dspace. It is possible to tag each learning object with your own tags, as well as rating them with a grade ranging from 1 to 5. All activity on the site will be logged, partly for uncovering usage patterns. What keywords would you use to describe the content of a learning object? These are the kinds of keywords I'm interested in. A way to determine what keywords you should use is to use the same words you would use in a Google search to retrieve the learning object. Other keywords one can use are personal ones, describing the lecture, like ”interesting”, ”difficult” etc.

How can you benefit?

Have you ever wished you were able to give more direct feedback on specific elements in the course than what is possible during course evaluation? Such continuous rating and tagging allows the professors to form an impression of how well-received the syllabus is during the course, instead of only relying on the end evaluation. This makes it possible to experience changes while still doing the course, instead of just helping next years class.

There are studies which indicate the fact that students who tag their learning objects will learn the subject matter in a more efficient way, by processing it in a different, subconscious manner[1], and it allows people to connect items, by ”placing hooks”, to provide their meaning in their own understanding.[2]

For the technically inclined: You will be some of the first to use a new system which maybe will be used by the whole school the coming years.

[1] Bateman, S., Brooks, C., Mccalla, G., & Brusilovsky, P. 2007. Applying Collaborative Tagging to E-Learning. In Proceedings of the 16th International World Wide Web Conference (WWW2007).

[2] Wal, T. V. 2007. Folksonomy coinage and definition. `http://www.vanderwal.net/folksonomy.html`

A.1.2 The Mail

This e-mail was sent to all students following the course at March 10. 2011, and its content was subsequently made available in a private Skype group the class had created.

Topic: To students following IMT4581: Network Security

From: christian.hochlin@hig.no

Date: Thu, March 10, 2011 10:35

To: IMT4581v11@hig.no

Hi all

You are now able to tag your lecture slides, audio and blackboard pictures as part of a master's thesis. Please visit http://www.stud.hig.no/ 091213/index.php?lang=en for a short introduction to the project, and directions on how to participate. Distance students are especially encouraged to participate, as this project may benefit future distance students.

Some more information I forgot to mention when presenting this in last week's lecture:

It is now possible to log in with your GuC-username and password. This enables you to delete tags you added by mistake.

In addition to adding tags, you can rate the quality of the slides using a grade between 1 and 5. This can give the professor an indication on what the class as a whole thinks of the lecture slides.

Available lecture slides: http://matuku.hig.no/overview

If you have any questions about this, do not hesitate to contact me.

Your participation is appreciated :)

Regards,

Christian Hochlin

A.2 Data Logs

This section contains the data taken from the DSpace logs.

A.2.1 View Count per Item

Views	Item
70	lecture 1
54	lecture 2
37	lecture 3
20	lecture 4
21	Lecture 5
23	lecture 6
23	lecture 7
20	lecture 8
17	lecture 9
18	lecture 10
16	lecture 1 audio
25	lecture 1 img
18	lecture 2 audio
14	lecture 2 img
4	lecture 3 audio
7	lecture 3 img
2	lecture 4 audio
4	lecture 4 img
3	lecture 5 audio
9	lecture 5 img
8	lecture 6 audio
4	lecture 6 img
6	lecture 7 audio
3	lecture 7 img
5	lecture 8 audio
3	lecture 9 img
5	lecture 9 audio
3	lecture 10 audio

A.2.2 Item Views

This table shows item views per date.

Date	Views
2/3	17
3/3	1
4/3	1
8/3	2
9/3	1
10/3	44
11/3	1
12/3	3
13/3	2
14/3	4
15/3	2
16/3	6
17/3	4
22/3	13
23/3	3
25/3	6
28/3	18
29/3	37
30/3	28
31/3	5
5/4	1
6/4	6
7/4	1
11/4	7
12/4	14
13/4	19
16/4	11
17/4	5
18/4	11
19/4	8
20/4	1
22/4	1
24/4	1
27/4	3
28/4	5
30/4	6
1/5	8
3/5	19
4/5	7
5/5	1
10/5	4
11/5	9
13/5	1
15/5	2
16/5	1
19/5	7
26/5	1
27/5	4
28/5	1
29/5	2
30/5	4
31/5	5
2/6	1
3/6	3

A.3 Tags Applied

This is a list of all tags applied, sorted on date.

Date	Tag Count
2/3	7
10/3	20
22/3	10
28/3	10
29/3	18
30/3	29
12/4	29
3/5	13
10/5	10
27/5	4

B Source Code

Imports are omitted from all source code.

B.1 DSpace

B.1.1 SQL

SQL for table creation

```
CREATE TABLE tag(id SERIAL NOT NULL, tag VARCHAR(32) UNIQUE,
PRIMARY KEY(id))

CREATE TABLE tagUse(id SERIAL, itemID INTEGER, tagID INTEGER,
userID INTEGER, time TIMESTAMP, visibility INTEGER,
deleted INTEGER, ip VARCHAR(32), PRIMARY KEY(id));

CREATE TABLE actions(actionid SERIAL NOT NULL, userid INTEGER,
tagid INTEGER, itemid INTEGER, time TIMESTAMP, action INTEGER,
ip VARCHAR(32), PRIMARY KEY(actionid));

CREATE TABLE rating(id SERIAL NOT NULL, rating INTEGER,
item INTEGER, person INTEGER, ip VARCHAR(32),
time TIMESTAMP, PRIMARY KEY(id));
```

B.1.2 Sitemap

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
<map:components>

  <map:transformers>
   <map:transformer name="Navigation" src="no.hig.xmlui.aspect.overview.Navigation"/>
   <map:transformer name="Overview" src="no.hig.xmlui.aspect.overview.Overview"/>
   <map:transformer name="ItemViewer" src="no.hig.xmlui.aspect.overview.ItemViewer"/>

   <map:transformer name="BrowseSingleTag"
                   src="no.hig.xmlui.aspect.overview.BrowseSingleTag" />
   <map:transformer name="BrowseOwnTags"
                   src="no.hig.xmlui.aspect.overview.BrowseOwnTags" />
   <map:transformer name="BrowseTags" src="no.hig.xmlui.aspect.overview.BrowseTags" />
   <map:transformer name="Statistics" src="no.hig.xmlui.aspect.overview.Statistics" />
  </map:transformers>

  <map:matchers default="wildcard">
            <map:matcher name="HandleTypeMatcher"
```

```
                  src="org.dspace.app.xmlui.aspect.general.HandleTypeMatcher"/>

               <map:matcher name="HandleAuthorizedMatcher"
                 src="org.dspace.app.xmlui.aspect.general.HandleAuthorizedMatcher"/>

  </map:matchers>
  <map:actions>
        <map:action name="DeleteTag" src="no.hig.xmlui.aspect.overview.DeleteTag"/>

        <map:action name="AddTags" src="no.hig.xmlui.aspect.overview.AddTags" />

        <map:action name="Log" src="no.hig.xmlui.aspect.overview.Logging" />
        <map:action name="Rate" src="no.hig.xmlui.aspect.overview.Rate" />
  </map:actions>

</map:components>
<map:pipelines>
        <map:pipeline>
        <map:generate/>

        <!-- Add to every page -->
        <map:transform type="Navigation"/>



                   <map:match pattern="addTags">

                     <map:act type="AddTags">
                                 <map:redirect-to uri="{url}" permanent="yes" global="true" />
                  <!-- User agent won't cache redirected page if not permanent is set -->
                         </map:act>

                   </map:match>

                   <map:match pattern="tagbrowser">

                     <map:transform type="BrowseTags"/>

                   </map:match>

                   <map:match pattern="overview">

                     <map:transform type="Overview"/>

                   </map:match>
```

```
          <map:match pattern="tagstatistics">

            <map:transform type="Statistics"/>

          </map:match>

          <map:match pattern="tagbrowser/*">
                  <map:act type="Log">
                          <map:parameter name="eventType" value="viewtag"/>
                  </map:act>
                  <map:transform type="BrowseSingleTag"/>
    </map:match>


          <map:match pattern="owntags">

            <map:transform type="BrowseOwnTags"/>

          </map:match>

           <!-- Add data to an item -->
           <map:match pattern="handle/*/*">
                   <map:match type="HandleAuthorizedMatcher" pattern="READ">
                <map:match type="HandleTypeMatcher" pattern="item">
                <map:act type="Log">
                           <map:parameter name="eventType" value="viewitem"/>
                          </map:act>
        <map:transform type="ItemViewer"/>
       </map:match>
   </map:match>
</map:match>

<!-- Delete a tag -->
           <map:match pattern="deletetag/*">

                   <map:act type="DeleteTag">
                        <map:redirect-to uri="{url}"
                         permanent="yes" global="true" />
        <!-- User agent won't cache redirected page if not permanent is set -->
                </map:act>

</map:match>

<!-- Rate item -->
<map:match pattern="rate">
```

```
                           <map:act type="Rate">
                                <map:redirect-to uri="{url}"
                                 permanent="yes" global="true" />
                <!-- User agent won't cache redirected page if not permanent is set -->
                        </map:act>

        </map:match>



    <map:serialize type="xml"/>

        </map:pipeline>
</map:pipelines>
</map:sitemap>
```

B.1.3 AddTags

```
        package no.hig.xmlui.aspect.overview;

/**
 * A method to add user generated tags to the database.
 *
 * @author Christian Hochlin
 *
 */

public class AddTags extends AbstractAction {

        public Map act(Redirector redirector, SourceResolver resolver, Map objectModel,
            String source, Parameters parameters) throws Exception, SQLException
    {
                Request request = ObjectModelHelper.getRequest(objectModel);
                Context context = ContextUtil.obtainContext(objectModel);

                String ip = request.getRemoteAddr();

                Boolean proceed = false;
                String tags, item;
                if (request.getParameter("tags") != null
                        && request.getParameter("itemhandle") != null)
                {
                        tags = request.getParameter("tags");
                        item = request.getParameter("itemhandle");
                        TagManager tagManager = new TagManager(context);

                        EPerson eperson = context.getCurrentUser();
                        int personid;
                        if(eperson == null)
                        {
                                personid = 0;
                        }
                        else
                        {
                                personid = eperson.getID();
                        }
                        tagManager.addTagsFromCSVString(personid,
                                Integer.parseInt(item), tags, ip);


                }
                else
                //Something wasn't filled out. Abort
                {


                }
                 //Takes care of redirecting
                String fromPage = request.getHeader("Referer");
                Map sitemapParams = new HashMap();
                sitemapParams.put("url", fromPage);

                return sitemapParams;


    }
}
```

B.1.4 BrowseOwnTags

```
        package no.hig.xmlui.aspect.overview;

/**
 *
 * Transformer that lets users browse their own tags.
 *
 * @author Christian Hochlin
 *
 */
public class BrowseOwnTags extends AbstractDSpaceTransformer {

         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {
         }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {
                 EPerson eperson = context.getCurrentUser();

                 TagManager tm = new TagManager(context);

                 ArrayList<Tag> tags = tm.getItemTagsByOwner(eperson);

                 for(Tag tag : tags)
                 {
                         Division d = body.addDivision("tag"+tag.getId());
                         Para p = d.addPara();
                         p.addXref(contextPath+"/tagbrowser/"
                                 +tag.getId(),tag.getTagName());

                         p.addContent(" used "+tag.getCount()+" time(s)");
                 }
          }
}
```

B.1.5 BrowseSingleTag

```
        package no.hig.xmlui.aspect.overview;
/**
 *  Transformer that lets users browse items tagged with a tag
 *
 * @author Christian Hochlin
 *
 */
public class BrowseSingleTag extends AbstractDSpaceTransformer {

         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {
     }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {
                  Tag tag = HiGHandleUtil.getTag(objectModel);

              Division division = body.addDivision("item-tags",null);
              if(tag.getId() == -1)
              {
                      division.setHead("No Tags");
                      division.addPara("This is a list of all items without tags");
              }
              else
              {
                      division.setHead(tag.getTagName());
                      division.addPara("This is a list of all items tagged with "
                                      +tag.getTagName().toUpperCase());
              }
               ArrayList<Item> items = tag.getTaggedItems(context);

               Division div = body.addDivision("temp");

               List list = div.addList("list");
               for(Item item : items)
               {
                       list.addItemXref("/handle/"+item.getHandle(), item.getName());
               }
          }
}
```

B.1.6 BrowseTags

```
        package no.hig.xmlui.aspect.overview;
/**
 * Transformer that lets users browse a list of all tags
 * @author Christian Hochlin
 *
 */
public class BrowseTags extends AbstractDSpaceTransformer {

         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {
     }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {
                   Division division = body.addDivision("item-tags",null);
               division.setHead("Browse tags");

               TagManager tm = new TagManager(context);
```

```
                        ArrayList<Tag> tags = tm.getTags();
                        if(tags.get(0).getId() == -1) //If no tags exist
                        {
                                division.addPara("Nothing here");
                        }
                        else
                        {
                                division.addPara("This is a list of all tags used");


                                Division div = body.addDivision("temp");

                                List list = div.addList("list");
                                for(Tag tag : tags)
                                {
                                        list.addItemXref(contextPath+"tagbrowser/"
                                                        +tag.getId(),tag.getTagName());
                                }
                        }
                }
}
```

B.1.7 DeleteTag

```
        package no.hig.xmlui.aspect.overview;
/**
 * Action to delete tags
 * @author Christian Hochlin
 *
 */
public class DeleteTag extends AbstractAction {

        public Map act(Redirector redirector, SourceResolver resolver, Map objectModel,
            String source, Parameters parameters) throws Exception, SQLException
    {
                Request request = ObjectModelHelper.getRequest(objectModel);
                Context context = ContextUtil.obtainContext(objectModel);
                String ip = request.getRemoteAddr();
                String uri = request.getSitemapURI();
                int id = Integer.parseInt(uri.split("/")[1]);

        EPerson loggedin = context.getCurrentUser();
        TagManager tm = new TagManager(context);

        //Takes care of redirecting
                String fromPage = request.getHeader("Referer");
                Map sitemapParams = new HashMap();
                sitemapParams.put("url", fromPage);
                Item item;
                 DSpaceObject dso = getObjectFromUri(fromPage, context);
                if (dso instanceof Item)
                {
                   item = (Item) dso;
                   try
                   {
                         //Deletes the tag, if EPerson has the rights to do so.
                           tm.deleteTag(item.getID(), loggedin.getID(), id, ip);
                   }
                   catch(SQLException e)
                   {
                   }
                }
            return sitemapParams;

    }

        //A modified version of handleutil
        private DSpaceObject getObjectFromUri(String uri, Context context)
        throws SQLException
        {
                final String HANDLE_PREFIX = "http://matuku.hig.no/handle/";

                 if (!uri.startsWith(HANDLE_PREFIX))
             // Dosn't start with the prefix then no match
             return null;

         String handle = uri.substring(HANDLE_PREFIX.length());

         int firstSlash = handle.indexOf('/');
         if (firstSlash < 0)
             // If there is no first slash then no match
             return null;

         int secondSlash = handle.indexOf('/', firstSlash + 1);
         if (secondSlash < 0)
             // A trailing slash is not nesssary if there is nothing after
             // the handle.
             secondSlash = handle.length();

         handle = handle.substring(0, secondSlash);
         return HandleManager.resolveToObject(context, handle);
        }
}
```

B.1.8 HiGHandleUtil

```
        package no.hig.xmlui.aspect.overview;

/**
 * A modified version of handleUtil, that works with tags
 *
 * @author Christian Hochlin
 *
 */
public class HiGHandleUtil {

        public static final int TAG = 1;

        public static Tag getTag(Map objectModel) throws SQLException
        {
                final String HANDLE_PREFIX = "tagbrowser/";

                Request request = ObjectModelHelper.getRequest(objectModel);

                 String uri = request.getSitemapURI();

                 if (!uri.startsWith(HANDLE_PREFIX))
             // Doesn't start with the prefix then no match
             return null;

         String handle = uri.substring(HANDLE_PREFIX.length());

         Context context = ContextUtil.obtainContext(objectModel);

         return (Tag)resolveToObject(context, handle, TAG);
        }

        private static Object resolveToObject(Context context, String handle, int type)
                throws SQLException
        {
                Object o = new Object();
                switch (type)
                {
                        case TAG:

                                TagManager tm = new TagManager(context);
                                o = tm.getTagFromID(Integer.parseInt(handle));
                        break;
                        default:

                }
                return o;
        }
}
```

B.1.9 ItemViewer

```
        package no.hig.xmlui.aspect.overview;

/**
 * This class adds an interface for viewing tags and adding new tags to an item's page.
 *
 * @author Christian Hochlin
 *
 */

//TODO Internationalize text
public class ItemViewer extends AbstractDSpaceTransformer {


        private static final Message T_dspace_home =
        message("xmlui.general.dspace_home");


        private Item _item;
        private EPerson _person;

         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {
         }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {

                 DSpaceObject dso = HandleUtil.obtainHandle(objectModel);
                 if (!(dso instanceof Item))
                    return;
                Item item = (Item) dso;
                _item = item;
                Division division = body.addDivision("item-tags",null);
```

```
division.setHead("Tags");
division.addPara("These are user generated tags for the current item "
        +"(number of occurrences in parentheses)");

addTags(division);

EPerson eperson = context.getCurrentUser();
_person = eperson;
division.addPara("Even when a tag you want to add exists,"
        +" don't hesitate to add it anyway.");

        Division addTagsDiv = division.addInteractiveDivision(
                "input-new-tag", "/addTags", "get");

        List list = addTagsDiv.addList("addTag",List.TYPE_FORM);
        Hidden itemHandle = addTagsDiv.addHidden("itemhandle");

        itemHandle.setValue(item.getID());
        //Not good practice according to dspace
        //(revealing internal IDs)


        list.addLabel("Add your own tags (comma separated)");
        list.addItem().addText("tags");
        Button button = list.addItem().addButton("button");


        button.setValue("Submit");

        Division rating = body.addDivision("rating");
        rating.setHead("Rate this learning object");
        double avgRating = RatingManager.getAverageRating(context, _item.getID());

        if(avgRating != 0)
        {
         rating.addPara("This object has an average rating of "+avgRating);
        }

        Division ratingDiv = rating.addInteractiveDivision("input-new-rating",
                "/rate", "post");
        Hidden itemHandleRating = ratingDiv.addHidden("itemhandle");
        itemHandleRating.setValue(item.getID());
               Para ratingPara = ratingDiv.addPara();
               Radio r = ratingPara.addRadio("rating");
               if(_person != null)
               {
```

```
                                 int personrating = RatingManager.getRatingForPerson(contex
                                  _person.getID(), _item.getID());
                                 addRating(r, personrating);
                                }
                                else
                                {
                                         r.addOption("1", "1 - Bad");
                                    r.addOption("2", "2");
                                        r.addOption("3", "3");
                                        r.addOption("4", "4");
                                        r.addOption("5", "5 - Great");
                                }
                                Button buttonRating = ratingPara.addButton("submitRating")
                                buttonRating.setValue("Submit");

                 if(_person != null)
                 {
                         addUserTags(division);
                 }
          }

         /**
          * Method to have the previously rated value selected when logged in
          * users enter the item.
          * @param r
          * @param rating
          * @throws WingException
          */
         private void addRating(Radio r, int rating) throws WingException
         {
                 //Huff
                 switch (rating)
                    {
                    case 1:
                            r.addOption(true,"1", "1 - Bad");
                               r.addOption("2", "2");
                               r.addOption("3", "3");
                               r.addOption("4", "4");
                               r.addOption("5", "5 - Great");
                            break;
                    case 2:
                            r.addOption("1", "1 - Bad");
                               r.addOption(true,"2", "2");
                               r.addOption("3", "3");
                               r.addOption("4", "4");
                               r.addOption("5", "5 - Great");
```

```
                        break;

                case 3:
                        r.addOption("1", "1 - Bad");
                           r.addOption("2", "2");
                           r.addOption(true,"3", "3");
                           r.addOption("4", "4");
                           r.addOption("5", "5 - Great");
                        break;

                case 4:
                        r.addOption("1", "1 - Bad");
                           r.addOption("2", "2");
                           r.addOption("3", "3");
                           r.addOption(true,"4", "4");
                           r.addOption("5", "5 - Great");
                        break;
                case 5:
                        r.addOption("1", "1 - Bad");
                           r.addOption("2", "2");
                           r.addOption("3", "3");
                           r.addOption("4", "4");
                           r.addOption(true,"5", "5 - Great");
                        break;
                default:
                        r.addOption("1", "1 - Bad");
                           r.addOption("2", "2");
                           r.addOption("3", "3");
                           r.addOption("4", "4");
                           r.addOption("5", "5 - Great");
                }
     }

     /**
      * Fetches tags the current user has used to tag the current item from
      * the TagManager-class
      *
      * @param Division d
      * @throws WingException
      */
     private void addUserTags(Division d) throws WingException, SQLException
     {
             TagManager tagManager = new TagManager(context);
             ArrayList<Tag> tags = tagManager.getItemTagsByOwner(_item, _person);
            if(tags.size() > 0)
            {
                Division owntags = d.addDivision("own-tags");
                owntags.addPara("Your own tags:");
                Para tagList = owntags.addPara();

                for(Tag tag : tags)
                    {
                        if(tag.getId() != -1)
                        {
                         tagList.addXref("/tagbrowser/"+tag.getId(),
                                                tag.getTagName());

                         tagList.addFigure(
                          "/themes/Reference/images/confidence/3-circleslash.gif"
                          "/deletetag/"+tag.getId(),"primary");

                        }
                        else
                        {
                                tagList.addContent("No tags");
                        }
                    }
        }
}
/**
 * Fetches tags related to the current item from the TagManager-class
 *
 * @param Division d
 * @throws WingException
 */
private void addTags(Division d) throws WingException, SQLException
{
        TagManager tagManager = new TagManager(context);
        ArrayList<Tag> tags = tagManager.getItemTags(_item);

        Para container = d.addPara();

        for(Tag tag : tags)
        {
                container.addXref("/tagbrowser/"+tag.getId(),
                        tag.getTagName()+"("+tag.getCount()+")");
                if(AuthorizeManager.isAdmin(context))
                {
                        container.addFigure(
                         "/themes/Reference/images/confidence/3-circleslash.gif",
                         "/deletetag/"+tag.getId(),"primary");
                }
                }
        }

}
```

B.1.10 Logging

```
        package no.hig.xmlui.aspect.overview;

/**
 * Action to log events. Invoked from Sitemap
 *
 * @author Christian Hochlin
 *
 */
public class Logging extends AbstractAction  {

        public Map act(Redirector redirector, SourceResolver resolver, Map objectModel,
                String source, Parameters parameters) throws Exception, SQLException
        {
                Request request = ObjectModelHelper.getRequest(objectModel);
                Context context = ContextUtil.obtainContext(objectModel);
                String ip = request.getRemoteAddr();

                String type = parameters.getParameter("eventType");
                EPerson person = context.getCurrentUser();
                int personID = 0;
                if(person != null)
                {
                        personID = person.getID();
                }

                if(type.equals("viewtag"))
                {
                        Tag tag = HiGHandleUtil.getTag(objectModel);
                        TagLog.logEvent(context, TagLog.ENTER_TAG, personID ,
                                tag.getId(), ip);
                }
                else if(type.equals("viewitem"))
                {
                        DSpaceObject dso = HandleUtil.obtainHandle(objectModel);
                        if ((dso instanceof Item))
                        {
                                Item item = (Item) dso;
                                TagLog.logEvent(context, TagLog.ENTER_ITEM,
                                        personID , item.getID(), ip);
                        }

                }
                return null;
        }
}
```

B.1.11 LogItem

```
        package no.hig.xmlui.aspect.overview;
/**
 * Represents an item in the log
 *
 * @author Christian Hochlin
 *
 */
public class LogItem {

        private int _id;
        private int _user;
        private int _tag;
        private int _item;
        private Date _date;
        private int _action;
        private String _ip;

        private long _count;
        public ArrayList<String> ipList;

        public LogItem(int id, int user, int tag, int item,
         Date date, int action, String ip)
        {
                _id = id;
                _user = user;
                _tag = tag;
                _item = item;
                _date = date;
                _action = action;
                _ip = ip;
        }

        public LogItem(Date date, long count )
        {
                _date = date;
                _count = count;
                ipList = new ArrayList<String>();
        }
```

```
public void addIP(String ip)
{
        if(!ipList.contains(ip))
        {
                ipList.add(ip);
                _count++;
        }

}

public String dumpTemporalCSV()
{
        int month = _date.getMonth() + 1;
        String info = _date.getDate()+"/"+month+","+_count;

        return info;
}

public String dumpInfo(Context context) throws SQLException
{
        String info = "user: ";
        EPerson person;
        Tag tag;
        Item item;
        if(_user != 0)
        {
                person = EPerson.find(context, _user);
                info += person.getFullName()+", ";
        }
        else
        {
                info += "Anonymous, ";
        }

        if(_tag != 0)
        {
                if(_tag == -1)
                {
                        info += "no tag, ";
                }
                else
                {
                        tag = Tag.find(context, _tag);
                        info += tag.getTagName()+", ";
                }
        }
```

```
                if(_item != 0)
                {

                        item = Item.find(context, _item);
                        info += item.getName()+", ";
                }
                int month = _date.getMonth() + 1;
                return info+"date: "+_date.getDate()+"."+month
                        +" - "+_date.getHours()+":"+_date.getMinutes();
        }
}
```

B.1.12 Navigation

```
package no.hig.xmlui.aspect.overview;
/**
 * Adds GUC-specific menu options
 *
 * @author Christian Hochlin
 */
public class Navigation extends AbstractDSpaceTransformer
{
    public void addOptions(Options options) throws SAXException, WingException,
            UIException, SQLException, IOException, AuthorizeException
    {
        List test = options.addList("hig");
        test.setHead("HiG");

        test.addItemXref(contextPath + "/tagbrowser","Tag Browser");
        test.addItemXref(contextPath+"/overview", "Item Browser");
        if(eperson != null)
        {
                test.addItemXref(contextPath + "/owntags", "Your tags");
                if(eperson.getID() == 9)
            {

                    test.addItemXref(contextPath+"/tagstatistics", "Stats");
            }
        }


    }
}
```

B.1.13 Overview

```
        package no.hig.xmlui.aspect.overview;
```

```
/**
 * A list of all the submitted papers. More convenient to navigate
 * to them from here, than going into each community.
 *
 * @author Christian Hochlin
 */
public class Overview extends AbstractDSpaceTransformer  {

         Item item;

         private static final Message T_dspace_home =
                message("xmlui.general.dspace_home");

         private static final Message T_heading =
                message("no.hig.xmlui.aspect.overview.header");

         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {
                 pageMeta.addMetadata("title").addContent("Overview");
                 pageMeta.addTrailLink(contextPath + "/",T_dspace_home);
                 pageMeta.addTrail().addContent("Overview");
     }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {
                 Division div = body.addDivision("header");
                 div.setHead("Item Browser");

                 Request request = ObjectModelHelper.getRequest(objectModel);
                 String t = "This is a list of all available items";
                 div.addPara(t);
                         Collection[] collections = Collection.findAll(context);
                         for(Collection collection : collections)
                         {
                                 if(collection.countItems() != 0)
                                 {
                                         ItemIterator items = collection.getAllItems();

                                         List list = div.addList(
                                                 "list"+collection.getID());
                                         list.setHead(collection.getName());
                                         Item item;
                                         do
```

```
                                        {
                                      item = items.next();
                                          list.addItemXref(contextPath+"handle/"
                                               +item.getHandle(), item.getName());
                                        }
                                        while(items.hasNext());
                                }
                        }

     }

         private void displayItems(Division div, ItemIterator items)
           throws SQLException, WingException
         {
                 Item item = items.next();
                 while(items.hasNext())
                 {
                         div.addPara(item.getName());
                 }

         }

}
```

B.1.14 Rate

```
        package no.hig.xmlui.aspect.overview;

/**
 * Action to rate an item.
 * @author Christian Hochlin
 *
 */
public class Rate extends AbstractAction {

        public Map act(Redirector redirector, SourceResolver resolver, Map objectModel,
            String source, Parameters parameters) throws Exception, SQLException
    {
                Request request = ObjectModelHelper.getRequest(objectModel);
                Context context = ContextUtil.obtainContext(objectModel);
                String ip = request.getRemoteAddr();

                int rating = Integer.parseInt(request.getParameter("rating"));
                int itemid = Integer.parseInt(request.getParameter("itemhandle"));

                EPerson loggedin = context.getCurrentUser();
                if(loggedin == null)
                {
                        RatingManager.rateItem(context, rating, itemid, ip);
                        TagLog.logEvent(context, TagLog.RATE_ITEM, 0 , itemid, ip);
                }
                else
                {
                        RatingManager.rateItem(context, rating, itemid,
                                loggedin.getID(), ip);
                        TagLog.logEvent(context, TagLog.RATE_ITEM,
                                loggedin.getID() , itemid, ip);
                }


                //Takes care of redirecting
                String fromPage = request.getHeader("Referer");
                Map sitemapParams = new HashMap();
                sitemapParams.put("url", fromPage);

                return sitemapParams;
    }
}
```

B.1.15 RatingManager

```
        package no.hig.xmlui.aspect.overview;
/**
 * Manager to manage adding and retrieving of ratings
 *
 * @author Christian Hochlin
 *
 */
public class RatingManager {

        public RatingManager()
        {

        }

        public static void rateItem(Context c, int rating, int itemid,
                int personid, String ip) throws SQLException
        {
                if(rating > 0 && rating < 6)
                {
                        if(personid != 0)
                        {
                                TableRow tr = DatabaseManager.querySingle(c,
                                "SELECT id FROM rating WHERE person = ?"
                                +"AND item = ?", personid, itemid);
                        if(tr == null)//If person hasn’t rated yet
                        {
                                DatabaseManager.updateQuery(c,
                                "INSERT INTO rating(rating, item, " +
                                "person, ip, time) VALUES(?,?,?,?, now())",
                                                rating, itemid, personid, ip);

                        }
                        else
                        {
                                DatabaseManager.updateQuery(c,
                                "UPDATE rating SET rating=?"
                                +" WHERE id = ?", rating, tr.getIntColumn("id"));
                        }


                }
                else
                {
                        DatabaseManager.updateQuery(c,
                        "INSERT INTO rating(rating, item,"+
                                "person, ip, time) VALUES(?,?,?,?, now())",
                                        rating, itemid, personid, ip);
                }


        }
}

public static void rateItem(Context c, int rating, int itemid, String ip)
throws SQLException
{
        rateItem(c,rating, itemid, 0, ip);
}

public static int getRatingForPerson(Context c, int personid, int itemid)
throws SQLException
{
        TableRow tr = DatabaseManager.querySingle(c, "SELECT id FROM rating
                WHERE person = ? AND item = ?", personid, itemid);
        if(tr == null)//If person hasn’t rated yet
        {
                return 0;
                }
                else
                {
                        return DatabaseManager.querySingle(c, "SELECT rating FROM rating
                        WHERE item = ? AND person = ? ", itemid, personid)
                        .getIntColumn("rating");

                }

        }

        public static double getAverageRating(Context c, int itemid) throws SQLException
        {

                long count = DatabaseManager.querySingle(c, "select count(id) AS no
                        FROM rating WHERE item = ?", itemid).getLongColumn("no");

                TableRowIterator it = DatabaseManager.query(c, "SELECT rating
                        FROM rating WHERE item = ?", itemid);

                int sum = 0;
                do
                {

                        if(it.hasNext())
                        {
                                TableRow row = it.next();
                                int rating = row.getIntColumn("rating");
                                sum += rating;

                        }


                }
                while(it.hasNext());

                if(sum != 0)
                {
                        double b = (double)sum/(double)count;
                        DecimalFormat twoDForm = new DecimalFormat("#.##");
                        return Double.valueOf(twoDForm.format(b));

                }
                else
                {
                        return 0;
                }
        }
}
```

B.1.16 Statistics

```
        package no.hig.xmlui.aspect.overview;
/**
 * "hidden" page to extract statistics
 * @author Christian Hochlin
 *
 */
public class Statistics extends AbstractDSpaceTransformer {
         public void addPageMeta(PageMeta pageMeta) throws SAXException,
     WingException, UIException, SQLException, IOException,
     AuthorizeException
     {

     }

         public void addBody(Body body) throws SAXException, WingException,
     UIException, SQLException, IOException, AuthorizeException
     {
                 EPerson eperson = context.getCurrentUser();
                 if(eperson != null && eperson.getID() == 9)
                 {
                         TagLog tl = new TagLog();
                         Division info = body.addDivision("info");
                         info.addPara("Tiden er en time feil");
                         Division itemStats = body.addDivision("itemstats");
                         itemStats.setHead("Item Statistics");

                         Division userStats = body.addDivision("userstats");
                         userStats.setHead("User Statistics");

                         Division tagStats = body.addDivision("tagstats");
                         tagStats.setHead("Tag Statistics");

                         Division itemsEntered = itemStats.addDivision("itemsEntered");

                         itemsEntered.setHead("Items entered");
                         itemsEntered.addPara("Last five entries:");

                         ArrayList<LogItem> itemsEnteredList =
                                 tl.getLogItems(context, TagLog.ENTER_ITEM, 5);

                         for(LogItem listItem : itemsEnteredList)
```

```
{
        itemsEntered.addPara(listItem.dumpInfo(context));
}

Division tagsEntered = tagStats.addDivision("tagsEntered");
tagsEntered.setHead("Tags Entered");
tagsEntered.addPara("Last five entries");

ArrayList<LogItem> TagsEnteredList =
        tl.getLogItems(context, TagLog.ENTER_TAG, 5);
for(LogItem listItem : TagsEnteredList)
{
        tagsEntered.addPara(listItem.dumpInfo(context));
}

ArrayList<LogItem> list =
        tl.getTemporalData(context, TagLog.ENTER_ITEM);
Division links = body.addDivision("links");
String text = "Entered items temporal\n";



for(LogItem l : list)
{
   text += l.dumpTemporalCSV()+"\n";
}
links.addPara(text);

ArrayList<LogItem> list2 = tl.getTemporalTagUsage(context);

text = "Temporal Tag usage \n";



for(LogItem l : list2)
{
       text += l.dumpTemporalCSV()+"\n";
}
links.addPara(text);

ArrayList<LogItem> list3 =
        tl.getTemporalVisitsByIP(context, TagLog.ENTER_ITEM);

text = "Temporal Tag usage by IP \n";

for(LogItem l : list3)
```

```
                              {
                                 text += l.dumpTemporalCSV()+"\n";
                              }
                              links.addPara(text);
                     }
           }
}
```

B.1.17 Tag

```
        package no.hig.xmlui.aspect.overview;
/**
 * A class that represents a tag
 *
 * @author Christian Hochlin
 *
 */
public class Tag {

        private int id; //-1 is "no tag"
        private String tag;
        private int count;
        public Tag(String tagName, int tagID)
        {
                id = tagID;
                tag = tagName;
                count = 1;
        }

        public static Tag find(Context context, int id) throws SQLException
        {
                TableRow tr = DatabaseManager.querySingle(context, "SELECT tag "
                        +"FROM tag WHERE id = ?", id);
                return new Tag(tr.getStringColumn("tag"), id);

        }
        public Tag(String tagName, int tagID, int count)
        {
                id = tagID;
                tag = tagName;
                this.count = count;
        }

        public String getTagName()
        {
                return tag;
        }
```

```java
public int getCount()
{
        return count;
}

public void setCount(int i)
{
        count = i;
}

public int getId()
{
        return id;
}

public ArrayList<Item> getTaggedItems(Context context) throws SQLException
{
        TableRowIterator iterator;
        if(id == -1)
        {
                iterator = DatabaseManager.query(context,
                                "SELECT item_id FROM item "+
                                "WHERE item_id NOT IN (SELECT itemid FROM taguse)");
        }
        else
        {


                iterator = DatabaseManager.query(context,
                        "SELECT Distinct(item_id) FROM taguse, tag, item " +
                        "WHERE" +
                                        " item.item_id = taguse.itemid " +
                                        " AND taguse.tagid = tag.id" +
                                        " AND taguse.deleted = 0" +
                                        " AND tag.id = "+id);
        }
        ArrayList<Item> items = new ArrayList<Item>();
        do
        {
                TableRow row = iterator.next();
                items.add(Item.find(context, row.getIntColumn("item_id")));


        }
        while(iterator.hasNext());
```

```
                return items;

        }

}
```

B.1.18 TagLog

```
        package no.hig.xmlui.aspect.overview;

/**
 * Manager for the log.
 *
 * @author Christian Hochlin
 *
 */
public class TagLog {

        public static final int DELETE_TAG = 1;
        public static final int ENTER_ITEM = 2;
        public static final int ENTER_TAG = 3;
        public static final int RATE_ITEM = 6;

        private static String stmt = "INSERT INTO actions(userid, tagid, itemid,
                time, action, ip) VALUES (?, ?, ?, now(), ?, ?)";

        private final String selectstmt = "SELECT * FROM actions
                WHERE action = ? ORDER BY time DESC";

        public static void logEvent(Context context, int action, int user,
                int itemOrTag, String ip) throws SQLException
        {
                int item = 0;
                int tag = 0;
                if(action == ENTER_TAG)
                {
                        tag = itemOrTag;
                }
                else if(action == ENTER_ITEM || action == RATE_ITEM)
                {
                        item = itemOrTag;
                }
                DatabaseManager.updateQuery(context, stmt, user, tag, item, action, ip);
        }
        public ArrayList<LogItem> getTemporalVisitsByIP(Context context, int action)
         throws SQLException
```

```
{
        TableRowIterator it;
        String selectstmt = "SELECT date_trunc('day',time) as date, ip
                FROM actions WHERE action = ? ORDER BY time ASC";
        it = DatabaseManager.query(context, selectstmt, action);


        ArrayList<LogItem> list = new ArrayList<LogItem>();
        Date currDate = new Date();
        currDate.setDate(34);
        int x = -1;
        do
        {
                if(it.hasNext())
                {

                        TableRow row = it.next();
                        Date nowDate = row.getDateColumn("date");
                        String ip = row.getStringColumn("ip");
                        //Hvis det er samme dato, sjekk ip
                        if(currDate.equals(nowDate))
                        {
                                //Hvis ip for den datoen eksisterer,
                                //increaser den ikke count
                                LogItem i = list.get(x);
                                i.addIP(ip);
                                list.add(x, i);

                        }else{
                                //Lag nytt logitem, og gi den en i count
                                x++;
                                currDate = nowDate;
                                LogItem item = new LogItem(nowDate,1);
                                item.addIP(ip);
                                list.add(x,item);

                        }
                }
        }
        while(it.hasNext());

        return list;
}
public ArrayList<LogItem> getTemporalTagUsage(Context context)
throws SQLException
{
        TableRowIterator it;

                String selectstmt = "SELECT date_trunc('day', time) as date,
                 COUNT(*) as antall FROM taguse GROUP BY date_trunc('day', time)
                 ORDER BY date ASC";

                it = DatabaseManager.query(context, selectstmt);

                ArrayList<LogItem> list = new ArrayList<LogItem>();
                do
                {
                        if(it.hasNext())
                        {
                                TableRow row = it.next();

                                list.add(
                                        new LogItem(row.getDateColumn("date"),
                                                row.getLongColumn("antall"))
                                );
                        }
                }
                while(it.hasNext());

                return list;
}

public ArrayList<LogItem> getTemporalData(Context context, int action)
        throws SQLException
{
        TableRowIterator it;

                String selectstmt = "SELECT date_trunc('day', time) as date,
                 COUNT(*) as antall FROM actions WHERE action = ?
                 GROUP BY date_trunc('day', time)
                 ORDER BY date ASC";

                it = DatabaseManager.query(context, selectstmt, action);

                ArrayList<LogItem> list = new ArrayList<LogItem>();
                do
                {
                        if(it.hasNext())
                        {
                                TableRow row = it.next();

                                list.add(
                     new LogItem(row.getDateColumn("date"),
                      row.getLongColumn("antall"))
                    );
                }
            }
            while(it.hasNext());

            return list;
    }

    public static void logEvent(Context context, int action, int user,
     int item, int tag, String ip) throws SQLException
    {

        DatabaseManager.updateQuery(context, stmt, user, tag, item, action, ip);
    }

    public static void logEvent(Context context, int action, int user, String ip)
    throws SQLException
    {

        DatabaseManager.updateQuery(context, stmt, user, 0, 0, action, ip);
    }

    public ArrayList<LogItem> getLogItems(Context context, int action, int limit)
    throws SQLException
    {
        TableRowIterator it;
        if(limit == 0)
        {
            it = DatabaseManager.query(context, selectstmt, action);
        }
        else
        {
            String selectstmt = "SELECT * FROM actions WHERE action = ?
             ORDER BY time DESC LIMIT "+limit;

            it = DatabaseManager.query(context, selectstmt, action);
        }


        return iteratorToList(it);

    }
```

```
        public ArrayList<LogItem> getLogItems(Context context, int action)
         throws SQLException
        {

                return getLogItems(context, action, 0);

        }


        private ArrayList<LogItem> iteratorToList(TableRowIterator it)
         throws SQLException
        {
                ArrayList<LogItem> list = new ArrayList<LogItem>();
                do
                {
                        if(it.hasNext())
                        {
                                TableRow row = it.next();
                                list.add(new LogItem(
                                                row.getIntColumn("actionid"),
                                                row.getIntColumn("userid"),
                                                row.getIntColumn("tagid"),
                                                row.getIntColumn("itemid"),
                                                row.getDateColumn("time"),
                                                row.getIntColumn("action"),
                                                row.getStringColumn("ip")
                                ));
                        }
                        else
                        {
                                list.add(new LogItem(0,0,0,0, new Date(), 0, ""));
                        }

                }
                while(it.hasNext());

                return list;
        }


}
```

B.1.19 TagManager

```
        package no.hig.xmlui.aspect.overview;
/**
```

```
 * This class manages everything that has to do with tags in this Dspace aspect
 *
 * @author Christian Hochlin
 */
public class TagManager {

        private Context _context;


        public TagManager(Context c)
        {
                _context = c;

        }

        /**
         * Adds tags to the item from a string of comma separated values.
         *
         *
         * @param Eperson user
         * @param String itemID
         * @param String tags
         * @throws WingException
         *
         */
        public void addTagsFromCSVString(int user, int itemID, String tags, String ip )
         throws SQLException, WingException
        {


                String[] tagArray = tags.split(",");
                for(String tag : tagArray)
                {

                        insertTag(user, itemID, tag.trim(), ip);
                }
        }

        /**
         *
         * Fetches all tags.
         * @return ArrayList<Tag>
         */
        public ArrayList<Tag> getTags() throws SQLException
        {
                TableRowIterator iterator = DatabaseManager.query(_context,
                        "SELECT tag.id, tag.tag FROM taguse, tag " +
                        "WHERE " +
                                        " tag.id = taguse.tagid" +
                                        " AND taguse.deleted = 0");

        return iteratorToList(iterator);
}

/**
 * Fetches a tag by its ID
 *
 * @param int id
 * @return Tag
 */
public Tag getTagFromID(int id) throws SQLException
{
        if(id == -1)
        {
                return new Tag("No_Tag", -1, 1);
        }
        else if(id == 0)
        {
                return new Tag("", -1,1);
        }
        else
        {
                TableRow tr = DatabaseManager.querySingle(_context,
                 "SELECT tag.id,tag.tag, count(taguse.tagid) FROM taguse, tag " +
                 "WHERE tag.id = ? AND tag.id = taguse.tagid " +
                 "GROUP BY tag.id, tag.tag", id);

                if(tr == null)
                {
                        return new Tag("", -1,1);
                }
                else
                {
                        return new Tag(tr.getStringColumn("tag"),
                                tr.getIntColumn("id"),
                                (int)tr.getLongColumn("count"));
                }

        }
}

/**
```

```
 * Fetches tags related to the item from the database
 *
 * @param item
 * @return ArrayList<Tag>
 */
public ArrayList<Tag> getItemTags(Item item) throws SQLException
{
        TableRowIterator iterator = DatabaseManager.query(_context,
                        "SELECT tag.id, tag.tag FROM taguse, tag, item " +
                        "WHERE " +
                                        " taguse.itemid = "+item.getID() +
                                        " AND item.item_id = taguse.itemid" +
                                        " AND tag.id = taguse.tagid" +
                                        " AND taguse.deleted = 0");

        return iteratorToList(iterator);
}

/**
 * Fetches tags the user has tagged an item with
 *
 * @param item
 * @return Tag[]
 */
public ArrayList<Tag> getItemTagsByOwner(Item item, EPerson person) throws SQLException
{

        TableRowIterator iterator = DatabaseManager.query(_context,
        "SELECT tag.id, tag.tag FROM taguse, tag, item, eperson " +
        "WHERE " +
                                "taguse.userid = "+person.getID()+
                        " AND taguse.itemid = "+item.getID() +
                        " AND taguse.userid = eperson.eperson_id" +
                        " AND item.item_id = taguse.itemid " +
                        " AND taguse.tagid = tag.id" +
                        " AND taguse.deleted = 0");

        return iteratorToList(iterator);


}

public ArrayList<Tag> getItemTagsByOwner(EPerson person) throws SQLException
{
        TableRowIterator iterator = DatabaseManager.query(_context,
                        "SELECT tag.id, tag.tag FROM taguse, tag, eperson " +
                    "WHERE " +
                                             "taguse.userid = "+person.getID(
                                  " AND taguse.userid = eperson.eperson_id"
                                  " AND taguse.tagid = tag.id" +
                                  " AND taguse.deleted = 0");

                    return iteratorToList(iterator);
}
/**
 * Method to create a list based on an iterator.
 * How Tag objects are made, may change in the future,
 * so I'm keeping all related code in one place.
 *
 * @param TableRowIterator
 * @return ArrayList<Tag>
 */
private ArrayList<Tag> iteratorToList(TableRowIterator it) throws SQLException
{

        HashMap<String, Tag> hm = new HashMap();

        ArrayList<Tag> list = new ArrayList<Tag>();
        do
        {
        //Need to have an easy way to identify unique tags for our tag cloud.
        //Hashmaps work better than lists for this
                if(it.hasNext())
                {
                        TableRow row = it.next();
                        String key = row.getStringColumn("tag");
                        Tag tag = new Tag(key, row.getIntColumn("id"));
                        if(hm.containsKey(key))
                        {
                                int count = hm.get(key).getCount();
                                tag.setCount(count + 1);

                        }
                        hm.put(key, tag);
                }
                else
                {
                        hm.put("No Tag", new Tag("No Tag",-1));
                }

        }
        while(it.hasNext());
```

```
                Object[] c = hm.values().toArray();

                for(int x = 0;x < c.length;x++)
                {
                        list.add((Tag) c[x]);
                }

                return list;
        }

        /**
         * Adds a tag to the database
         *
         */
        private void insertTag(int person, int item, String tag, String ip)
        {
                int isPublic = isPublic(tag);
                int tagID = 0;
                tag = tag.toLowerCase();

                try
                {
                        //Sjekker om tagen allerede eksisterer.
                        //trusting UNIQUE in DB throws an error,
                        //preventing the remaining statements from completing
                        TableRow tr = DatabaseManager.querySingle(_context,
                         "SELECT id FROM tag WHERE tag = ?", tag);

                        if(tr == null)
                        {
                                DatabaseManager.updateQuery(_context,
                                 "INSERT INTO tag(tag) VALUES (?)", tag);

                                TableRowIterator it = DatabaseManager.query(_context,
                                 "SELECT currval('tag_id_seq') AS id");

                                tagID = (int)it.next().getLongColumn("id");
                        }
                        else
                        {
                                tagID = tr.getIntColumn("id");
                        }


                        TableRow tr2 = DatabaseManager.querySingle(_context,
                 "SELECT tagid FROM taguse " +
                 "WHERE tagid = ? AND itemid = ? AND userid = ?" +
                 " AND deleted = 1", tagID, item, person);

                if(tr2 != null)
                {
                //Tag is already there. Undelete it.
                        DatabaseManager.updateQuery(_context,
                        "UPDATE tagUse SET deleted=0" +
                        " WHERE itemid = ? AND tagid = ?" +
                        " AND userid = ?", item, tagID, person);

                }
                else
                {
                        //Tag doesn't exist. Create it.
                        DatabaseManager.updateQuery(_context,
                         "INSERT INTO tagUse(itemid,tagid,userid,time,
                           visibility, deleted, ip)" +
                         " VALUES (?, ?, ?, now(), ?, 0, ?)", item,
                           tagID, person, isPublic, ip);

                }

        }catch(SQLException e)
        {
        }
}

/**
 * Method to determine if a tag is public or not.
 *
 * @return true, false
 */
private int isPublic(String tag)
{
        return tag.startsWith("*") ? 0 : 1;
}
/**
 *
 * Deletes a tag, checking if the current user id allowed to delete the tag.
 *
 */
public void deleteTag(int itemID, int personID, int tagID, String ip)
 throws SQLException
{
```

```
                if(AuthorizeManager.isAdmin(_context))
                {
                        DatabaseManager.updateQuery(_context,
                         "UPDATE tagUse SET deleted=1" +
                         " WHERE itemid = ? AND tagid = ?", itemID, tagID);
                }
                else
                {
                        DatabaseManager.updateQuery(_context,
                         "UPDATE tagUse SET deleted=1" +
                         " WHERE itemid = ? AND tagid = ? AND userid = ?",
                           itemID, tagID, personID);

                }
                TagLog.logEvent(_context, TagLog.DELETE_TAG, personID, itemID, tagID, ip);


        }


}
```

B.2 Retrieval Algorithms

B.2.1 HITS

```
package HITS;

public class ModifiedHITS {

        private HashMap<Integer, Document> dokumenter;
        private HashMap<Integer, Tag> tagger;

        public static int HIGH_WEIGHT = 5;
        public static int LOW_WEIGHT = 1;
        public static int MEDIUM_WEIGHT = 3;


        private Matrix hubWeight;
        private Matrix authWeight;
        private ArrayList<Integer> sourceTags;
        private ArrayList<Integer> extTags;


        private int iterateTimes = 2;

        public ModifiedHITS(HashMap<Integer, Document> docs, HashMap<Integer, Tag> tags)
        {
```

```
        dokumenter = docs;
        tagger = tags;

        authWeight = new Matrix(docs.size(), 1);
        hubWeight = new Matrix(tags.size(), 1);
}


public HashMap<Integer, Integer> findRelatedDocuments(int id)
{

        Document d = dokumenter.get(id);

        HashMap<Integer, Integer> result = new HashMap<Integer, Integer>();
        sourceTags = new ArrayList<Integer>();

        //Preload sourceTags-array with IDs
        for(int x = 0;x < d.getNumberOfTags();x++)
        {
                sourceTags.add(d.getTag(x).getId());
        }

        for(int x = 0;x < d.getNumberOfTags();x++)
        {

                //For hver tag i dokumentet
                Matrix weight = getAuthorityVector(d.getTag(x));

                result = addWeightVector(weight, result);


        }
        return result;

}
/**
 * Adds the supplied weight to an existing hashmap of weights.
 * @param weight
 * @param target
 * @return
 */
private HashMap<Integer, Integer> addWeightVector(Matrix weight,
        HashMap<Integer, Integer> target)
{
        for(int x = 0;x < weight.documentSize;x++)
        {
```

```
                        int docID = weight.getRowID(x);

                        int newValue = weight.getValue(x, 0);
                        if(target.containsKey(docID))
                        {
                                int value = target.get(docID);
                                value += newValue;
                                target.put(docID, value);
                        }
                        else
                        {
                                target.put(docID, newValue);
                        }

                }

                return target;

        }
        /**
         * This method generates a list of tag IDs for use when weighting, based on a given set.
         * @return ArrayList<Integer>
         */
        private ArrayList<Integer> generateListOfTagsByID(HashMap<Integer, Document> set)
        {
                ArrayList<Integer> tagList = new ArrayList<Integer>();
                Iterator it = set.entrySet().iterator();
            while (it.hasNext()) {
                    Map.Entry<Integer,Document> pairs = (Map.Entry<Integer, Document>)it.next();
                    Document dok = pairs.getValue();


                        //Preload sourceTags-array with IDs
                        for(int x = 0;x < dok.getNumberOfTags();x++)
                        {
                                if(!tagList.contains(dok.getTag(x).getId()))
                                        tagList.add(dok.getTag(x).getId());
                        }
            }
            return tagList;
        }


        /**
         * Gets the authority vector from the supplied tag
```

```
 * @param tag
 * @return
 */
private Matrix getAuthorityVector(Tag tag)
{
        //Get Root set based on given tag
        HashMap<Integer, Document> set1 = getListOfDocumentsWith(tag);

        HashMap<Integer, Document> extendedSet = new HashMap<Integer, Document>()

        //Set of all tags related to documents in extendedSet
        ArrayList<Integer> tagSet = new ArrayList<Integer>();

        //Iterate over each document in set 1
        //to get all related tags and documents
        Iterator it = set1.entrySet().iterator();
    while (it.hasNext()) {
            Map.Entry<Integer,Document> pairs =
                    (Map.Entry<Integer, Document>)it.next();
            Document dok = pairs.getValue();


            //Processing documents
            for(int x = 0;x < dok.getNumberOfTags();x++)
                {

                        //Checks if tag already exists
                        if(!tagSet.contains(dok.getTag(x).getId()))
                                tagSet.add(dok.getTag(x).getId());

                        //Add all documents related to current tag
                        extendedSet =
                                getListOfDocumentsWith(dok.getTag(x), extendedSet
                }
    }

    Matrix matrix = createMatrix(extendedSet, tagSet);

    Matrix initWeight = createInitialWeightVector(extendedSet, tagSet);

    Matrix result = iterativeWeightComputation(matrix, initWeight);



   return result;
```

```
        }


        /**
         * Creates the initial weight vector.
         *
         * @return Matrix
         */
        private Matrix createInitialWeightVector(HashMap<Integer, Document> extendedSet,
                ArrayList<Integer> tagSet )
        {
                //Create initial weight array
            Matrix initWeight = new Matrix(extendedSet.size() + tagSet.size(), 1);
            initWeight.setDocumentSize(extendedSet.size());
            initWeight.setTagSize(tagSet.size());

            int[] lowWeightRow = {LOW_WEIGHT};
            int[] highWeightRow = {HIGH_WEIGHT};
            int[] zeroWeightRow = {0};

            Iterator it2 = extendedSet.entrySet().iterator();
            while (it2.hasNext()) {
                    Map.Entry<Integer,Document> pairs = (Map.Entry<Integer, Document>)it2.next();
                    Document dok = pairs.getValue();


                    initWeight.addRow(lowWeightRow, Matrix.DOCUMENT, dok.getId());
            }

            //Add differing weight for tags in different "sets"
            for( int tagID : tagSet)
            {
                    if(tagID == 9)
                    {
                            //Quick fix for a "tag" which wasn't possible
                            //to remove from all the documents in the blacklist
                            initWeight.addRow(zeroWeightRow, Matrix.TAG, tagID);
                    }
                    else if(sourceTags.contains(tagID))
                    {
                            //Give greatest weight to this tag
                            initWeight.addRow(highWeightRow, Matrix.TAG, tagID);
                    }

                    else
                    {
```

```
                    //Give this low weight
                    initWeight.addRow(lowWeightRow, Matrix.TAG, tagID);

                }

            }

        return initWeight;
    }

    /**
     * Computes the hub and Authority weights
     * @param matrix
     * @return
     */
    private Matrix iterativeWeightComputation(Matrix matrix, Matrix weight)
    {

        Matrix result = new Matrix(weight);

        hubWeight = new Matrix(weight);
        for(int x = 0;x < iterateTimes;x++)
        {


            Matrix matrixT = matrix.transpose();

            result = matrixT.multiplyBy(hubWeight);
            result.normalize();

            hubWeight = new Matrix(result.getRows(), 1);
            hubWeight.setDocumentSize(result.documentSize);
            hubWeight.setTagSize(result.tagSize);


            hubWeight = matrix.multiplyBy(result);
            hubWeight.normalize();

        }

        return result;




    }
```

```
/**
 * Creates a HITS matrix based on the supplied tags and documents
 * @param docSet
 * @param tagSet
 * @return
 */
private Matrix createMatrix(HashMap<Integer, Document> docSet,
        ArrayList<Integer> tagSet)
{
        //Lag ny matrise i korrekt str
    int size = docSet.size() + tagSet.size();
    Matrix matrix = new Matrix(size, docSet.size(), tagSet.size());

    //Fill document cluster
    matrix.fillDocumentCluster(docSet);

    //Fill tag Cluster
    for (int tagId : tagSet){

            int[] row = new int[size];

            Arrays.fill(row, 0);

            int x = 0;
            Iterator it1 = docSet.entrySet().iterator();
            //iterate horizontally
            while (it1.hasNext()) {
                    Map.Entry<Integer,Document> pairs2 =
                            (Map.Entry<Integer, Document>)it1.next();
                    Document dok = pairs2.getValue();
                    if(dok.hasTagById(tagId))
                    {
                            row[x] = 1;
                    }
                    x++;
            }

            matrix.addRow(row,Matrix.TAG,tagId);


    }

        return matrix;

}
```

```java
/**
 * Gets all documents containing the given tag and appends
 * them to the supplied list.
 *
 * @param tag, list
 * @return Array of document IDs
 */
private HashMap<Integer, Document> getListOfDocumentsWith(Tag tag,
        HashMap<Integer, Document> map)
{

        Iterator it = dokumenter.entrySet().iterator();
    //iterer horisontalt
        int count = 0;
    while (it.hasNext()) {
            Map.Entry<Integer,Document> pairs =
                    (Map.Entry<Integer, Document>)it.next();
            Document dok = pairs.getValue();

            int x = 0;
                while(x<dok.getNumberOfTags())
                {
                        if(tag.getId() == dok.getTag(x).getId())
                        {

                                //If it doesn't exist, add the document
                                if(!map.containsKey(dok.getId()))
                                {
                                        count++;
                                        map.put(dok.getId(), dok);

                                }

                                break;
                        }
                        else
                        {
                                x++;
                        }
                }

        }


        return map;
        }


        /**
         * Gets all documents containing the given tag.
         *
         * @param tag
         * @return Map of documents with their id as key
         */
        private HashMap<Integer, Document> getListOfDocumentsWith(Tag tag)
        {
                HashMap<Integer, Document> map = new HashMap<Integer, Document>();
                return getListOfDocumentsWith(tag, map);
        }


}
```

B.2.2 Matrix

Custom class to represent a matrix, with some convenience methods for use by hits

```
package HITS;

public class Matrix {

        private int[][] matrix;
        private int rows;
        private int columns;
        private int lastInsertedRow;
        private int lastInsertedColumn;

        public int documentSize;
        public int tagSize;
        public int[] documentID;
        public int[] tagID;

        public static final int DOCUMENT = 1;
        public static final int TAG = 2;

        /**
         * Clone constructor
         * @param Matrix
         */
        public Matrix(Matrix m) {
                matrix = new int[m.getRows()][m.getColumns()];
                for(int x = 0;x < m.getRows();x++)
                {
```

```
                for(int y = 0;y < m.getColumns();y++)
                {
                        matrix[x][y] = m.getValue(x, y);
                }
        }

        this.rows = m.rows;
        this.columns = m.columns;
        this.lastInsertedColumn = m.lastInsertedColumn;
        this.lastInsertedRow = m.lastInsertedRow;


        this.documentSize = m.documentSize;
        this.tagSize = m.tagSize;


        documentID = new int[m.documentSize];
        tagID = new int[m.tagSize];
        for(int x = 0;x < m.tagSize;x++)
        {
                tagID[x] = m.tagID[x];
        }
        for(int y = 0;y < m.documentSize;y++)
        {
                documentID[y] = m.documentID[y];
        }

  }

/**
 * Normalizes the hub and authority-values.
 * Should only be used on weight-matrices
 */
public void normalize()
{

        if(columns > 1)
                throw new RuntimeException("Not a weight Matrix");

        int x = 0;
        double authSum = 0;
        double hubSum = 0;
        while(x < documentSize)
        {
                authSum += this.getValue(x, 0);
                x++;
        }

        while(x < tagSize + documentSize)
        {

                hubSum += this.getValue(x, 0);

                x++;
        }

        hubSum = Math.sqrt(hubSum);
        authSum = Math.sqrt(authSum);

        x = 0;
        while(x < documentSize)
        {

                double value = (this.getValue(x, 0)/authSum)*10;
                this.insertValue(x, 0, (int) value);
                x++;
        }
        while(x < tagSize + documentSize)
        {
                double value = (this.getValue(x, 0)/hubSum)*10;
                this.insertValue(x, 0, (int)value);
                x++;
        }
}

/**
 * Creates a n-by-n matrix, and sets number of documents and tags
 * @param n
 * @param documents
 * @param tags
 */
public Matrix(int n, int documents, int tags)
{
        this(n,n);
        tagSize = tags;
        documentSize = documents;
        documentID = new int[documents];
        tagID = new int[tags];

}
public Matrix(int n, int m)
```

```
{
        lastInsertedRow = 0;
        lastInsertedColumn = 0;
        //Creates an n-by-m matrix of zeros
        matrix = new int[n][m];
        this.rows = n;
        this.columns = m;
        for(int x = 0;x < n;x++)
        {
                for(int y = 0;y<m;y++)
                {
                        this.matrix[x][y] = 0;
                }
        }
}

/**
 * Convenience method to fill the document part with zeros
 */
public void fillDocumentCluster(HashMap<Integer, Document> docs)
{
        if(docs.size() != documentSize)
                throw new RuntimeException("Document sizes do not match");

        Iterator it = docs.entrySet().iterator();
    while (it.hasNext()) {
            Map.Entry<Integer,Document> pairs =
                    (Map.Entry<Integer, Document>)it.next();
            Document dok = pairs.getValue();

            int[] row = new int[columns];
            Arrays.fill(row, 0);
            addRow(row, DOCUMENT, dok.getId());

    }
}

public void setTagSize(int t)
{
        tagSize = t;
        tagID = new int[t];
}
public void setDocumentSize(int d)
{
        documentSize = d;
        documentID = new int[d];
        }

        public int[] HubWeight(int size)
        {

                if(columns != 1)
                        throw new RuntimeException("Invalid matrix size");

                int initPos = rows - size;
                int[] hubWeight = new int[size];

                for(int x = 0;x < size;x++)
                {
                        hubWeight[x] = matrix[initPos + x][0];
                }
                return hubWeight;

        }

        public int[] AuthorityWeight(int size)
        {
                if(columns != 1)
                        throw new RuntimeException("Invalid matrix size");

                int[] authorityWeight = new int[size];

                for(int x = 0;x < size;x++)
                {
                        authorityWeight[x] = matrix[x][0];
                }
                return authorityWeight;
        }

        public int getRows()
        {
                return rows;
        }
        public int getColumns()
        {
                return columns;
        }
        public int getValue(int row,int column)
        {

                return matrix[row][column];
        }
```

```
public void insertValue(int row, int column, int value)
{
        matrix[row][column] = value;
}

private void incrementLastInsertedRow()
{
        if(lastInsertedRow >= rows)
                throw new IndexOutOfBoundsException("Can't add more rows.");

        lastInsertedRow++;

}
private void incrementLastInsertedColumn()
{
        if(lastInsertedRow >= columns)
                throw new IndexOutOfBoundsException("Can't add more columns.");

        lastInsertedColumn++;

}



public int getRowID(int row)
{
        if(row >= documentSize)
        {
                System.out.println(row);
                return tagID[row-documentSize];
        }
        else if(row < documentSize)
        {

                return documentID[row];
        }
        else
                throw new IndexOutOfBoundsException("Position out of bounds");
}
private void setTagID(int position, int id)
{
        tagID[position-documentSize] = id;
}
private void setDocumentID(int position, int id)
{
        documentID[position] = id;
```

```
    }

    public void addRow(int[] row, int type, int typeID )
    {
            addRow(lastInsertedRow, row, type, typeID);
            incrementLastInsertedRow();
    }
    public void addRow(int position, int[] row, int type, int typeID)
    {
            //Preserves the id related to each row
            if(type == TAG)
            {
                    setTagID(position, typeID);
            }
            else if(type == DOCUMENT)
            {
                    setDocumentID(position, typeID);
            }
            else
            {
                    throw new IllegalArgumentException(
                            "Type needs to refer to either tag or document");
            }


            //If the row differs from the matrix size
            if(columns != row.length)
                    throw new IllegalArgumentException(
                             "The supplied length of the row is not "+
                             "compatible with the matrix size");

            //If position is out of bounds
            if(position < 0 || position > rows -1)
                    throw new IndexOutOfBoundsException("Position out of bounds");


            for(int x = 0;x < columns;x++)
            {
                    matrix[position][x] = row[x];
            }


    }

    public Matrix transpose()
    {
        Matrix transposedMatrix = new Matrix(columns,rows);

        for(int x = 0;x < columns;x++)
        {
                for(int y = 0;y < rows;y++)
                {
                        transposedMatrix.insertValue(x, y, getValue(y,x) );
                }
        }
        return transposedMatrix;
}



public Matrix multiplyBy(Matrix matrix)
{
        if(matrix.getRows() != getColumns())
                throw new RuntimeException("Invalid matrix sizes");


        Matrix newMatrix = new Matrix(matrix);
        for(int x = 0;x < getRows();x++)
        {
                for(int y = 0;y < matrix.getColumns();y++)
                {

                        int value = 0;
                        for(int z = 0;z < getColumns();z++)
                        {
                                value += getValue(x,z) * matrix.getValue(z,y);

                        }
                        newMatrix.insertValue(x, y, value);
                }
        }

        return newMatrix;
}

public String toString()
{
        for(int x = 0;x < rows;x++)
        {
                String s = getRowID(x)+"";

                if(s.length() == 1)
```

```
                        {

                                System.out.print(getRowID(x)+"  ");
                        }
                        else if(s.length() == 2)
                        {

                                System.out.print(getRowID(x)+"  ");
                        }
                        else
                        {
                                System.out.print(getRowID(x)+" ");

                        }

                        for(int y = 0;y<columns;y++)
                        {

                                System.out.print(this.matrix[x][y]+" ");
                        }
                        System.out.println(" ");

                }
                return "";

        }

}
```

B.2.3 Vector Space

```
package plain;

public class VectorScoring {

        private VectorWrapper vw;
        private HashMap<Integer, Document> docList;
        public VectorScoring(HashMap<String, Tag> tagList, HashMap<Integer, Document> docList)
        {
                vw = new VectorWrapper(tagList, docList.size());
                this.docList = docList;
        }

        public HashMap<Document, Double> getRelevantDocuments(Document d)
        {

                Vector<Double> queryVector = vw.createWeightVector(d);
                HashMap<Document, Double> scoreMap = new HashMap<Document, Double>();

                Set<Map.Entry<Integer, Document>> postingSet = docList.entrySet();
                for (Map.Entry<Integer, Document> entry : postingSet) {
                        Document doc = entry.getValue();
                        double score = computeScore(queryVector,
                                vw.createWeightVector(doc));
                        scoreMap.put(doc, score);
                }

                return scoreMap;

        }
        private double computeScore(Vector<Double> query, Vector<Double> D)
        {
                double value = 0.0;
                double qAbs = getAbsoluteValue(query);
                double tAbs = getAbsoluteValue(D);
                for(int x = 0;x < query.size();x++)
                {
                        value += query.get(x)*D.get(x);
                }

                double r = tAbs*qAbs;
                return value/r;
        }
        private double getAbsoluteValue(Vector<Double> v)
        {
                double qAbs = 0.0;
                for(Double t : v)
                {
                        qAbs += t*t;
                }
                return Math.sqrt(qAbs);
        }
}
```

B.2.4 Vector Wrapper

Convenience class to create a vector for Vector Space

```
package plain;

public class VectorWrapper {

        private HashMap<String, Tag> tagList;
        private int collectionSize;
        public VectorWrapper(HashMap<String, Tag> tagList, int collectionSize)
        {

                 this.tagList = tagList;
                 this.collectionSize = collectionSize;

        }

        /**
         * Create a vector with weights, based on the supplied Document d
         * @param d
         * @return Vector<Double>
         */
        public Vector<Double> createWeightVector(Document d)
        {
                Vector<Double> queryVector = new Vector<Double>();
                queryVector.ensureCapacity(collectionSize);

                Set<Map.Entry<String, Tag>> postingSet = tagList.entrySet();
                for (Map.Entry<String, Tag> entry : postingSet) {
                        Tag tag = entry.getValue();

                        //W = tf * log (N/df)

                        double N = (double)collectionSize;
                        double termFrequency = (double)d.getTagFrequency(tag.getId());
                        double documentFrequency = (double)tag.getCount();
                        double weight = termFrequency*Math.log10(N/documentFrequency);
                        queryVector.add(weight);


                }
                return queryVector;

        }

}
```

B.2.5 Document

```
package reading;
/**
 * Class representing a Document
 *
 * @author Christian Hochlin
 *
 */
```

```java
public class Document implements Serializable {

        private int id;

        private String name;


        private ArrayList<Tag> tags;
        private HashMap<Integer, Integer> tagFrequency;

        public int getTagFrequency(int id)
        {
                if(tagFrequency.get(id) == null)
                {
                        return 0;
                }
                else
                {
                        return tagFrequency.get(id);
                }
        }
        public Document(int id, String name)
        {
                this.id = id;
                this.name = name;
                tags = new ArrayList<Tag>();
                tagFrequency = new HashMap<Integer, Integer>();
        }
        public boolean hasTagById(int tagid)
        {
                for(Tag tagIterate : tags)
                {
                        if(tagIterate.getId() == tagid)
                        {
                                return true;
                        }
                }
                return false;
        }
        public boolean hasTag(Tag tag)
        {
                return hasTagById(tag.getId());
        }

        public void addTag(Tag tag)
        {
                if(hasTag(tag))
                {

                        int count = tagFrequency.get(tag.getId());
                        count++;
                        //If exists, increase count
                        tagFrequency.put(tag.getId(), count);
                }
                else
                {

                        tags.add(tag);
                        tagFrequency.put(tag.getId(), 1);
                }
        }

        public int getId()
        {
                return id;
        }

        public String getName()
        {
                return name;
        }
        public ArrayList<Tag> getTags()
        {
                return tags;
        }

        public Tag getTag(int pos)
        {
                return tags.get(pos);
        }
        public int getNumberOfTags()
        {
                return tags.size();
        }
}
```

B.2.6 Tag

```
package reading;

/**
 * Class representing a tag
 *
 * @author Christian Hochlin
 *
 */
public class Tag implements Serializable {

        private int id;
        private String name;
        private boolean added;

        //Documents containing tag
        private int count;

        public Tag(int id, String name)
        {
                this.id = id;
                this.name = name;
                this.count = 1;
        }

        public void increaseCount()
        {
                count++;
        }
        public int getCount()
        {
                return count;
        }
        public String getName()
        {
                return name;
        }

        public int getId()
        {
                return id;
        }

        public void setAdded(boolean newValue)
        {
                added = newValue;
        }
        public boolean isAdded()
        {
                return added;
        }
}
```

B.2.7 Index

```
package reading;

/**
 * Keeps an index of all terms. Generates new ones if an index doesn't exist.
 *
 * @author Christian Hochlin
 */
public class Index {

        public static final String DBPATH = "test";
        public static final String ANN_PATH = DBPATH + "/annotations_complete_eng";

        public static final String SEPERATOR_REGEXP = "\\s++|\\.|\\,|\\;|\\:|\\’|\\\"|\\‘|\\’";

        //List of stop words
        private static final Set<String> BLACKLIST = new HashSet<String>(
    Arrays.asList(new String[] { "a", "about", "above", "above", "across", "after", "afterwards"
, "again", "against", "all", "almost", "alone", "along", "already", "also","although"
,"always","am","among", "amongst", "amoungst", "amount",  "an", "and", "another", "any"
,"anyhow","anyone","anything","anyway", "anywhere", "are", "around", "as",  "at", "back"
,"be","became", "because","become","becomes", "becoming", "been", "before", "beforehand"
, "behind", "being", "below", "beside", "besides", "between", "beyond", "bill", "both"
, "bottom","but", "by", "call", "can", "cannot", "cant", "co", "con", "could", "couldnt"
, "cry", "de", "describe", "detail", "do", "done", "down", "due", "during", "each", "eg"
, "eight", "either", "eleven","else", "elsewhere", "empty", "enough", "etc", "even", "ever"
, "every", "everyone", "everything", "everywhere", "except", "few", "fifteen", "fify", "fill"
, "find", "fire", "first", "five", "for", "former", "formerly", "forty", "found", "four", "from"
, "front", "full", "further", "get", "give", "go", "had", "has", "hasnt", "have", "he", "hence"
, "her", "here", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "him", "himself"
, "his", "how", "however", "hundred", "ie", "if", "in", "inc", "indeed", "interest", "into"
, "is", "it", "its", "itself", "keep", "last", "latter", "latterly", "least", "less", "ltd"
, "made", "many", "may", "me", "meanwhile", "might", "mill", "mine", "more", "moreover"
, "most", "mostly", "move", "much", "must", "my", "myself", "name", "namely", "neither"
, "never", "nevertheless", "next", "nine", "no", "nobody", "none", "noone", "nor", "not"
, "nothing", "now", "nowhere", "of", "off", "often", "on", "once", "one", "only", "onto"
, "or", "other", "others", "otherwise", "our", "ours", "ourselves", "out", "over", "own"
,"part", "per", "perhaps", "please", "put", "rather", "re", "same", "see", "seem", "seemed"
, "seeming", "seems", "serious", "several", "she", "should", "show", "side", "since"
, "sincere", "six", "sixty", "so", "some", "somehow", "someone", "something", "sometime"
, "sometimes", "somewhere", "still", "such", "system", "take", "ten", "than", "that"
, "the", "their", "them", "themselves", "then", "thence", "there", "thereafter", "thereby"
, "therefore", "therein", "thereupon", "these", "they", "thickv", "thin", "third", "this"
, "those", "though", "three", "through", "throughout", "thru", "thus", "to", "together"
, "too", "top", "toward", "towards", "twelve", "twenty", "two", "un", "under", "until"
, "up", "upon", "us", "very", "via", "was", "we", "well", "were", "what", "whatever"
, "when", "whence", "whenever", "where", "whereafter", "whereas", "whereby", "wherein"
, "whereupon", "wherever", "whether", "which", "while", "whither", "who", "whoever"
, "whole", "whom", "whose", "why", "will", "with", "within", "without", "would", "yet"
, "you", "your", "yours", "yourself", "yourselves", "the"}));

        protected HashMap<String, Tag> tagList;
        protected HashMap<Integer, Document> docList;
        protected ArrayList<String[]> entries; // [location, content]

        public void printIndex() {
                Set<Map.Entry<String, Tag>> postingSet = tagList.entrySet();
                for (Map.Entry<String, Tag> entry : postingSet) {
                        System.out.println(entry.getValue().getName());
                }

        }
        public Index() {

                tagList = new HashMap<String, Tag>();
                docList = new HashMap<Integer, Document>();


                if (!indexExists()) {
                        DirReader leser = new DirReader();
                        entries = leser.harvestFileContent(ANN_PATH, true);
                        generateIndex();

                        saveIndex();
                } else {

                        loadIndex();
                }

        }

        public HashMap<String, Tag> getTagList()
        {
                return this.tagList;
        }
        public HashMap<Integer, Document> getDocumentList()
        {
                return this.docList;
        }

        private void generateIndex() {
```

```
int index = 0;
int innerIndex = 0;
for (String[] text : entries) {
        String[] words1 = text[1].toLowerCase().split(
                        SEPERATOR_REGEXP);

        Document doc = new Document(index, text[0]);
        docList.put(index, doc);


        for (String word : words1) {
                if (!BLACKLIST.contains(word)) {

                        if (!tagList.containsKey(word)) {

                                Tag tag = new Tag(innerIndex, word);
                                tagList.put(word, tag);
                                doc.addTag(tag);
                                innerIndex++;
                        //Increase innerindex when new tags are added,
                        //to push id up one.

                        }
                        else
                        {
                         Tag t = tagList.get(word);

                         //Increases count of tag only on first occurrence.
                         //jmf bilde 14 i lecture 3, indexing
                         if(!doc.hasTagById(t.getId()))
                         {
                                 t.increaseCount();
                                tagList.put(word, t);
                                doc.addTag(tagList.get(word));

                         }
                         else
                         {
                                //Increase count when tag already exists
                                doc.addTag(tagList.get(word));
                         }
                        }
                }
        }
        ++index;
}
```

```
}

private boolean indexExists() {
        return new File("index.dat").exists()
                && new File("indexTags.dat").exists();

}

@SuppressWarnings("unchecked")
private void loadIndex() {

        FileInputStream fis = null;
        ObjectInputStream in = null;
        try {
                fis = new FileInputStream("indexTags.dat");
                in = new ObjectInputStream(fis);
                tagList = (HashMap<String, Tag>) in.readObject();
                in.close();

                fis = new FileInputStream("index.dat");
                in = new ObjectInputStream(fis);
                docList = (HashMap<Integer, Document>) in.readObject();
                in.close();
        } catch (IOException ex) {
                ex.printStackTrace();
        } catch (ClassNotFoundException ex) {
                ex.printStackTrace();
        }

}

private void saveIndex() {
        try {
                FileOutputStream fout = new FileOutputStream("index.dat");
                ObjectOutputStream oos = new ObjectOutputStream(fout);
                oos.writeObject(docList);
                oos.close();

                FileOutputStream fout2 = new FileOutputStream("indexTags.dat");
                ObjectOutputStream oos2 = new ObjectOutputStream(fout2);
                oos2.writeObject(tagList);
                oos2.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
        }



}
```

B.2.8 Reader

A class for reading data from the test set.

```
package reading;

public class DirReader {

        public static final int ANN_ARR_DOCNO = 0;
        public static final int ANN_ARR_TEXTDATA = 1;
        public static final int ANN_ARR_IMAGEFILE = 2;


        public static final String ANN_ENCODING = "ISO-8859-15";
        public static final String ANN_FILE_POSTFIX = ".eng";
        public static final String ANN_FILE_PREFIX = "annotations";
        public static final String IMAGE_FILE_POSTFIX = ".jpg";
        public static final String IMAGE_FILE_PREFIX = "images";

        protected ArrayList<String[]> textArray;
        protected HashMap<String, String> textMap;

        ArrayList<String[]> array;
        Document d;

        public DirReader() {
                textArray = new ArrayList<String[]>();
                array = new ArrayList<String[]>();


        }

        /**
         * Reads a directory, and returns all content from files.
         *
         * @param directory
         *            The directory to access.
         * @param recursive
         *            Indicates if the method should traverse all sub-directories.
         *
         * @return An ArrayList<String> with the content of each file as entries.
```

```
 */
public ArrayList<String[]> harvestFileContent(String directory,
                boolean recursive) {


        File myDir = new File(directory);
        File[] files = null;
        if (myDir.exists() && myDir.isDirectory()) {
                files = myDir.listFiles();
                readDir(files, recursive);
        } else {
                System.out.println("Directory location not valid");
        }

        return textArray;

}




/**
 * Creates an Image from a path, and passes it to the Histogram-class to
 * create histogram
 *
 * @param subPath
 *            , string in the form of "00/25"
 *
 *
 */
private void processImage(String subPath) {
        String path = Index.DBPATH + "/" + subPath;
        BufferedImage image = null;
        try {

                File file = new File(path);
                image = ImageIO.read(file);

        } catch (IOException e) {
                System.out.println("Could not read image: " + path);
                e.printStackTrace();
        }


}

private void readDir(File[] files, boolean recursive) {
```

```
                for (File file : files) {
                        // Read all files in the list that are not
                        //directories and has the right prefix.
                        if (file.exists() && file.isFile()
                                        && file.getName().endsWith(ANN_FILE_POSTFIX)) {

                                String[] annotation = parseXml(file);

                                processImage(annotation[ANN_ARR_IMAGEFILE]);

                                textArray.add(annotation);

                        }
                        // Otherwise descend into the next directory.
                        else if (file.exists() && file.isDirectory() && recursive) {
                                File[] subDir = file.listFiles();
                                readDir(subDir, true);
                        }
                }

        }

        private String[] parseXml(File xmlAsText) {

                String[] ret = null;

                XMLInputFactory inputFactory = XMLInputFactory.newInstance();
                try {
                        FileInputStream inputStream = new FileInputStream(xmlAsText);
                        XMLStreamReader reader = inputFactory.createXMLStreamReader(
                                        inputStream, ANN_ENCODING);

                        ret = new String[3];

                        StringBuilder data = new StringBuilder();

                        int event = 0;
                        while (reader.hasNext()) {
                                event = reader.next();
                                if (event == XMLStreamConstants.START_ELEMENT) {
                                 String elementName = reader.getName().toString();
                                 if (elementName.equalsIgnoreCase("DOCNO")) {
                                         ret[ANN_ARR_DOCNO] = reader.getElementText();
                                 } else if (elementName.equalsIgnoreCase("TITLE")) {
                                         data.append(reader.getElementText());
                                 } else if (elementName.equalsIgnoreCase("DESCRIPTION")
                                        || elementName.equalsIgnoreCase("LOCATION")
                                        || elementName.equalsIgnoreCase("NOTES")) {
                                        data.append(" " + reader.getElementText());
                                 } else if (elementName.equalsIgnoreCase("IMAGE")) {
                                        ret[ANN_ARR_IMAGEFILE] = reader.getElementText();
                                 }

                                }
                        }
                        ret[ANN_ARR_TEXTDATA] = data.toString();
                } catch (FileNotFoundException e) {

                        System.out.println("Something went wrong, file not found.");
                        e.printStackTrace();
                } catch (XMLStreamException e) {
                        System.out.println("Problem reading the XML file: "
                                        + xmlAsText.getName());
                        e.printStackTrace();
                }


                return ret;
        }

        private String readFile(File file) {
                String textInFile = null;

                StringBuffer contents = new StringBuffer();
                BufferedReader reader = null;

                try {
                        reader = new BufferedReader(new FileReader(file));

                        while ((textInFile = reader.readLine()) != null) {
                                contents.append(textInFile);

                        }
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                if (reader != null) {
                                        reader.close();
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }

                return contents.toString();

        }

}
```

B.2.9 Main

The main Method of the application

```
package HITS;

public class main {

        /**
         * @param args
         * @throws FileNotFoundException
         */
        static HashMap<String, Tag> tagger;
        static HashMap<Integer, Document> dokumenter;
        public static void main(String[] args) throws FileNotFoundException {

                reading.Index indexer = new reading.Index();

                tagger = indexer.getTagList();
                dokumenter = indexer.getDocumentList();
                HashMap<Integer, Tag> tags = new HashMap<Integer, Tag>();

                int x = 0;

                Set<Map.Entry<String, Tag>> postingSet = tagger.entrySet();
                for (Map.Entry<String, Tag> entry : postingSet) {
                        tags.put(x, entry.getValue());
                        x++;
                }

                //int dokumentid = 1200; pingviner til Kim
                int dokumentid = 632;//Kaktuser

                Document d = dokumenter.get(dokumentid);
                System.out.println(d.getName());
                for(Tag t : d.getTags())
                {
```

```
                System.out.println(t.getId()+" "+t.getName());
        }

        long start = System.currentTimeMillis();

        VectorScoring vs = new VectorScoring(tagger, dokumenter);
        HashMap<Document, Double> scores =
                vs.getRelevantDocuments(dokumenter.get(dokumentid));

        ModifiedHITS mHITS = new ModifiedHITS(dokumenter, tags);
        //HashMap<Integer, Integer> scores =
        //        mHITS.findRelatedDocuments(dokumentid);

        Map<Document, Double> map = sortByValue(scores);
        long end = System.currentTimeMillis();
        Set<Map.Entry<Document, Double>> scoreSet = map.entrySet();
        for (Map.Entry<Document, Double> entry : scoreSet) {

         System.out.println(entry.getKey().getName()+
                 " - Score: "+entry.getValue());
        }

        System.out.println("Execution time was "+(end-start)+" ms.");

}
static Map sortByValue(Map map) {
     List list = new LinkedList(map.entrySet());
     Collections.sort(list, new Comparator() {
          public int compare(Object o1, Object o2) {
               return ((Comparable) ((Map.Entry) (o1)).getValue())
              .compareTo(((Map.Entry) (o2)).getValue());
          }
     });

    Map result = new LinkedHashMap();
    for (Iterator it = list.iterator(); it.hasNext();) {
        Map.Entry entry = (Map.Entry)it.next();
        result.put(entry.getKey(), entry.getValue());
    }
    return result;
}



public static void readFile() throws FileNotFoundException
{
```

```
                File fFile = new File("tagliste.txt");
                Scanner scanner = new Scanner(new FileReader(fFile));
                   try {
                     //first use a Scanner to get each line
                     while ( scanner.hasNextLine() ){
                       processLine( scanner.nextLine() );
                     }
                   }
                   finally {
                     //ensure the underlying stream is always closed
                     //this only has any effect if the item passed to the Scanner
                     //constructor implements Closeable (which it does in this case).
                     scanner.close();
                   }
        }


        public static void processLine(String aLine){
            //use a second Scanner to parse the content of each line
            Scanner scanner = new Scanner(aLine);
            scanner.useDelimiter(",");
            if ( scanner.hasNext() ){
              String document = scanner.next();
              String tag = scanner.next();
              dokumenter.get(Integer.parseInt(document))
                        .addTag(tagger.get(Integer.parseInt(tag)));
            }
            else {

            }
            //no need to call scanner.close(), since the source is a String
          }
}
```