Interactive Multimedia Search and Exploration

Joachim Tingvold†, Denny Stohr*, Dominik Schneider*, and Andrè B. Amundsen†
† University of Oslo
joachim@tingvold.com, andrebam@ifi.uio.no
* University of Mannheim
dennystohr@gmail.com, dominik.schneider@students.uni-mannheim.de

Abstract—Flash content and multimedia content in general is quite popular today and widely used. Still there exists no reasonable way to actually search for, find and explore flash movies on the net. Thus the main objective of this paper is to describe an intuitive to use search user interface (UI) and an interactive result presentation. For this purpose the *fulgeo* prototype (fulgeo: "flash" in latin) is developed and implemented to provide a showcase for an easy-to-use multimedia content search UI. Furthermore with more than 1300 files in database the *fulgeo* search engine has the largest set of indexed files in comparison to other flash search engines.

Index Terms—Flash retrieval, Multimedia search engine, User Interface, UI.

1 Introduction

Multimedia documents like PowerPoint presentations or Flash documents are widely adopted in the internet and they exist in context of lots of different topics. However at the moment there is no user friendly way to explore and search for this content. Thus, the aim of the project is to address this issue by developing an easy-to-use user interface (UI) and a prototype search engine with the focus on Flash documents. We created *fulgeo* based on scientific findings in the context of multimedia search that allows users to explore more than 1300 Flash elements as a potential canditate for a user friendly multimedia search engine.

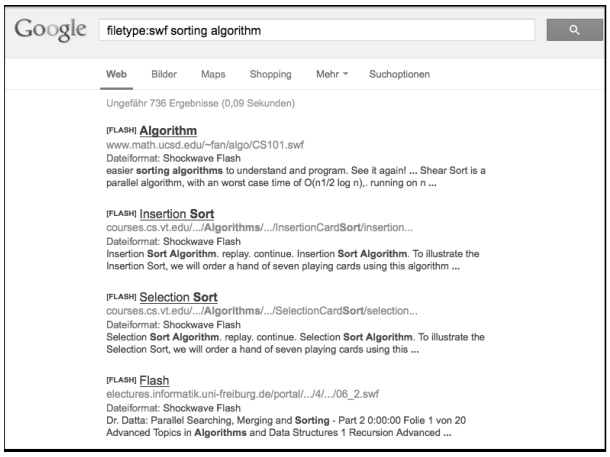

Fig. 1: Google UI

1.1 Background

The specialty of multimedia content is that it can be images, videos, sounds and text (Candan and Sapino (2010) [10]). Further, these media types can be animated or rearranged by interactions, which affects time and space. Due to this special feature, it's not enough to provide only a text based search results page, like for example Google (Fig. 1), but you have to address all aspects of multimedia. To do this you have to come up with a diverse set of features that stimulate different visual senses and give an intuitive idea of the documents content.
Furthermore different ways to query for these documents efficiently can exceed keyword based query by far. Query-by-example techniques for pictures and videos can improve the finding of relevant results, as well as special interfaces for audio (Hearst (2009) [6]). Also query of what animation takes place is possible (Yang et al (2007) [2])
However, in this paper we only use keyword based query, as the focus lies on the part after

the query is entered - the results page.

2 Related Work

2.1 Multimedia Search

Although flash was, and still is, a frequently used multimedia format on the internet, the work on specialized search engines for flash movies, and multimedia content in general, is still limited.

The FLAME (Flash Access and Management Environment) framework by Yang et al (2007) [2] can be considered as the most comprehensive work on this topic. The focus in this paper was the indexing and retrieval of flash movies by a semantic context. The initial search user interface (UI) consists of an input field for keyword-queries with a ”Search” and a ”Browse” button, which provides a keyword based query that looks for matching text in the flash movies. The results of the search are displayed below this rudimentary search field (Fig. 2). The retrieved flash movies are

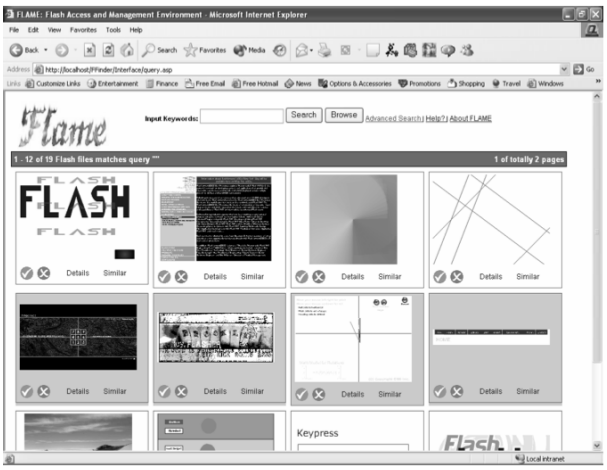

Fig. 2: FLAME main UI

illustrated with thumbnails, a link to the details of this movie and a button to find similar flash movies. On one page, twelve results are displayed. Furthermore they provided an ”Advanced Search” page (Fig. 3) where the user can enter more sophisticated queries via an input form.

The work of Ding et al (2003)[3] has the focus on the content-based retrieval problem, and especially on the creation of a relevance ranking of search results based on co-occurrences of

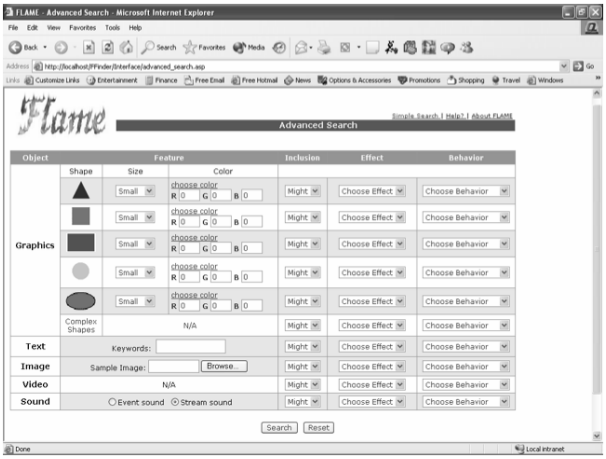

Fig. 3: FLAME advanced search UI

elements among flash files. The search UI looks quite similar to the one of FLAME, although it doesn’t provide a separate advanced search page.

In contrast, the work of Chung et al (2004) [4] has a look on the query costs for flash movie retrieval in combination with the Structural Join Index Hierarchy (SJIH) indexing technique. The queries in this prototype are entered as XML objects. Also the result is displayed in a human readable XML format.

The newest work on content retrieval of flash animations comes from Yinghua et al (2010) [5]. Their approach is to create a XML document tree of the flash file, and trim this tree to a final object fragment tree. To query a document one has to construct a object characteristic tree for the demanded characteristics and enter it in the UI.

2.2 User Interfaces for Search in General

More work has been carried out in the field of general search engine UI design, and the characteristics and behaviors of people who use a search engine.

Hearst (2009) [6] states that there are three main search behaviors: *fact finding* (looking for specific facts or pieces of information), *information gathering* (the collection of information form multiple sources) and *browsing* (visiting web pages without a particular goal). Furthermore there are common search patterns people tend to use, as Morville and Callender (2010) [7] mentioned. In fact it’s important that a search

engine easily allows to formulate queries, adapt and change them according to the users needs and preferences and direct the user to results, even if the user doesn't know exactly what he or she is looking for.

As generally in software development, the design of the user interface is one of the most important steps in search engine development (Stone et al (2005) [11]). The UI has to support the user in their intentions and adapt to different search patterns. There are several different search patterns one has to take into account when designing the UI of a search engine. These patterns are *Quit*, *Narrow*, *Expand* and *Pearl Growing*. Additionally there are anti-patterns as *Pogosticking* and *Trashing*, as described by Morville and Callender (2010) [7]. At *Quit* the user searches and exits after the first query. This can be avoided by providing a proper no results page, so that the user is guided to a better result set. *Narrow* is the second most common search pattern. Here the user adds search terms to the query to specify what he or she's looking for. Providing a bigger input box for the search query is helpful here, as then there is more space for search terms that lead the user to relevant results. The opposite, *Expand*, is less common, as the user normally doesn't remove terms from his query. A way to broaden the search if there are not enough results, is to show related search terms next to the input box and, if more than one search term, propose the user to search for one less term. *Pearl Growing* is a technique for advanced searchers, where the user retrieves new search terms from the content and the metadata of one good early search result. A way to aid this pattern in the UI is to provide a "similar results" button. A tradeoff in the design is to provide enough relevant search results on each page, and also provide enough information about individual search results to avoid *Pogosticking*, where the user often switch between the actual results and the results page of the search engine. Finally *Trashing*, the continued partial refining of a search term, can be avoided by autocomplete that aids the users in spelling mistakes and autosuggest that may lead the user away from an initial wrong guess of the search terms (Morville and Callender (2010) [7]).

Spink and Jansen's work (2005) [9] suggest further that users tend to view only one result page (approx. 85%), and only open a very few documents on that one page. This is despite very low use of advanced search options like boolean and quoted terms. Only a small part of search queries are refined to improve results. Also results with thumbnails are considered to be more relevant than other results on a page (Hearst (2009) [6]). Spink and Jansen (2005) [9] also note that there is growing trends towards more diverse terms, longer queries and more use of advanced functions.

Currently most search engines focus on keyword retrieval methods. Contrary, in a multimedia context, content-based retrieval methods are more interesting. An actual used retrieval method for images is query by example. Unfortunately one has to know much about the visual property of the image one is searching for to get proper results. Furthermore there are limited possibilities to query for metadata, for example to search by director or creator of a video (Hearst (2009) [6]).

Finally there are general guidelines concerning the design of search engine UIs. Thus it is important to offer informative feedback to the user, as the current status of the system and to display the search results immediately. Next, the system should support the user control, for example through a relevance ranking. Also simple error handling, as avoiding an empty result set, and a consistent UI should be a goal of search UI designers (Hearst (2009) [6]). Furthermore it is helpful to provide snippets of the found items, so that the user gets an idea of the content and creates links to similar results (Morville and Callender (2010) [7]). A common pitfall of search engines is to provide no proper "No Search Results" page which provides the user "a way out" (Nudelman (2011) [8]). Another bad mistake is to provide a graphical or numerical relevance score next to the results, as they don't add relevant information to the user and may even confuse them (Hearst (2009) [6]).

3 Requirements

The requirements of the search engine are all related to the user interface, as this is the

focus of our work. We wanted to build a UI that allows the user gets an impression of the content of the multimedia presentation at the first glance. Thus we made a brainstorming of possible features and agreed on the following ones. Following that, we created user stories for each requirement and implemented them in the front-end.

1) Search Page
   a) The user should be able to enter a keyword based query: The search interface itself should basically only support text based query for contents, since we only support queries based on the containing text in the flash file. An advanced search interface, that may even confuse the user – as is the case in the FLAME prototype (Fig. 3) (Yang et al (2007) [2]), is not necessary here. Furthermore a drawback of advanced search methods like query by example is that the searcher already has to know much about the result he or she's looking for (Hearst (2009) [6]).
   b) The user should be able to require the results to contain video or audio content, animations or interactional parts: In addition to the text based query, we agreed on the possibility to filter the flash files for those that contain audio or video files, animations or interaction. This is a contribution to the specialty of multimedia content, as a user might want to require a flash file to have for example audio inside, when he searches in a specific music related search domain.

2) Search Result Page
   a) The user should get a quick impression of the flash document content: One of our initial requirements was that the user gets a first impression of the found movies right away. Thus every result should have a thumbnail. This is a major lack of the current way to search for flash content on Google, which provides only text output as a result (Fig. 1). Evidence suggests that thumbnails are scanned faster than text, and improve the performance of the user to find good results (Hearst (2009) [6]).
   b) The user should be able to interactively explore the flash documents: We wanted the results to be interactive, so that you get a better idea of the actual content of the flash file. Thus at mouseover of the thumbnail, a small animation should present the different views of the document. Like this, a first step against *Pogosticking* is made (Morville and Callender (2010) [7]).
   c) The user should get a quick overview whether audio or video content, animations or interactional parts are included in the flash documents: We came up with the idea to provide more information about different media content types like audio or video and whether the document includes animations or supports interactions. This adds important information in the multimedia context of flash movies, and reduces short term memory load from the user because he or she doesn't need to take that in mind when comparing different results - one more reduction of *Pogosticking*.
   d) The system should display the relevance of the flash documents in an intuitive way: We wanted the search engine to have a visual and intuitive relevance ranking. Because results with bigger thumbnails seems to be more important to the user (Nudelman (2011) [8]) we came up with the idea of the thumbnail size as indicator for the absolute relevance in context of the query term. The images should also be ordered from top to bottom to view a relative relevance between the different results. Like this also people from other cultural backgrounds, who may read from left to right, get an intuitive idea of the ranking. Through the absolute and relative relevance the user gets double feedback which results are the most important.
   e) The user should be able to explore more details of the flash document: We wanted to allow users to explore more details of the flash document within a details view by clicking on the thumbnail of a result. This view should contain again

the indicators for media contents, animations and interactional parts. We also initially planned to change the audio and video icons in the backend at mouseover to ”play” buttons and play the media when clicking on it. We discarded this requirement due to the large number of short sequences of audio and video in the flash files we tested it. Further it should contain a bigger thumbnail of the result. This thumbnail size is the same every time, so that even the lower ranked results with a smaller thumbnail can be viewed easily without recognition problems. Next to the thumbnail the text contained in the document should be displayed to let the user make his own impression of the pertinence of this document. The search terms that appear in the text should be highlighted, so that the searcher sees them in their context in the result. Finally the view should also contain a button to close the details view and go back to the normal result page.

f) The user should be able to view the document: Of course the user also needs to view and maybe then download the document. For this a link to the document should be placed directly next to the thumbnail in the search results page. Also a button to view the document should be placed in the details view. Additionally the document will be opened when clicking on the big thumbnail inside the view, as this is more intuitive than to extra click on a button for that.

4 *fulgeo*

The *fulgeo* prototype (fulgeo: ”flash” in latin) is a prototype intended to provide a novel and intuitive user interface to search for and explore flash movies, as representatives of multimedia content.

4.1 Implementation

The prototype is split into two parts; the front-end and the back-end. The first is what the user sees when visiting the web page. It only handles how things should be displayed, and whenever the user search for something, it sends the queries to the back-end, which does all the work, and sends the results back to the front-end to be displayed.

Fig. 4: Initial search UI

4.1.1 Frontend

The first thing you see when you open the *fulgeo* search engine, is the text box for the search term, the checkboxes to require special multimedia contents, animations or interactional parts and the current number of entries in the database (Fig. 4).

We made sure to provide a large text box for query terms, as longer text boxes encourage the user to enter larger queries. Like this the user finds more relevant results and the common *Narrow* search pattern is supported (Morville and Callender (2010) [7]). Also the number of entries shows the actual status of the system, and like this offers informative feedback to the user. When entering a query and hitting the search button the system also shows that it’s working through a progress indicator, another contribution to the design guidelines of Hearst (2009) [6].

After the system finds the results and passes them to the front-end, they are displayed in a dynamic grid layout, produced by the Masonry jQuery plugin (Fig. 5). In combination with our already mentioned relevance indication by the size of the thumbnails, this gives the user a good overview of the results, as per requirement 2a and 2d.

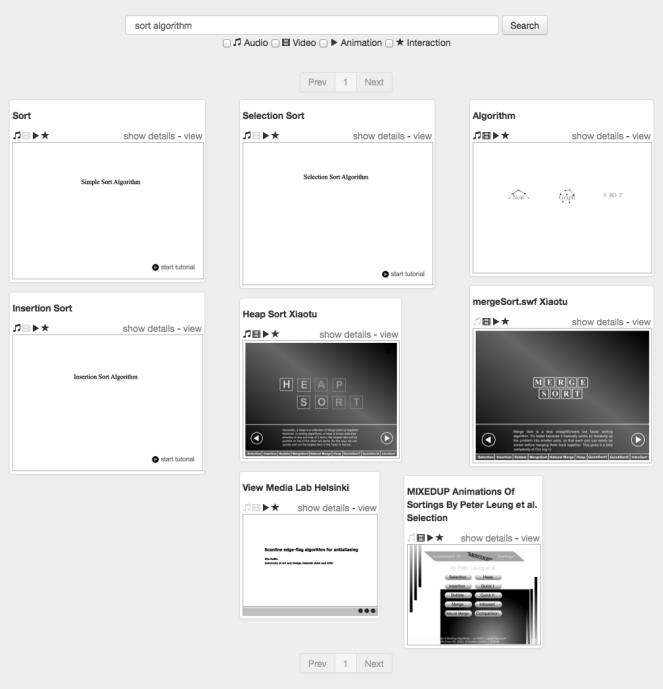

Fig. 5: Search Results Page

To limit the number of results per page we introduced a pagination mechanism that splits up the results (Fig. 6). Per page 20 items are displayed, so that the user gets a reasonable amount of results without increasing *Pogosticking* as enough information per result are available (Nudelman (2011) [8]).

Fig. 6: Pagination on the Search Results Page

The search results layout needs to be compact, yet informative, to fit as much information as possible into the users view. Within a discussion with experts in the field of user interface design and multimedia, we agreed to show only the most important elements of each file as a thumbnail image with the title and compact links to the actual file, as well as the indicators for type of content (music, video, animation and interaction). This should satisfy requirements 2a, 2c, 2e, and 1b. The results seemed a bit flat, and lacked a real impression of the file, so animated thumbnails were added refer requirements 2b and 2e.

The details view allows continued browsing of other results as well as providing an interactive way of looking into the details of a result (Fig. 7). It also gives two ways to view the flash file itself. The combination of the bigger interactive thumbnail with the text is similar to the approach of the *Voxalead* search engine (http://voxaleadnews.labs.exalead.com/). This is according to requirements 2c, 2e and 2f.

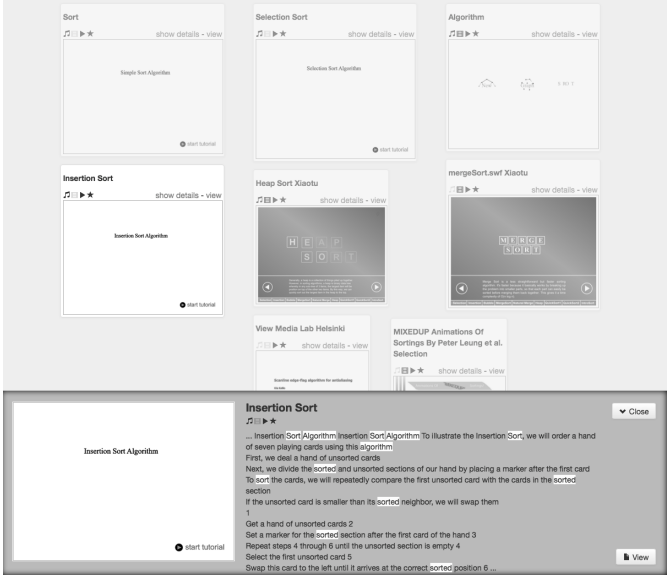

Fig. 7: Details View

The technical part of the front-end is JavaScript-heavy with focus on HTML5 and CSS3. Support for old browsers and mobiles was not considered, as it was outside the scope of this project[1]. The front-end was written using jQuery[2], simple.js[3], Masonry[4] (a jQuery plug-in) and Bootstrap[5]. jQuery is a standard javascript library to simplify DOM manipulation tasks, and should be familiar to anyone creating web-pages. Simple.js is a minimalistic approach to a MV* framework. It helps with structuring code and includes some useful functions for task like AJAX calls. Bootstrap is used for easy paging, formatting of some HTML elements, CSS reset and other handy CSS. The JavaScript is divided into models and views. The three views each have responsibility for a part of the web-page. The masterView.js is perhaps

1. Due to jQuery and masonry it does work decently on some old browsers and some phones.
2. http://jquery.com/; Version 1.9.1
3. http://simplejs.org/; Version 0.1.1
4. http://masonry.desandro.com/; Version 2.1.07
5. http://twitter.github.com/bootstrap/; Version 2.3.1

unnecessary but was added to easily change between different search result-displays if we decided to create more than one. searchView.js and resultsView.js are what they sound like. searchView.js contains the code for the search bar, and handling the user input. This input is then sent to queryModel.js, which handles the AJAX connection with the back-end. Displaying the search results is handled in resultsView.js via event calls from the queryModel.js. queryModelStatus.js is a very small model to get an up to date number of files in the database.

4.1.2 Backend

There are two main services that need to be handled by the backend of the multimedia search engine: fetching and processing flash content, and providing search access to the stored information about the flash content. To address those needs, PostgreSQL[6] as a database system for information storage, and software developed in Python are used. The selection of the Python[7] script language allowed fast development leveraged by having readily available libraries to access the Google Search Engine[8], implement web services[9] and interacting with the PostgreSQL datababase[10]. In the following, the different modules of the multimedia search engine backend are further explained.

As as first step, the Google custom search API is queried specifically for flash content using the

```
filetype: swf
```

operator combined with a desired keyword. This request, in case elements are available, is returned as a JSON formatted list, consisting of different attributes such as website title, URL and description. Next, using the URL, the flash content is downloaded by the script for further processing. For each result an instance of the object FlashElement is created. At creation, seven further processing steps, which make use external software, are started. (1) The first one renders a thumbnail of the downloaded flash file by calling the swfdec-thumbnailer[11] command. The result is stored on the file system, and can be accessed by the front-end later on. (2-6) Next, by using swfdecoder (included in the swftools[12] package; Version 0.9.2) and swfmill[13], the flash file is analyzed for containing audio, video, animation, interaction and text contents. (7) In the last step, several images from within the flash animation are rendered and stored in order to provide data for the quickview of elements in the front-end by using the swfrender (also part of swftools) program. After finishing the processing of the flash element, the data is inserted in the database.

The web service, which provides database access for the front-end, has been implemented using the Tornado library. When starting the backend.py script with the -server parameter, an instance of the web server is launched. There are three services implemented in the backend.

The first one, which can be accessed using the URL *http://server-name/search*, runs a search on the database using the parameters *query* and *page*, as well as four boolean values: audio, video, interaction and animation. The query parameter defines the desired keyword that should be searched in the database. Users can combine different keywords that have to be included in the result by separating them with blanks and can also exclude keywords by using - in front. In case a users selects one or more of the search refinement select boxes (Fig. 8) in the frontend, the query is adjusted to return only results that must include the selected attributes. The database ranks the results based on their

Fig. 8: Selectboxes in Search UI

cover density (Clarke et al (2000) [1]), and returns a list of 20 results including html tags for highlighting the search terms in the front-end, as well as an ranking value normalized between 0 and 1 that is used for the calculation

6. http://www.postgresql.org/; Version 8.4.16
7. http://www.python.org/; Version 2.7
8. https://code.google.com/p/google-api-python-client/; Version 1.1;
9. http://www.tornadoweb.org/en/stable/; Version 3.01
10. pyPgSQL: https://pypi.python.org/pypi/pyPgSQL; Version 2.5
11. http://swfdec.freedesktop.org/wiki/; Version 0.8.4
12. http://wiki.swftools.org/
13. http://swfmill.org/; Version 0.3.0

of the thumbnail size in the front-end. Next, the service accessible on the URL *http://server-name/searchcount* returns the number of results that are available for a certain search request. This is required to provide paging in the front-end. It seems strange to have this as a separate query, but the optimization of queries in the database makes it impossible to get this number together with the first search-query.

Last, *http://server-name/status* returns the total number of elements in the database to be displayed when the user accesses the search front page.

4.2 Testing

For performance evaluation we loaded more than 1300 entries in the database and tested with several different queries of various query terms and in different search domains. Further we did a rough usability testing to make sure that the UI isn't missing intuitive functions.

5 Discussion

In our work on user interfaces for multimedia search engines, we started with a list of requirements for our *fulgeo* prototype. We believe that we included all of the requirements from the beginning of the work in our final prototype, and arranged it in an intuitive and user-friendly way. This can be said especially in comparison with the FLAME prototype (Yang et al (2007) [2]), that only provides a static way of exploring the search results. Also it doesn't contribute to the possible variety of multimedia content in the flash document, in contrast to our approach of providing indicators for the contents of different media types and animations or interactional parts. Also, other existing flash search engines, like the work from Ding et al (2003)[3], Chung et al (2004) [4] and Yinghua et al (2010) [5], fall behind our UI solution either through a static interface, similar to the one of FLAME, or already through a cumbersome and user-hostile way of query formulation.

Furthermore, with the great number of more than 1300 files in our database, we outperform previous flash search engines by far. Previously, the FLAME prototype with around 200 (Yang et al (2007) [2]), and the prototype of Ding et al (2003)[3] with 224 files, were considered as the biggest flash search engines available.

Initially we had three ideas for displaying detailed information of a selected search result. The first one was an in page pop-up along the lines of Facebook's image gallery (Fig. 9). We

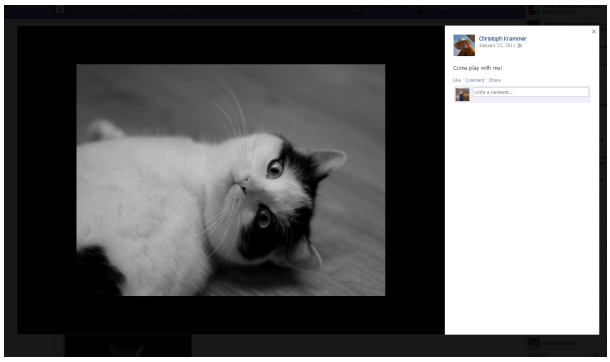

Fig. 9: Facebook image gallery

discarded that, since pop-ups may be perceived as hindering by users, due to the fact that they block the access of all other content on the site, even though it is still shown in the background. The second idea was to switch out the entire search results page with one large detailed view of the selected item. We didn't like this either, as the user had no overview over the other results anymore, and also had to close this view in order to select other search results - thus we'd create a danger of *Pogosticking* on our own site. Our third idea was to expand the thumbnail into a larger area, but still in line with the other results, which is is what Google does for their image search. This way we could display more information, and the user would not have to close any windows in order to select another result from the search. This approach would however be difficult to structure properly with the thumbnails being different sizes. We ended up going with a fourth idea. The details view should slide in from the bottom of the page, covering some of the other results. All the other results should fade out, but still be visible. This way the user could select another result from the search, and the details for this result would replace the previous details.

We originally wanted a form of infinite scrolling for the search results page, like that of Google image search. We did however not find

any plug-in or framework for that which fit the rest of the workings of the front-end, so we left it out. Implementing something like that could have helped combat the issue of most users not looking past the first page, by providing them with *one* long page. Morville et. al. states that infinite scrolling my be confusing for users that are used to the standard model[7]. So infinite scrolling would have been a trade-off at any rate.

A small drawback of our work was the late switch from the algorithm domain to a broader field of multiple search domains. Due to this, we late discovered that the feature to preview the audio or video content when hovering and clicking on the multimedia indicators in the details view actually makes sense. We excluded it in the first place due to our initial focus and testing in the algorithm domain, where the documents only contained small pieces of sound and video - and where the feature was useless. We found that this feature makes much more sense in a search domain that focuses on music and visual content, so we have added it to the future work, and it will be considered in the next version.

6 Conclusion

This paper investigated the problem of providing an intuitive and easy-to-use search user interface for multimedia content. We presented general requirements and guidelines one has to consider in the design of search user interfaces and finally presented our prototype *fulgeo*, providing a new user interface which aims on allowing users to intuitively scan the multimedia content and find the most relevant documents for their purpose. Thus, *fulgeo* is a possible solution that allows easy exploration of multimedia contents.

6.1 Future Work

Future work on our research foremost includes to test on an independent group of people whether the user interface really improves the usability and findability for users in comparison to common solutions to find multimedia content like Google.

Further work has to be done also on the front-end of the prototype. We intentionally excluded autocomplete, autosuggest and partial match functions from the feature list due to the limited time frame for this work and the higher importance of other features. An autocomplete function for the query term suggestion, to aid the user in finding proper search phrases that relate to his own thoughts and like this offer informative feedback (Hearst (2009) [6]), is part of the future work. Also part of future work is an autosuggest function that shows the user related search queries in form of links for forward navigation. These can be included in the ”no search results” page, to provide a ”way out”. Also a partial match function that automatically excludes some of the search terms the user entered, if it’s more than one term and there are no matches (Nudelman (2011) [8]) will is included here. This improvement of the ”no search results” case can also help to avoid the *Quit* search pattern (Morville and Callender (2010) [7]). Additionally the excluded preview feature for audio and video content should be implemented in the next version, as stated in the discussion section. Another improvement for the front-end is the inclusion of a history library like History.js, so that the user can use the back and forward buttons of his browser, which currently is not possible. This can be quite annoying to a user, but as this is only a prototype with limited time for development, we decided to leave this out. Also a mobile themed and IE-compatible version of the prototype can be added to this list. Currently it works as intended on the latest versions of Chrome, Firefox, Maxthon, Safari and IE 10.

The back-end should have support for other languages than english, and also add the extra ’count’-query within the regular search handler (to avoid two database connections, when only one is needed). The back-end should also move away from running it’s own web-server, as *fulgeo*, in most cases, would be used together with another web server serving the front-end. Now the back-end is configured to listen on it’s own port, which the front-end should be able to reach (and thus, also the client). Current testing was done with the back-end listening on port 8888, however, in some places,

the client would be behind a firewall, and port 8888 would not be available. In such a scenario, the user would be presented with a timeout-error on the front-page, since the client can't reach the back-end on port 8888.

In conclusion, we developed a new approach to intuitively search for, and explore, multimedia content, and implemented the approach in our *fulgeo* flash search engine prototype. It can be considered as the most comprehensive and intuitive to use flash search engine out there, and also the one with the largest amount of files indexed. Finally this could be a first step to more user friendly search interfaces for flash retrieval, and even multimedia content in general.

ACKNOWLEDGMENTS

The authors would like to thank the chair of Prof. Dr. Wolfgang Effelsberg (Praktische Informatik IV) and especially Dr. Stephan Kopf for organizing the Teleseminar, as well as Lydia Weiland and Juniorprof. Dr. habil. Ansgar Scherp for supervising and providing us with feedback troughout the seminar project. We'd also like to thank Prof. Thomas Plagemann from the Department of Informatics, University of Oslo, for organizing the course this project has been a part of.

REFERENCES

[1] C. L. Clarke, G. V. Cormack, and E. A. Tudhope, "Relevance ranking for one to three term queries" *Information Processing & Management*, vol. 36, no. 2, pp. 291–311, 2000.

[2] J. Yang, Q. Li, L. Wenyin, Y. Zhuang, "Content-based Retrieval of Flash movies: Research Issues, Generic Framework, and Future Dircetions" *Multimedia Tools and Applications* , 2007, vol. 34, pp. 1-23, .

[3] D. Ding, Q. Li, B. Feng, L. Wenyin, "A Semantic Model for Flash Retrieval Using Co-occurrence Analysis" *MULTIMEDIA '03 Proceedings of the eleventh ACM international conference on Multimedia* , 2003, pp. 239-242.

[4] E. Hon Chung, C. Fung, Q. Li, "Rich Media Retrieval in an Object XML Framework: a Case Study with Flash Movies ausing Structural Join Index Hierarchy" *28th Annual International Computer Software and Applications Conference* , 2004, pp. 51-52.

[5] N. Yinghua, Y. Liyong, M. Yongjin, J. Bingyao, "Research on Content Retreival of Flash Animation Based on XML" *Ubi-media Computing (U-Media), 2010 3rd IEEE International Conference* , 2010, pp. 64-67.

[6] M.A. Hearst, *Search User Interfaces* New York, NY: Cambridge University Press, 2009.

[7] P. Morville, J. Callender, *Search Patterns* Sebastopol, Calif.: O'Reilly Media , 2010.

[8] G. Nudelman, *Designing Search - UX Strategies for eCommerce Success* Indianapolis: Wiley Pub, 2011.

[9] A. Spink, B.J. Jansen. *Web Search: Public Searching of the Web* Springer, 2005.

[10] K.S. Candan and M.L. Sapino. *Data Management for Multimedia Retrieval* New York, NY: Cambridge University Press, 2010.

[11] D. Stone, C. Jarrett, M. Woodroffe, S. Minocha. *User Interface Design and Evaluation* San Francisco: Morgan Kaufmann Publishers, 2005.