U N I V E R S I T Y O F T A R T U
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Computer Science speciality

Peeter Jürviste

Supporting Internet Search by Search-Log Publishing

Master Thesis (30 EAP)

Supervisors: Prof. Eero Vainikko, PhD,
Ulrich Norbisrath, PhD

Author: ..... "....." August 2012
Supervisor: ..... "....." August 2012

Allowed to defence
Professor: ..... "....." August 2012

TARTU 2012

Contents

Acknowledgements

Hereby I would like to express my sincere gratitude to everyone who have helped me during my Master studies. A big thank you to Dr. Ulrich Norbisrath for all his all-round support and wisdom. Many innovative ideas have emerged from my numerous conversations with him. I am very grateful to Prof. Dr. Eero Vainikko for providing technical advice and supervising my writing. Last but not least I want to thank Georg Singer who inspired me to take a plunge into exploratory search and information management.

I am very fortunate to have received a lot of support from my parents and I wish to thank them for their continued love and prayers.

Finally, I would like to acknowledge that the research was supported by European Social Fund and Internationalization Programme DoRa.

Introduction

Now that the size of World Wide Web is growing exponentially, there is an increasing demand for better information retrieval algorithms as well as Internet search supporting tools to find the relevant information especially in case of complex exploratory search tasks. As part of a collective ongoing effort to support Internet search, the aim of this thesis is to provide the tools for detailed search task log creation, annotation and publishing. It is a part of the information management efforts and can be located in the research field of Information Retrieval (IR), although it must be noted this is not a traditional IR project.

I decided to undertake this project because I was somewhat frustrated with the state of the art in traditional query-based search methods which are inherently not suited for bigger and more difficult search tasks. What also motivated was being a part of a research group made up of likely minded people who all share my ambition and want to bring a change to how we think about Internet search. The main research problem of my thesis is engineering a new type of search task logging and publishing framework which would provide a better alternative for existing browser plug-in based methods.

Let us consider the following practical scenario. Jade is a high school graduate with a goal of finding a top European university on a monthly budget of 1500 EUR where to study machine engineering in English. She starts by googling "machine engineering Europe universities" and goes through ranked lists of universities and visits each of their home pages for more information. This takes a lot of time but Jade is not happy with the results. She decides to change the search query to "top universities in Europe machine engineering".

This time she has more luck and finds the QS World University Rankings. As she is building a list of potential candidates, Jade is confused by the fact that universities in different countries are difficult to compare: some have tuition fees, some do not have; they provide different entry criteria and financial support packages. She changes her search query a few more time to include related terms "mechanical engineering" and "electrical engineering". Her list of universities gets longer but also the list of discovered gaps in her knowledge. After hours of fruitless search Jade gives up.

Jade's scenario is not an isolated example of complex search tasks gone wrong. Prevalent Internet search are reasonably good for simple lookup operations. However, we could do better when things get more complicated and users want to combine multiple queries in one search task. An important characteristic of these scenarios is users learning, investigating and exploring the subject matter while they are conducting the search. This means that it is even possible that they do not have the "right words" for what they are searching for and do not know what they are going to find out. As a result, such search tasks of complex nature tend to be long-winded and unsatisfactory.

In order to tackle this problem, I propose a proxy-based method for logging user

search behaviour across different browsers and operating systems. I also compare it with an existing plug-in based search monitoring solution for Mozilla Firefox 3.6 and other similar solutions. The logs created by my solution are subsequently annotated by the user and made publicly available on a dedicated Internet blog called the Search Task Repository. Users can search against the already annotated and published Internet search logs. Ideally this would mean reduced complexity of search tasks for the users which in turn saves time.

To illustrate the benefits of my solution, I will highlight what would be different in Jade's scenario. Assuming that her computer is already set up to use my solution, Jade would start by visiting my search task repository and search like she would normally do. If there were no publicly available related search task logs by members of the community, my solution would not be of any help to Jade. However, she could be the first in the community to publish a university-related search task log with annotations so that the next person in a similar situation could benefit from it. If lucky, Jade would find at least one matching search task log with ideas how to proceed from other users. Having found her best university, Jade would choose to publish her own annotated search logs too, and the next person looking for a university would have even more search advice to consider.

There are many challenges ahead in search information management. For example, we need even more user-friendly search monitoring and supporting systems that can work in the background without user interaction. Efforts requiring lots of user activity are unlikely to be actively used unless the users are sufficiently rewarded for their part in annotating search logs. A number of these challenges actually have their roots in the society. Things like the user's location and cultural upbringing, information freshness and reliability, willingness to adapt to new search paradigms all have to be taken into account if we are to maximize the benefits from logging Internet search and making decisions based on logged information.

From a technical perspective, we can clearly comprehend that one of the most challenging tasks ahead would be indexing and ranking search logs. And even though my thesis is centred on designing, implementing and testing out new search information management solutions, one cannot neglect challenges related to Internet security nor the actual performance and scalability of them. All in all, the management of complex Internet search tasks will continue to be at the forefront of research for years to come.

My thesis is divided into four chapters. After this introduction, the first chapter defines complex search tasks in the framework of exploratory search and describes related work and projects. It is followed by a chapter providing a detailed overview of the proxy-based search logging solution as well as the search task repository. A guide to new users is given in Chapter three. The last chapter is dedicated to future work. Links to all relevant resources are provided in Appendix A.

Chapter 1

Theoretical background and related work

In this chapter, I introduce the meaning of complex search tasks and review the attempts that have been made to record user search behaviour. I start from defining and explaining some of the basic terms from classical Information Retrieval in the context of my thesis. Also, I give an overview of related research in search behaviour analysis. The possibilities of crowdsourcing are analysed in detail to determine its suitability for exploratory search studies in the future.

1.1 Important definitions and concepts

For the sake of clarity I will give a few definitions for most frequently used and/or relevant terms in my thesis.

Information Retrieval (IR) A broad area of computer science that deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogues, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be set up in a way to provide the users with easy access to information of their interest [6, p. 1].

Information need The topic about which the user desires to know more [37, p. 5]. For example, a high school graduate might be needing information about university life, offered curricula, tuition fees, and living expenses. This forms a base for his/her Internet search actions and behaviour.

Query The user input conveyed to the computer in an attempt to communicate the information need [37, p. 5]. The simplest query is a single-word query. Matching Web pages will have one or more occurrences of the query word. A single word is not sufficient to express all information needs [33, pp. 197-200]. A multi-word query is comprised of two or more words with optional context information. Most Web search engines offer both multimedia and text search capability. In keyword-based search the contents of multimedia files are not used directly: the text in a neighbourhood of the multimedia file represents its content and it is assumed that a description of the file will be found in its neighbouring text [33, pp. 197-200]. Another dominant interface type for searching and browsing large

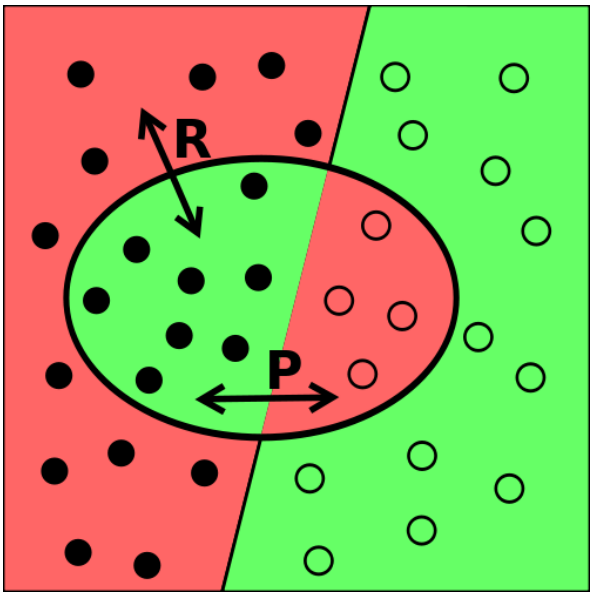


Figure 1.1: The terms *precision* and *recall* illustrated (taken from [57])

image collections is searching by overall similarity to sample images. Yee et al. designed an image access interface that allows users to navigate a large collection using hierarchical faceted metadata in a flexible manner. This enables users to navigate along conceptual dimensions that describe the images [64]. *Query length* is the number of terms (sometimes just "words") in a query. According to Jansen and Spink, *term* is a series of alpha-numeric characters separated by white space or other delimiter [29].

Relevance A document is relevant if it is one that users perceive as containing information of value with respect to their personal information need [37, p. 5].

Effectiveness The quality of the search results. It is often assessed by the following two key statistics about the system's returned results for a query, like explained in the next two points: *precision* and *recall* [37, p. 5].

Precision The fraction of the returned results that are relevant to the information need [37, p. 5]. For example, in Figure 1.1 precision is the quotient of the left green region by the whole oval where the relevant items are to the left of the straight line while the retrieved items are within the oval. Precision can be seen as a measure of exactness or quality, and in even simpler terms, high precision means that an algorithm returned more relevant results than irrelevant [57].

Recall The fraction of the relevant documents that were returned by the system [37, p. 5]. For instance, in Figure 1.1 recall is the quotient of the left green region by the whole left region where the relevant items are to the left of the straight line while the retrieved items are within the oval. Recall is a measure of completeness or quantity and high recall means that an algorithm returned most of the relevant results [57].

Search task Marchionini defines "search tasks" as information-seeking activities risen from different layers of information needs, each with associated strategies and tactics that might be supported with computational tools. He distinguishes three

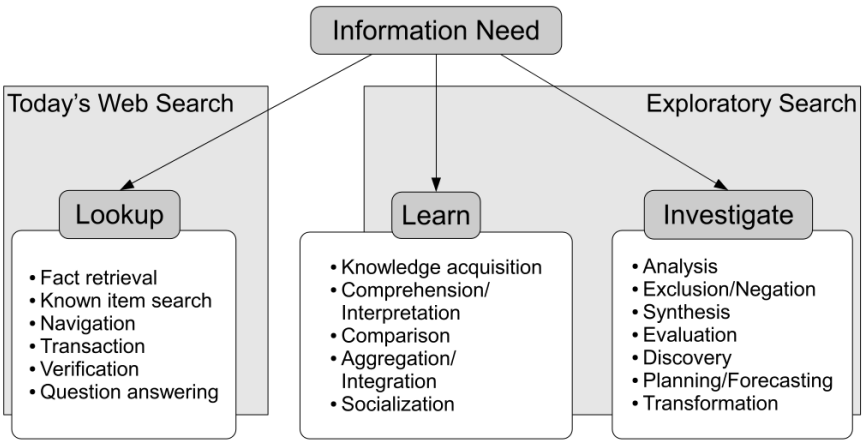


Figure 1.2: Marchionini's model of search activities[38] (taken from [43])

basic categories of such activities: information lookup, learning and investigating (Figure 1.2). These three are often overlapping because people may engage in multiple kinds of search in parallel, and some activities may be embedded in others. A good example of this phenomenon are lookup activities which are often embedded in learn and investigate activities [38]. In my thesis, I define it also as a collection of purposeful information-seeking activities that can be decomposed into an ordered set of simpler lookup, learning and investigating activities.

Information lookup Lookup is the most basic kind of search task, the focus of development for database management systems. These are akin to fact retrieval and question answering, and are satisfied by short, discrete pieces of information such as numbers, short statements, names, or names of files or Web sites. In journalism, lookups are related to questions of who, when and where [38]. According to Marchionini, lookup tasks are generally suited to analytical search strategies that begin with carefully specified queries and yield precise results. Thus, the need for result set examination and item comparison is minimal [38, 6, p. 22].

Exploratory search A class of information seeking tasks pertinent to learn and investigate search activities (see Figure 1.2). Learning searches require more than one query-response pair, and require scanning and reading multiple information items by the searcher as well as synthesizing content to form new understanding. Investigating is seen as a longer-term process that involves multiple iterations over a potentially very long time span and they encompass critical assessment and validation of results [38, 6, p. 22]. In journalism, exploratory search is related to questions of what, how, and why [38]. According to Ryen W. White from Microsoft Research, "exploratory search describes an information-seeking problem context that is open-ended, persistent, and multifaceted, and information-seeking processes that are opportunistic, iterative, and multitactical" [54]. Exploratory searchers aim to solve complex problems and develop enhanced mental capacities while exploratory search systems support this through symbiotic human-machine relationships that provide guidance in exploring unfamiliar information landscapes [54].

Complex search task Aula and Russell from Google suggest two undiscussed dimensions to information searches: complexity of the information need and clarity of

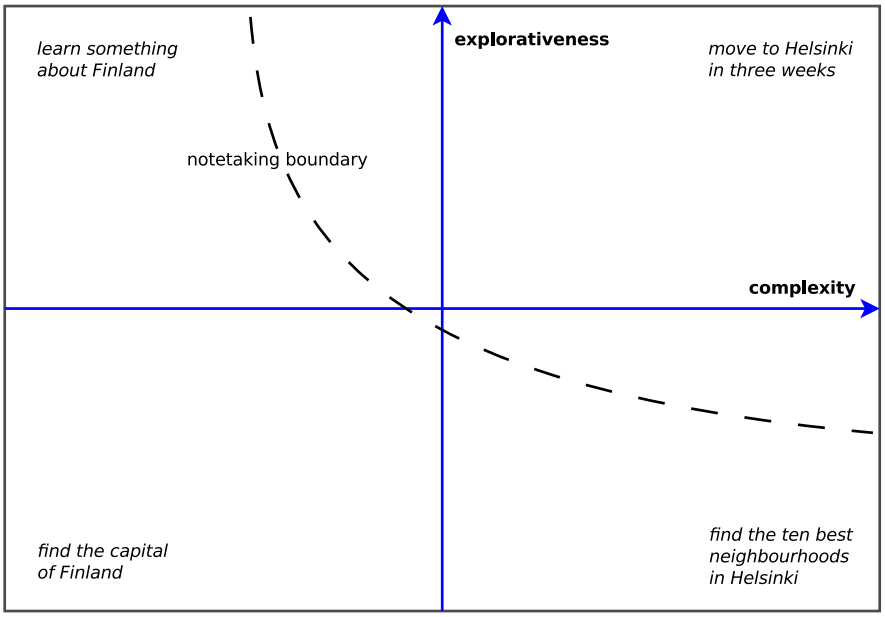


Figure 1.3: Search task space according to Aula and Russell [5]

what is needed to satisfy the information need [5]. It is the number of steps that are required to collect the needed information that among other factors affect the complexity of the information need. Storing found information becomes increasingly important as the complexity of the task increases since there is a limit to the amount of information to hold in human memory. They also show how "exploratory search may sometimes be complex, but is not necessarily so, and is characterized more accurately by the degree of clarity the searcher has about the goal" [5]. According to Aula and Russell, "complex search tasks often include exploring the topic, but do not necessarily require exploration or may require exploration only in certain phases of the search process". Figure 1.3 illustrates the difference between exploratory and complex search tasks [5]. Some researchers do not draw a distinct line between the two terms. For instance, Singer et al. define *complex search* as "a multi-step and time consuming process that requires multiple queries, scanning through many documents, and extracting and compiling information from multiple sources" [44]. Thereby, a complex search task is the one that leads to a complex search activity [44].

Social search Contrary to the solitary activity of an individual searcher, social search is a form of information seeking in which social interactions play an important role throughout the search process [16]. This notion encompasses collaborative co-located search, as well as remote and asynchronous collaborative and collective search [16]. Social search can be aided through instant messaging access to the searchers' personal connections alongside the search box [16].

Session Jansen, Spink, Blakely, and Koshman define a session "from a contextual viewpoint as a series of interactions by the user toward addressing a single information need" [30]. They explored three alternative methods for detecting session boundaries and conclude that "using IP address, cookie, and query-content changes, appears to provide the most detailed method for session identification with both session length and session duration" [30]. As this method does not rely on prob-

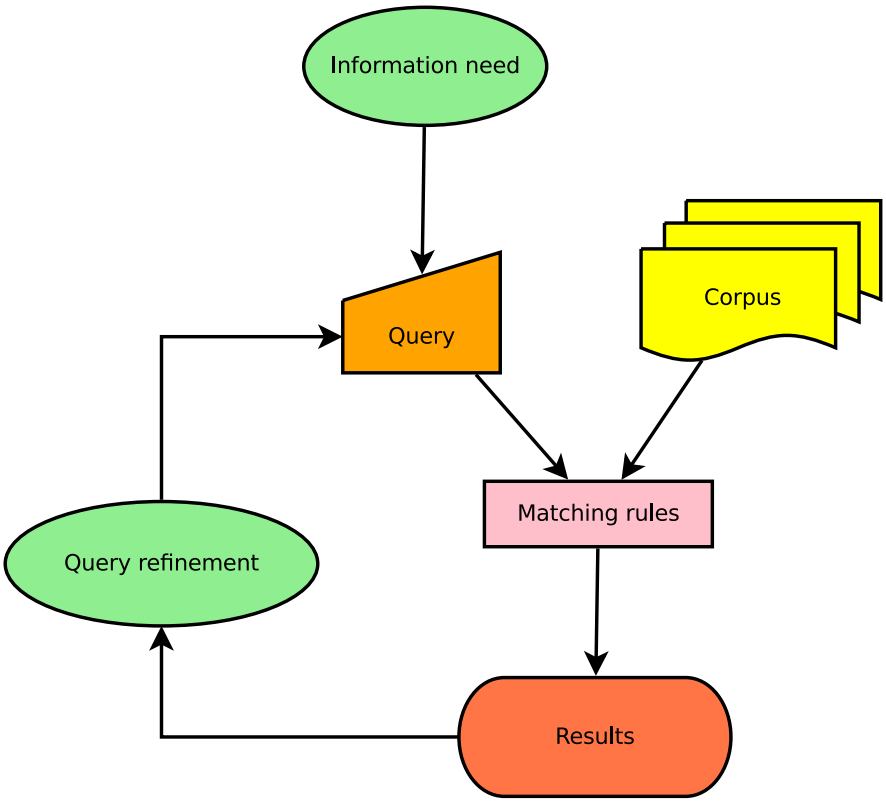


Figure 1.4: The classic model for information retrieval [10]

ability methods, it can be calculated in real time with near 100% precision of new session identification [30]. *Session length* can be formulated as the number of queries that a searcher submits in one episode with a Web search engine [29].

1.2 Web search taxonomies

Current information retrieval literature shares the assumption that Web searches are motivated by an information need [10]. According to the classic model for information retrieval (Figure 1.4), a user, driven by an information need, constructs a query in some query language. Then the query is submitted to a system that selects from a collection of documents (also known as a corpus), those documents that match the query as indicated by certain matching rules. If not satisfied, the user can refine the old query to create new queries and/or to refine the results [10].

This basic model can be augmented as in Figure 1.5, to reflect the human-computer interaction factors and the applicable cognitive aspects. In essence, we recognize that the information need is associated with some task. This need is verbalized (not necessarily loud) and translated into a query conveyed to a search engine [10].

Andrei Broder argues that the need behind a Web search is often not informational (users want to find information on a certain topic) – it might be also navigational (seeking the URL of the Web site I want to reach) or transactional (the intent is to perform some Web-mediated activity, e.g. do online shopping, download a file, or find a map) [10]. Modern search engines need to deal with all three types (informational, navigational, and transactional queries) although each type is best satisfied by very different results [10].

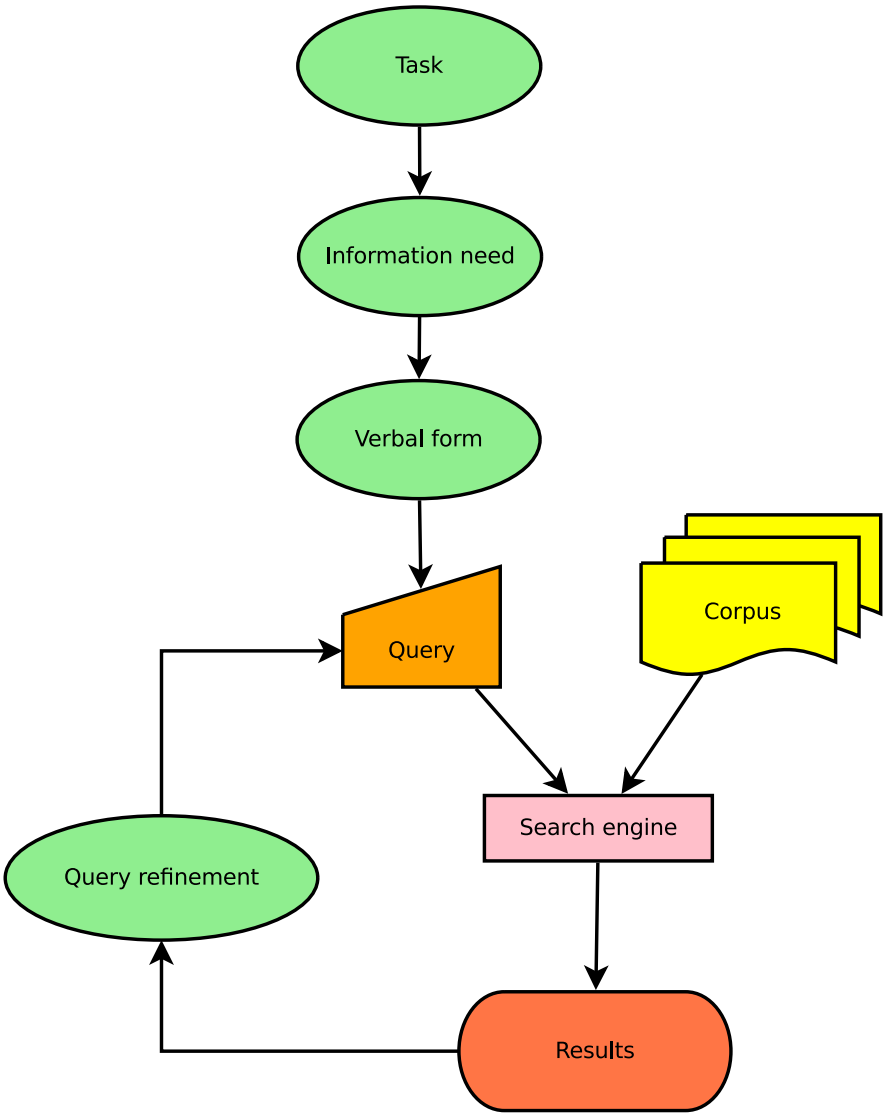


Figure 1.5: Augmented IR model for the Web [10]

Peter Morville and Jeffery Callender [39, p. 25] provide a simple, user-centred view to search as a process in general. They call it "The Anatomy of Search" as in Figure 1.6. Their map to search features five elements: users, creators, content, engine, and interface [39, p. 25].

1.3 Search engine user behaviour

Knowing how people search and what affects their search behaviour is an important factor in designing advanced search engines, interfaces and information management support systems. There is a large number of research done in this field, most involving medium to large scale user studies. Only a fraction of these will be covered in the following subsection.

Prof. Dirk Lewandowski from Hamburg University of Applied Sciences, Germany describes the typical behaviour of the Web search engine user: "a user only types in one or a few keywords and expects the search engine to produce relevant results in an instant" [36]. In fact, his opinion is to a certain degree backed up by a number of

user studies [35, 29]. Back in 2008 he acknowledged that the major problem in Web searching was still the same as it had been 10 years before: relevance [36]. Today, the same probably holds true as search engines have surely improved a great deal over the years, but their results are still far from perfect.

Jansen and Spink (2006) compare the changes in session length, query length, operator usage, and number of results pages viewed across these nine studies from 1997 to 2002. Their comparative analysis shows that for the US-based search engines the percentage of one-term queries is holding steady, within a range of 20–29% of all queries. For the European-based search engines it is within a range of about 25–35% [29]. Lewandowski finds that the average query length is 1.7 words for German language queries [35]. Other studies have shown that queries in English are a bit longer, but he thinks that this has to do with the specifics of the German language, where there is heavy use of compound words [36].

When compared to some more recent studies, there seems to be a trend such that search queries on all the major search engines are starting to get longer and longer. Hitwise has established (see Figure 1.7) that from January 2008 to January 2009 one and two-word search engine queries became slightly less popular, while the number of three-word queries remained flat. Instead, a growing number of users are now opting to use longer queries [51]. This could indicate that searchers are not totally happy with the results they get from short and unspecific queries and are becoming more sophisticated in how they structure their queries. Interestingly, Lewandowski maintains that search engine users are indeed easily satisfied [36]. When asked about the quality of the results they achieve, they usually express satisfaction with their searching strategies [36].

According to Jansen and Spink (2004), session durations are in most cases 15 minutes or less, although the average is a couple of hours [48, p. 185]. They also note that a substantial percentage of Web sessions are even less than five minutes [48, p. 185]. Although already dated, this is an interesting finding that could shed some light on the distribution of search session duration. We can expect to see two types of search sessions: simple information lookup which often takes less than five minutes, and exploratory search taking hours. Authors also mark a growth in multitasking sessions, which are sessions that include at least two topics [48, p. 185].

Another interesting related area is assessing user search behaviour when deciding which links to follow in rank-ordered result lists. Mark T. Keane, Maeve O'Brien, and Barry Smyth show people do manifest some bias, favouring items at the top of results lists, nevertheless they also seek out high-relevance items listed further down a list [32].

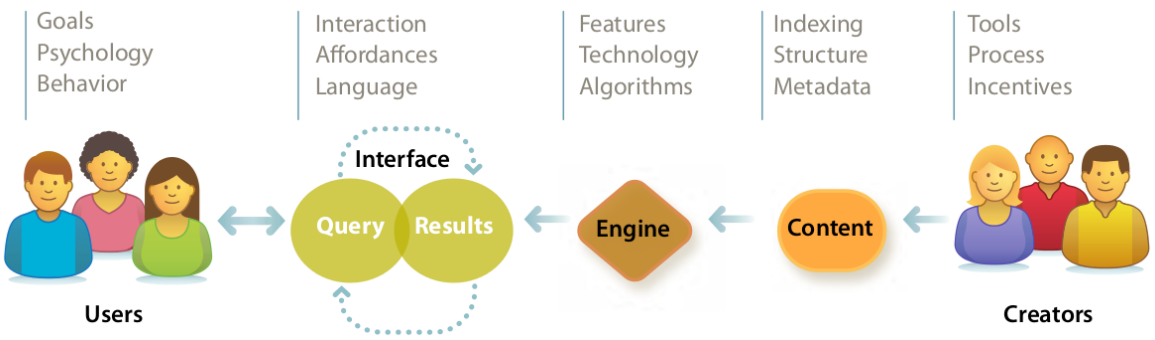


Figure 1.6: The anatomy of search (taken from [39])

Percentage of U.S. clicks by number of keywords				
Subject	**Jan-08**	**Dec-08**	**Jan-09**	**Year-over-year percent change**
1 word	20.96%	20.70%	20.29%	**-3%**
2 words	24.91%	24.13%	23.65%	**-5%**
3 words	22.03%	21.94%	21.92%	**0%**
4 words	14.54%	14.67%	14.89%	**2%**
5 words	8.20%	8.37%	8.68%	**6%**
6 words	4.32%	4.47%	4.65%	**8%**
7 words	2.23%	2.40%	2.49%	**12%**
8+ words	2.81%	3.31%	3.43%	**22%**
Note: Data is based on four-week rolling periods (ending Jan. 31, 2009; Dec. 27, 2008; and Jan. 26, 2008) from the Hitwise sample of 10 million U.S. Internet users.				
Source: Hitwise, an Experian company				

Figure 1.7: Search query length findings from Hitwise[51]

Their experiment shows that when lower-relevance-ranked items are placed on top of the results list, they are chosen more often by searchers, despite their limited relevance [32]. On the contrary, when highest-relevance items are placed last, they are being chosen by users considerably less often [32].

Granka et al. conducted eye-tracking analysis which seem to indicate that attention paid to links ranked 1 and 2 is almost equal [21]. After the second link, fixation time drops off sharply. In addition, there is an interesting dip around results 6 and 7, both in the viewing time as well as in the number of clicks [21]. This phenomenon can be justified by the fact people are intrinsically lazy and only the first 5-6 links are visible without scrolling. Unlike for ranks 2 to 5, the abstracts ranked 6 to 10 receive approximately equal attention [21]. Various studies [29, 36, 21, 32] point out that a sharp drop occurs after link 10, as ten results are usually displayed per page and most users do not go past the first results list.

1.4 Related work in exploratory search and search task logging systems

Georg Singer, Dmitri Danilov, and Ulrich Norbisrath [44] argue that the exploratory search concepts of *aggregation*, *discovery*, and *synthesis* are today the most time consuming activities, especially when fulfilling a complex information need. After a thorough theoretical analysis, they come to a conclusion that these three terms – aggregation, discovery, and synthesis – are the main steps or "three pillars" in the process, that Marchionini calls exploratory search [44].

Singer et al. view *aggregation* as the "support for selecting, storing, and accessing the relevant documents for a certain aspect of a search need" [44] and maintain that it is supported to some extent by all systems which they investigated, including the classic search interfaces of Google and Bing. There is also support as far as dynamic query interfaces, faceted browsing, collaborative search tools, social search, and universal

search interfaces are concerned. However, they are of the opinion that even the support for aggregation is just in its roots and could theoretically serve users better [44].

Discovery means the support for finding unknown relevant documents or information especially in new and unknown categories [44]. When it comes to discovery, the authors have found that "discovery is nearly not supported in standard Web search interfaces and that first support for information discovery is given by dynamic query interfaces, collaborative search tools, faceted browsing, and universal search interfaces" [44]. This also means that we are about to see major breakthroughs in this area, one possible field being graph visualization techniques that will tremendously impact the field of information discovery in the middle and long term.

Singer et al. understand *synthesis* as the support for compiling multiple documents into one and extracting relevant information [44]. Interestingly, when it comes to synthesis, their survey shows that this aspect of exploratory search is not supported at all in present search systems. Information synthesis comprises amongst others the steps information ordering, automatic editing, information fusion, and information compression which all require heavy use of artificial intelligence and research in those areas has not progressed far enough to come up with commercially usable solutions [44]. However, new and promising innovations can be seen for all three activities.

Singer et al. [45] present a new methodology and corresponding tools to evaluate the user behaviour when carrying out exploratory search tasks as well as to describe and quantify their complexity. These tools consist of a client called Search-Logger, and a server side database with front-end and an analysis environment [45]. The client is a plug-in for Firefox Web browsers. In order to evaluate the system, they compiled a pilot user study consisting of seven search tasks performed by ten participants mainly with an academic background. The main findings of their experiment are that when carrying out exploratory search tasks, classic search engines are mainly used as a gateway to the Web [45]. After that "users work with several search systems in parallel, they have multiple browser tabs open and frequently use the clipboard to memorize, analyse and synthesize potentially useful data and information" [45]. The idea of my Search Logger arose from this project to try and overcome its certain limitations (see the comparison of my solution with the original Search Logger in Section 2.4 for more information). In a related project for evaluating mobile search, Gleb Stsenov adapted the Webkit-based browser itself, enabling intense monitoring on the client side without confusing security messages [50].

Singer, Norbisrath, and Lewandowski [46] took the above-mentioned pilot user study [45] a step further by carrying out an experiment with 60 people in Hamburg, Germany. Their main goal was to investigate the characteristics of complex information needs in the context of Web search engines. Again, they use the Search-Logger plug-in for Firefox to record the user behaviour. As a result, they present a number of simple measures for complex search tasks which also serve as proxies to characterize this kind of task. For example, all time-based measurements (i.e. search time, time on search engine results pages, reading time) are higher than in simple search tasks; the average number of sessions in complex search tasks is larger which may indicate users having difficulty in completing the search tasks; and users are entering longer queries for complex search tasks as well as refining their queries more times when they may be unsure about how to express their information need [46]. Yet, as far as the final outcome of the search process is concerned, it seems to be difficult to distinguish successful from unsuccessful search behaviour [46]. They claim that successful tasks

are carried out with more strategy; users start issuing a query and then either narrow the query or extend it, which can be interpreted as a sign of strategy and playing with the result space. They also spend less times on search engine result pages. This leads us to a conclusion that experienced searchers seem to be able to get an overview of the results quicker than inexperienced searchers [46].

In his upcoming PhD thesis Georg Singer outlines three main causes impacting search task complexity [43]:

- a lot of information needs to be processed, read, qualified and collected;
- the collected information needs to be synthesized into a single document, causing a lot of effort;
- the openness of a task where a lot of effort is needed to discover the dimensionality of the result space and the main aspects of the task.

He also outlines the ATMS approach [43] to improve the support for complex search in search engines:

1. Build awareness
2. Offer a task-based search structure
3. Monitor the search process and help if needed
4. Share best practices with other users

I have found several other tools designed mainly for the same purpose of logging user events. The first tool to look at is called "Wrapper" by Bernard J. Jansen [28]. It addresses a fundamental issue in exploratory search evaluation in that user may seek information over an extended period and on multiple information systems. Being a desktop application, it logs a wide range of user interactions, such as interactions with the browser toolbars, interactions with the system clipboard, scrolling of results listing or documents, and numerous implicit feedback actions, such as bookmark, copy, print, save, and scroll [28]. Although this broad logging functionality is an advantage, the software does not collect explicit user feedback [45] and it only works on the Windows operating system. Wrapper also does not have the functionality built in, to relate a set of logs to a certain search task [45]. The concept of search task does not exist in this approach [45].

Another tool is a browser plug-in that was created by Fox et al. in 2005 [19]. Fox's approach was implemented as an add-on for the Internet Explorer [19]. It was the first tool to gather explicit as well as implicit information during searches at the same time [45]. Explicit feedback was collected at two levels of detail: for individual result visits and for the overall search session. A state machine was developed to prompt the user for feedback at appropriate times [19]. Their data was collected in a workplace setting among Microsoft employees during a period of six weeks to improve the generalizability of the results. Unfortunately, Fox's Internet Explorer add-on is not publicly available [45] and is not designed to work on other platforms and browsers.

A third tool is an open-source project called "Lemur Query Log Toolbar" [52]. It is a Web browser plug-in that captures user search and browsing behaviour to support research on information seeking behaviour, learning to rank, and related topics [52].

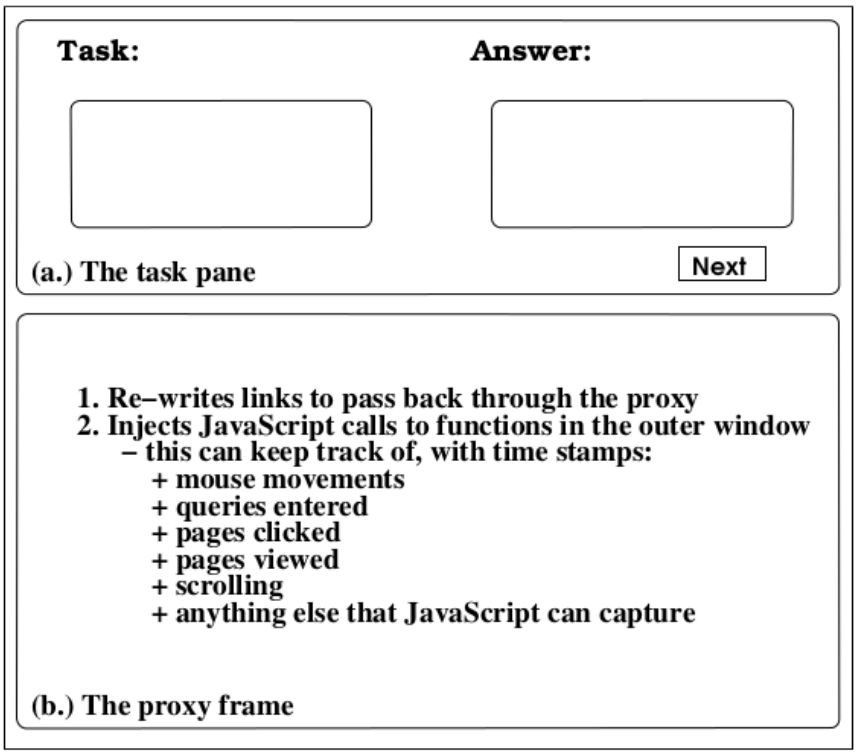


Figure 1.8: The proxy set-up for the study of Feild et al. (taken from [17])

The solution is currently available for Firefox and Internet Explorer. The creators of the toolbar have put a lot of emphasis on user privacy. For instance, the toolbar allows the user to set up phrases and items that can be "blacklisted" – that is, if a URL, copied text, or search result has the found blacklisted item in it, the matched item will be replaced by the generic text "##–##" [52]. Also, users can generate a random session ID when they choose to upload data to the server and, of course, they encrypt user data for privacy [52]. Nevertheless, it does not collect explicit information from the user and is not designed for exploratory search tasks.

Henry Feild et al. [17] address the relationship between searcher self-efficacy assessments and their strategies for conducting complex searches. From their initial experiments of using Amazon Mechanical Turk to conduct experiments in this area they discovered that "it is too much to expect Amazon Mechanical Turk (see 1.5.3) workers to install new software in their browsers, especially software that may inadvertently violate the worker's privacy in other browsing, unrelated to the assigned search task" [17]. Thus, they set up an environment on an Amazon Mechanical Turk HIT page in which they show each "turker" a page with a task pane and an embedded frame, which points to the proxy (Figure 1.8). The proxy frame is directed to a modified search engine interface and it rewrites all links on every page that passes through so that those pages are redirected via the proxy as well. Their solution utilizes JavaScript injection in a way that events, such as pages visited and mouse movements, can be logged [17]. Similarly with my solution, search logs from users on a variety of browsers can be successfully captured using their approach. What is different, however, is that users are forced to browse the Web for solving search tasks inside the proxy frame which can be just a small part of the browser window. And even more importantly, this method severely limits users' freedom to search in a way and in a surrounding they

are used to. Feild et al. reported themselves that some users had chosen not to adhere to the rules by doing all the search in a non-logged environment and just copying over the answers [17].

Henry Feild, James Allan, and Joshua Glatt present "a novel framework for collecting, storing, and mining search logs in a distributed, private, and anonymous manner" [18]. Their project is called CrowdLogging and it is motivated jointly by a need to have search logs available to researchers outside of large search companies and a need to instil trust in the users that provide search data [18]. The framework gives researchers access to up-to-date search data and also allows users an unprecedented amount of control over their data [18].

CoScripter [23, 56] is a macro recorder implemented as an extension for the Mozilla Firefox browser. It records user actions and saves them in semi-natural language scripts. The scripts made are saved in a central wiki for sharing with other users. CoScripter was created by a research team at IBM led by Allen Cypher, and it allows to record user actions on the Web, play them back, and share them with others [7]. For instance, one popular script quickly automates the process of adding a phone number to the national do not call registry.

Further tools, which I will not discuss here, are: The HCI browser [11], The Curious Browser [12], and Weblogger [42]. Some of these are either discontinued, or do not fulfil my requirements enough to be included in this comparison. All tools mentioned, have some features with my proxy-based Search Logger in common (i.e. the logging of user-triggered events), while some aspects are not covered at all. Most of the tools were developed for evaluating the query level while my solution was purely developed for evaluating the exploratory search task level. None of the tools have the built-in functionality to have a user-defined set of exploratory search tasks carried out and published to a dedicated repository of search stories by a test group in a non laboratory environment without any time constraints. Realistic user study results for exploratory search experiments will only be possible if the participants are free of any time constraints and can search in a way and in a surrounding they are used to [45].

1.5 Crowdsourcing feasibility study

A theoretical feasibility study was carried out in fall 2010 to study the meaning of crowdsourcing, related projects, and possible benefits and shortcomings as a means for user-based relevance evaluation in an academic environment. At that time I decided not to go forward with it for two reasons: 1) I deemed porting our existing tools to a crowdsourcing platform technically too complex and way out of scope; and 2) human experimentation in the lab seemed a good low-tech alternative. Nevertheless, this section presents the most interesting findings about crowdsourcing in the context of my thesis.

1.5.1 Motivation

The research in IR is often criticized for two main reasons: 1) lacking a more formal framework as a basic foundation; and 2) lacking robust and consistent testbeds and benchmarks [6, pp. 159-160]. Baeza-Yates and Ribeiro-Neto argue that the first of these criticisms is difficult to dismiss "due to the inherent degree of subjectiveness associated with the task of deciding on the relevance of a given document to an information

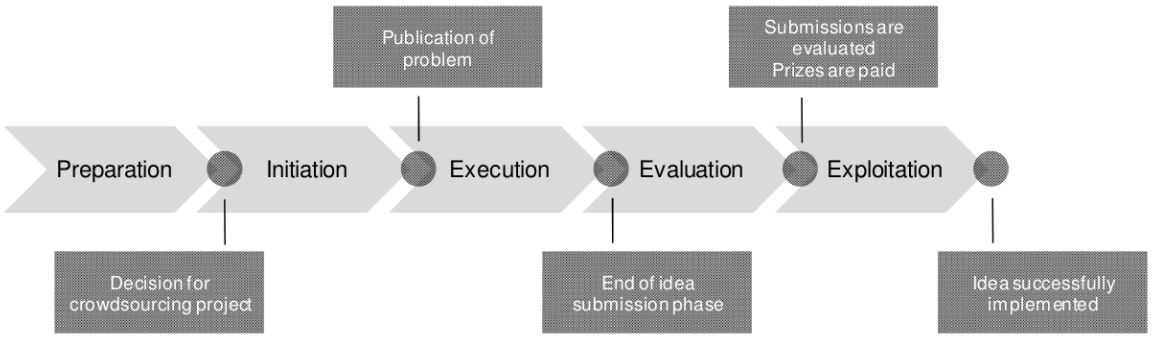


Figure 1.9: The crowdsourcing process (taken from [49])

need" [6, pp. 159-160]. Thus, they do not expect any developments on this front any time soon. However, there are attempts to deal with the second main problem. Text REtrieval Conference (TREC) is an annual promoted conference dedicated to experimentation with large datasets comprising millions of documents or even more [6, pp. 159-160]. A set of experiments is designed for each TREC conference which are held since November 1992. The goal of the conference series is to encourage IR research from large text applications by providing a big test collection, uniform scoring procedures, and a forum for organizations interested in comparing their outcomes. Each TREC workshop consists of a set tracks, areas of focus in which particular retrieval tasks are defined. TREC 2012 tracks include a Contextual Suggestion Track, a Crowdsourcing Track, a Knowledge Base Acceleration Track, a Legal Track, a Medical Records Track, a Microblog Track, a Session Track, and a Web Track [53].

In 1950s Cyril Cleverdon, a librarian at the College of Aeronautics at Cranfield in the UK, initiated a series of experiments out of which came the systematic evaluation of IR systems. These experiments provide a foundation for the evaluation of information retrieval systems, and are now called the Cranfield experiments [6, p. 132]. However, as far as my created tools for detailed search task log creation, annotation and publishing are concerned, proper evaluation of the user interface and of the interactions initiated by the searchers requires methods that go beyond the framework of Cranfield experiments. Accordingly, I determined that user-based evaluation is best suited for this task.

Crowdsourcing has emerged as a promising alternative for relevance evaluation because it combines the flexibility of the editorial approach (assembling a large body of editorial staff to judge the relevance of search results as suggested in the Cranfield experiments) at a much larger scale [2]. By definition, crowdsourcing is a term used to describe tasks that are outsourced to a large group of people (also known as "workers") instead of performed by an employee or contractor [6, p. 170]. It is an open call to solve a problem or carry out a task and this service is usually paid by the benefiting organization. Yet, the lower cost of running experiments makes this approach very attractive for a variety of purposes and environments [6, p. 170].

According to Stanoevska-Slabeva, there are various attempts to give an overview of crowdsourcing related processes identifying and analysing the underlying characteristics [49]. An aggregated view on the crowdsourcing process is provided by Gassmann et al. [20], who consider 5 steps (see Figure 1.9): 1) Preparation; 2) Initiation; 3) Execution; 4) Evaluation; and 5) Exploitation.

1.5.2 Crowdsourcing in academia

Since 2006, when the term was coined, various papers have been published about applying the notion of crowdsourcing to different fields of computer science, mainly for user-based evaluation in IR. Here I aim to provide a detailed overview of the main results of selected related studies.

Omar Alonso, Daniel E. Rose, and Benjamin Stewart describe a new approach to relevance evaluation called TERC, based on the crowdsourcing paradigm, in which many online users, drawn from a large community, each performs a small evaluation task [2]. In contrast to traditional evaluation approaches, their solution has fast turnaround (all the HITs on Amazon Mechanical Turk completed in a couple of days), low cost ($125 for their task), flexibility and high quality (by getting several opinions and eliminating the noise) [2].

Requiring feedback from workers enables researchers to get justifications for certain answers. Post-processing this feedback can help improve instructions and systems [6, p. 170]. Interestingly, some studies claim that "turkers [workers in Amazon Mechanical Turk] not only are accurate in assessing relevance but in some cases were more precise than the original experts" [1]. Also, it has been established that only a small number of non-expert annotations per item are necessary to equal the performance of an expert annotator for producing annotations in natural language texts [47]. Thus, many large labelling tasks can be effectively designed and executed in this method at a fraction of the usual expense [47].

Daren C. Brabham surveyed 651 people at iStockphoto, in essence, a giant, royalty-free stock photography agency. His main findings are that the desire to make money, develop individual skills, and to have fun were the strongest motivators for participation at iStockphoto, and that the crowd at iStockphoto is quite homogeneous and elite [9]. Unlike general purpose social networks, Brabham's survey indicates that friendship and other social networking features are secondary to individual fulfilment and profit in the crowdsourcing context [9]. He portrays the typical iStocker as a white, married, middle to upper-class, higher educated, 30-something, working in a so-called "white collar" job with a high-speed Internet connection in the home [9]. Geographically, 56.4 percent of respondents hailed from North America, 33.4 percent from Europe, and 4.1 percent from Australia and New Zealand [9].

Jiang Yang, Lada A. Adamic, and Mark S. Ackerman examine the behaviour of users on one of the biggest Witkey Websites in China, Taskcn.com. They observed several characteristics in users' activity over time and tried to find interesting behavioural patterns. For instance, most users become inactive after only a few submissions while others keep attempting tasks [63]. Over time, users tend to select tasks where they are competing against fewer opponents to increase their chances of winning [63]. Some users are tempted to select tasks with higher expected rewards [63]. Yet, on average, those users do not increase their chances of winning, and in some categories of tasks, their chances actually decrease [63]. The authors have found that there is a very small core of successful users who manage not only to win multiple tasks, but to increase their win-to-submission ratio over time. This group of people proposes nearly 20% of the winning solutions on the site [63].

Often it just comes to finding the right facts in the crowd. Community Question Answering (example of this is Yahoo! Answers) requires effective answer retrieval. Jiang Bian, Yandong Liu, Eugene Agichtein, and Hongyuan Zha present a general ranking framework for factual information retrieval from social media [8]. They also

demonstrate that their method is highly effective at retrieving well-formed, factual answers to questions, as evaluated on a standard factoid Question Answering benchmark [8]. The authors provide result analysis to gain deeper understanding of which features are significant for social media search and retrieval [8].

Tingxin Yan, Vikas Kumar, and Deepak Ganesan have developed CrowdSearch, an accurate image search system for mobile phones. Their CrowdSearch combines automated image search with real-time human validation of search results [62]. Automated image search is performed using a combination of local processing on mobile phones and backend processing on remote servers [62]. Amazon Mechanical Turk (see 1.5.3) is used for performing human validation. What makes their solution particularly useful, is a novel predictive algorithm that determines which results need to be validated, and when and how to validate them [62]. Thus, they were able to show that CrowdSearch achieves over 95% precision across multiple image categories, provides responses within minutes, and costs only a few cents [62].

Vamshi Ambati, Stephan Vogel, and Jaime Carbonell have adopted active learning and crowdsourcing techniques to machine translation. Their experiments with crowdsourcing on Amazon Mechanical Turk have shown that it is possible to create parallel corpora using non-experts and with sufficient quality assurance, a translation system that is trained using this corpus approaches expert quality [4]. They also proposed Active Crowd Translation (ACT), a new paradigm where active learning and crowdsourcing come together to enable automatic translation for low-resource language pairs [4].

Harry Halpin, Daniel M. Herzig, Peter Mika, et al. discuss the retrieval of objects in response to user formulated keyword queries – the ad-hoc object retrieval [22]. In semantic search like in many other domains it is necessary to evaluate the relevancy of search results but by this time there still is no good non-human evaluation of relevancy of search results. Therefore the authors of this paper opted for crowdsourcing. They posted near 600 HITs using Amazon Mechanical Turk (see 1.5.3) in 2 days and have spent only $350 for conducting the whole experiment. Each HIT consisted of 12 query-result pairs for relevance judgements. Their main findings from the crowdsourcing experiment are that irrelevant results are the easiest to agree on, followed by perfect results [22]. To see how this agreement compares to the more traditional setting of using expert judges, they also used expert judges for the same tasks. Authors claim that there is practically no difference between the two and three-point scales, meaning that expert judges had much less trouble using the middle judgement (potentially relevant results) [22]. All in all, the experiment was both fast and cost-effective [22].

1.5.3 Crowdsourcing platforms

Nowadays crowdsourcing is widely used as a means for solving complex scientific problems, producing new design and ideas, finding competent and motivated human resources and freelancers, forecasting market trends, utilizing open innovation for new initiatives. Here I introduce some of the biggest crowdsourcing platforms.

InnoCentive InnoCentive (http://www2.innocentive.com/) is a crowdsourcing platform with a community of over 200,000 members. InnoCentive (see Figure 1.10) posts a wide range of problems from organizations in areas such as engineering, computer science, chemistry, life sciences and business [14]. They connect people interested

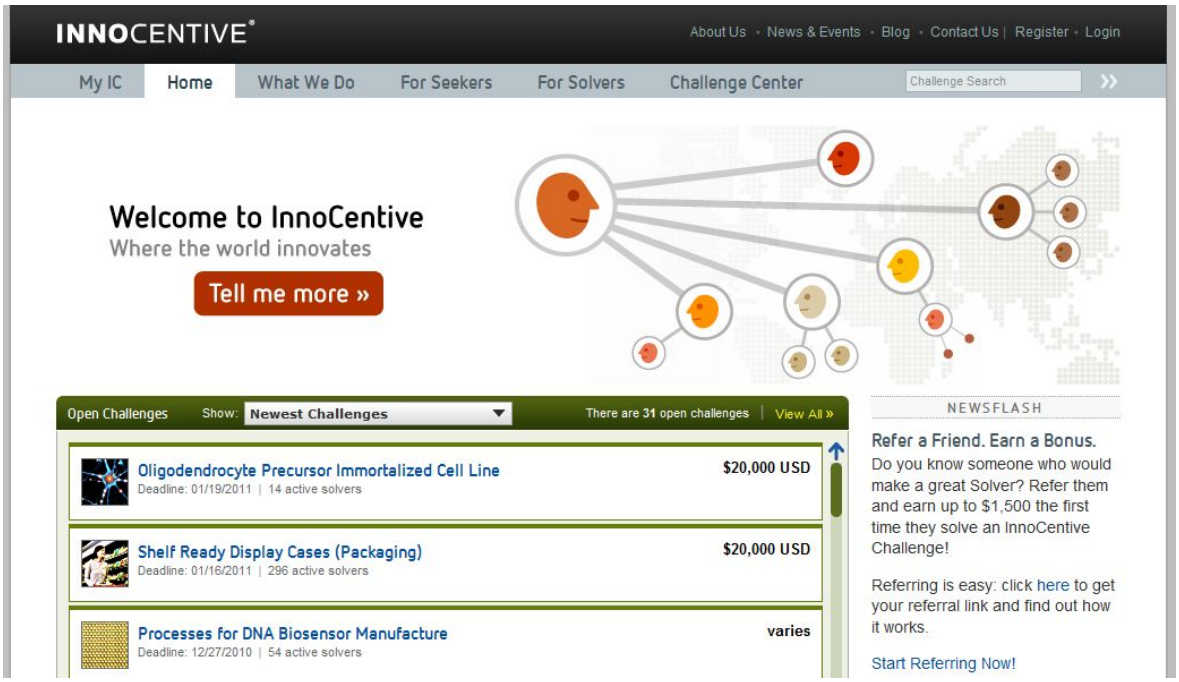


Figure 1.10: InnoCentive connects seekers and solvers.

in solving critical and pressing problems (solvers) with organizations seeking answers (seekers) [27]. As of November 2010, InnoCentive's Web site features an award from the non-profit Prize4Life foundation for $1 million for finding a biomarker that measures ALS disease progression [25]. Nature.com requests novel insecticidal proteins, or genes encoding insecticidal proteins, for $1,250,000 USD [26].

TopCoder TopCoder (http://www.topcoder.com/) is a crowdsourcing platform with a community of nearly 270,000 members. TopCoder (see Figure 1.11) posts computer programming problems from organizations in the form of competitions. The community are then asked to design, develop, and test the software developed in the competitions [14]. For example, TopCoder hosts 1) algorithms competitions in which competitors are given a set of algorithmic problems and have 75 minutes to correctly solve as many as they can; 2) design competitions where participants are given a set of user requirements and attempt to convert them into a usable software design specification; 3) development contests where people are given a set of design specification and attempt to write software components that match those specification; 4) Marathon Matches in which contestants are given a particularly difficult algorithmic problem and the scoring is done by computer based on criteria specifically suited to the problem; 5) studio contests where people are asked to show off their creative skills in a competitive environment; 6) bug races, and more [59]. "The work in design and development produces useful software which is licensed for profit by TopCoder. Competitors involved in the creation of these components are paid royalties based on these sales. The software resulting from algorithm competitions – and the less-frequent marathon matches – is not usually directly useful, but sponsor companies sometimes provide money to pay the victors [59]".

Peer recognition is given via developer ratings which are publicly available. TopCoder provides this in the form of detailed ratings and performance metrics are kept to track a developers standing within the community. This includes skill ratings and

Figure 1.11: TopCoder posts computer programming problems from organizations in the form of competitions.

history of submissions. TopCoder also acts as a recruitment centre where companies can find the best software developers regardless of their location [15].

Amazon Mechanical Turk Amazon Mechanical Turk (https://www.mturk.com/) is a platform with a community of over 100,000 members (see Figure 1.12) [14]. The name Mechanical Turk comes from "The Turk" – a chess-playing automaton of the 18th century made by Wolfgang von Kempelen. It toured Europe beating the likes of Napoleon Bonaparte and Benjamin Franklin. It was later revealed that this "machine" was not an automaton at all but was in fact a chess master hidden in a special compartment controlling its operations [55].

They give businesses and developers access to an on-demand scalable workforce [3]. The platform co-ordinates the use of human intelligence to perform tasks which computers are unable to do. The Requesters are able to pose tasks known as HITs (Human Intelligence Tasks), such as finding the email address for the company and Website, translating text across multiple languages, or assessing the relevance of the given search results [3]. Workers (called Providers in Mechanical Turk's Terms of Service) can then browse among existing tasks and complete them for a monetary payment set by the Requester.

To place HITs, the requesting programs use an open Application Programming Interface (API), or the more limited Requester site. Requesters, typically corporations, pay 10 percent over the price of successfully completed HITs to Amazon [55]. To the application, the transaction looks similar to a remote procedure call: the application sends the request, and the Web service returns the results [6, p. 171]. Because HITs are typically simple repetitive tasks and users are paid often only a few cents to complete them, some have criticized Amazon Mechanical Turk as a "virtual sweatshop" [55].

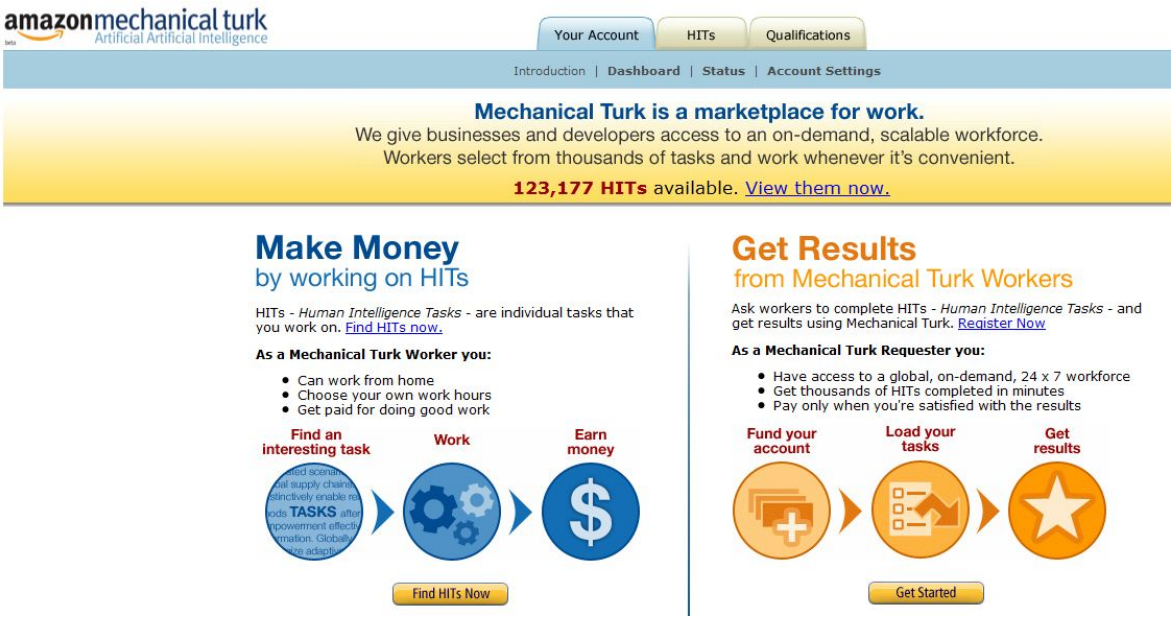


Figure 1.12: Amazon Mechanical Turk allows anybody to post tasks that need to be completed like writing product descriptions or finding broken links on a Website.

1.5.4 Practical scenarios for exploratory search

In this subsection, a few practical scenarios are envisioned to highlight the possible benefits of incorporating crowdsourcing methodology to logging, annotating and evaluating exploratory search tasks. These scenarios are purely hypothetical and are not implemented as discussed.

Pre-conditions Ann and Diana both have a similar information need. Ann and Diana are going to use the search task repository to seek for existing search logs and publish their own annotated ones. We will assume that they are both participating in a user study and their computer systems are fully set up to log their Web activity. Bruno and Colin are monitoring new HITs on Amazon Mechanical Turk. They do not need to have the Search Logger installed and running.

Ann publishes a search log Ann wants to take dance classes in Tartu. First, she searches from existing search logs in the search task repository. She happens to be the first person to be looking for dance classes in Tartu. Thus, there are no relevant search results. Next she starts up the Search Logger, describes her search task in the pre-form, and starts searching as usual. She types "dance classes in Tartu" into Google search box and finds two promising matches. Then she decides to refine her query a few times to get more relevant results. Now she has pinpointed four classes in Tartu area to check out in person. Happy with the results, she reports her findings in the post-form and publishes her search logs.

Bruno and Colin validate Ann's log Ann's new search task has just appeared on Amazon Mechanical Turk. They can see Ann's search task description and are asked to select the log parts which seem to be helpful for solving the initial problem as well as the obviously irrelevant ones. In addition, Bruno and Colin also get to validate the

ultimate search result posted by Ann. That way both the proposed answer as well as sections from the published search logs get either positive or negative feedback from Bruno, Colin and others doing the evaluation. Bruno and Colin get $0.5 for their work.

Diana uses Ann's log to speed up her search session Diana also wants to take dance classes in Tartu. First, she searches from existing search logs in the search task repository. She happens to be not the first person to be looking for dance classes in Tartu. She stumbles upon Ann's validated search logs. It is easy to interpret this information because the most relevant Web pages visited by Ann as well the most helpful parts in her proposed answer are in a much bigger font than the irrelevant information. This was made possible by the crowd's generalized feedback at Amazon Mechanical Turk. Nevertheless, Diana wants to carry out some additional searching on her own. All of her search activity is logged by the Search Logger, and at the end Diana publishes her own logs with annotations, too. They become a new HIT on Amazon Mechanical Turk.

1.6 Roundup

Here I gave a theoretical overview of what is already done in relation to my thesis. This overview included giving and explaining important definitions in IR, related work in exploratory search, Internet search behaviour and search task logging systems, a crowdsourcing feasibility study, and more.

In essence, the exploratory search concepts of *aggregation*, *discovery*, and *synthesis* are today the most time consuming activities, especially when fulfilling a complex information need. Thus, we need to develop new search support systems targeted at solving problems arising from these activities. Most of the current search task logging research is focused on developing browser plug-ins, often requiring frequent updating from developers among other annoyances. None of the studied search task logging solutions was intrinsically operating system and browser independent. Also, they are not linked to a search task repository to facilitate easy publishing and sharing of users' search task stories. It was determined that crowdsourcing can be used to organize large scale user studies in exploratory search.

Chapter 2

Implementation and evaluation

This chapter outlines the main architecture and design of my search task logging and publishing software. It also gives practical scenarios to better understand how it works from a human perspective. Last but not least, it describes the evolution of this project throughout its history.

2.1 Practical scenarios

Here I give two main practical scenarios from the user's perspective.

2.1.1 Publishing a search task

As a user of the search task logging and reporting solution, I can log, edit, annotate and finally publish my own search tasks. Here I describe how publishing a search task works in practice because it includes all the previously mentioned activities.

First, I make sure everything in my local machine is set up correctly (see 3.1). Before deciding to publish a new search task I try to find a related search task that can already be found on the Web site of the search task repository (see 2.1.2). If this does not satisfy my information need, I open up my Web browser and go to the page of the main Web interface. I am prompted to fill in the pre-form which I will do. After that logging is automatically enabled by the software but I can temporarily turn it off for any reason (such as doing something personal or taking up another topic). I conduct my search task and after doing so fill-in the post-form. Now I can choose whether I want to publish the generated and collected logs right away or edit them. If I choose to edit and annotate my search logs, I can use the simple to use Search Task Log Editor interface. Then I can publish my edited logs for the world to see and use.

The logs get uploaded to the search task repository. This completes my scenario.

2.1.2 Finding a search task

As a user of the search task logging and reporting solution, I can search and read publicly available search stories without having to set up everything.

To begin with, I go to the Web site which contains all search logs published by all the members of the community. I think of a way to phrase my information need into a query. Then I press the "Go" or "Search" button to go to the results page matching my query. I can choose to study some of the published logs, or I may also reformulate

my search query to get a better match. Once I have found the relevant search tasks, I can use their synthesized knowledge to my advantage.

This completes my scenario. Alternatively, I may choose to publish my own search report as described in 2.1.1.

2.2 Requirements analysis

The following section outlines the requirements that were set for my search task logging, annotating and publishing solution. More information about the history of the development process can be found in Section 2.6. In addition, since a fair share of the requirements arose from the limitations of the existing browser plug-in based SearchLogger, these are discussed in Section 2.4.

From the side of functional requirements, the following activities must be supported by the solution:

- automatically logging at least two types of events
- logging starts and stops automatically during a search session
- logging can be easily turned on and off manually by the users
- crucial meta-information about the goals of the search task and their perceived success rate must be collected from the users
- the users have to be able to describe and tag their search task
- the users must be able to edit and annotate all collected and automatically logged information about their search session before publishing their search stories
- all published content must go to a public repository to enable sharing search experience
- performing searches from published search stories
- filtering and browsing all published search task logs by tags

Additionally, the solution has to comply with these non-functional requirements:

- search task logging, editing, annotating and publishing is fully usable in a browser window, no need to run anything from command line or use external text editors
- support multiple Web browsers (at least Google Chrome, Mozilla Firefox, and Internet Explorer) by architecture
- support multiple operating systems (at least Linux, Windows, OS X, and Solaris) by architecture
- logger set-up must be easy and have no more than five steps
- all common scenarios must be adequately documented
- use modular design to allow superior maintainability, extendibility and scalability

- none of the components may not cause any other running services and applications (i.e. Web browser) to behave abnormally and/or crash
- the solution has to be transparent enough to allow for authentic Web browsing in most cases
- the chosen architecture can be easily configured to block irrelevant data such as news and ads
- the chosen architecture can be modified to serve multiple clients in a Local Area Network (LAN)
- all components must be tested internally and known problems reported

In retrospective, it were mainly the non-functional requirements that affected to architectural decisions the most. The chosen proxy-based approach allowed me to offer cross-platform compatibility while also being very stable, maintainable and extendible. As for blocking irrelevant data such as news and ads, Privoxy is a powerful tool for this purpose and can be configured to accomplish this.

2.3 Architecture and design

Right from the start, the proxy-based search task reporting system has been a complex engineering challenge involving code written in multiple programming languages, interactions planned across many software modules (some of which have already been existing large projects themselves), and a Linux operating system configured to ease the set-up process for the user. This was the decision process to make sure that this solution is reliable, extendible and maintainable in the future.

The solution is comprised of two large units, which are the search task repository and the search task logging and publishing unit. The search task repository is a remote component, essentially a fairly simple WordPress blog, which enables search stories to be published automatically over XML-RPC protocol, search queries to be served, and search task logs to be displayed to the searcher. The logging system is much more complex: it consists of three sub-components (Figure 2.1).

First of these is the Web interface that mediates all relevant actions between all other components and the user. These are for example collecting explicit data from the user about the search task, controlling the execution of the logger, serving an easy to use log editing functionality, and eventually implementing the client side of XML-RPC (MetaWeblog API for WordPress was used) to publish the edited and user-annotated logs online. The Web interface is mainly written in PHP and HTML but also uses very little CSS. The second sub-component, written in Python, is the logger itself which listens to HTTP requests from the visited JavaScript-injected Web sites.

The third sub-component is a third-party open-source Web proxy Privoxy. According to its developers, Privoxy is a non-caching Web proxy with advanced filtering capabilities for enhancing privacy, modifying Web page data and HTTP headers, controlling access, and removing ads and other unwanted Internet junk [41]. Privoxy has a flexible configuration and can be customized to suit individual needs and tastes and it has application for both stand-alone systems and multi-user networks [41]. I took the challenge of making a working configuration for Privoxy to modify visited Web pages in

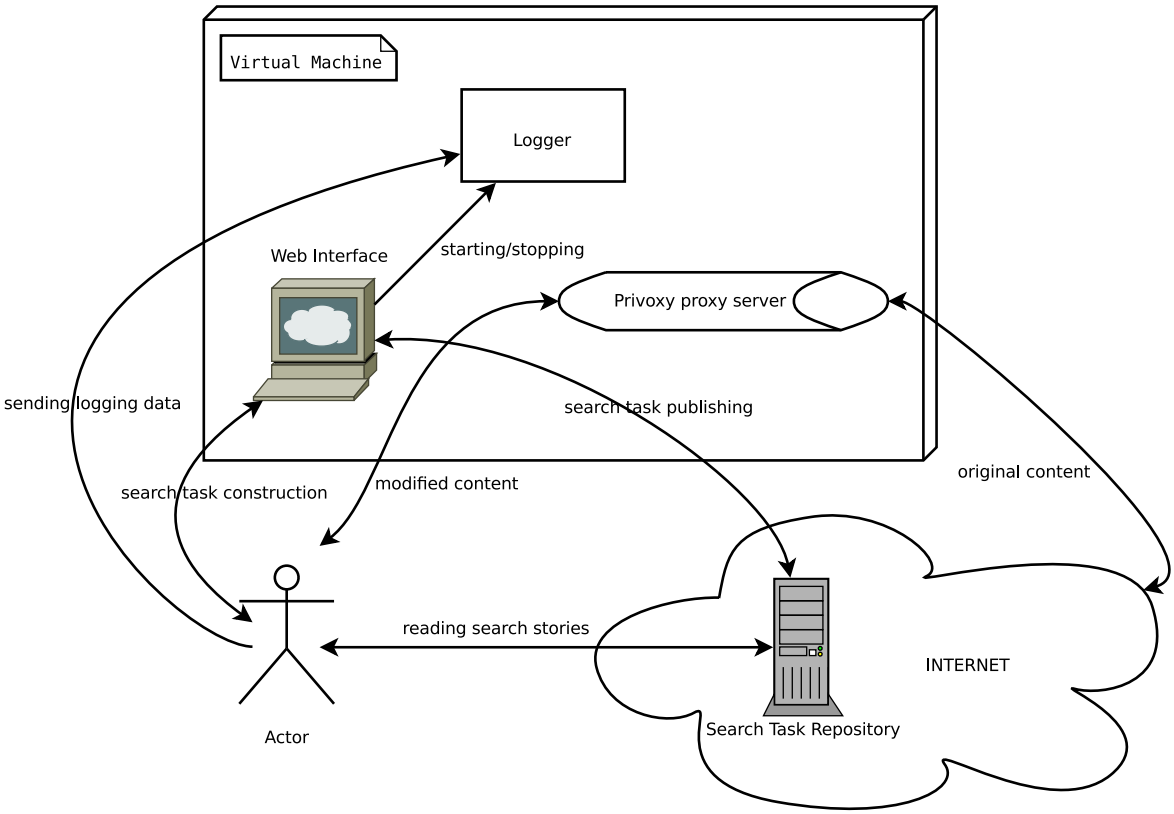


Figure 2.1: Architectural overview at a component level with interactions between components

a way that they send out meta-information to the logger sub-component. A JavaScript injection approach was chosen because it allows us to capture client-side information at a fairly low level. Currently only two types of events (page load and unload) are supported but related studies [17] have proved that much more could be logged via this approach (see section 4.2). In order to describe and enforce the JavaScript injection rules, I had to do some Perl programming.

All these three sub-components within my search task logging system are made to communicate with each other. The fact that certain settings are rather host system specific also meant difficulties with setting up the solution on new systems. To tackle this problem and to propagate the software on other computers besides my test environment, I took a Linux (Debian) operating system and set everything up as a VirtualBox image (see Figure 2.2). This way the users can benefit from a quick and easy set-up process without complicated installation and configuration without a significant trade-off in operating performance.

An interesting detail about this step is how I solved the networking problem between the host and the guest machine while maintaining Internet connectivity. It appears to be a common practice to define two network adapters: a NAT adapter for Internet and a host-only adapter for communications between the host and the guest machine. Host-only adapter is useful for creating private networks, where machines need access to one another, but not necessarily outside this subnet [13]. VirtualBox also supports other network adapter types, including a bridged adapter. This, however, was not a suitable option for my scenario. The main reason is that a bridged adapter requires a second IP-address for the bridge which might be tough to acquire in a WAN. Consequently,

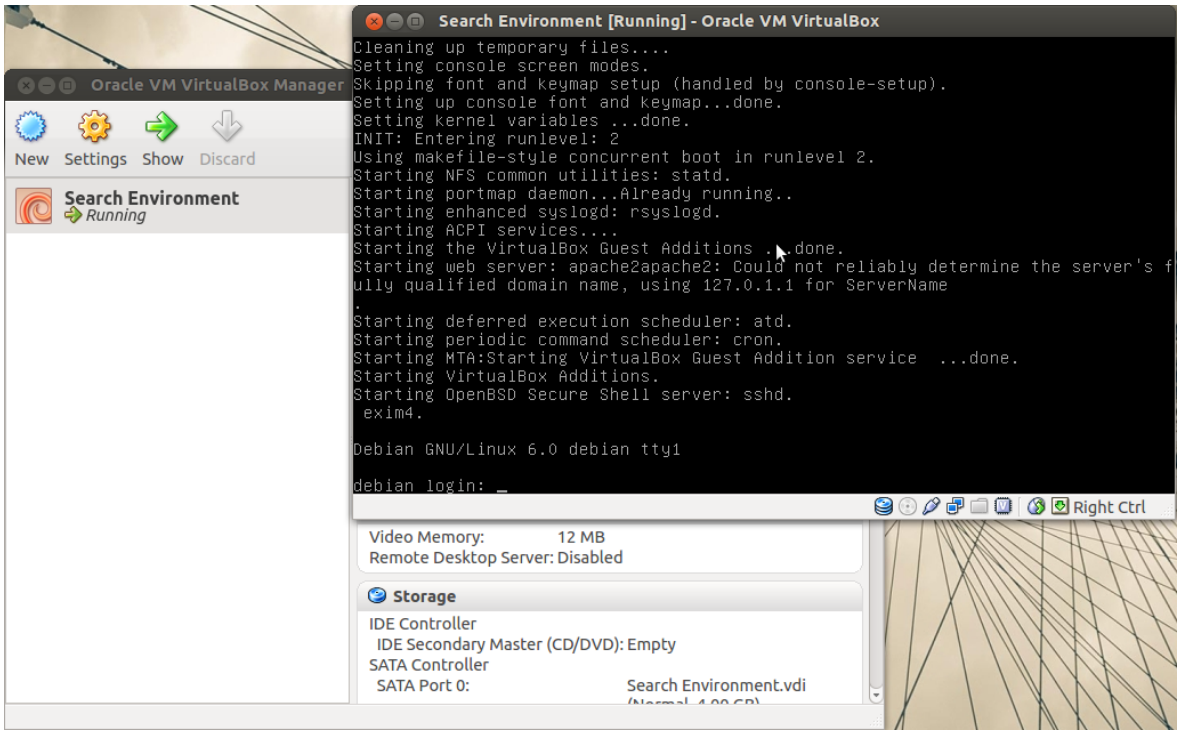


Figure 2.2: The solution for seamless distribution – a VirtualBox virtual machine

computers with direct Internet access (nowadays USB modems are widely used for mobile broadband) may not be able to use bridged networking because of the absence of a router [13].

Finally, the main activity conducted by the users using my solution needs to be clarified again. It is the process of logging, editing and publishing a new search task log to the search task repository (see Figure 2.3). At the beginning we have some users with specific information needs that can be just lookup tasks, yet these are likely to be related to exploratory or complex search tasks. Upon opening up the Web interface, they fill in the pre-form where they define, describe and tag their upcoming search task. My solution gives the users the possibility to express themselves relatively freely, so they may find this small additional work useful for their personal needs as well when they may want to come back to their already published search task to study how they found the answer. Also, this helps them to better understand and structure their intentions for the next step. Then, the logger is automatically enabled and the users can satisfy their information need as usual. After that, the users complete a small post-form to let the searching community know how successful they perceived themselves to be. Now, they can either publish their search logs immediately or choose to edit any information collected throughout this process (for example, removing some personal data or annotating the logs). If they choose to use the search task log editor, they can publish their search stories after performing some manual modifications. Now, the search logs get published in the repository of all contributed search tasks.

2.4 Comparison with the original Search Logger

The idea of developing a proxy-based search task logging and publishing solution came from out of necessity, because the existing logging solution [45] had significant prob-

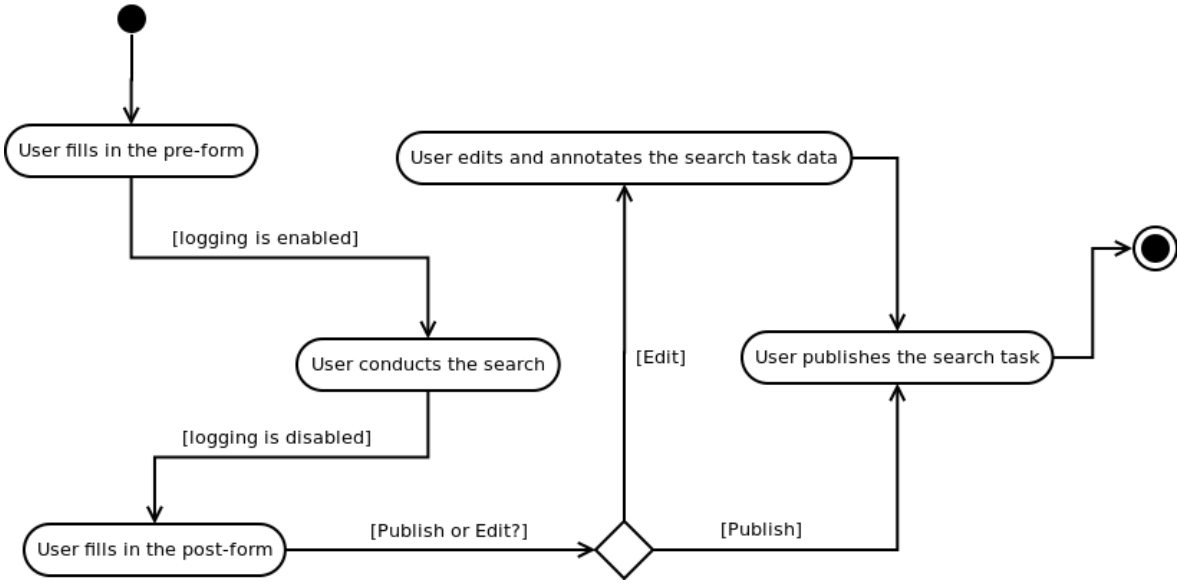


Figure 2.3: Activity diagram with the main activity

lems with maintainability, literally meaning that it only worked on a Mozilla Firefox browser with a certain version and tended to have a rather unpredictable behaviour across different operating systems. At first, I tried to solve these issues but with no avail. Then, I started looking at alternative options to achieve better cross-platform compatibility. A new type of search task logging solution was born. Here I compare the two approaches: browser plug-in and proxy based evaluation of exploratory search tasks.

Both of the Search Logger tools can be used to carry out user studies for search tasks independent of a laboratory environment. They collect implicit user information by logging a number of significant user events. Explicit information is gathered via user feedback in the form of questionnaires before and after each search task. While the plug-in based approach has the advantage of catching more user generated events, the proxy-based approach ensures platform and browser independence.

In case of the plug-in based approach, the Search Logger consists of a browser plug-in for Firefox, a remote log storage database and an analysis environment. The architecture of my proxy-based Search Logger is given in Section 2.3. Unlike its predecessor, this solution does not currently have an analysis environment. Instead, it provides a Web repository for uploading search task logs and conducting searches.

The plug-in can be distributed to the study participants by either publishing it on a Web-page or by sending the plug-in as an email attachment. Once the plug-in is received, it can be installed into an existing Firefox browser by dragging and dropping the plug-in onto the browser. After the installation is finished, the browser needs to be restarted. The users interact with Search Logger via a little icon that appears in the status bar of the browser at bottom right. A click on this icon opens up a window. In this window the user can start, stop and pause search tasks. Depending on the chosen state, the icon in the browser status bar either blinks if in logging mode or shows a still green logo indicating that logging is stopped. The questionnaires are implemented as HTML pages and can be edited with a standard HTML editor.

The proxy version is a bit more difficult to set up. The user needs to start the browser of choice, for example Firefox and then go to the “Options” menu. Under

	proxy	plug-in
logging behaviour	fewer events are logged (logging many more events can be implemented), but also less irrelevant data such as news and ads (altering Privoxy settings can block most of that as well)	all user-triggered events are logged, but also irrelevant requests not directly triggered by the user
installation	slightly more complicated, but the VirtualBox image takes most of the hassle away from the users; the trade-off is in larger download size (a 225 MB archive)	drag and drop
stability	very stable	plug-in can cause trouble in browser
compatibility	no compatibility issues as independent of browsers and operating systems	depending on actual browser release
usability	users do not notice the proxy	some security measures could not be turned off and create unneeded clicks for the study participants

Table 2.1: Comparison of the two ways for logging search tasks

the connection settings he can change the manual proxy settings to route all traffic through the proxy. As outlined in Table 2.1, the proxy based as well as the plug-in based solutions have their own advantages and disadvantages. While the plug-in based solution can log more user events, and the installation is a bit easier than in case of the proxy based solution, those advantages do not outweigh the disadvantages. As the plug-in based solution is tightly connected to the browser structure, keeping the plug-in compatible with the latest releases is a significant maintenance effort. In addition the plug-in based version can occasionally create trouble by making the browser a bit unstable. The plug-in based solution also has the disadvantage of logging too much information as any access to the Internet also from other plug-ins or JavaScript mechanisms (like pulling RSS feeds, news, or advertisements) is logged. This significantly clutters the results and complicates the evaluation.

Finally, an enhancement of proxy-based search task logging can potentially reduce the organizer's effort significantly in a lab environment where only one server is needed. This requires some small technical changes to be made in my solution and is not yet supported but I provide some insight into this modification in Future work (see Section 4.2).

2.5 Implementation details

In the following section, I will clarify some particular implementation details. This list includes the mechanism for injecting JavaScript into HTML pages to make them loggable, the inside of the search task logger, and details about how data is published

to the remote repository of user search stories. Finally, the WordPress platform choice for the repository is explained.

2.5.1 Rewriting Web pages

The corresponding functionality of injecting JavaScript into all Web pages that pass through the Privoxy proxy server had to be defined by Perl substitutions. Currently, I can log two types of events: when a Web page is loaded and when it is unloaded. Due to a lot of effort going to other areas of research, supporting more events was left as a task for the future (more on this in Section 4.2). My first goal was to write these commands in JavaScript:

```
<script type="text/javascript">
function loggerLoad() {
    sendLogEntry("pageload");
}
function loggerUnload() {
    sendLogEntry("pageunload");
}
function sendLogEntry(action) {
    request = new ajaxRequest();
    request.open("GET", "http://192.168.56.101:3128/?data="
        + action + "|+0+|" + window.location +
        "|+0+|" + document.referrer, true);
    request.send(null); }
function ajaxRequest() {
    try {
        var request = new XMLHttpRequest();
    } catch(e1) {
        try {
            request = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e2) {
            try {
                request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch(e3) {
                request = false;
            }
        }
    }
    return request;
}
</script>
```

This code defines the JavaScript functions which have to be placed somewhere inside the <head></head> tags. Function calls are defined as parameters for the <body> tag. For example, to achieve the effect of a message going to the logger sub-component immediately after a page is loaded, we would need to have some code defined inside the <head></head> tags as well as a function call like this: <body onload="loggerLoad()">.

Things get more interesting when trying to achieve exactly this by defining our desired behaviour with Perl substitution statements. Explaining each symbol in the following code would take too long but in my case rewriting the <body> tag was more difficult because this tag can have multiple properties and, what is even worse, there may be existing function calls in what I am about to rewrite myself, i.e. <body onload="some_function()" onunload="another_function()">. In this case, I insert

my function calls to the beginning, followed by a semicolon, and then append what was originally called. In practice, it looks like this:

```
s/(<body)(?(?=[^>]*onload=\"[^\"]*\")([^>]*)onload=\"([^\"]*)\"|([^>]*))/
    $1$2 onload=\"loggerLoad();$3\" $4/is

s/(<body)(?(?=[^>]*onunload=\"[^\"]*\")([^>]*)onunload=\"([^\"]*)
    \"|([^>]*))/$1$2 onunload=\"loggerUnload();$3\" $4/is

s/(<head[^>]*>)/$1<script type=\"text\/javascript\"> function loggerLoad
    () { sendLogEntry(\"pageload\"); } function loggerUnload() {
    sendLogEntry(\"pageunload\"); } function sendLogEntry(action) {
    request = new ajaxRequest(); request.open(\"GET\", \"http
    :\/\/192.168.56.101:3128\/?data=\" + action + \"|\+0\+|\" + window.
    location + \"|\+0\+|\" + document.referrer, true); request.send(null);
     } function ajaxRequest() { try { var request = new XMLHttpRequest();
    } catch(e1) { try { request = new ActiveXObject(\"Msxml2.XMLHTTP\"); }
     catch(e2) { try { request = new ActiveXObject(\"Microsoft.XMLHTTP\");
     } catch(e3) { request = false; } } } return request; } <\/script>/is
```

To enable this configuration, I created a new filter configuration file called user.filter in the configuration directory of Privoxy (its location on Linux is /etc/Privoxy). This file needed to be enabled in the main configuration file, and its actions triggered in the user.action file, all located in the same directory.

2.5.2 Search task logger

The purpose of my search logger is listening to incoming HTTP GET requests and intercepting messages coming from modified Web pages. I have developed two simple log creators capable of producing output for different purposes: one for "humans" and another for "geeks". Currently, only the "human" version is used. Here are two different examples of automatically generated search logs.

- Log output for "humans":

  At 2011-11-30 14:11:10 a user opened http://jobview.monster.com/Data-Analyst-Position-Job-Orange-County-CA-US-104170607.aspx. The user IP was 127.0.0.1. The user came from http://jobs.monster.com/v-business-q-data-analyst-jobs.aspx.
  It was 2011-11-30 14:11:22 when the user from 127.0.0.1 moved away from http://jobview.monster.com/Data-Analyst-Position-Job-Orange-County-CA-US-104170607.aspx.
  It was 2011-11-30 14:11:22 when the user from 127.0.0.1 moved away from http://fast.monster.demdex.net/dest2.html.
  At 2011-11-30 14:11:24 a user opened http://jobs.monster.com/v-business-q-data-analyst-jobs.aspx. The user IP was 127.0.0.1. The user came from http://www.google.ee/url?sa=t&rct=j&q=data%20analyst%20career&source=web&cd=4&sqi=2&ved=0CDkQFjAD&url=http%3A%2F%2Fjobs.monster.com%2Fv-business-q-data-analyst-jobs.aspx&ei=pzjWTuCFKeT04QTS-O3QAQ&usg=AFQjCNEyVah9vo4PhMlbQWW_Ur-2FaTFfA.
  It was 2011-11-30 14:11:27 when the user from 127.0.0.1 moved away from http://fast.monster.demdex.net/dest2.html.
  It was 2011-11-30 14:11:27 when the user from 127.0.0.1 moved away from http://jobs.monster.com/v-business-q-data-analyst-jobs.aspx.
  At 2011-11-30 14:11:28 a user opened http://a836.g.akamai.net/7/836/35746/v0001/manpower.download.akamai.com/35746/jobboards/experis/experis_jbt_banner_v1.html. The user IP was 127.0.0.1.

The user came from http://jobview.monster.com/Business-Analyst-Data-Analyst-Job-Columbus-OH-US-104148407.aspx.

At 2011-11-30 14:11:28 a user opened http://jobview.monster.com/Business-Analyst-Data-Analyst-Job-Columbus-OH-US-104148407.aspx. The user IP was 127.0.0.1. The user came from http://jobs.monster.com/v-business-q-data-analyst-jobs.aspx.

- Log output for "geeks":

```
['2011-11-30 13:42:13', '127.0.0.1', 'pageload', 'http://www.find-me-a-gift.co.uk/christmas-gifts/', 'http://www.google.com/url?sa=t&rct=j&q=christmas%20present&source=web&cd=7&ved=0CE8QFjAG&url=http%3A%2F%2Fwww.find-me-a-gift.co.uk%2Fchristmas-gifts%2F&ei=qDLWTszcFY6IhQe75oxP&usg=AFQjCNHtFu0X5axyZqEmUYWMCJx0dYrHpA&cad=rja']
['2011-11-30 13:43:00', '127.0.0.1', 'pageunload', 'http://www.find-me-a-gift.co.uk/christmas-gifts/', 'http://www.google.com/url?sa=t&rct=j&q=christmas%20present&source=web&cd=7&ved=0CE8QFjAG&url=http%3A%2F%2Fwww.find-me-a-gift.co.uk%2Fchristmas-gifts%2F&ei=qDLWTszcFY6IhQe75oxP&usg=AFQjCNHtFu0X5axyZqEmUYWMCJx0dYrHpA&cad=rja']
['2011-11-30 13:43:02', '127.0.0.1', 'pageload', 'http://www.find-me-a-gift.co.uk/modules/thawte_logo.asp', 'http://www.find-me-a-gift.co.uk/christmas-gifts/christmas-gifts-for-her.html']
['2011-11-30 13:43:03', '127.0.0.1', 'pageload', 'http://www.find-me-a-gift.co.uk/christmas-gifts/christmas-gifts-for-her.html', 'http://www.find-me-a-gift.co.uk/christmas-gifts/']
['2011-11-30 13:43:10', '127.0.0.1', 'pageunload', 'http://www.find-me-a-gift.co.uk/christmas-gifts/christmas-gifts-for-her.html', 'http://www.find-me-a-gift.co.uk/christmas-gifts/']
['2011-11-30 13:43:10', '127.0.0.1', 'pageunload', 'http://www.find-me-a-gift.co.uk/modules/thawte_logo.asp', 'http://www.find-me-a-gift.co.uk/christmas-gifts/christmas-gifts-for-her.html']
['2011-11-30 13:43:11', '127.0.0.1', 'pageload', 'http://www.find-me-a-gift.co.uk/modules/thawte_logo.asp', 'http://www.find-me-a-gift.co.uk/bath-underwater-light-show.html']
['2011-11-30 13:43:12', '127.0.0.1', 'pageload', 'http://platform.twitter.com/widgets/tweet_button.html']
```

The logger itself is written in Python language and is fairly straightforward. In essence, the logger is listening for incoming messages at a designated network interface and port. Then it checks whether the message contains the query string parameter named "data". If it does, certain information is extracted and passed on to one of the log creators in order to produce a nice formatted search log.

```
try:
    class Logger(SimpleHTTPServer.SimpleHTTPRequestHandler):
        def do_GET(self):
            # field separator is |+0+|
            # time |+0+| client ip |+0+| action |+0+| url |+0+| referrer
            if '?data=' in self.path:
                client_data = re.sub(r'.*/\?data=', '', self.path)
                log(time.strftime("%Y-%m-%d %H:%M:%S", \
                    time.gmtime(time.time())) + \
                    "|+0+|" + str(self.client_address[0]) + \
                    "|+0+|" + client_data)
            # wrong request
            else:
                print 'We should not be getting this:', self.path
    httpd = SocketServer.ForkingTCPServer(('192.168.56.101', PORT), Logger)
    print "Logger serving at port", PORT
```

```
    httpd.serve_forever()
except KeyboardInterrupt:
    print '\n^C received, shutting down logging server'
    httpd.socket.close()
    print 'Bye!'
```

2.5.3 XML-RPC data transfer

Another key aspect of my solution is the data transfer from the virtual machine to the remote search task repository. This is achieved by incorporating a third-party library from Incutio Limited. The Incutio XML-RPC library (IXR) incorporates both client and server classes, and is designed to hide as much of the workings of XML-RPC from the user as possible [24]. A key feature of the library is automatic type conversion from PHP types to XML-RPC types and vice versa [24]. To clarify things further, XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism. Remote procedure calls give developers a mechanism for defining interfaces that can be called over a network [34, pp. 2-3].

The following code effectively describes how to send search stories to the repository. To get an IXR_Client instance, which is needed before any other actions can be executed, run the function initXMLRPC(). Function sendPost($client, $user, $pass, $title, $data, $categories, $tags) will perform the actual data transfer to the remote server. The server part of XML-PRC is already implemented by WordPress and there was no need for me to change that.

```
<?php
        // include the library
        include("IXR_Library.php");

        // initialize the client
        function initXMLRPC() {
                $xmlrpc = 'http://searchtaskrepo.wordpress.com/xmlrpc.php';
                $client = new IXR_Client($xmlrpc);
                return $client;
        }

        // publish a new post
        function sendPost($client, $user, $pass, $title, $data, $categories, $tags) {
                // define data array
                $data = array(
                        'title' => $title,
                        'description' => $data,
                        'mt_allow_comments' => 0,
                        'mt_allow_pings' => 0,
                        'post_type' => 'post',
                        'mt_keywords' => $tags,
                        'categories' => $categories
                        );
                // send it
                if (!$client->query('metaWeblog.newPost', '', $user, $pass, $data,
                    true)) {
                        die( 'Error while creating a new post ' . $client->
                            getErrorCode() . " : " . $client->getErrorMessage());
                }
                $id = $client->getResponse();
                // return the post ID
                if ($id) {
                        return $id;
                }
                return null;
        }
?>
```

2.5.4 WordPress search task repository

My initial goal was to find a development platform for the repository which would let me see some quick results and concentrate on developing the core parts of the solution instead of having to program user account management, search task post management, key security and privacy issues on the server-side, remote procedure call interfaces, basic search functionality. WordPress as a well-known and widely used dynamic content management system (CMS) seemed like a logical choice for my needs.

A key factor in making this choice lies in the fact that WordPress is infinitely extensible [60] and many plug-ins and themes are already made for it. In the context of this project, this will mean that there is a good chance of replacing the current search engine by a more customized one, hand-tailored for processing people's search stories (see 4.3). At the time of writing this thesis, the repository is still fairly basic. In fact, not a single line of code was written by the author to get it to its current state (see Figure 2.4). The current hosting choice (WordPress.com) may need to be changed

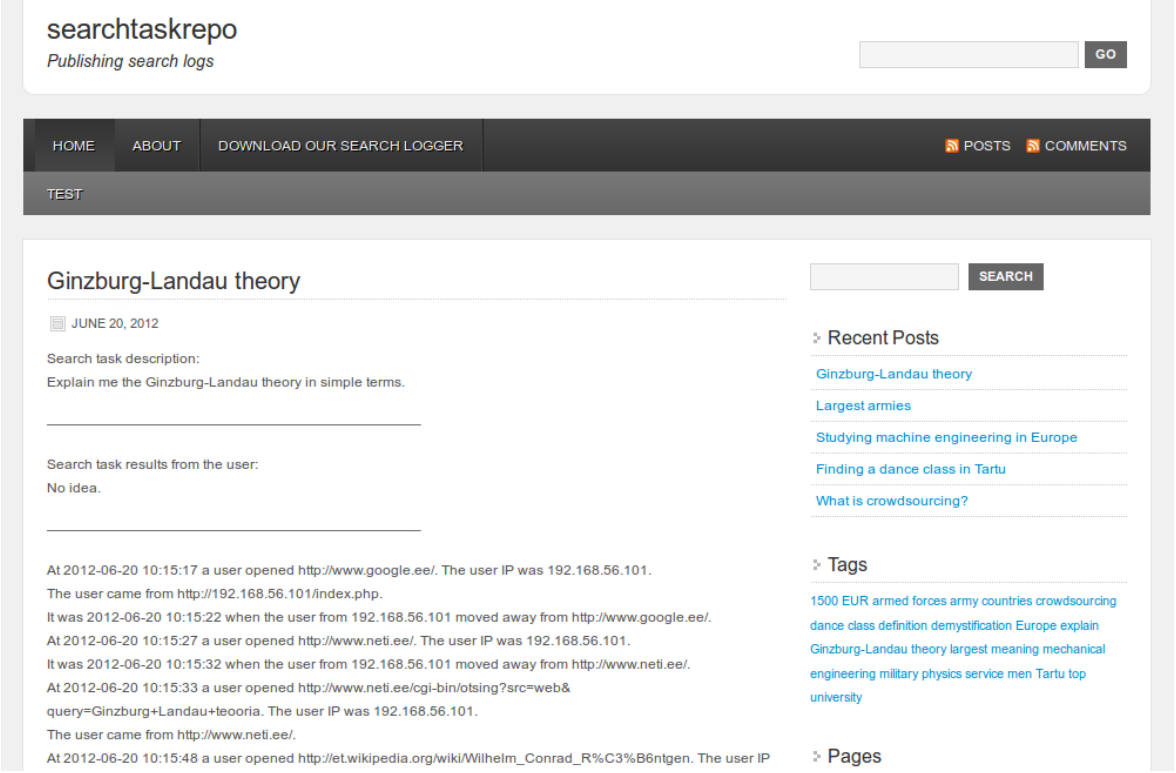


Figure 2.4: Search stories on the search task repository

in the future as the project evolves, since it is debatable whether the repository fully meets their Terms of Service.

2.6 Evolution of the solution

It was in June 2010 that I decided to undertake the challenge of engineering and developing this search support solution. The actual work started in autumn that year. Throughout this period the solution as well as my exact working tasks changed and evolved many times. Here I will shed some light on the history of the project.

The actual development process can be roughly divided into four evolution phases:

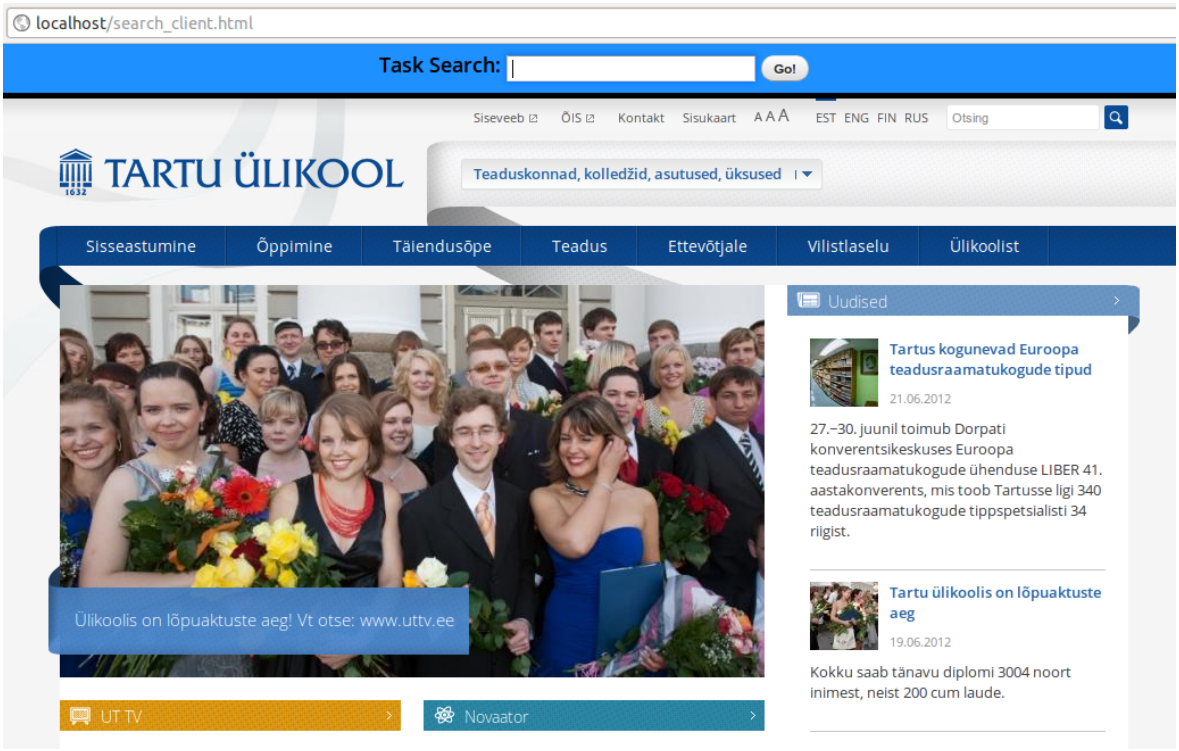


Figure 2.5: The SearchBar

Phase I: developing a reporting tool on top of the original Search Logger Our research group had a functioning Search Logger tool [45] at our disposal. However, it had to be integrated with a new component which would help us effectively conduct large user studies about helping users with their complex exploratory search tasks. As described in Section 2.4, this tool had its problems which needed to be solved first. For example, every time a new version of Mozilla Firefox came out, the logging solution as a plug-in was likely to be broken. What is more, I observed a different behaviour under Linux and Windows operating systems. I soon abandoned the plug-in based search task logging solution to start over from scratch.

Phase II: AJAX and PHP SearchBar I started my search task logging endeavour by something we later never needed nor used: having a bar on top of the browser window for all Internet searches, the requested pages appearing below (see Figure 2.5). In short, the page loading behaviour was implemented mainly in AJAX (Asynchronous JavaScript and XML), whereas user activities were logged using PHP. The single main disadvantage of this approach is forcing users to conduct their searches through my search bar which would not constitute a normal search behaviour by any means. The lessons learned from this phase led me to the proxy-based search task logging framework.

Phase III: building my own Internet proxy server Initially, I did not think that search task reporting at this scale would be possible with a purely proxy-based solution, but the possibilities of JavaScript enable to capture relevant user behaviour. The proxy server rewrites the HTML pages requested and amends them with special JavaScript to capture similar events as the ones that were captured directly from the Firefox plug-in. At first, I implemented a simple but working proxy server capable

of handling HTTP GET requests (but not much more) in Python which was able to do just that. After some internal testing this approach (but not the idea of using a proxy) was also dropped because it was vital to have a very stable yet versatile proxy server for my search task logging solution. The latter would have required too much developing and testing effort, and luckily there were better existing Internet proxies already available.

Phase IV: Privoxy-based search task logging, editing and publishing Then I did some research about existing Internet proxy servers. I discovered a variety of existing Web proxies, in particular Apache's mod_proxy, Squid and Privoxy. Each one of them has their own strengths in different real world scenarios. Ultimately, the selection was narrowed down to one – Privoxy. It checked all the boxes for me for the following reasons:

- frequently used ("in combination with Tor or the Vidalia project or Squid and can be used to bypass Internet censorship" [58]) and known for its reliability;
- advanced existing filtering capabilities which can also be defined by user scripts;
- cross-platform;
- Free Software and licensed under the GNU GPL version 2 [41];
- a lot more complex than my own proxy – supports HTTP and HTTPS traffic.

As a result, I carried on with Privoxy as the proxy server and this approach serves as a basis for the current architecture of my solution. More information about how exactly pages are re-written can be found in Subsection 2.5.1.

2.7 Evaluation

The created solution allows research groups to carry out evaluation experiments for exploratory search tasks and extend the evaluation scope from the query level to the task level. Largely due to the fact that developing these tools took considerably longer than initially hoped and expected, my proxy-based search task logger and the search task repository have yet not been used in planning and conducting such experiments. However, all parts of my solution are available for public testing and academical use since mid-June 2012 from the search task repositories Web page (http://searchtaskrepo.wordpress.com/). Our research group has already carried out and published user studies using the Search-Logger plug-in version [45, 46] that has many shared characteristics with this proposed system but is created not by myself.

A practical usability study was carried out in fall 2011 to test one of the key subcomponents presented in my thesis – the search task logger. To do that, I conducted a number of exploratory search tasks and used my software to log the Web activity. In essence, I sought an answer to the question "How to do a search report?". The other goal was to find, document and fix bugs in the existing logging and reporting software and come up with ideas about new features. Many of these insufficiencies discovered by this self-review have been fixed or redesigned since then.

The existing solution was tested on six exploratory search tasks such as finding a foreign university on a budget where to study machine engineering, finding 5 clever

Christmas present ideas for under 10 Euros per item, finding a ballroom dance school in Tartu area. These tasks and how they worked out with my proposed software can be found on my blog in more detail (http://searchtaskrepo.blogspot.com/). In addition, I made then a short screencast for presentation purposes which is also linked to.

There is currently interest from researchers at University of Tartu as well as one foreign university in Hamburg, Germany to use my presented solution for large user studies about Internet search behaviour. In order to do that, some tweaks to the search task logging and publishing solution might be desirable. They are looked at in Section 4.2. What is more, searchers from the general public can use my current solution for their real world inspired problems due to the freedom to publish user-defined search stories, and all this can bring forward new unforeseen opportunities for researchers.

2.8 Roundup

In this chapter, I explained the core features of my proposed search task logging, editing and publishing solution both from a technical and a human point of view. Additionally, I compared my proxy-based search logging method with the existing browser plug-in based approach. A brief overview was given about how my search support tools have been tested and how they can be used in upcoming academic user studies.

Chapter 3

User manual

This chapter explains how to set up and use my search monitoring and supporting systems. These instructions are purposefully explained in a detailed and clear way, coupled with a number of illustrations to make the set-up process easy for new users. In order to organize large scale classroom experiments, it would be feasible to set up and run just one proxy server for all traffic in the Local Area Network (LAN). This would necessitate some tweaks to my search support solution that are discussed in Future work (see Section 4.2).

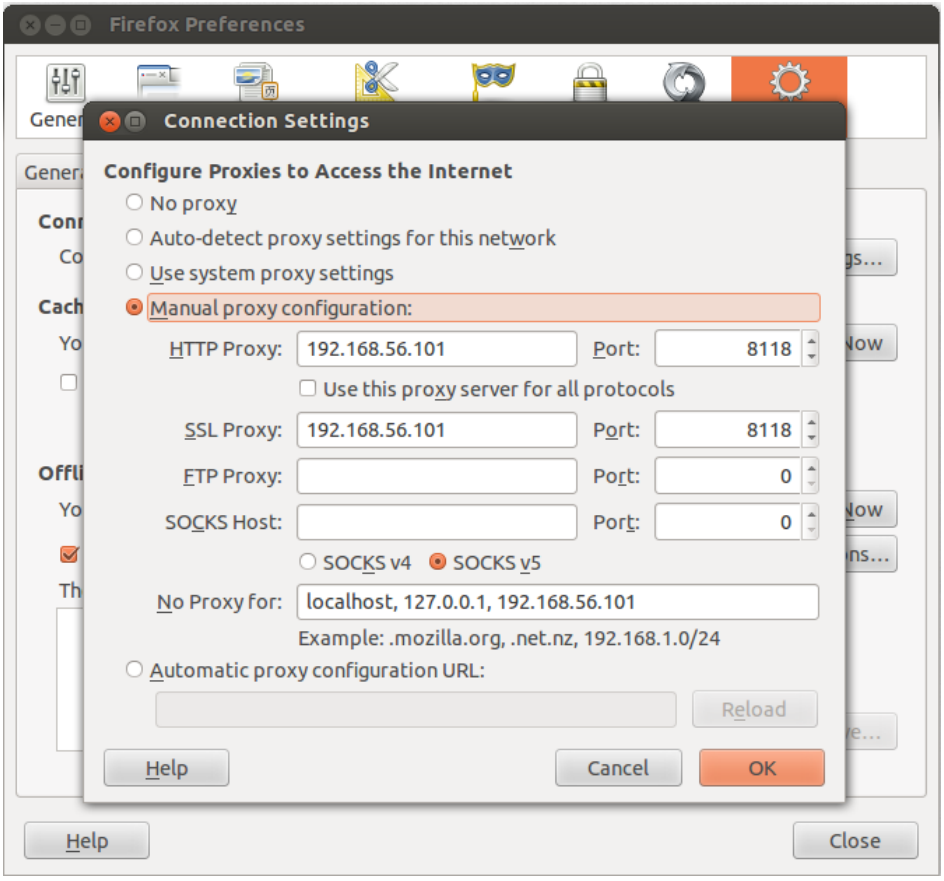


Figure 3.1: Correct browser configuration

3.1 Set-up tutorial

For using the repository of search tasks, just go to http://searchtaskrepo.wordpress.com/ and satisfy your information need. To make contributions to my repository, you will need to follow these simple steps:

1. Make sure you have Oracle VM VirtualBox installed. "Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7), DOS/Windows 3.x, Linux (2.4 and 2.6), Solaris and OpenSolaris, OS/2, and OpenBSD" [40].

2. Download the 7-Zip archive from http://searchtaskrepo.wordpress.com/download-our-search-logger/.

3. Unpack this archive to the virtual machines folder of your VirtualBox installation.

4. Before running any search tasks, you will want to configure your browser(s) to use Privoxy as an HTTP and HTTPS (SSL) proxy. Correct settings are directing HTTP and HTTPS traffic to 192.168.56.101:8118 as shown in Figure 3.1. Additionally you may like to add 192.168.56.101 to the "no proxy" list if you do not want see interactions between the search task logging portal and your local computer being logged as well. This is the one configuration step that must be done!

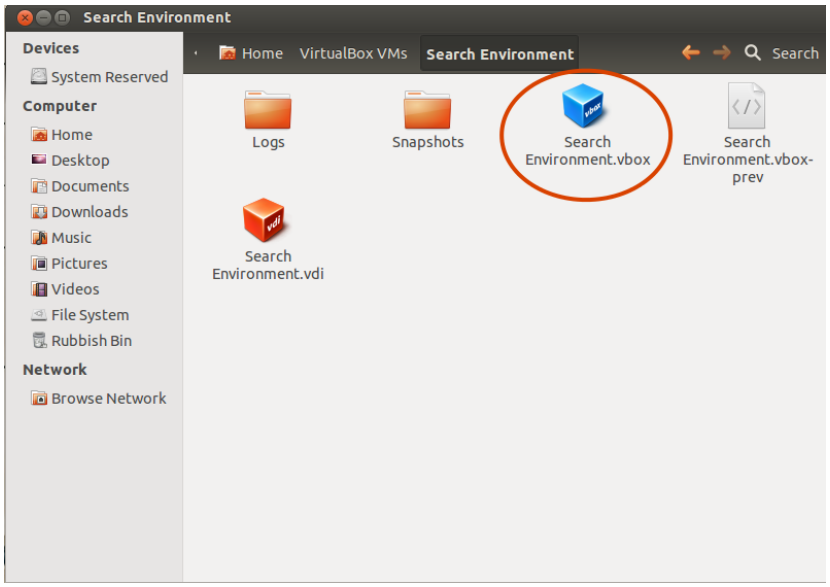


Figure 3.2: Running the search task logging software

5. Upon starting VirtualBox a new virtual machine called "Search Environment" shall be visible on the list of all VMs. Having selected the correct virtual machine, click on the green "Start" button or just click on the Search Environment.vbox file in the previously unpacked directory (see Figure 3.2). Wait until the the login prompt appears and you are ready to go! NB! You do not have to log in but if you want to, both the username and password are "peeter". Please be warned that making changes there may lead to a non-operable virtual system state.

3.2 Recording and publishing a search task

First, it would make sense to verify that you have the search task reporting solution set up as described previously. This tutorial will walk to though the process of compiling and publishing a search task.

1. Type "192.168.56.10" into the address bar of your browser. This will bring up the search task logging Web interface (see Figure 3.3). You can control all aspects of the solution via this interface, including starting and stopping the logger, and editing your search story to be published.

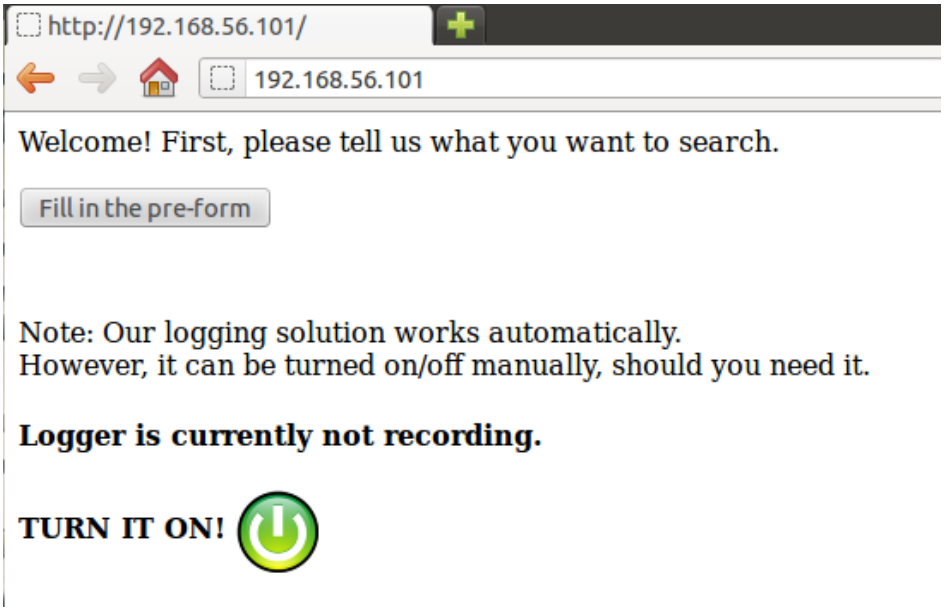


Figure 3.3: The Web interface of my search task logging software

2. Fill in the pre-form to define your upcoming search activity (see Figure 3.4). This will help you as well because you will then have a better understanding of what exactly you are looking for.

3. Start searching as you would normally do (Figure 3.5). While searching, use the "resultspad" to keep track of your answers.

4. Fill in the post-form to give us feedback about how your search session went. This feedback is important for offering better value for searches made at the search task repository.

5. Now you can either publish your search task straight away or view, edit and annotate your search logs as shown in Figures 3.6a and 3.6b.

6. Once you have published your search report, close the browser. If you have no further search tasks to complete, either send a shut-down signal to the VirtualBox image or save its state. Now you will also have to revert your browser proxy settings back to normal to be able to browse the Web.

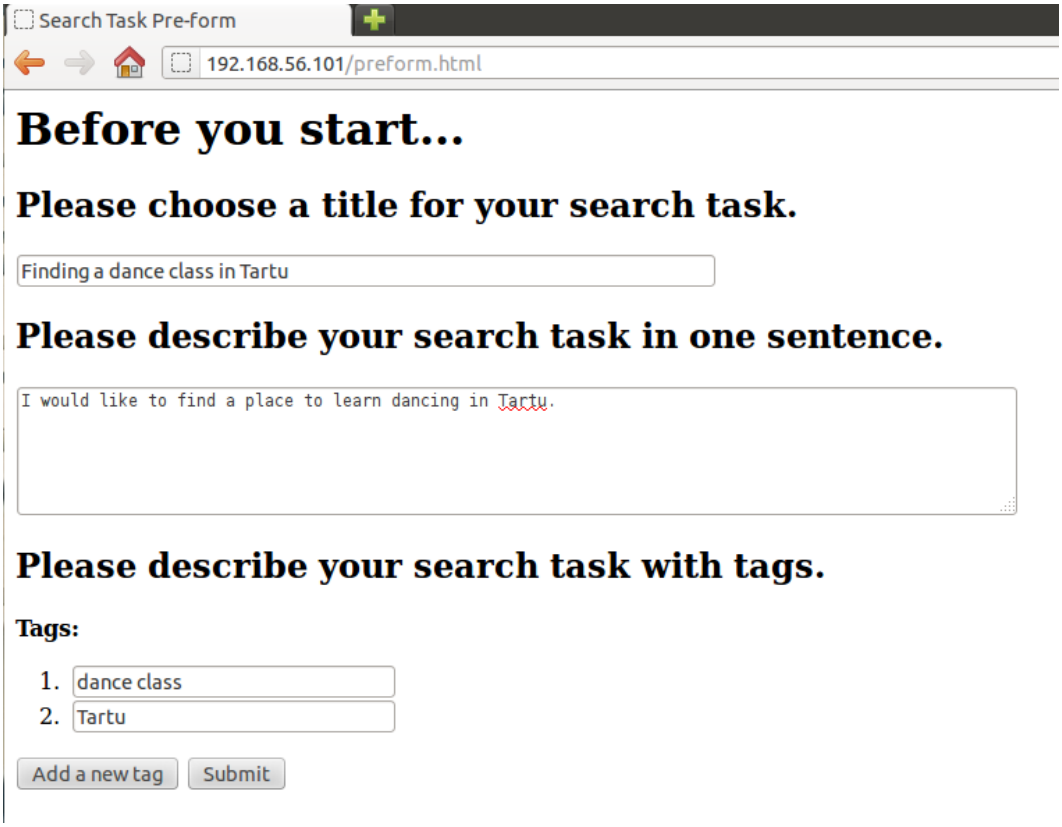


Figure 3.4: Filling in the pre-form

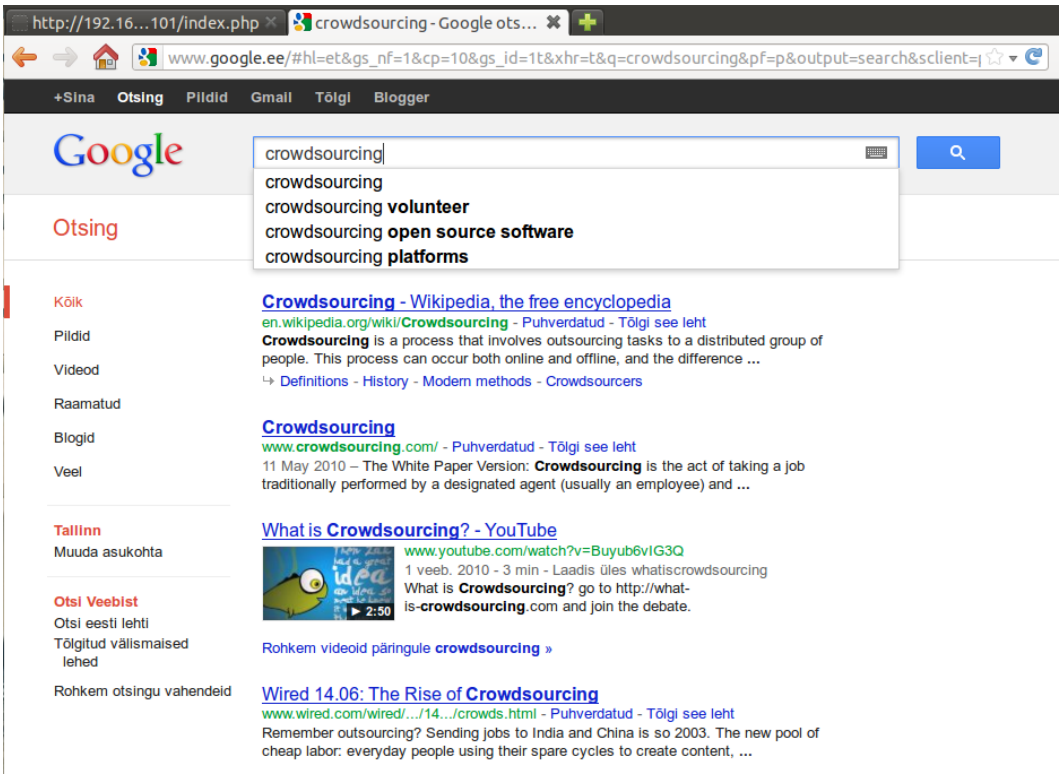
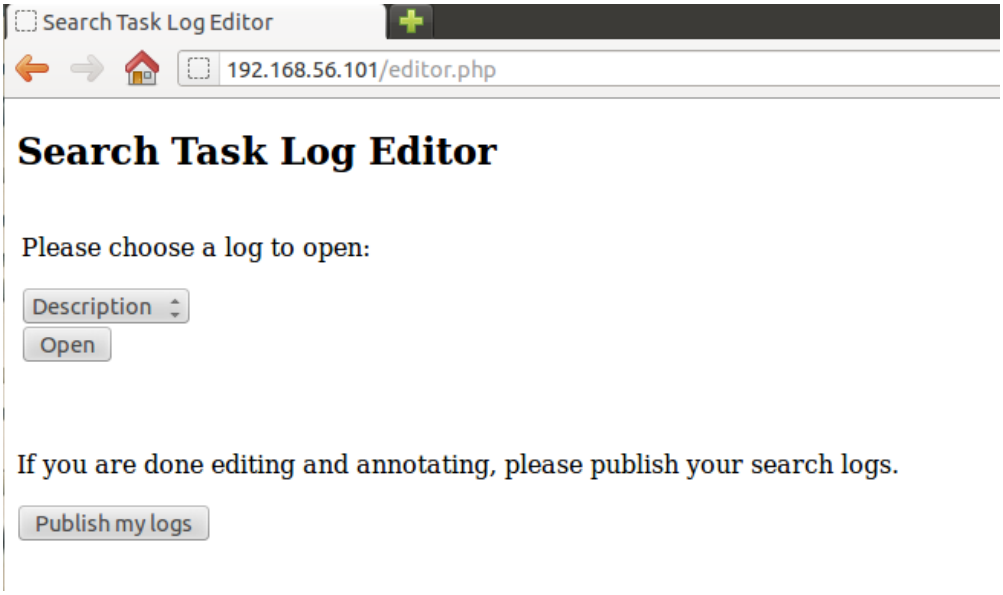
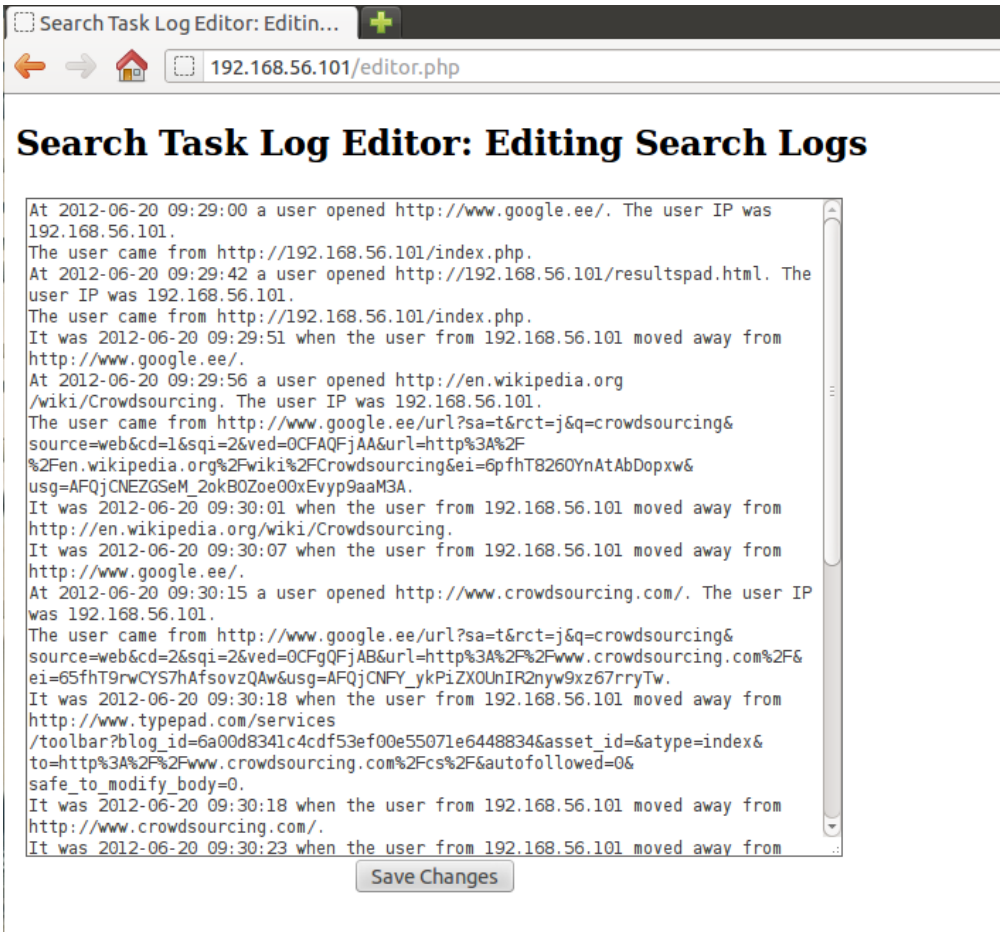


Figure 3.5: In the majority of cases the proxy-based search logging does not change the usual search experience.



(a) Search task log editor gives you full control over what you publish.



(b) Machine-generated logs can be annotated manually.

Figure 3.6: Search Task Log Editor

Chapter 4

Future work

In this chapter, I take a look at the future: how can search logs make solving exploratory search tasks easier, and what are the features I would like to add to my search logger and search task repository. I also envision the broader benefits of my search support system to our society.

4.1 Vision for search log aided search

To my mind, this solution for supporting complex search tasks has the potential of being the joint platform for bringing together people from different fields of expertise in the society. While it would still primarily be the means for the academic community to conduct information retrieval experiments, and especially those studying more complex and less pre-defined information needs, I do not believe that search logs have no other ground. By giving people the freedom of freely defining and expressing their own search tasks, we could be gaining more trust from other target groups as well, be it a project manager in a large enterprise conducting market research or a housewife looking for a new hobby. As an indirect outcome of letting searchers truly express themselves and their information needs, the academic community could have a working platform which contains large amounts of representative search data to study further. I would call this enabling symbiosis between researchers in IR and the society.

Let us take a look at this practical scenario. Marcus has a complex search problem which could take hours or even days to complete. Instead of sacrificing that much time, he first decides to visit the search task repository to see if people have been there before. Given the popularity of this Web site he gets lots of relevant search stories with answers from the community as well as well-annotated search logs, helping Marcus to clearly see how the proposed answers were synthesized. This gives him new ideas for areas he wants to explore himself in more detail. Knowing more in advance about the dimensionality of the search space helps him save valuable time. Now, Marcus enables logging of his search behaviour and spends some time exploring the unknown aspects of this problem. During all that time he does not have to rely on his memory and can note down his personal answers while he is searching. At the end, Marcus chooses to publish his search log for his own future reference as well as for the searching community. Not only has he benefited from my solution, so did researchers who can use this data for Internet search behaviour analysis, and fellow explorers who may save time from the time-consuming activities of aggregation, discovery and synthesis. Even businesses may

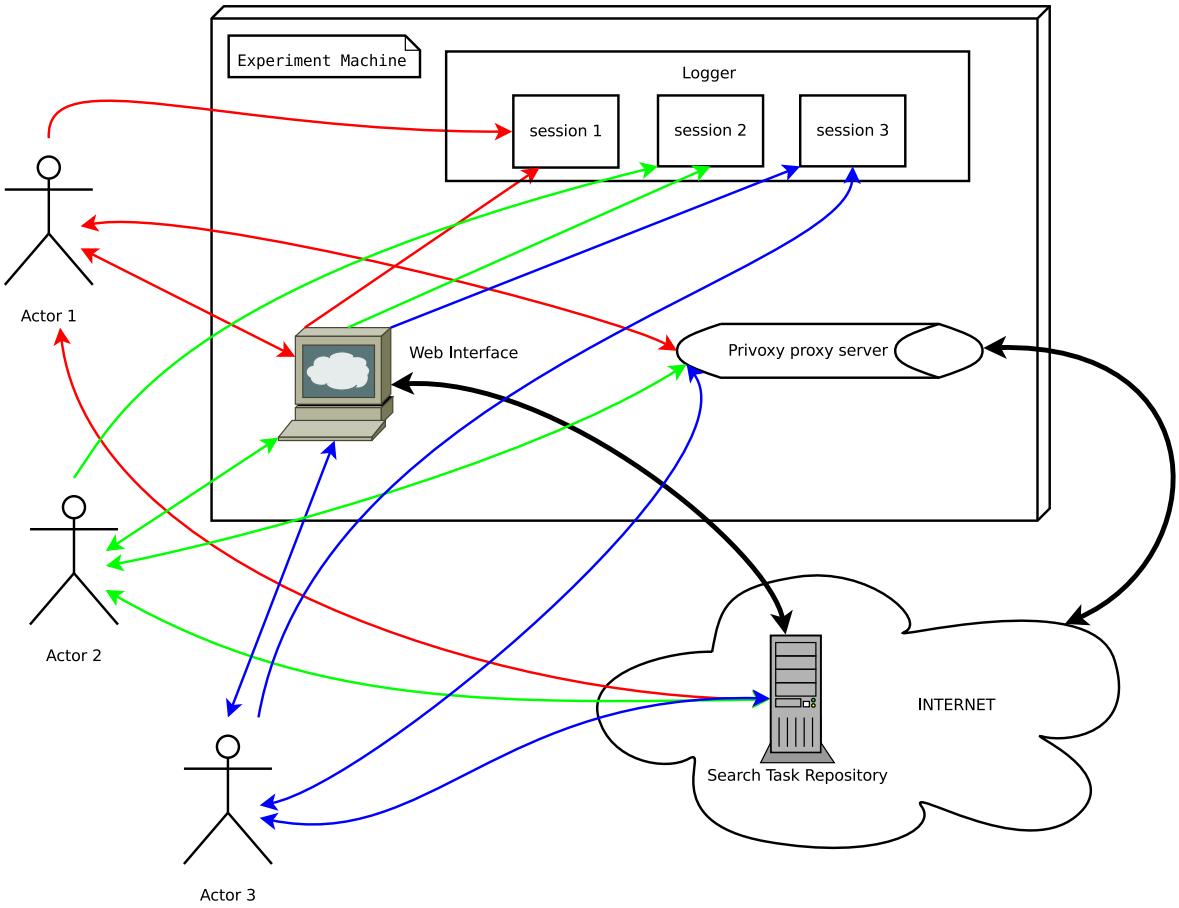


Figure 4.1: The modified architecture for classroom experiments

benefit if they find a way to use this publicly available resource about their potential customers to their advantage.

4.2 Improving proxy-based search logging

One of the suggested future additions to the search task logging framework would be adapting it for classroom experiments. This is already possible by the currently proposed set-up but the idea is not having to install VirtualBox virtual machines on every computer. The following modified architecture (Figure 4.1) achieves just that by having one central experiment machine which then serves computers in its Local Area Network (LAN). I removed interaction labels to get a less congested picture, the labels from Figure 2.1 can be used for reference since they are essentially the same. Shared connections are between the Web interface and the remote repository, and the outgoing connection from Privoxy.

Having studied the inner workings of Privoxy configurations, it is easy to configure different Web page re-writing rules for individual computers or subnets in an LAN. With some additional programming, this can be even done dynamically. However, this problem does not need even that. Probably the simplest solution to implement would allocate each participant a unique sub-directory in the experiment machine in which both the search logger and the Web interface sub-components can save their logs. Although totally enough for a lab experiment, this approach is subject to a simple

malicious attack condition from someone in the LAN falsifying their IP, since these folders would be created and accessed according to each client's sent information about their own IP. A more sound way would also create unique directories for all participants but would additionally let each participant enter their WordPress credentials for the search task repository at the beginning of the experiment. This data could be stored locally during the experiment as a cookie, and only sent out in an encrypted form over the LAN to uniquely identify the participant.

Another previously mentioned future need is adding support for a greater number of programmatically loggable events. Some researchers [17] have already practically shown that JavaScript injection enables to logs much more than my search logging solution is currently capable of. For instance, they also logged mouse movement and scrolling as well as pages clicked. I would add detecting currently active browser tabs to this list by logging window.onfocus and window.onblur events. However, this method may not always be reliable [31]. As a means for site developers to programmatically determine the current visibility state of the page in order to develop power and processor efficient web applications, World Wide Web Consortium (W3C) along with leading Internet browser developers are currently working on the Page Visibility standard [61]. Logging these events might be a more reliable alternative to the previously discussed window. onfocus and window.onblur events.

There are a few known recurring bugs and annoying usability issues at the time of writing this thesis which have been unsolved:

1. `Error while creating a new post -32700 : parse error. not well formed`

   This error may occur when the user is trying to publish the search log to the WordPress repository of search tasks. I did some research about this problem: it seems to be a common issue with the Incutio XML-RPC library version 1.7.4 which I am using, and is allegedly related to an ill-formed remote procedure call response message. However, the solution is not known to me. If this error occurs, users are encouraged to press the "Back" button on the browser and try to publish again. I have also observed that this error is more likely to occur if the search story contains non-ASCII letters which in turn would imply an encoding problem. Nothing is confirmed yet.

2. ```
   File "./logger.py", line 48, in do_GET
   "|+0+|" + str(self.client_address[0]) + "|+0+|" +
      client_data)
   File "./logger.py", line 35, in log write_human(g, fields)
   File "./logger.py", line 18, in write_human
   if information[4]: IndexError: list index out of range
   ```

   This is a known exception from the search task logging module but it remains concealed from the end user because the user does not run this component and will not see the error messages from the Python interpreter. As it seemed not to be affecting the overall performance of my solution over a long testing cycle, I decided not to fix it in the presence of more urgent development matters.

3. Regrettably, I was unable to ensure that the logging solution would always exactly replicate the same search experience as if there were no proxy present. Most
   ```

Figure 4.2: Some Web sites do not display correctly when logging is enabled in Privoxy.

tested sites functioned impeccably with the proxy and logging enabled, however, there are exceptions (see Figure 4.2). Disfigured Web sites often make extensive use of JavaScript scripts which in some very rare occasions may not function after my own JavaScript injections to make proxy-based search task logging possible. This remains on top of the efforts to make proxy-based logging less intrusive but it has to be said that different Web sites may require different measures.

4. Another slight inconvenience is that some "noise" gets logged as well. In the current development phase, all JavaScript events, which are received, are also logged by my solution. For instance, it is quite common to see loading and unloading records of Facebook plug-ins published in search stories unless the user edits this information out manually. Privoxy provides sophisticated methods for filtering and blocking ads, pop-ups, and other kinds of generally unwanted components seen on many of the current Web sites. In favour of providing maximally authentic search experience, I decided to disable these measures. However, these can be easily re-enabled from Privoxy configuration files by interested users. A more promising option is to design and implement some sort of more intelligent customized filtering in my logger. This will remain a task for the future.

4.3 Improving search task repository

My repository for search tasks is currently fairly basic. The most important features to come should in my opinion be related to the way searches are performed in it. My solution can already collect quite a lot of structured data about each search task: search

task title, user description, tags and categories associated with the search task, results and feedback from the user, machine-generated search logs.

At present I use standard WordPress search capability to produce results for search queries. However, it is possible to invent a better way for indexing this textual information to reflect the peculiarities of the data collected and published. On top of that, we would need a new algorithm for ranking search stories to yield more relevant results. This would deem some properties of a search task log more important than the others. For example, if the author of the search story admitted that she did not find what she was looking for, then her published search logs should probably appear lower at the results page. There are very many factors to ranking search logs. For example, the up-to-dateness of a search task log has a role to play as well, illustrating the dynamic nature of ranking search stories and constructing answer sets.

Once the repository has a larger user base and more search tasks published, one very promising option is evaluation based on the analysis of clickthrough data, which can be obtained by observing how frequently the users click on a given search task log when it is shown in the answer set for a given query. There are existing algorithms for collecting unbiased clickthrough data so that there exists a connection between user clickthrough data and relevance of the proposed answers to the users [6, pp. 172-173].

Alternatively, a crowdsourcing approach could be used to cross-validate the search task logs by users themselves. One problem with this as far as Amazon Mechanical Turk is concerned, is the fact that "turkers" are not very keen on downloading and installing software [17]. One theoretical scenario which would work without them having to do more than read and think, is proposed in subsection 1.5.4. As discussed in subsection 1.5.2, crowdsourcing for relevance evaluation can produce quick results at a low cost with good flexibility and high quality [2].

Another promising idea for making the search task repository a more useful resource for fulfilling an information need would be running different clustering techniques and algorithms on user-entered tag data to automatically group posts based on their topic. At present, all posts share the "test" category. One could ideally develop a successful recommender system on top of this extracted information of connections to suggest potentially relevant or at least interesting search stories on a similar area.

Having WordPress as a target platform provides a great opportunity to make full use of users' profile information. This opportunity is currently not used at all. The repository has just two accounts: one for the administrator and one for all authors of the search tasks. This way I am losing interesting behavioural and social data about the actions of different searchers and an opportunity to cluster users based on their interests, topics of their contributions to the repository, physical location. Using this opportunity would help us tackle the challenges related to a user's location. For instance, it is quite obvious that when someone is looking for gyms, they are unlikely to be interested in search stories from people who are thousands of kilometres away. Harvesting profile (or even social profile) information can help us customize the search to individual needs.

Finally, giving so much freedom to the search task publishers can backfire, leading to a repository mixed with purposeful search reports and unwanted content such as advertisements, spam and other types of indiscreet material. To avoid that, it may be necessary to invite dedicated volunteers for moderation tasks similarly how existing collaboratively edited Web sites (e.g. Wikipedia) tackle this problem. Due to the structured nature of search logs, simple but automatic content checking scripts could

be created and enforced to flag potentially unrelated submissions. Alternatively, it would be possible to enable user voting which could serve two purposes: helping to identify useful search task stories to rank higher, and spotting unwanted content to remove from the repository.

4.4 Roundup

In this chapter, I looked into the future: how can my search task logging solution along with the repository cope with a greater number of users? What features is it currently missing and what are the known bugs? Here I presented my own vision concerning the role annotated search task reports can play in the search market of tomorrow.

Conclusion

The main research problem of my thesis was engineering a new type of search task logging and publishing framework which would provide a better alternative for existing browser plug-in based methods. Right from the start, the proxy-based search task reporting system has been a complex engineering challenge involving code written in multiple programming languages, interactions planned across many software modules (some of which have already been existing large projects themselves), and a Linux operating system configured to ease the set-up process for the user. This was the decision process to make sure that this solution is reliable, extendible and maintainable in the future. My research goal was completed successfully.

In my thesis, I proposed a proxy-based method for logging user search behaviour across different browsers and operating systems. I also compared it with an existing plug-in based Search Logger for Mozilla Firefox and other similar solutions. The idea of developing a proxy-based search task logging and publishing solution came from out of necessity, because the existing logging solution had significant problems with maintainability. The logs created by my solution are subsequently annotated by the user and made publicly available on a dedicated Internet blog called the Search Task Repository. Users can search against the already annotated and published Internet search logs. Ideally this would mean reduced complexity of search tasks for the users which in turn saves time. User studies to confirm this are still pending but there is confirmed interest from Tartu researchers as well as from one foreign university to use my solution in their search experiments.

The proposed solution is comprised of two large units, which are the search task repository and the search task logging and publishing unit. The search task repository is a remote component, essentially a fairly simple WordPress blog, which enables search stories to be published automatically over XML-RPC protocol, search queries to be served, and search task logs to be displayed to the searcher. My logging system is configured as a VirtualBox virtual machine. It is much more complex, consisting of three sub-components: the main Web interface, the search task logger, and the Privoxy Web proxy specially configured for my needs. Logging can be started and stopped at a user's will in the main Web interface. What is more, this sub-component also gives them absolute control over what gets published online by providing an editing and annotating functionality for all search task data, both implicitly and explicitly logged.

A comprehensive theoretical overview was given in my thesis about the state of the art, explaining basic related concepts in Information Retrieval and recent developments in Exploratory Search and search task logging systems. In contrast with existing browser plug-in based search task logging methods, my proposed proxy-based approach ensures platform and browser independence while also being very stable. By giving searcher's the opportunity to freely define and annotate their own search tasks, my search support solution is setting a new standard.

In the final chapter, I conducted a thorough analysis about future work and presented my own vision about the future opportunities for this search support methodology. A modified architecture for more convenient laboratory experiments was outlined as an important task for the future. In conclusion, my proxy-based search task logging, editing and publishing framework can be extended further to log more JavaScript events. The search task repository is a large open area with lots of opportunities for future extensions.

Internetiotsingu toetamine otsingulogide jagamise meetodil

Peeter Jürviste

Magistritöö (30 EAP)

Sisukokkuvõte

Antud väitekiri on osa jätkuvast kollektiivsest uurimistööst, laiema eesmärgiga eeskätt parandada Internetiotsingu tuge keeruliste ja aeganõudvate ning tihti uurimusliku loomuga otsinguülesannete kiiremaks ja efektiivsemaks läbiviimiseks. Töö peamine uurimisprobleem on uut tüüpi otsinguülesannete logimise ja Internetis jagamise raamistiku väljatöötamine, olles alternatiiviks brauseri pistikprogrammide põhistele olemasolevatele meetoditele. Tegu oli keerulise insenertehnilise ülesandega, mille käigus tuli autoril täita mitmesuguseid programmeerimise, planeerimise, süsteemi komponentide integreerimise ja konfigureerimisega seotud ülesandeid. Püstitatud eesmärk sai edukalt täidetud.

Väitekirjas pakuti välja proksipõhine meetod kasutajate otsingukäitumise logimiseks, mis on ühtlasi lihtsasti kohaldatav erinevatele veebilehitsejatele ning operatsioonisüsteemidele. Lahendust võrreldi varasemate sarnaste süsteemidega. Meetod sündis reaalsest vajadusest leida kergemalt hallatav ning porditav asendus varem väljatöötatud tarkvarale, mis kujutas endast pistikprogrammi Mozilla Firefox veebilehitsejale, kuid mida tuli parandada pärast iga uue brauseri versiooni väljatulekut.

Teostus koosneb kahest suuremast komponendist, millest esimene ja tehniliselt keerulisem, otsinguülesannete logide koostamise ja jagamise süsteem, paikneb VirtualBox'i virtuaalses masinas. Teine on WordPress'il põhinev otsingulogide repositoorium, võimaldades lisaks kasutaja poolt annoteeritud logide avaldamise ka neist lihtsamaid otsinguid teostada. Süsteeme on põhjalikult testitud, kuid neid pole veel rakendatud Internetiotsinguga seotud kasutajauurimustesse. Autorile on teada, et selline huvi on olemas nii Tartu Ülikooli sees kui ka ühe välismaise partnerülikooli poolt.

Lokaalselt paiknev otsinguülesannete koostamise ja jagamise süsteem koosneb kolmest võrdselt tähtsast alamkomponendist. Nendeks on Python'i keeles realiseeritud otsinguülesande logija; peamiselt PHP'd ja HTML'i kasutav veebiliides, mis muuhulgas võimaldab kasutajal eelpoolmainitud logijat sisse ja välja lülitada, aga ka kõiki otsinguülesandega seotud andmeid käsitsi muuta ja täiendada; ja antud ülesandeks spetsiaalselt konfigureeritud Privoxy veebiproksi server.

Töös antakse põhjalik ülevaade olemasolevast tarkvarast, teaduspublikatsioonidest ja teoreetilistest alustest seoses väitekirja uurimisprobleemiga. Võrreldes olemasole-

vate meetoditega eristub autori pakutud proksipõhine otsinguülesannete logimise ja jagamise raamistik peamiselt kahel põhjusel. Esiteks, meetod tagab platvormist ja brauserist sõltumatuse, olles ühtlasi väga stabiilne. Teiseks, kasutajatele antav vabadus oma otsinguülesannet vabalt defineerida ning annoteerida on oluliseks uueks tähiseks.

Väitekirja viimases peatükis käsitletakse tööga seotud tulevikuväljavaateid ja avatud probleeme. Üks neist on väljapakutavaga võrreldes muudetud arhitektuur, mis võimaldaks korraldada väiksema vaeva ja ajakuluga laboriekspérimente. Internetiotsingu logimise süsteemi saab edasi arendada, lisades tuge enamatele JavaScript'i sündmustele. Otsingulogide repositoorium, olles veel üsna algeline, pakub hulgaliselt võimalusi täiendusteks tulevikuks.

Bibliography

[1] O. Alonso and S. Mizzaro. Can we get rid of TREC assessors? using mechanical turk for relevance assessment. In *Proceedings of the SIGIR 2009 Workshop on the Future of IR Evaluation*, page 15–16, 2009.

[2] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, November 2008. Available from: `http://doi.acm.org/10.1145/1480506.1480508`, `doi:10.1145/1480506.1480508`.

[3] Amazon. *Amazon Mechanical Turk*. June 2012. Available from: `https://www.mturk.com/`.

[4] V. Ambati, S. Vogel, and J. Carbonell. Active learning and crowd-sourcing for machine translation. *Language Resources and Evaluation (LREC)*, 2010.

[5] A. Aula and D. M. Russell. Complex and exploratory web search. In *Information Seeking Support Systems Workshop (ISSS 2008), Chapel Hill, NC, USA*, 2008.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval: the concepts and technology behind search, Harlow*. Addison-Wesley, Pearson, 2nd edition, 2011.

[7] C. Beard. *CoScripter*. March 2012. Available from: `http://mozillalabs.com/blog/2007/09/coscripter/`.

[8] J. Bian, Y. Liu, E. Agichtein, and H. Zha. Finding the right facts in the crowd: factoid question answering over social media. In *Proceeding of the 17th international conference on World Wide Web*, page 467–476, 2008.

[9] D. C. Brabham. Moving the crowd at iStockphoto: the composition of the crowd and motivations for participation in a crowdsourcing application. *First Monday*, 13(6):1–22, 2008.

[10] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, September 2002. Available from: `http://doi.acm.org/10.1145/792550.792552`, `doi:10.1145/792550.792552`.

[11] R. Capra. HCI browser: A tool for studying web search behavior. *Proceedings of the American Society for Information Science and Technology*, 47(1):1–2, 2010. Available from: `http://onlinelibrary.wiley.com/doi/10.1002/meet.14504701444/abstract`, `doi:10.1002/meet.14504701444`.

[12] M. Claypool, P. Le, M. Wased, and D. Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, page 33–40, New York, NY, USA, 2001. ACM. Available from: http://doi.acm.org/10.1145/359784.359836, doi:10.1145/359784.359836.

[13] Dedoimedo.com. Network & sharing in VirtualBox - full tutorial. http://www.dedoimedo.com/computers/virtualbox-network-sharing.html, June 2012. Available from: http://www.dedoimedo.com/computers/virtualbox-network-sharing.html.

[14] S. Dhansay. *Crowdsourcing Platform*. November 2010. Available from: http://www.opentechnologist.com/2010/05/03/crowdsourcing-platform/.

[15] S. Dhansay. *People Participation in Crowdsourcing Platforms*. November 2010. Available from: http://www.opentechnologist.com/2009/05/15/people-participation-in-crowdsourcing-platforms/.

[16] B. M. Evans and E. H. Chi. Towards a model of understanding social search. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, page 485–494, New York, NY, USA, 2008. ACM. Available from: http://doi.acm.org/10.1145/1460563.1460641, doi:10.1145/1460563.1460641.

[17] H. Feild, R. Jones, R. C. Miller, R. Nayak, E. F. Churchill, and E. Velipasaoglu. Logging the search self-efficacy of amazon mechanical turkers. In *Proceedings of the ACM SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation (CSE 2010)*, page 27–30, 2010. Available from: http://ciir.cs.umass.edu/~hfeild/publications/feild2010lss.pdf.

[18] H. A. Feild, J. Allan, and J. Glatt. CrowdLogging: distributed, private, and anonymous search logging. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, page 375–384, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/2009916.2009969, doi:10.1145/2009916.2009969.

[19] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, April 2005. Available from: http://doi.acm.org/10.1145/1059981.1059982, doi:10.1145/1059981.1059982.

[20] O. Gassmann, M. Daiber, and L. Muhdi. Der crowdsourcing prozess. *Gassmann (Edt.): Crowdsourcing-Innovationsmanagement mit Schwarmintelligenz. Carl Hanser Verlag München*, page 31–55, 2010.

[21] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in WWW search. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, page 478–479, New York, NY, USA, 2004. ACM. Available from: http://doi.acm.org/10.1145/1008992.1009079, doi:10.1145/1008992.1009079.

[22] H. Halpin, D. M. Herzig, P. Mika, R. Blanco, J. Pound, H. S. Thompson, and D. T. Tran. Evaluating ad-hoc object retrieval. *Proceedings of IWEST*, 2010.

[23] IBM Research. CoScripter. http://coscripter.researchlabs.ibm.com/coscripter, June 2012. Available from: http://coscripter.researchlabs.ibm.com/coscripter.

[24] Incutio Limited. The incutio XML-RPC library for PHP. http://scripts.incutio.com/xmlrpc/, June 2012. Available from: http://scripts.incutio.com/xmlrpc/.

[25] InnoCentive. *ALS Biomarker*. November 2010. Available from: https://gw.innocentive.com/ar/challenge/8305421.

[26] InnoCentive. *Identifying & Sourcing Novel Insecticidal Proteins*. November 2010. Available from: https://gw.innocentive.com/ar/challenge/9582167.

[27] InnoCentive. *What We Do*. November 2010. Available from: http://www2.innocentive.com/about-innocentive.

[28] B. J. Jansen, R. Ramadoss, M. Zhang, and N. Zang. Wrapper: An application for evaluating exploratory searching outside of the lab. *EESS 2006*, page 14, 2006.

[29] B. J. Jansen and A. Spink. How are we searching the world wide web? a comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248–263, 2006.

[30] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman. Defining a session on web search engines: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 58(6):862–871, April 2007. Available from: http://dx.doi.org/10.1002/asi.v58:6, doi:10.1002/asi.v58:6.

[31] Kantor, I. Focus/blur methods and events. http://javascript.info/tutorial/focus, June 2012. Available from: http://javascript.info/tutorial/focus.

[32] M. T. Keane, M. O'Brien, and B. Smyth. Are people biased in their use of search engines? *Commun. ACM*, 51(2):49–52, February 2008. Available from: http://doi.acm.org/10.1145/1314215.1314224, doi:10.1145/1314215.1314224.

[33] M. Konchady. *Text Mining Application Programming (Programming Series)*. Charles River Media, Inc., 2006.

[34] S. S. Laurent, J. Johnston, and E. Dumbill. *Programming Web Services With XML-RPC*. O'Reilly Media, Inc., 2001.

[35] D. Lewandowski. Query types and search topics of german web search engine users. *Information Services and Use*, 26(4):261–269, 2006.

[36] D. Lewandowski. Search engine user behaviour: How can users be guided to quality content? *Information Services and Use*, 28(3):261–268, 2008.

[37] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[38] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[39] P. Morville and J. Callender. *Search Patterns: Design for Discovery*. O'Reilly Media, Inc., 1st edition, 2010.

[40] Oracle. Oracle VM VirtualBox. https://www.virtualbox.org/, June 2012. Available from: `https://www.virtualbox.org/`.

[41] Privoxy Developers. Privoxy - home page. http://www.privoxy.org/, June 2012. Available from: `http://www.privoxy.org/`.

[42] R. W. Reeder, P. Pirolli, and S. K. Card. WebEyeMapper and WebLogger: tools for analyzing eye tracking data collected in web-use studies. In *CHI '01 extended abstracts on Human factors in computing systems*, CHI EA '01, page 19–20, New York, NY, USA, 2001. ACM. Available from: `http://doi.acm.org/10.1145/634067.634082`, `doi:10.1145/634067.634082`.

[43] G. Singer. *Web Search Engines and Complex Information Needs*. Ph.D. thesis, University of Tartu, Estonia, 2012.

[44] G Singer, D Danilov, and U Norbisrath. Complex search: aggregation, discovery, and synthesis. *Proceedings of the Estonian Academy of Sciences*, 61(2):89, 2012. Available from: `http://www.kirj.ee/?id=20506&tpl=1061&c_tpl=1064`, `doi:10.3176/proc.2012.2.02`.

[45] G. Singer, U. Norbisrath, E. Vainikko, H. Kikkas, and D. Lewandowski. Searchlogger analyzing exploratory search tasks. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, page 751–756, New York, NY, USA, 2011. ACM. Available from: `http://doi.acm.org/10.1145/1982185.1982350`, `doi:10.1145/1982185.1982350`.

[46] Georg Singer, Ulrich Norbisrath, and Dirk Lewandowski. Ordinary search engine users carrying out complex search tasks. *arXiv:1206.1492*, June 2012. Available from: `http://arxiv.org/abs/1206.1492`.

[47] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, page 254–263, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. Available from: `http://dl.acm.org/citation.cfm?id=1613715.1613751`.

[48] A. Spink and B. J. Jansen. *Web Search: Public Searching On The Web*. Springer, July 2004.

[49] K. Stanoevska-Slabeva. Enabled innovation: Instruments and methods of internet-based collaborative innovation. 2011.

[50] Stsenov, G. *Measuring mobile search tasks on Android platform*. B.Sc. thesis, University of Tartu, Estonia, 2011.

[51] M. Tatham. Google received 72 percent of U.S. searches in january 2009. Technical report, Hitwise, New York, February 2009. Available from: `http://image.exct.net/lib/fefc1774726706/d/1/SearchEngines_Jan09.pdf`.

[52] The Lemur Project. Lemur project components: Query log toolbar. http://www.lemurproject.org/toolbar.php, June 2012. Available from: http://www.lemurproject.org/toolbar.php.

[53] TREC organisers. *TREC Tracks*. June 2012. Available from: http://trec.nist.gov/tracks.html.

[54] R. W. White and R. A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.

[55] Wikipedia contributors. *Amazon Mechanical Turk*. Wikimedia Foundation, Inc., June 2012. Page Version ID: 497448764. Available from: http://en.wikipedia.org/w/index.php?title=Amazon_Mechanical_Turk&oldid=497448764.

[56] Wikipedia contributors. CoScripter, January 2012. Page Version ID: 392861139. Available from: http://en.wikipedia.org/w/index.php?title=CoScripter&oldid=392861139.

[57] Wikipedia contributors. *Precision and recall*. Wikimedia Foundation, Inc., June 2012. Page Version ID: 497112923. Available from: http://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=497112923.

[58] Wikipedia contributors. Privoxy, June 2012. Page Version ID: 457510113. Available from: http://en.wikipedia.org/w/index.php?title=Privoxy&oldid=457510113.

[59] Wikipedia contributors. *TopCoder*. Wikimedia Foundation, Inc., June 2012. Page Version ID: 495297360. Available from: http://en.wikipedia.org/w/index.php?title=TopCoder&oldid=495297360.

[60] WordPress.org. Extend WordPress. http://wordpress.org/extend/, June 2012. Available from: http://wordpress.org/extend/.

[61] World Wide Web Consortium. Page visibility draft. http://www.w3.org/TR/2011/WD-page-visibility-20110602/, June 2012. Available from: http://www.w3.org/TR/2011/WD-page-visibility-20110602/.

[62] T. Yan, V. Kumar, and D. Ganesan. CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones. 2010.

[63] J. Yang, L. A. Adamic, and M. S. Ackerman. Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce*, page 246–255, 2008.

[64] K. P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 401–408, 2003.

Appendix A

Resources

The search task repository described in my thesis is available for public use and scrutiny at http://searchtaskrepo.wordpress.com/. The proxy-based logging solution can be downloaded from http://searchtaskrepo.wordpress.com/download-our-search-logger/. It is set up on a Debian operating system and requires Oracle VM VirtualBox to operate. The latter can be downloaded from https://www.virtualbox.org/wiki/Downloads. A lot of useful information about the development process of this software can be found at my blog dedicated to this research area: http://searchtaskrepo.blogspot.com/.