Optimizing a large dynamically generated website for search engine crawling and ranking

Research and implementation of a search engine optimization solution for a Fredhopper implementation

Johan Köhne

Department of
Media- and Knowledge Engineering
Technical University of Delft

2006

Optimizing a

large dynamically generated website

for search engine crawling and ranking

A thesis submitted in partial satisfaction

of the requirements for the degree of

Master of Science

presented at

Delft University of Technology,

Faculty of Electrical Engineering,

Mathematics, and Computer Science,

Department of Media- and Knowledge Engineering,

Man-Machine Interaction Group.

December 2006

Student Administration

Thesis title:	Optimizing a large dynamically generated website for search engine crawling and ranking
Student name:	Johan Köhne
Studentnumber:	1015389
Study:	Technische Informatica (Computer Science),
	Faculty of Electrical Engineering
Date of presentation:	19th of December 2006
Supervising committee:	drs. dr. L.J.M. Rothkrantz
	dr. ir. C.A.P.G. van der Mast
	dr. K. van der Meer
	ir. F.H.C. Nieuwenhuys (Director Fredhopper)

Abstract

In the past decade, businesses have started to use the internet as a sales channel. Naturally, the number of online visitors determines conversion rates and therefore sales. With the evolution of search engines, the large amount of traffic generated by these has become very important for online sales channels.

Fredhopper is a company that developed a software solution for large online sales channels. Their *Fredhopper Access Server* allows fast and intuitive navigation and search within any set of products. Clients may include this software in their own web front-end, using it as a webservice, or they may allow users to directly browse the catalogue on the *Fredhopper Access Server*. In the latter case, openly exposed to the internet, search engines were found to have trouble indexing the catalogue as presented, resulting in minimal incoming traffic from search engines: a missed opportunity with a financial impact.

Literature describes how the inability of search engines to effectively crawl and index dynamically generated websites is a common issue, which is hard to solve on the search engine side.

The work presented in this thesis analyzes the underlying causes of the issue, and provides design guidelines for possible solutions, which are both universally applicable to (large) dynamically generated websites. The analysis consists of a broad review of crawler techniques and search engine ranking algorithms. Proposed improvements and design heuristics include techniques as URL rewriting, site structure simulation and keyword optimization. A Fredhopper specific solution is derived and implemented for a representative live case. The effectiveness of the solution is substantiated by the results of empirical studies, which show a large improvement in page crawlability and prove a high potential for improved ranking.

Contents

Acknowledgements

This Master's thesis is the summary of my graduation project at the Man-Machine Interaction Group, Faculty of EEMCS, Delft University of Technology, on which I worked from October 2005 to November 2006. A major part of the work presented in this thesis was done at Fredhopper Benelux, in Amsterdam. The work presented in this thesis is my own, but I could not have accomplished this without the help of others. I want to take this opportunity to express my gratitude to the following people for their contribution:

- Frederik Nieuwenhuys, for providing an excellent opportunity to do my thesis work in Amsterdam, in a stimulating and instructive environment, and his many useful suggestions on my work;
- Leon Rothkrantz, for being there when needed although he is a very busy person, and his restful approach when providing me with feedback and suggestions.

Next to contribution to my work, I would like to thank the following people for their support during this year of hard work and travelling, for making invaluable comments on my thesis, and for their personal involvement:

- Eva van der Wulp,
- Jelle ten Hoeve, Michel Meulpolder, and
- Christiaan Mourik.

1 Introduction

As this thesis discusses two subjects that will the reader will likely be unfamiliar with, this introduction describes both “search engine optimization” and “Fredhopper”. However, this chapter will first outline the contents of this thesis.

1.1 Outline

Fredhopper is a company that developed a software solution for presenting catalogues online, allowing fast searching and intuitive navigation. This software is called the *Fredhopper Access Server*. Clients may include this software in their own web front-end, using it as a webservice, or they may allow users to directly browse the catalogue on the *Fredhopper Access Server*.

The first type of implementation allows clients to integrate the Fredhopper output as they like, and leaves all responsibility for layout and interface with the client. However, for the open implementations, the *Fredhopper Access Server* itself is exposed to the world wide web. Unfortunately, search engines were found to have trouble indexing the catalogues presented, resulting in minimal incoming traffic from search engines for these implementations.

Literature indicates that the inability of search engines to effectively crawl and index dynamically generated websites is a common issue, which is hard to solve on the search engine side. This is in short the problem this thesis addresses, and is more precisely formulate in the next chapter.

1.1.1 Problem relevance

Scientific Impact

As previously mentioned, literature describes the crawlability and indexing problems that occur with large and dynamic websites. Chapter 3 will review this existing literature. During my research we have not found any such extensive and broad review of the issues related to search engine optimization as we have collected and reviewed them in [21].

In addition to this contribution to science, this thesis work provides solution design guidelines that are applicable for many implementations. Empirical evidence shows its effectiveness and reliability.

Societal relevance

The issues that this thesis addresses are very actual and common, contributing to the relevance of this work. Software engineers and webdesigners working on projects that face the same issues will find the guidelines in this thesis very specific and readily applicable. Furthermore, end-users on the internet will ultimately find things easier on the internet, as the work in this thesis contributes to increased visibility and presence of webpages.

Economic benefits

The economic benefits of this work are twofold. This thesis models and implements a solution around a Fredhopper implementation, which is a true online sales channel. With the increase of traffic generated by search engines, more potential clients are attracted, which should ultimately result in increased online sales. Additionally, Fredhopper finds itself with another module that can be sold with their product.

1.1.2 Solution

This thesis work will ultimately model a solution which both simulates a crawler friendly site structure, and uses crawler friendly URLs at the same time. It will also focus on keyword presence and presentation. This combination of approaches should help increase the visibility of webpages as crawlers will now be able to easily index them. At the same time, potential is created for higher rankings as keyword presence is increased in both URLs and page contents.

The solution that is the subject of this thesis is actually implemented in a live case, Quelle France, providing empirical evidence that the solution works, is effective and is reliable. This case is very representative for the problematic Fredhopper implementations, and also resembles many other cases present on the internet.

1.2 Background

1.2.1 Fredhopper

As a scientific thesis will allow many different contexts, this gave me the opportunity to choose my own experience. I had decided to experience a corporate environment. After being shown different companies and having done several interviews, Fredhopper offered me an internship that interested me very much: both the subject of their business, as well as their professional and corporate way of work.

Company

The company Fredhopper started its business in 1999 in the USA, but soon changed its focus to Europe. Today the headquarter and development is located in Amsterdam, Netherlands - sales and support offices are spread over Western Europe and the USA. The development is supported by the Dutch program WBSO[1], which supports innovative companies in research and development. Fredhopper was regularly ranking in top positions of Deloitte's Technology Fast-50 as one of the fastest-growing high-tech companies [12],[13]. Customers are major sales channels in Europe and the United States like Philips, Conrad Electronics and Thomas Cook.

Figure 1: Fredhopper logo

Fredhopper is developing search, navigation and merchandizing software for online sales channels [15]. Their product, Fredhopper Access Server, offers intuitive and fast ways of browsing large data sets in several ways, as well as management tools for control and reporting. Websites with such navigation increase their sales, because end-users easier find products that they want to buy. Studies on Fredhopper's clients show that they normally increase their sales by 30-40%.

Fredhopper Access Server

To be able to formulate the objective of my internship, some understanding of Fredhopper Access Server (FAS) is required. An extensive and more technical description will be given in chapter 0. Fredhopper Access Server, abbreviated throughout this report as FAS, is software that makes large databases of items easy and fast to browse over the internet. To achieve this, a (usually dedicated) machine loads a complete database into memory and maintains a search tree to allow fast searching in many dimensions.

The database that is loaded needs to be in Fredhopper database format. This format has tables dedicated to the products, product categories, product attributes, assets, prices, languages and availability. By specifying a format, FAS can load and interpret the database in a semantically correct and useful way.

[1] http://www.senternovem.nl/wbso/

Communication with the Fredhopper Access Server uses SOAP, a protocol for exchanging XML-based messages over a computer network[2]. Both requests and replies are therefore in XML format. Requests are typically searches for products, whereas replies are usually a list of products along with their attributes and categories.

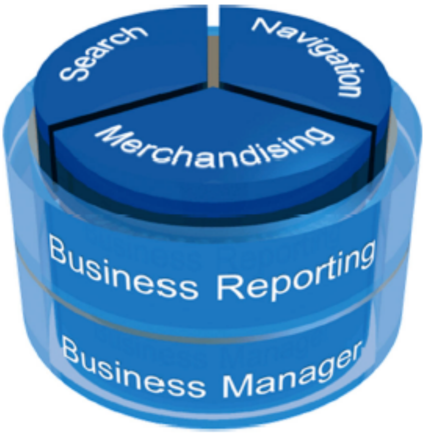


Figure 2: Fredhopper Access Server components

Fredhopper Access Server is a Java 2 EE application running on JBoss, an open source J2EE application server. JBoss has the Apache Tomcat webserver embedded, making it an excellent platform for FAS and its SOAP communication.

While the output of FAS is data formatted in XML, the server includes an XSLT transformer, which allows the output XML to be transformed server-side, before being returned to the client. XSLT is a language written in XML that defines transformations from XML to XML. This allows developers to transform the output XML into XHTML server-side and forward the output directly to the visitor's browser. Other Fredhopper clients however decide to use their own front-end for processing the output XML and implement it in their own website.

An influential choice in the design of FAS was the goal to make FAS stateless. This means the server does not maintain any information on visitors and the pages they visit. This will potentially save many resources when there are many visitors, since no information needs to be stored server-side. However, FAS can not relate subsequent page-views by visitors, and requires the whole domain of input parameters for each page-view. These input parameters are communicated in the URL as query parameters. These are normally visible to visitors in URLs after the question mark, e.g.:

[2] http://en.wikipedia.org/wiki/SOAP

`http://www.fredhopper.com/public/services.php?cat=2`

where cat=2 is the query string, indicating the parameter `cat` has value `2` for this request.

Apart from the catalogue features, used to navigate and search products, FAS includes management features as well. It automatically reports business information on product views, online purchases and promotion efficiency. This information can be used to change the configuration of FAS by using the *business manager*. The *business manager* is a feature in FAS that allows a broad range of settings to be configured, including the appearance of specific promotions and the configuration of the search engine.

1.2.2 Search Engine Optimization

For businesses that sell on the internet, success depends on visitors. While only a small part of these visitors may turn out to be an actual customer, this means that more visitors will ultimately result in more customers. It's a numbers game. Generating traffic is therefore an important goal in all online activity.

In the last decade, new players emerged that have become most influential spectators: search engines. The amount of traffic generated by search engines is different for every business, but in some reported cases makes up for 80% of all visitors. While it is possible to pay for traffic by means of advertisements, traffic from search engines is free. Search engine optimization is the - relatively new - field that specializes in obtaining, increasing and maintaining this free traffic. Businesses that offer "search engine optimization" or "search engine marketing", offer to help improve the visibility of a website on search engines.

Figure 3: Google in a box, Google's intranet solution

Search engine optimization (abbreviated throughout this report as SEO) may sound like a paradox to some: How is it possible to influence the search engine results for a specific query, while search engine are supposed to be completely objective? The following example will shed some light on this matter. Search any of the biggest 3 search engines (MSN, Yahoo! and Google) for the term "`miserable failure`" and the first result that comes up is the "Biography of George W. Bush". To stress that they are not actively involved in this joke, Google has given an explanation of the phenomenon on their official blog[3].

[3] http://googleblog.blogspot.com/2005/09/googlebombing-failure.html

The high ranking was obtained by a large amount of incoming links with specific anchor text (Bush antagonists chose "`miserable failure`" on purpose). In addition, the odd result got much attention and its popularity was picked up by the search engine ranking algorithms: a simple mathematical equation. Search engines have their algorithms designed to relate phrases to websites, and order the results by popularity. Over the few years this joke has been running now, the biography now even shows up in the top 3 for the extremely common phrase of "`failure`" alone and the phrase "`miserable failure`" has become a commonly known and understood internet phenomenon[4]. While the actual purpose of SEO companies will be different, the strategies and techniques employed by these optimizers are exactly the same.

[4] http://en.wikipedia.org/wiki/Miserable_failure

2 Problem Definition

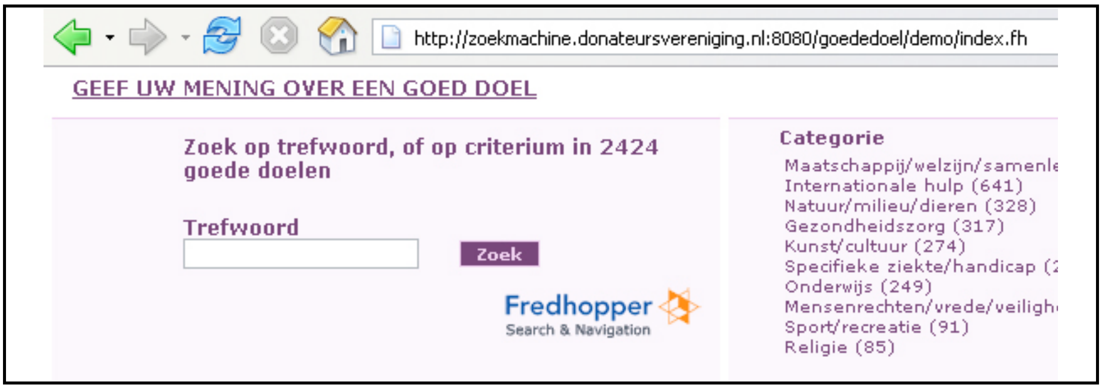
With the subjects of SEO and Fredhopper explained, this chapter will outline the problem domain and specify the objectives of my project.

2.1 Problem outline

While Fredhopper Access Server works perfectly in navigation and search, the pages that it generates will not show up in Google. With the relatively recent focus on search engine generated traffic, an increasing amount of Fredhopper clients demands search engine friendly pages. Naturally, this problem occurs for those clients that do not integrate the FAS in their own front-end, where they could address this issue themselves. The distinction between such enclosed implementations and open implementations is explained and visualized in detail in paragraph 4.2.1. This thesis addresses the issues that are typical with such open implementations, as these are for all other webplatforms that generate a large amount of pages dynamically, from a large product catalogue. The problem thus applies to a large set of webapplications, especially those with FAS characteristics like parametric browsing and a large amount of navigational links.

Thus, the problem can be defined as:

Given a large webplatforms that generate a large amount of pages dynamically. A problem with these platforms is that they are typically badly indexed by search engines, if at all, and thus loose much potential traffic and income.



http://zoekmachine.donateursvereniging.nl:8080/goededoel/demo/index.fh

GEEF UW MENING OVER EEN GOED DOEL

Zoek op trefwoord, of op criterium in 2424 goede doelen

Trefwoord

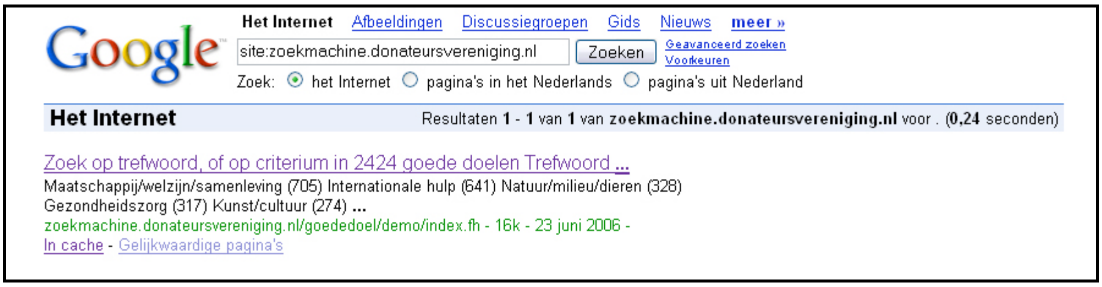
Categorie

Maatschappij/welzijn/samenle
Internationale hulp (641)
Natuur/milieu/dieren (328)
Gezondheidszorg (317)
Kunst/cultuur (274)
Specifieke ziekte/handicap (2
Onderwijs (249)
Mensenrechten/vrede/veiligh
Sport/recreatie (91)
Religie (85)

Figure 4: Fredhopper implementation of the Donateursvereniging

This thesis will focus on open Fredhopper applications as a typical case where this problem applies. Refer to Figure 4 for a visualization of the problem. It shows a FAS start page that states there are 2424 items in the system. However when we search a big search engine like

Google for all pages from the FAS base URL, only this start page is included in the index (refer to Figure 5), and not one link to a single item is maintained. This implies that searches through Google will only be able to return the starting page, while all relevant pages remain invisible. Obviously, this is very much undesirable for Fredhopper clients: the implementation of online sales channels is done specifically to increase the visibility of individual products, not just their website.



site:zoekmachine.donateursvereniging.nl

Resultaten **1** - **1** van **1** van **zoekmachine.donateursvereniging.nl** voor . (**0,24** seconden)

Zoek op trefwoord, of op criterium in 2424 goede doelen Trefwoord ...
Maatschappij/welzijn/samenleving (705) Internationale hulp (641) Natuur/milieu/dieren (328)
Gezondheidszorg (317) Kunst/cultuur (274) **...**
zoekmachine.donateursvereniging.nl/goededoel/demo/index.fh - 16k - 23 juni 2006 -
In cache - Gelijkwaardige pagina's

Figure 5: Google index for zoekmachine.donateursvereniging.nl

2.2 Objectives & Constraints

The target of my research and contribution at Fredhopper was to investigate the given problem. Fredhopper development was aware of the issue and formulating plans to include search engine optimization support in the product itself. However, for my research no changes were to be made in the code of the Fredhopper Access Server. Development of FAS is strictly supervised and implemented by a dedicated department of programmers. As this development department was reviewing possibilities of integrating SEO solutions, the findings and outcome of my project would be reviewed and valued in this context.

In summary

The objectives of my project at Fredhopper:

- analyze the cause of the problem by reviewing literature on the subject;
- develop a solution model;
- implement this model, observing the identified constraints;
- review if the implementation is successful.

The constraints:

- the solution may not change the FAS application itself;
- the solution should not be a separate application;
- the solution is only required for open FAS implementations.

3 Literature, Theory & Previous work

In this chapter existing theory and work on the problem will be reviewed. Much of the contents of this chapter are extracted from my previous research into the subject, bundled in [21]. The reader is encouraged to refer to this work when interested in more and more in-depth information on the subject. The information presented here, however, will cover all of the knowledge required for understanding this thesis work.

To provide information that will help achieve the objectives described previously, this chapter focuses on Search Engine Optimization theory. It will:

- analyze the workings of modern-day search engines;
- here from deduce theoretically promising SEO techniques;
- review relevant aspects of current SEO practice.

3.1 Search engine anatomy

Search engines can semantically be divided into two separate parts: the crawlers and its indexing & search engine. While the engine stores and retrieves information about websites and their contents, crawlers are the eyes of a search engine. They are small little programs that, very much like an organism, wander the web and make notes of what they encounter. These notes are passed to the engine, which stores the information for retrieval.

3.1.1 Crawlers

A crawler that visits a webpage, will typically search for any links to other documents on the web and strip its contents down to the bare textual. This textual contents is stored for the engine, and the links are remembered for future visiting: crawlers maintain a list of pages to visit, which may only be one page as the crawler starts its expedition, but grows exponentially with every document, as most pages on the internet contain many links. The basic crawling algorithm can thus be expressed as in Figure 6.

Basic Crawling algorithm
Input: starting url: seed URL
Procedure:

```
enqueue(url queue, starting url)
while (not empty(url queue))
    url = dequeue(url queue)
    page = crawl page(url)
    enqueue(crawled pages, (url, page))
    url list = extract urls(page)
    foreach u in url list
            enqueue(links, (url, u))
            if (u/?url queue and (u,-)/?crawled pages)
                    enqueue(url queue, u)
reorder queue(url queue)
```

Function description:

`enqueue(queue, element)`	append element at the end of queue
`dequeue(queue)`	remove element at the beginning of queue and return it
`reorder queue(queue)`	reorder queue using information in links

Figure 6: Basic Crawling Algorithm [9]

While the workings of a crawler may seem very trivial, its implementation features many challenges. Castillo compares crawling the web and building a crawlers, to exploring the surface of mars with a vehicle (in [6]): "Implementing a Web crawler is, in a certain way, like building a vehicle for exploring the surface of mars: you need to build the vehicle to explore the terrain, but once you have tested it, you know more about the terrain and you have to modify your vehicle's design accordingly." Although some of these implementation challenges are of no relevance to our subject, some are of direct influence on the field of SEO. Therefore in the remainder of this paragraph, those relevant issues will be discussed.

Visiting order & depth

The document structure of a website can be visualized by graphs. Each node represents a document, and directed paths visualize links. Every site has a starting node, most often the root document of a website, that represents the root. It will be clear that loops and dead-ends are common, and need to be accounted for. Figure 7 shows a simplified example site structure, which is modelled as a directed graph. Crawlers apply common tree-walk algorithms to find a path through such a structure. Bread-first and depth-first are common implemented approaches. The figure visualizes how the perceived depth of a document will vary according to the algorithm used. It needs be stated that the figure omits some links that are likely to occur in reality; for example, often there are links to the homepage (root node) from almost every document.

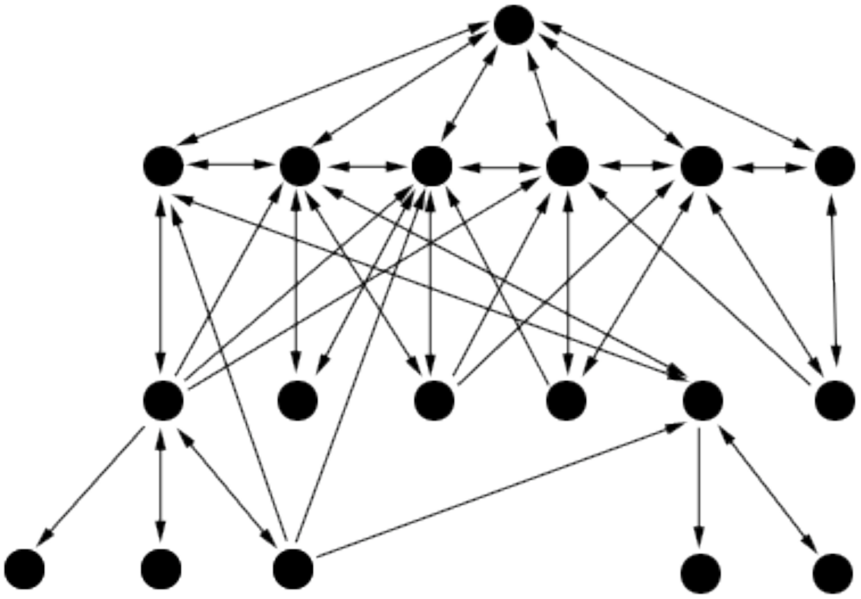


Figure 7: Simplified site structure graph

Depending on the path taken through a site, some documents will be encountered earlier than others. The depth of a document can be described as the minimum number of clicks a user would need to reach this document. Crawlers of large search engines are known to limit visiting depth for some sites. For example Googlebot, Google's crawler, is known to limit visiting depth [Matt[5]], depending on its PageRank. The assumption is that these pages are less important, because of their remote placement.

In addition, the depth of a document influences its perceived importance quantitatively. Rankings algorithms (discussed later this chapter) will numerically value documents encountered early on more, than those hidden in the corners and niches of a site.

It can be concluded that a site's structure needs to be carefully planned. The most important documents need all to be up-front, within a few clicks reach from the root to ensure indexing and decent valuation.

Query strings (URL parameters)

As most recent websites serve dynamic content, this usually implies that additional information is required to process a page request. In a most simple example, a website features a single document, which fetches the content based on a page name or ID passed. Advantages to webdesigners and programmers are clear: layout information, as well as other non-changing information is only stored once, and redundancy is avoided. This

[5] http://mattcutts.com/blog/q-a-thread-march-27-2006/

additional information can be passed among page-views in several ways, of which the most common and easy method is through the URL as query string. This query string is glued to the original URL with a question mark.

The problem for crawlers that arises with this situation is their inability to understand the meaning of the different parameters. While requests to `index.php?page=welcome` and `index.php?page=faq` will produce two pages with different contents, the requested document in both cases is `index.php`. However, the parameter could just as well be meaningless (imagine a parameter maintaining a visitors assigned ID or the date), and the pages produced are semantically identical. The biggest problem with the situation is the risk of crawlers being caught in a *spidertrap*: by assuming pages are different, based on their parameters, crawlers can get stuck in a loop of pages that are actually the same.

A solution may seem simple: compare the generated contents of pages. However, much contents on pages changes for every visit (advertisements, date & time) or simply updates very quickly like the contents of news-sites. In the end crawlers prefer not indexing a page to the risk of getting stuck in a *spidertrap*.

Sessions & Cookies

Sessions and cookies are another practiced way of maintaining information between different page views. Sessions are kept server-side, while cookies are stored locally on the visitor's computer. However, even with sessions the visitor is expected to keep track of its assigned session ID. The risk of a *spidertrap* is here just as big as with query strings. In addition client-side session and cookie maintenance would require much additional resources and code complications for crawlers.

Hidden web

Many websites use forms as a portal to more information. For specific input, a page is generated with the requested information. Often the displayed information is retrieved from a database. Obviously, classic crawlers techniques will never reach this "hidden information". The amount of hidden information on the internet, called the *hidden web* is estimated to be up to 500 times the size of the *shallow web [24]*.

It is important to realize the existence of the hidden web, and the obstacles that are hiding it from crawlers. Forms are the biggest obstacles in this regard, but there are others, like script generated links or clicks.

Robots exclusion protocol

Crawlers can be told what files or folders not to index on a website. The robots exclusion protocol from 1994 states that any crawler will read the `robots.txt` file in the root of a website and follow instructions in it. These instructions consist of an agent identification combined with restricted or allowed file and path expressions. The specifics of the syntax can be found online[6]. This way certain pages may be kept out of the index and secret to the public eyes. However, providing a robots.txt file does not mean that the excluded files will remain hidden. On the contrary, it makes them even *more* visible because of the listing.

```
# Disallow all crawlers access to certain pages.

User-agent: *
Disallow: /exec/obidos/account-access-login
Disallow: /exec/obidos/change-style
Disallow: /exec/obidos/flex-sign-in
Disallow: /exec/obidos/handle-buy-box
Disallow: /exec/obidos/tg/cm/member
Disallow: /gp/cart
Disallow: /gp/flex
Disallow: /gp/product/e-mail-friend
Disallow: /gp/product/product-availability
Disallow: /gp/product/rate-this-item
Disallow: /gp/sign-in
```

Figure 8: Contents of /robots.txt on Amazon.com

Robots META-tags

While the `robots.txt` file specifies for a whole website which URLs should not be visited, this can also be achieved for each individual page using crawler specific META tags. These META tags are placed in the HTML header of a page and allow webmasters to specify for each individual document, whether crawlers may index it and if the links on the page should be followed.

Link exclusion

Modern search engine algorithms reward external pages that links to your site. It can be easily understood that this invokes a desire in many webmasters and online activists to obtain as many inbound links as possible. The result is a phenomenon called 'comment-spam'.

Comment-spam is well described by an author from Slashdot.org: "Its defining characteristic is that spammers abuse websites where the public can add content (blogs,

[6] http://www.robotstxt.org/

wiki's, forums, and even top referrer lists) to increase their own ranking in search engines."[7] Since most comments allow links in it, this feature has been abused to generate links to sites. In an attempt to fight this abuse and help webmasters protect their site from such abuse, the three biggest search engines (Google, Yahoo! And MSN) agreed on a new attribute for link tags[8]:

<a href="{src}" **rel="nofollow"**>

With the added `rel="nofollow"` a link can easily be marked for crawlers to be ignored. Although this technique was invented for other purposes, it may prove useful in SEO.

Technical workaround

In addition to the previously described official techniques to manage visiting crawlers on your website, there are other possibilities to control crawlers. Technical barriers can be created for crawlers that make it impossible for crawlers to detect and follow a link. To illustrate this, review the following example from Kelkoo in Figure 9. Although this technique surely works, it is the least aesthetic of all. The links themselves are made unreadable, although clicking them will take you to the correct page.

```
HTML (on page)
<a href="javascript:link('/ctl/do/co', 'mpare?catI', 'd=12843182&p', 'id=1235')">
Olympus µ Digital 800
</a>


JAVASCRIPT
// Scripts to display non-crawlable Javascript links
// Links to the page given by a splitted URL.
// The arguments are the tokens forming the splitted URL.
function link()
{
   var url = '';
   for (i = 0; i < arguments.length; i++)
   {
      url = url + arguments[i];
   }
   go(url);
}
```

Figure 9: Kelkoo[9] creates links that can not be followed

[7] http://it.slashdot.org/article.pl?sid=05/01/19/0516246&tid=111&tid=217
[8] http://blogs.msdn.com/msnsearch/archive/2005/01/18/nofollow_tags.aspx
[9] http://www.kelkoo.nl/

3.1.2 Ranking

The theory and practice of ranking algorithms originates in the field of information retrieval. Rankings algorithms in the mid nineties were based merely on keywords, as theories and algorithms from classic information retrieval were used. Work in this field goes back to as early as the seventies [28], [29].

Information retrieval can be defined as the field of work that is concerned with retrieving specific information in a much larger set of information. The IR (information retrieval) system processes this query, and searches its index for relevant documents. Two measures have been introduced in IR theory to measure the quality of retrieval: precision and recall. Precision is the ratio of relevant documents retrieved over the total amount of retrieved documents. Recall is the ratio of retrieved relevant documents over the total amount of relevant documents [28].

Search engines have always used this fundamental work in information retrieval. In addition, meta-data present on the WWW is used in search engine ranking algorithms. In particular, work that focuses on the 'naive' and short queries that searchers issue is highly relevant for the specific context of the WWW. Conventional IR research has traditionally focused on long queries with comparatively expert searchers [11].

Early ranking algorithms

Early ranking algorithms were keyword based. Reviewing WebCrawler, the first full-text search engine for the WWW, we learn about the basics of ranking algorithms [26]. The first and most basic factor is the count of the number of appearances from keywords in the query in the document. "For *w* words in the query, the score for each document would be

$$relevance(doc, query) = MIN\left(\sum_{i=1}^{w} tf(i), maxweight\right),$$

where *tf(i)* is the frequency of the word in the particular document, computed as the number of occurrences of a word divided by the total number of words in the document. The rationale for using this weighting is that the frequency of a term in a document is a good measure of the importance of that term to the document [26] ."

Improving on this basic relevancy indication, several strategies have been employed. Remaining with our example of WebCrawler, example strategies are: word domain narrowing and a stop-list, which would filter out words that appeared in too many documents to be of any importance. Word domain narrowing can be considered the opposite: very specific query keywords, which would only appear in a few documents, are given considerably more weight. The formula for relevancy would thus become:

$$relevance(doc, query) = MIN\left\{ \sum_{i=1}^{w} tf(i).\left(1 - \frac{\log(\text{docs containing i})}{\log(\text{total \# of docs})} \right), maxweight \right\}.$$

As WebCrawler improved it started using a third-party full-text system. In this software, some of the currently common improvements on keyword based relevancy calculations were implemented.

Improvements to classic ranking

Improvements to the classic ranking algorithms use other information from the documents in addition to just the contents to determine keyword relevancy. Commonly adapted techniques weigh keywords more when they appear closer to the top of the document. For multi-keyword searches, proximity of the found keywords increases the relevancy too. These factors were given more importance as the algorithms to determine a document's relevancy was improved.

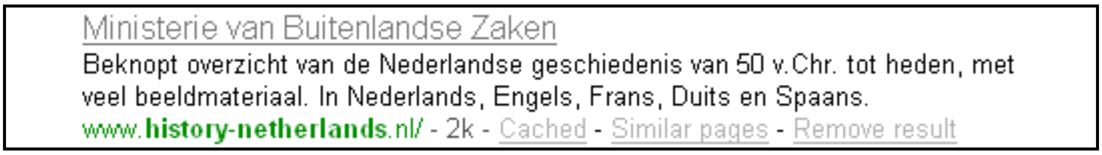
The documents that need to be indexed were all HTML pages, formatted for specific visual presentation. New strategies were created that would use the information contained within this visual presentation. Typically certain HTML tags would visually emphasize its contents (like the header tags `<h1>` … to `<h6>`). These header tags are known to be given extra weight [25].

In addition META tags, optional descriptive tags for HTML documents were scanned for keywords. This is not surprising, as these tags were specifically implemented to help authors provide META information about their documents. This was considered valuable information, until META tags were starting to be abused more than they were used for its intended purpose. The specification of the relevant META tags is presented in Figure 10.

META name="description"	content="Everything about Tristan da Cunha, from its history and demographics to unique stamps."
META name="keywords"	content="Tristan da Cunha, remote island, Atlantic islands, rare stamps”

Figure 10: META tags (keywords and description)

The name of the document itself was found to present interesting information on its contents too. A page named `history_of_computers` will likely be relevant for someone that enters the search-phrase: "history computers". In addition the physical location (for example: the directory on server) of a document would tell even more. Google conveniently visualizes this: when the searched-for keywords appear in the URL of a document, this phrase is highlighted in the URL (see Figure 11).



Ministerie van Buitenlandse Zaken
Beknopt overzicht van de Nederlandse geschiedenis van 50 v.Chr. tot heden, met veel beeldmateriaal. In Nederlands, Engels, Frans, Duits en Spaans.
www.**history-netherlands**.nl/ - 2k - Cached - Similar pages - Remove result

Figure 11: Google using keyword presence in URLs (query: "history netherlands")

More META-information used to calculate relevancy for specific keywords, was found in the pagetitles. The information contained in pagetitles was already known to be a very useful indicator of the contents of a page (early indexing-systems like Jumpstation in 1993 actually used *only* the pagetitles to index documents).

The text that is used to describe a link, also called anchor text, is another source of information about a document. The focus here is on the content of incoming links, more specifically, links contained in other documents [19].

Pagerank

A revolutionary change in ranking algorithms came from Lawrence Page & Sergey Brin [4]. Their PageRank algorithm, named after Page himself, expanded the scope of classic rankings. In addition to the contents of a page, and the META-data contained in the document and URL, they worked out a technique that would use information from the WWW as a whole.

The basic idea of PageRank is that it introduces a measure of authority, derived only of the graphical (linking) structure of the WWW [2],[4]. Without going too deep inside the mathematical and algorithmic background of PageRank, it is interesting to review the basic approach and calculations involved.

PageRank counts each link from a page on the WWW to another page. A page that has many incoming links is considered an important page on the web. This is based on the assumption that relevant websites will be linked to by many others.

The authors themselves can best explain: "[They] assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor, which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

$$PR(A) = (1 - d) + d(PR(T1) / C(T1) + ... + PR(Tn) / C(Tn))$$

Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one" [4].

The success of Google is well known. Their popularity was fuelled by their search results relevancy, which was very high thanks to PageRank. It helped improve the relevancy of the results, by ordering the results from a classic keyword relevancy search. Webpages with a higher PageRank are considered more authorative and ranked above less popular pages.

It is a common mistake to confuse PageRank with relevancy. PageRank is a statement of a web page's importance, in relation to all web pages on the Internet. It is not a declaration of relevancy for search terms. It is important to understand the meaning of PageRank, the basic principles in its calculation and the application in ranking. Typically a keyword relevancy search is executed on a search engine, and these results are ordered using the PageRank value in addition to keyword scores.

The idea of using the structure of the web itself to improve relevancy for results is now applied by virtually all search engines. The algorithm is known to have changed over time. Two important changes in relation to SEO are important to identify. Firstly, PageRank is now known to weigh incoming links with respect to the PageRank of the page that features the outgoing link. Secondly, the value of outgoing links is distributed over the amount of outgoing links. It is relevant to realize the effect of these changes.

Beyond PageRank

Search engines have been developing ever since they first appeared. Algorithms like PageRank eventually need to be improved, as the contents of and behaviour on the WWW changes. Several improvements have been proposed and implemented, ranging from algorithmic (performance) improvements [22] to weighted PageRank algorithms [30]. It is also known that Google itself is working on improvements: initiatives like TrustRank and Hilltop.

TrustRank is known to be an extension to PageRank that propagates the trust-factor of a webpage through its links. The idea is that links from trusted sites are worth more, than links from doubtful or even spamming sites. This was initiated to fight the phenomenon of "search engine spam". The possibility of fighting web-spam using TrustRank is extensively researched in [18]. A related algorithm is the Hilltop ranking factor, which focuses on links from authoritative pages on a specific topic[10]. It attempts to categorize a query into a specific topic, and increases the ratings of sites that are known to be authoritative in this area.

Much of the algorithms search engines use remain well-kept secrets. By making them public, their algorithms can more easily be cheated and the relevancy of their search will decrease. The power of search engines remains in their objective judgement by organic analysis of the web. By exception, a relatively new area of Google's techniques was revealed by a patent request. In this request, which can be viewed online[11], it is explained how Google inventively identifies more META data. The request, which is very hard to understand with all the legal terms used, has been analyzed and reviewed by seomoz.org[12]. Apparently Google uses quite some additional historical information to refine the PageRank calculation: the page's ranking history, the development of link count (both inbound and outbound) and update frequency of a site to name a few. This implies that older sites are generally assumed to be more reliable. In addition its outbound links are valued more.

Search engines are presenting their ranked results to their users. The ranking algorithms incorporate many factors. The results itself however, have shown to be another interesting source. Search engines can easily acquire more information about the relevancy of webpages by tracking the user clicks on results. This "click-through data" has been shown to be very useful information in for example [20]. In general this strategy assumes that users will click on results that are relevant. This information provides search engines for free with human judgement information. More recently, search engines allowed users to rate and exclude results directly from the search engines results pages. Naturally, this information too is stored and used for ranking purposes.

Ranking overview

In the past paragraphs about ranking, the evolution of ranking algorithms has been reviewed. All known factors that influence rankings have been highlighted. Obviously this information will be critical to improving the rankings on search engines. For better overview and future reference, a summary of these factors is presented here:

- pagetitles;
- early in the document (close to the top);

[10] http://www.cs.toronto.edu/~georgem/hilltop/

[11] http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&co1=AND&d=PG01&s1=20050071741&OS=20050071741&RS=20050071741

[12] http://www.seomoz.org/articles/google-historical-data-patent.php

- URLs (domain- and document-name);
- emphasizing tags: <h1> … to <h6> or bold;
- META tags description & keyword;
- proximity to other keywords;
- anchor-text of inbound links;
- PageRank of referrer;
- historical data, including the development of inbound links, age of domain, update frequency of documents;
- clickthrough data from search engine results pages.

3.2 The concept of SEO

Reviewing the knowledge from the previous paragraphs, we can formulate techniques that are expected to improve search engine rankings. First, three different angles from which search engine optimization can be viewed are identified. Secondly, search engine techniques described previously are analyzed and strategies for SEO are deduced.

3.2.1 Classification of different SEO approaches

In this paragraph the approaches to SEO will be classified along three dimensions: paid versus organic, on page versus off page, and ranking versus crawlability. These will be shortly discussed.

Paid vs. Organic

First of all, there is the need to distinguish between organic and paid optimization. Paid optimization is straight-forward: for a specific (set of) keyword(s), businesses can make their link appear with the search results. Search engines each have their own implementation. Google for example, places the paid results above the organic (natural) results on a blue background, with a maximum of two per page. Natural optimization on the other hand, deals with obtaining a high(er) ranking in the 'normal' results pages.

On page vs. Off page

Secondly there is a difference between optimization on page and off page. Reviewing the workings of search engines, it can be observed that the factors of influence can be either on the ranked page itself (on page), or external (off page). This distinction helps identify the areas where optimization can be effectively done and which party is responsible. Off page factors include (among many more) the amount of links and anchor text within these links to a website, the age of a domain, the quality of referring sites, and the click-through data from search engines. On page factors include the contents body of a website, the physical site structure, titles and headers and the content update frequency.

Ranking vs. Crawlability

Third and last, it is important to distinguish two different approaches to optimization. Regarding crawlers, we need to improve crawlability of websites, to ensure the scanning and indexing of all content. Regarding ranking, we need to improve our keyword presence and presentation. In addition, inbound links influence ranking. Ranking and crawlability optimization are the two roads along which we can find ways to achieve search engine optimization.

Concluding

The discussed dimensions span the domain of SEO. Table 1 displays the 8 different areas of approach that these dimensions span. For each of these area it names the applicable optimization practice.

	Organic		Paid	
	On page	Off page	On page	Off page
Ranking	Keyword optimization	Link building	Outsourced keyword optimization	Search engine ad programs, buying incoming links
Crawlability	Crawlability optimization	Search engine sitemaps	Outsourced crawlability optimization	Using external proxy's and landing pages

Table 1: Dimensions of SEO

3.2.2 Theoretical improvements

Document layout, presentation and contents

The workings of search engines are clearly text-based. Crawlers are not (yet) able to categorize what is depicted in images or movies. Multimedia contents therefore don't have any influence on search engine rankings. If at all, the effect will be negative by missing opportunities to promote keywords and readability.

For a few years now, sites are being designed fully as a movie, using techniques like Macromedia Flash® or Shockwave®. These flash-sites, although their contents may be large and very interesting, do very badly at search engine visibility. From a theoretical point of view, we can therefore say that before anything a site should be textual.

Keywords

Keyword selection

Keywords should be carefully selected. The process of keyword selection is extensively described in [32]. Wall describes the importance to realize that keywords are not how a business describes its own contents; keywords are the words that users or clients will use to describe what they are looking for. This is an import realization.

Many keywords return a long list on the search engine. Keywords can therefore be expected to be more effective if they are not so frequently used. If business or site-contents allows, niche keywords can be expected to be easier to rank high for. These 'more unique' keywords can best be used in combination with the more common keywords that specify a general topic or service.

Keyword placement

It was previously reviewed how keywords are treated by ranking algorithms. After a selection of keywords has been made, important keywords should be stressed. Regarding the valuation of keywords from a ranking point of view, the list in paragraph 3.2.2 sums up all locations where keywords should be placed for extra attention from ranking algorithms.

The on-page factors are straightforward. However, an extra word on the off-page factor is required. The anchor-text of inbound links is an external factor that we have no influence on. By choosing a domain name with keywords in it, keyword presence in inbound URLs can be expected to increase. All other factors listed are on-page factors, considering the domain name free to choose.

URL keyword optimization may require physical changes. All other factors can be optimized for within one document. In the following paragraph on crawlability, more will be said about the possibilities of URL keyword optimization.

Over-optimization

It can be expected that a site that has only one or two words repeated everywhere is given a bad rating by search engines. Keyword optimization is theoretically very effective, but search engines have taken countermeasures to stop keyword spamming. Keyword spamming, along other over-the-top optimization strategies are referred to as over-optimization. The practice and related penalties are discussed in paragraph 3.3.

Crawlability

Identifying issues

Crawlers prefer plain and simple (HTML) documents, residing in a simple and straightforward structure. Many large websites that generate pages with scripting languages and databases will have trouble presenting such a site. In particular, the following issues pose a challenge:

- parameters in URL's, used to transfer information through URL's;
- sessions and cookies, used to store and transfer information between pages;
- JavaScript links or forms;
- crawlers with limited search depth;
- malformed document format;
- information "hidden" behind forms or scripted navigation elements.

Regarding crawlability it seems there are 3 types of issues that we can distinguish. First of all, pages need to be linked to. Secondly, the URL needs to be technically requestable and pass any filters that crawlers may use. Third and last: the document needs to be of a format that crawlers can actually read without problems. These are the issues that we need to pay attention to:

- document is linked (linking);
- URL can be requested (requesting);
- document can be read (reading).

Sitemaps

As it is obviously important to make crawlers reach all documents on a website, it seems a logical step to create a page that actually contains links to every document on a website. These pages provide a map of a whole site and are commonly called sitemaps. Including a link to such a sitemap from the bottom of every page, may be an effective way to increase the visibility of documents. Google has even started a sitemap-submit service[13], which allows webmasters to directly submit a list of URL's to all documents to Google. The implementation of a sitemap can be automated for dynamically generated sites: a sitemap page can easily generate links to all documents on request. Sitemaps help solve problems typed as 'linking' problems.

[13] https://www.google.com/webmasters/sitemaps/login

URL rewriting

The 'requesting' problems exist because of crawlers being unable or unwilling to request acquired URLs. The most common cause here is URL parameters, which create the risk of *spidertraps* (described earlier in this chapter). In addition, the extension of a document may tell that the contents are dynamically generated; naturally crawlers must treat such documents more carefully, as redundant content is a big risk.

Instead of creating several physical unique files to eliminate these problems, this can be solved by *pretending* to serve physically unique files. For example: imagine a document `index.php` that takes a numeric parameter `showpage` through the query string. This document seems singular, but may in reality generate a unique page for every `showpage` parameter. To crawlers, things would be much clearer if this contents would be served through separate physical files, for example: `showpage1.php`, `showpage2.php`, … and `showpage[N].php`. While this would surely be tedious to actually create such files, URL rewriting is the tool that allows the simulation of such structure.

A rewrite rule for the above example could simply translate `showpage[number].php` by filtering the number out, and placing it at the end of the internal URL `index.php?showpage=[number]`. Rewrite modules use regular expressions to implement these transformations. The regular expression rewrite rule for this example is:

A simple rewrite rule for the example from this paragraph:

```
RewriteRule ^showpage([0-9]+).php$ /index.php?showpage=$1
```

It could even be pretended that plain HTML files are used:

```
RewriteRule ^showpage([0-9]+).html$ /index.php?showpage=$1
```

Figure 12: URL rewrite example

This technique can be applied to solve a wide range of problems. Theoretically any problem that was earlier categorized as a 'requesting' problem can be solved by using URL rewrites to create (or rather pretend) crawler-friendly URL's. In addition, the regular expression format allows webmasters to stuff any words in the URL that will be ignored, thus opening doors for easy keyword presence in URLs.

Architecture

Another issue to review is the site depth. Some crawlers only follow links to a certain depth. In addition, documents found deep in a site, are likely to be regarded less important. Although search engines don't reveal their algorithms regarding crawl depth limitation, it can be expected that large sites are treated differently than smaller sites. By providing deep links or using sitemaps (previously described) the risk is decreased that documents are not indexed because they are too deep in a website.

Problems with information being hidden behind forms are another architectural issue that requires attention. As with site depth, it is important for the designers of a website to be aware of the issue.

Crawler detection

It can be readily agreed that crawlers have strict requirements on documents that they visit. While documents may be very valuable and interesting to regular visitors, crawlers may find several reasons to stop processing and indexing a document and thus stop it from reaching search engine indexes. From a theoretical point of view, one could try to detect crawler visits and serve crawlers different documents, which would typically be easier to read and index. Such an approach should focus on the question of how a crawler can be identified. It seems there are two ways to identify a crawler visit: by its name and by its behaviour.

Crawler identification by User Agent

Any visitor to a website presents the webserver with a user-agent string. For most regular internet users this will be a string that contains the browser and its version number. For crawlers however, it is their policy to identify themselves and include (contact) information just in case the crawler causes problems. This user-agent string can easily identify crawlers. The user-agent string is contained in the HTTP header information. It needs to be remarked that such a header can easily be spoofed and therefore can never be relied on. The user-agent string for GoogleBot looks like:

```
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
```

Crawler identification by behaviour

While crawlers don't necessarily behave differently from other users, they are known for accepting neither cookies nor sessions. Although the identification by user-agent is very reliable, crawlers have been identified for a long time by testing whether cookies or sessions are accepted. Chances are that if neither is accepted, webservers are dealing with a crawler.

3.3 The practice of SEO

3.3.1 Categorizing approaches

Intuitively a distinction can be made between strategies that are morally correct and strategies that obviously try to achieve their goals in a less ethical manner.[14] In the field of SEO a clear distinction is made between *whitehat* (ethically correct) and *blackhat* SEO.

Archer describes an interesting and slightly more nuanced distinction in his article[15] about the ethics in search engine optimization. He proposes three classes of optimization techniques: optimization, exaggeration and deception. These are described as follows:

- Search engine optimization: Ensuring that your code and content is appropriately organized and easy for search engines to interpret accurately.
- Search engine exaggeration: Reinforcing your desired keywords through frequent repetition, hidden keywords, etc.
- Search engine deception: Creating content, pages, etc., that aren't intended for human consumption, but are instead designed only to pull in search engine traffic.

The parallel with blackhat and whitehat SEO is clear to see. Optimization qualifies as whitehat, while deception qualifies as blackhat. Exaggeration covers the grey area in between.

3.3.2 Blackhat SEO techniques

In this paragraph, blackhat practices will be identified and reviewed. Although these are not useful strategies for the subject of this thesis, they highlight weaknesses of search engines and show us which paths to actively avoid.

Hidden text or hidden links

Since crawlers don't have any visual interpretation of a document on the internet, this fact can be easily abused by placing invisible text or links on pages. While users will not be able to see this content, crawlers do. This can be abused to achieve a high (false) keyword density or link to other documents that only crawlers will visit.

Cloaking

Some malicious websites attempt to serve entirely different pages to crawlers and users. This technique – called *cloaking* – tries to detect crawler visits and pretends to feature different content. When users click to indexed documents from the search engine results

[14] http://www.ahfx.net/weblog/55

[15] http://www.returnofdesign.com/81/ethical-seach-engine-optimization.html

pages, they see completely different content from what they expected. This technique not only directly deceives users, but also the rankings algorithms.

Redirects

Somewhat related to cloaking, but slightly less offensive, are abused redirects. Pages indexed on search engines may show as expected, but redirect to another page after a short time or - in the worst case - immediately.

Load pages with irrelevant words (spam)

Keywords are important in reaching a high ranking. Some websites try to reach higher rankings by bluntly showing a large amount of keywords all over a document. A technique referred to as *spam*, which shows not much creativity.

Multiple pages, subdomains, or domains with duplicate content

Google has a big interest in keeping its index clean. When many results point to exactly the same content, whether it be on the same site or on different sites, these results are not useful to users. Duplicate content may have many legal and plausible uses, such as: making a domain available both with and without the www prefix, an article being hosted on multiple sites or using multiple domain names for the same site for marketing purposes. However, search engines don't like duplicate content for a good reason. While an active use for deceitful SEO is not so obvious, search engines are known to penalize pages with duplicate content on the internet.

Link-farms

The importance of inbound links has been discussed with relation to PageRank. Link-farms are websites that exist for the sole reason of linking. Websites can freely add links to their pages on these link-farms, and in some cases after paying a fee. The effect is partly reduced by the changes to PageRank algorithms and its variations: the value of incoming links is determined by the PageRank of the site from which the link originates, and decreases with the amount of outgoing links from this page.

"Doorway" pages

Doorway pages can be defined as traffic drawing advertising pages for a website. They are not an integral part of the contents of a website, nor are they linked to by normal navigation. They are meant to be found by crawlers, and ranked high. Users that visit the doorway pages from the search engine results pages are expected to navigate from it to visit the rest of a site, because doorway pages itself typically do not provide any contents.

Shadow domains

One common scam is the creation of 'shadow' domains that funnel users to a site by using deceptive redirects. These shadow domains often will be owned by the SEO who claims to be working on a client's behalf. However, if the relationship sours, the SEO may point the domain to a different site, or even to a competitor's domain. If that happens, the client has paid to develop a competing site owned entirely by the SEO.[16]

3.3.3 Search engines and SEO

So what abuse are the large search engines actually fighting? And how are they fighting this abuse? This paragraph will answer these questions.

Search engines will try to fight all of the blackhat techniques listed before. It is easily seen that some of these strategies can be detected without much complexity, while others are much harder. This paragraph will shortly review detection of blackhat practices and discuss penalties and other solutions.

Keyword spam, redirects and cloaking

Keyword spam, redirects or cloaked pages cannot be hidden. On the contrary, it is all about visibility. These abuses are therefore easy to spot. However, detection is not as hard as is determining whether it is malicious or not. Algorithms need careful tweaking, but luckily this is exactly what search engines programmers have much experience in.

Identifying link-farms

A more complicated issue that has received attention in order to help search engines fight malicious SEO is detection and identification of *link-farm* or link-*spam*. In [14] as well as [33] techniques are described that are able to identify such pages. Such detection uses graph- and context-analysis and has been found to be successful.

Doorway pages

From the properties of doorway-pages it can quickly be understood that such pages can be easily identified. Two characteristic properties are uncommon for other documents: doorway pages are not linked to from normal navigation, and doorway pages show a significantly different part of a website.

[16] Explanation taken from http://www.appnetdesigns.com/searchenginesGoogle.htm

Penalties

Search engines can issue different penalties against a website, varying from engine to engine. These penalties have been listed by Green in [16] and will be summed up here:

- Removal from the search index. Also called the oblivion drop, this penalty removes (usually permanently) a website from the search index.
- SERPS Drop. This penalty results in a web page being permanently banned from the top 50 – 100 search results.
- PageRank Penalty. Google has been known to occasionally assign a permanent PageRank of 0 to websites that are found to be participating in linking schemes that are designed to artificially inflate PageRank.
- IP Ban. This penalty involves a search engine banning a specific IP address from even accessing their service. This type of penalty can be incurred if an IP address is found to be taking up an excessive amount of bandwidth on the search engine's servers. For example performing excessive rankings checks or using an application that is specifically advised against by the search engine.
- IP Block Ban. This penalty is the same as above except that the search engine bans an entire IP block (to the octet or "C" block) from using their services.
- Black Hat Penalty. This rare penalty involves a search engine banning an SEO company from accessing the search services as well as the permanent removal of all identified clients of the SEO from the search index.
- PR Parse Penalty: This penalty prevents a website from being able to pass PageRank to another website.

Green remarks that most penalties are applied for 3 months, 6 months or permanently and that some search engines can occasionally be persuaded to drop a penalty once the undesirable activity has been remedied.

There are many ways to deceive search engines and their crawlers. However, search engines are the ones that constitute the rules and have the last say. They are known to fight the abusive techniques. Therefore, malicious strategies can be expected to loose effectiveness in the long term. This is well summarized by the following quote from James Archer[17]: "It takes a lot of time and effort to 'stay ahead of the law' with deceptive SEO, however, and we've found that a long-term approach to SEO requires much less ongoing maintenance (and associated cost) — and it's frankly a lot more fun!"

[17] http://seo.phpmagazine.net/2005/09/ethical_search_engine_optimiza.html

3.4 Overview

As some of the strategies described in this chapter will be used for designing a solution, this chapter will conclude with an overview of all the strategies and techniques described, and rate them on several aspects.

A distinction needs to be made between techniques and tools. The techniques listed may be implemented in many different ways, using a variety of different tools. To give an example: using a tool such as URL rewrites, one could:

- reduce the number of query string parameters, by encoding them in the URL;
- create keyword presence in URLs;
- or redirect human users and crawlers to different pages.

Therefore, two figures are presented: one listing and rating the available techniques, and one listing and rating the available strategies.

On the very left is described the *type* of the approach, which determines its use. Although all types have a different application, they are listed together as they will be rated on the same scales. The ratings vary from none (-) to 5 stars (*****), where more stars are always a positive indication. The aspect named *simplicity* was used instead of its opposite *complexity* to maintain this *more-stars-is-better* indication. Simplicity thus stands for the transparency of a certain technique and its ease of implementation.

Although this chapter discussed several blackhat SEO techniques, the (ab)use of these is naturally undesirable. Apart from the penalties risked, there is typically a big need for maintenance, and much uncertainty. They are listed as a group in the strategies table for completeness.

Strategies

Type	Strategy	On/Off page	Effectiveness	Simplicity	User-friendliness	Proven techmology
Crawlability	Reduce query string parameters	On	****	****	*****	****
	Stop use of sessions and cookies	On	**	****	**	***
	Provide sitemaps	On	***	***	****	**
	Optimize architecture	On	*****	*	****	****
Ranking	Decrease perceived site depth	On	***	*	***	**
	Optimize keyword presence and placement	On	*****	*****	*****	*****
	Optimize keywords presence in URL (domain / filename)	On	*****	**	*****	*****
	Generate incoming links	Off	*****	N/A	N/A	*****
	Historical data	Off	****	N/A	N/A	**
	Clickthrough data	Off	***	N/A	N/A	*
	User ratings	Off	***	N/A	*****	*
Blackhat SEO techniques	(for ethical, security, economical and corporate reasons not suited, as the ratings indicate; refer to the techniques from paragraph 3.3.2)	On/Off	-	****	-	-

Table 2: SEO strategies overview and ratings

Techniques						
Type	Technique	On/Off page	Effectiveness	Simplicity	User-friendliness	Proven technmology
Crawler detection	Identification by user-agent	On	*****	****	N/A	*****
	Identification by behaviour	On	***	-	N/A	**
Crawler controlling	robots.txt	On	*****	**	*****	***
	Robots META tag	On	*****	****	*****	****
	Link exclusion (nofollow)	On	*****	*****	*****	**
	Technical barrier (e.g. javascript generated)	On	****	*	-	***
URL modification	URL rewriting	On	*****	-	*****	*****

Table 3: SEO techniques overview and ratings

4 Fredhopper Environment

In previous chapters the problem definition has been given, and theoretical backgrounds have been provided. This knowledge will help find a solution. However, the Fredhopper environment for which a solution needs to be found and implemented is very specific. It requires a carefully constructed solution, overcoming all predefined constraints.

In this chapter an extensive technical description will be given of this Fredhopper environment, which we should really call: the inner workings of Fredhopper Access Server and the environment of its implementation. These are in fact two different aspects defining the domain in which a solution needs to be found. Therefore, this chapter will first treat the Fredhopper Access Server itself, and discuss the possible implementations in client environments secondly.

The difficulty for many innovative products is that they include idea's that have not commonly used names. The following paragraph will provide explanations for many terms widely used by Fredhopper. The reader is informed of the glossary at the end of this thesis.

4.1 Fredhopper Access Server

4.1.1 FAS database model

Fredhopper has a generic model for cataloguing items into a structure of categories. At the top of the data-model are **universes**. While for many shops one universe will be enough, some will want to define a universe for e.g.: music, one for books and one for cosmetics. Each universe may contain a set of **categories**, children of a universe or of other categories, allowing the creation of a hierarchical category structure. While universes may not be stored in their own table, as can be seen in Figure 13, they are represented by top-level categories.

In each category **items** are placed. Items typically are the heart of the data, they are the products that need to be showcased. For each item additional information may be stored in separate tables: prices, availability, attributes and assets. Refer to Figure 13 again. As can be seen in the model, **prices** may be represented in different currencies and taxing is natively supported. **Availabilities** may be stored in advance for a specific period.

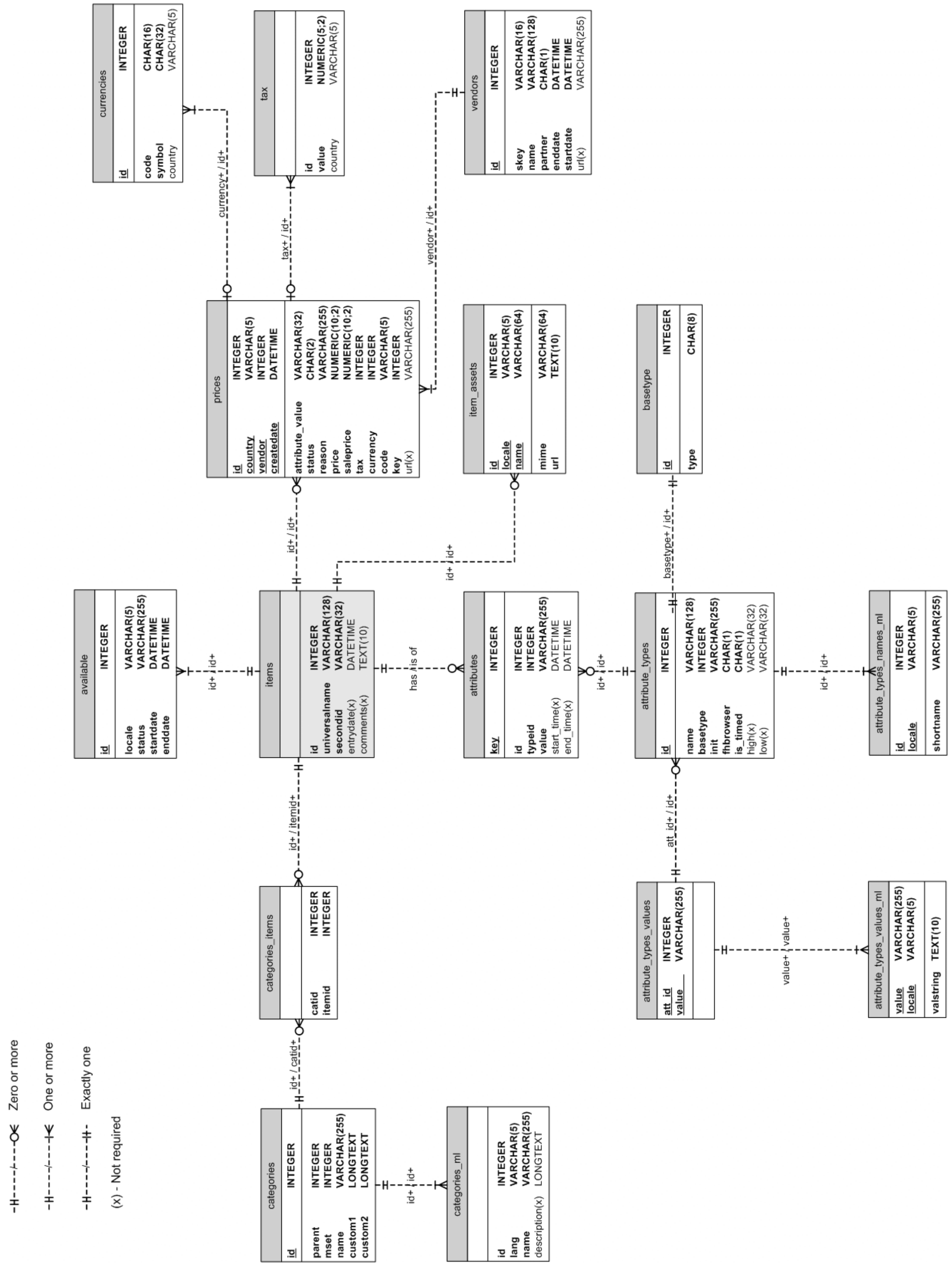


Figure 13: FAS catalogue database layout

Attributes and **assets** differ in the way they are treated: assets can only be searched; attributes may be used for navigation and filtering as well. While attributes are typically ordered fields, with numerical values or textual values from a predefined set, assets are less structured. They are the fields that items are usually not ordered by or filtered on. Usually long text values like an item description or history are stored as assets, while fields like length or colour are typically stored as attributes.

The structured nature of an attribute is reflected in the additional tables linked to attributes: *attribute_types*, *attribute_types_values* and *basetype*. Each attribute is stored as a certain attribute-type. Each attribute-type in its turn is of a certain basetype, which may be of the type: *integer*, *float*, *set*, *list*, *text* or *ref*. Each type has its own implicit search configuration. For *set* and *list* a predefined set of possible values must be given in *attribute_types_values*.

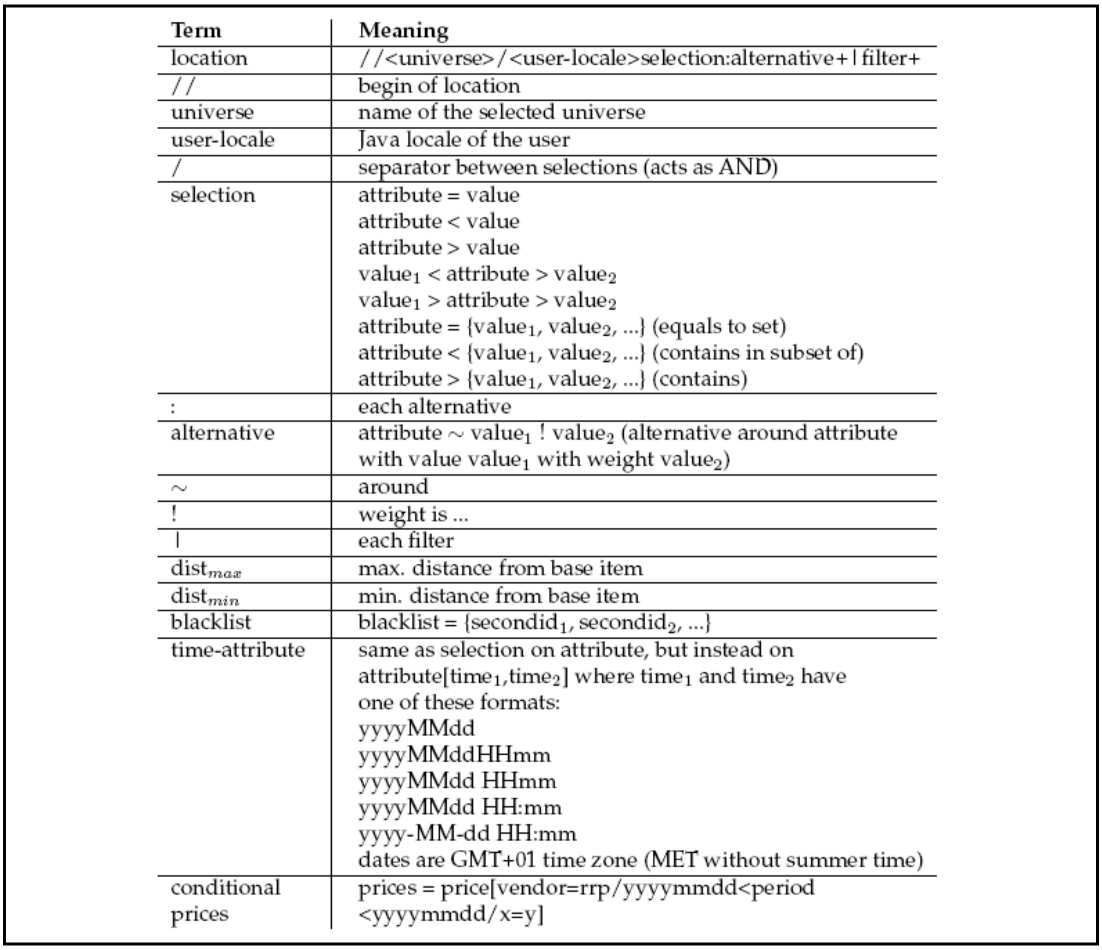
Lastly there are several tables with the suffix *_ml*. These contain multi-lingual versions of names and values stored in the related tables. Fredhopper has a strong focus on international products, clients and deployment.

4.1.2 Search & Navigation

Query string parameters

As mentioned earlier, FAS is stateless software. The advantage is a reduced strain on memory usage, especially when the server-load is high. The *dis*advantage however, is that FAS is not able to relate subsequent pageviews. Therefore, each page request needs to provide FAS with the full scope of environment and query variables. These are communicated through the URL as the query string.

The specific parameters communicated can be anywhere between none and many. As intuition expects, requesting a page from FAS without any additional parameters will take you to the main page. However, the powerful set of query string parameters allows the generation of any page, with any subset of items. The most important part of the query is the *location string*: it is the parameter that defines the domain of the subset of items that are requested. With the location string, one could for example request all items that have the colour red, are priced above $100, from the category women's clothing. A full definition of the location string syntax is given in Figure 14. All supported query string parameters are listed in Table 4.



Term	Meaning
location	//<universe>/<user-locale>selection:alternative+ | filter+
//	begin of location
universe	name of the selected universe
user-locale	Java locale of the user
/	separator between selections (acts as AND)
selection	attribute = value attribute < value attribute > value value$_1$ < attribute > value$_2$ value$_1$ > attribute > value$_2$ attribute = {value$_1$, value$_2$, ...} (equals to set) attribute < {value$_1$, value$_2$, ...} (contains in subset of) attribute > {value$_1$, value$_2$, ...} (contains)
:	each alternative
alternative	attribute $\sim$ value$_1$! value$_2$ (alternative around attribute with value value$_1$ with weight value$_2$)
$\sim$	around
!	weight is ...
|	each filter
dist$_{max}$	max. distance from base item
dist$_{min}$	min. distance from base item
blacklist	blacklist = {secondid$_1$, secondid$_2$, ...}
time-attribute	same as selection on attribute, but instead on attribute[time$_1$,time$_2$] where time$_1$ and time$_2$ have one of these formats: yyyyMMdd yyyyMMddHHmm yyyyMMdd HHmm yyyyMMdd HH:mm yyyy-MM-dd HH:mm dates are GMT+01 time zone (MET without summer time)
conditional prices	prices = price[vendor=rrp/yyyymmdd<period <yyyymmdd/x=y]

Figure 14: Location string syntax

fh_session	URL encoded location query //....	0..1	ID of session on Fredhopper server, can have value "unknown". If the fh_session parameter is specified in the url, it will be copied in this element.
fh_location	URL encoded location query //....	0..1	Location query
fh_secondid	secondid equal to secondid in items table	0..N	Specifies the secondid of an item to be desplayed in the detail or compare page. For the compare page, multiple fh_secondid parameters can be specified.
fh_view	view mode	0..1	One of the Fredhopper views: lister/summary/detail
fh_refview	previous view		Previous facet clicked by user (for reporting)
fh_reftheme	previous theme		Previous promotion clicked by user (for reporting)
fh_reffacet	previous facet	0..1	Previous facet clicked by user (for reporting)
fh_eds	ß	0..1	To check browser character encoding.
fh_search	URL encoded search term	0..1	Search terms
fh_search_pass	literal, near, and, synonym_and, partial_and, fuzzy_and,or, synonym_or, partial_or, fuzzy_or	0..1	Selects the specified search pass.
fh_search_field	assetname or attribute_type	0..1	Searches only the specified search field. Note that attribute_type or asset must be in the search index.
fh_start_index	Integer	0..1	Start item position in lister
fh_view_size	Integer ALL	0..1	Number of items to show in lister Show all items in lister
fh_sort_by	@<assetname> or attribute_type	0..1	Sort lister columns by asset or attribute_type. If asset not prefixed by @ sign then system first tries to find attribute_type with that name. If not found, it scans through assets.

fh_sort_order	-1 or 1	0..1	Forward or backward sorting of lister.
fh_usertype	client specific usertype as defined in the Business Manager	0..1	Usertype that can be used for usertype specific promotions and to have usertype specific reporting
fh_nolog	Do not write log line	0..1	Do not write a log line in the business log file when fh_nolog=1

Table 4: URL parameters used in FAS

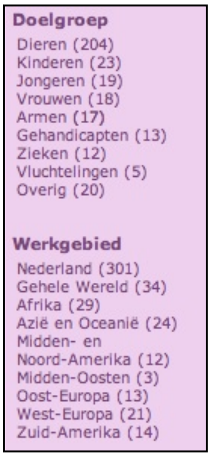
Breadcrumb

The location string is translated to a *breadcrumb* on every page. This allows users to quickly overview their navigated path, as well as step back to another point in this navigation history. Since FAS itself does not store this visitor information, all information is extracted from the breadcrumb.

Facets

Facets are the elements that implement most of the navigational functions of FAS. They are typically listed on the left side of the screen, in a vertical list. Facets are attribute or categories that may be further explored. For every facet the values are listed, commonly sorted on the amount of items in it. An example will be useful to visualize this abstract description:

Imagine a user that entered the category home appliances and navigated to refrigerators. On the left a list of facets is now shown, that provide several ways to explore the catalogue further: price, brand and volume. The facet price shows several price ranges and the number of items in it, e.g.: up to $100 (5 items), $100 to $150 (14 items), $150 to $200 (7 items), $200 to $250 (6 items), and above $250 (9 items). For a visual example, refer to Figure 15.



Doelgroep

Dieren (204)
Kinderen (23)
Jongeren (19)
Vrouwen (18)
Armen (17)
Gehandicapten (13)
Zieken (12)
Vluchtelingen (5)
Overig (20)

Werkgebied

Nederland (301)
Gehele Wereld (34)
Afrika (29)
Azië en Oceanië (24)
Midden- en
Noord-Amerika (12)
Midden-Oosten (3)
Oost-Europa (13)
West-Europa (21)
Zuid-Amerika (14)

Figure 15: Facets

Promotions

Promotions come in various forms and can be configured through the business manager. They can be used to highlight specific items, or to draw attention to an upcoming event by a visual banner. Promotions are configured to appear at certain places or times.

Page types

In FAS there are 5 basic page types. Every request may define a page type, specifying a desired page formatting, with its related promotions and functionality. If no specific page type is requested, FAS will automatically determine a type, based on the navigated path and the size of the active subset of items.

Home

The page where all users typically start. When requested, this page usually returns an overview of the top-level categories, promotions, or the most popular items in the catalogue.

Summary

Users will arrive at summary pages when they navigate deeper into the catalogue. Summary pages typically feature location related promotions, and top-level facets.

Lister

Having navigated deep into the catalogue, the user will now be exposed to the full list of items in the active subset of the catalogue. Lister pages may include promotions, and show every item and some of its attributes. Lister pages are usually paginated and will show facets that can even further refine the navigation.

Detail

A detail page highlights one single item. Full details on attributes are given and usually a large sized image is shown.

Compare

The compare page is a page slightly out of the typical navigational route. Compare pages show several items, with their attributes lined up to allow easy comparing.

4.1.3 FAS Core

The core of FAS consists of several conceptually distinguishable parts. For a better understanding of the software and the context of this thesis these will be shortly discussed.

The **data storage manager** maintains the search-tree. As FAS starts up the catalogue database is read, and post-processed, which allows the creation of additional, virtual data (a process called *enriching*). The result is kept in memory as a hypercube to allow multi-dimensional searching and selecting. During runtime, the information is exclusively read from the in-memory representation of this component, never directly from the Fredhopper catalogue storage database.

At the very heart of the application is the **query engine**. It is the element that returns a list of items for each request. Incoming requests (see next paragraph) are mapped against the search-tree, filtered on availability and sorted as requested.

The **breadcrumb module** generates the hierarchical link of the actual browsing position, e.g. *home* > *clothes* > *summer* > *t-shirts*. Users get a better understanding of their browsing location with breadcrumbs, and can browse up on their chosen path.

The **page XML** is a collection of Java Server Pages and acts as the controller. It uses the other components to generate XML documents, which are delivered for client requests. XML is not only generated in page XML, but other components also create XML blocks, which are glued together in page XML.

The **facet map generator** uses the results of the query engine and creates the depending facets. Again, the hypercube is used to find possible browsing paths. The promotion module supplies the facet map generator with additional promotional offers called *themes*.

The **promotion module** manages additional advertisement information. Items can be boosted or blocked in the search engine, e.g. to support own- labels. Depending on the browsing position, related themes can be shown. For example, if a user browses on a website of a travel agency to the slice hotels in Paris, the site should show special offers for weekend trips. This reflects the fact that big cities are popular for short holidays.

The **item detail module** provides information about one selected item: attributes, assets, and conditional prices. This component is needed to generate item detail pages and compare pages.

4.1.4 Platform

FAS is a J2EE application and therefore runs on a J2EE compatible application server like Orion or JBoss. After many FAS implementations over the last few years, JBoss is now the platform of choice for all Fredhopper clients.

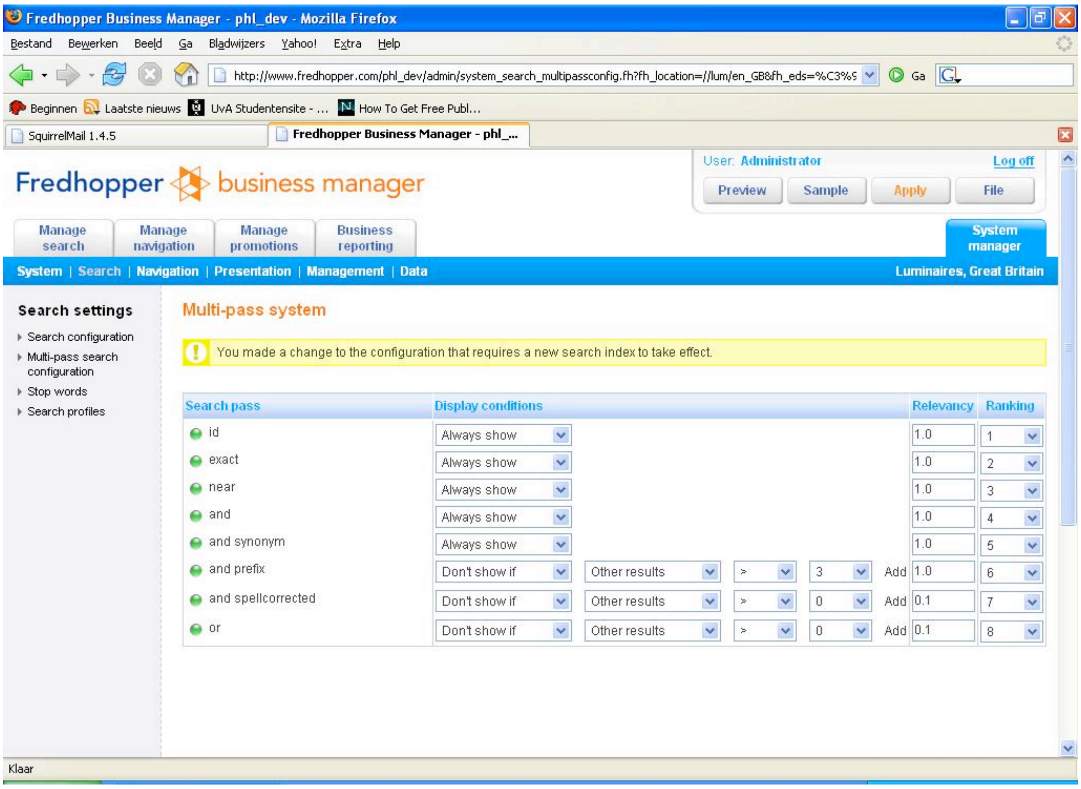
FAS implements JavaServer Pages (JSP), which are served using the Tomcat platform, included in the JBoss application server. All output generated by the JSP files is in XML.

4.1.5 Business Manager

The Business Manager is an administration tool included in FAS. It allows clients to configure many settings of FAS, like the previously mentioned promotion configuration. In addition, users can:

- configure the search; this includes the weighing and in-/exclusion of attributes;
- configure promotions;
- configure synonyms, and redirects for the search;

- configure the facets; this includes the depth at which they appear and ordering of values;
- configure the page flow; when to choose for specific page types when not defined;
- display view statistics;
- display conversion rates.


Fredhopper Business Manager - phl_dev - Mozilla Firefox
Bestand Bewerken Beeld Ga Bladwijzers Yahoo! Extra Help
http://www.fredhopper.com/phl_dev/admin/system_search_multipassconfig.fh?fh_location=//lum/en_GB&fh_eds=%C3%
Ga
Beginnen Laatste nieuws UvA Studentensite - ... How To Get Free Publ...
SquirrelMail 1.4.5
Fredhopper Business Manager - phl_...
Fredhopper business manager
User: Administrator
Log off
Preview
Sample
Apply
File
Manage search
Manage navigation
Manage promotions
Business reporting
System manager
System | Search | Navigation | Presentation | Management | Data
Luminaires, Great Britain
Search settings
Search configuration
Multi-pass search configuration
Stop words
Search profiles
Klaar

Multi-pass system

You made a change to the configuration that requires a new search index to take effect.

Search pass	Display conditions					Relevancy	Ranking
id	Always show					1.0	1
exact	Always show					1.0	2
near	Always show					1.0	3
and	Always show					1.0	4
and synonym	Always show					1.0	5
and prefix	Don't show if	Other results	>	3	Add	1.0	6
and spellcorrected	Don't show if	Other results	>	0	Add	0.1	7
or	Don't show if	Other results	>	0	Add	0.1	8

Figure 16: FAS Fredhopper Business Manager

4.2 Client implementations and environments

4.2.1 Front-end integration architecture

As long as items need only be navigated and searched, visitors could directly browse FAS. It would only require an XSLT stylesheet definition, to serve the desired layout. All required query parameters would be provided by the server, and could easily be added to URLs for continuous browsing with the illusion of states (remember, FAS is stateless, and every request is a separate query – no relation between consecutive queries is assumed, while they're known to exist).

However, this would allow for no additional functionality, which is often desired, if not the sole purpose of a Fredhopper implementation: FAS is commonly used as an online catalogue for shopping. Unless used by a museum or another type of client that merely wants to present its catalogue, FAS is usually integrated in a client front-end. This allows for integration of functionalities like a shopping-basket. We can distinguish open and enclosed integration.

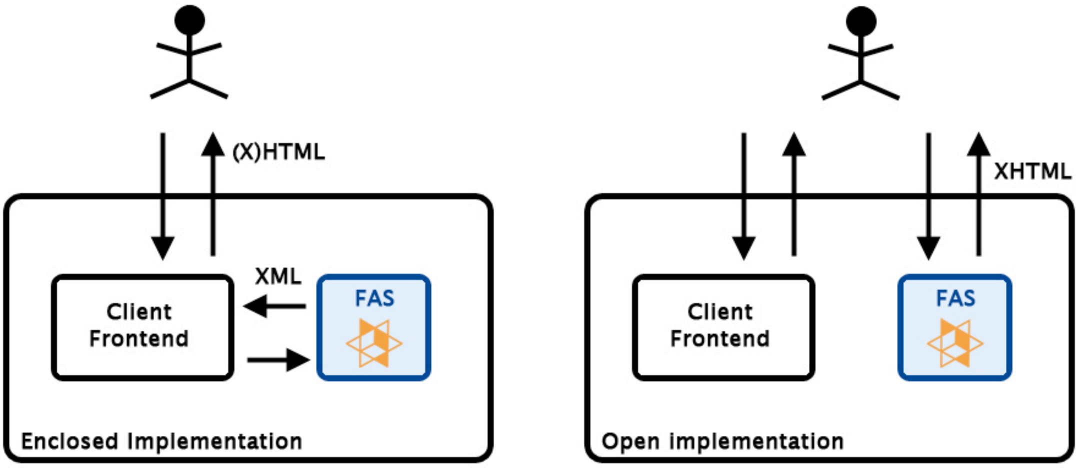


Figure 17: Enclosed vs Open integration

Open integration is easier for clients, since they can just install Fredhopper as is, and link for example to their own shopping system by including links to their own front-end in the output generated by Fredhopper. Typically, Fredhopper's integrated XSLT processor is used to produce ready-to-serve XHTML output.

Enclosed integration is usually the integration of choice in larger-scale projects, websites that integrate many different techniques and services. It helps maintain a clear overview of the architecture of a website and keeps control centralized. In these cases the Fredhopper XML output is processed by the client's front-end that will ultimately serve it to the visitor.

4.2.2 Databases

Typically, clients choosing to use FAS already have their own databases running. To run FAS however, they will need to serve the contents from a Fredhopper Catalogue database. Usually the information in a client's database is perfectly suited to present with FAS, but is not yet in the Fredhopper catalogue database format. Mapping it onto this format is usually the most complicated part of any FAS implementation.

To optimize flexibility, clients often choose to continue using their own database, and have the Fredhopper database be generated regularly from this database. There are several practiced techniques to achieve this. The most straightforward is a SQL script which is run at certain times or intervals. Other approaches work over a layer of XML: the client database is exported as XML and imported in Fredhopper format in the catalogue database format.

Whichever way the Fredhopper database is filled, it will always be merely a snapshot, unless changes to the original database are also translated and maintained in the Fredhopper database. This additional work would not pay off though, since the database is only read as FAS starts up. While the new generation of FAS allows dynamic updating of the data in memory, the current version does not have this functionality.

4.2.3 System architectures

Most Fredhopper clients will choose to implement FAS in a bigger system environment. While most websites run on one shared or dedicated server, this would pose to big a risk to businesses that rely on their online presence for revenue. Most clients therefore integrate FAS in a server architecture that includes load balancers, firewalls, one or more database servers, several (redundant) production (FAS) servers, a FAS test server and – depending on the integration – servers for their own front-end. Backup servers are typically placed as redundant productive servers, as they may help reduce load instead of catching dust. A typical example will be given by Figure 22 in chapter 6 (page 63).

5 Solution model

This chapter will highlight the issues with FAS regarding search engines and review several considerations and solution alternatives.

5.1 Domain of solution

It is important to realize the differences between previously mentioned open and closed integrations. While closed integrations are completely integrated into a client's own front-end, open integrations directly expose an end-user to the browsing and navigational structure of FAS. As already described in the problem definition, this thesis deals with the SEO issues for open integrations.

Furthermore, from Table 1 in paragraph 3.2.1 we learn that SEO can be practiced from several different angles. The domain of the solution that is the subject of this thesis is a subset of this domain. While *off-page* optimization may achieve as much as *on-page* optimization, it is more a subject of PR and marketing. The problem definition clearly states that a solution is desired on the product side: a solution that can be integrated or connected to the current product. This narrows down the domain in another dimensions as well: it is expected that the solution helps achieve better visibility and rankings in the *organic* results. This leaves ad-programs and other forms of *paid* optimization out. Regarding the 3rd and last dimension, that of *crawlability* versus *keyword* optimization, it must be realized that a solution must provide optimization for both.

In summary, the desired solution provides: a **technical tool** that **cooperates with FAS** to help **open integrations** improve their **ranking** and **crawlability**.

5.2 Identifying issues

From chapter 3 we learned about the relevant SEO problems regarding the Fredhopper Access Server. After the extensive review of the product itself in the previous chapter, this paragraph will list the problems that FAS poses to search engines, and where optimization is required.

5.2.1 Site structure

Near infinite possible paths

FAS creates no physical site structure; all pages are generated on demand. In fact, FAS does only provide a few physical JSP pages that generate every page an end-user will see. Every possible subset of items may be presented for a specific set of parameters. Naturally many of these generated pages contain redundant information; they may even be semantically identical as the same query may be formulated in different ways. For example, users browsing the category "cars" and choose a price range of €2000-€3000 see the same subset of items as users choosing the same price range and choosing the category "cars" afterwards. However, since the order of these selections differs, both pages have a different location string. The possible combinations of location strings depend on the number of facets and items in the database; in practice they are therefore near endless. This navigational structure is visualized in the simplified schematic model Figure 18. Every path in this graph represents a possible navigational route through FAS.

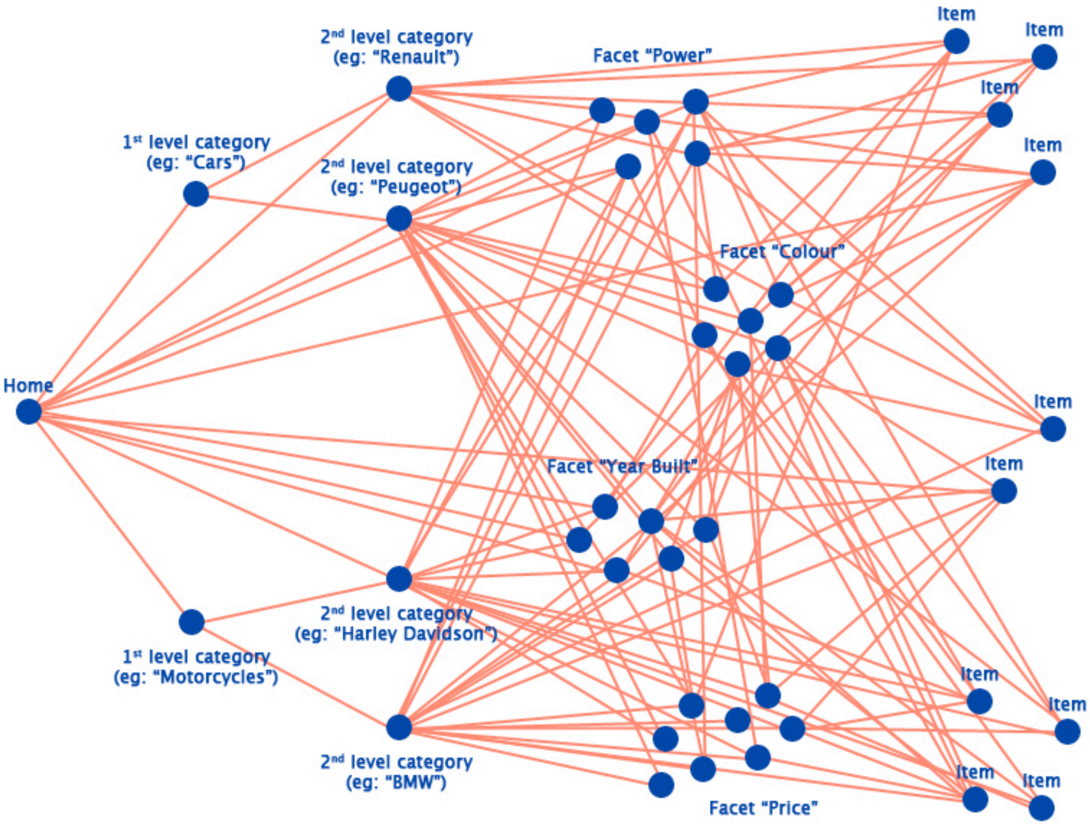

Home
1st level category (eg: "Cars")
1st level category (eg: "Motorcycles")
2nd level category (eg: "Renault")
2nd level category (eg: "Peugeot")
2nd level category (eg: "Harley Davidson")
2nd level category (eg: "BMW")
Facet "Power"
Facet "Colour"
Facet "Year Built"
Facet "Price"
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item

Figure 18: Modelled navigational graph of a FAS implementation

Detail pages

It is an important realization at this stage that the pages that are ultimately desired to end up in the search engine indexes are the *detail pages*. These can be considered end nodes in the tree or graph that represents the site structure. Refer to the "item" nodes in Figure 18. Detail pages display detailed product information. Any visitor interested in a specific (type of) product will ultimately make its way to the detail pages for detailed information, and possibly to purchase the product. These are the pages where we want to direct users from search engines; these are the pages that are semantically unique.

However, the path to these detail pages may be long. Depending on the number of steps taken from the main FAS page. In addition, those detail pages that appear in a paginated list of products (FAS' lister pages) may only be reached after clicking through several pages. And, as is elaborately explained in the previous paragraph, the path to a detail page is not unique. There are multiple ways to reach a product. The perceived depth of the documents that really matter is therefore far from optimal. We learned from chapter 3 that this perceived depth does influence the perceived importance of documents.

5.2.2 URL parameters

All navigation in FAS is controlled by the information transferred through URL parameters. For a complete list of these parameters, refer back to Table 4 (page 49). The problems that these parameters pose to crawlers are obvious. However, this issue will be hard to solve, as FAS cannot do without them. Naturally, the reporting parameters are not essential, although their functionality is on the long term. However, the location string *is* essential, as it tells FAS what subset of items to retrieve and display.

So, while it is true that not all are required to navigate FAS, at least some are – and as it is not yet understood how much or what type of parameters crawlers avoid, it should be preferred to not use them at all. Leaving some out to potentially increase the crawlability of a site is not an acceptable long-term solution.

Anticipating the discussion of possible solutions later in this chapter, the technique of URL rewriting looks promising. Using this, there might be possibilities to encode the parameters within the URL.

5.2.3 Keyword optimization

There do currently not exist any problems caused by documents that are not keyword optimized. Not having documents optimized must however be regarded as a missed opportunity. It was stressed in chapter 3, that keyword presence and presentation has a considerable impact on the ranking algorithms. So, while keywords do not make documents more visible, they do improve their ranking for search engine results. As the next part of this chapter will describe the actual design of a solution, the possibilities of keyword optimization are included in this design.

5.3 Solution Design

5.3.1 Design process

There are three aspects to the situation that require the modelling of a solution, in parallel with the identified issues: URL parameters, site structure and keywords.

The modelling and analysis of different heuristics that will follow, is presented in an evolving form. For all three aspects an initial approach is formulated based on the strategies and techniques reviewed in chapter 3. Reviewing this initial approach, several obstacles are identified. These are then addressed under a new header, presenting a next step in evolution of a solution model.

This approach is particularly present in the first two paragraphs about URL parameters and site structures. The third paragraph on keywords, is significantly shorted since its subject is more trivial and design heuristics are readily available from the contents of chapter 3.

5.3.2 URL parameters

This paragraph will investigate possible solutions that help remove parameters from the Fredhopper URLs. Since it is not possible to navigate FAS without any contextual information transferred, the information contained within the parameters needs to be communicated in a different way. Much of this information could be maintained in a session, although FAS was consciously designed without session support. However, building a separate session management system around FAS is like building a bedroom separate from the house in the garden. If sessions are the solution, this will need to be implemented in FAS itself. Still, as discussed in chapter 3, sessions do pose a serious problem for crawlers as well. And even though sessions do remove the need to communicate the full context and query for each page, they still need to be updated with respect to the user's navigation.

It seems the only possibility is to encode this information is in the URL itself, as a virtual path. To visualize this approach, imagine a document `index.jsp` that requires a parameter `colour` to be communicated for every view. Now, we could for example simulate a virtual path `index.php/purple` that is internally processed as `index.jsp?colour=purple`. This practice of parameter encoding requires URL rewriting to translate such requests.

Full parameter encoding

Paragraph 4.1.2 discussed the extensive list of URL parameters that FAS uses. Although FAS does not use all these parameters for every request, it does typically use a large subset of them. For example, review the following request URL:

```
http://zoekmachine.donateursvereniging.nl/goededoel/demo/index.fh?fh_sort_by=doelgro
ep&fh_refpath=facet_2&fh_reffacet=_doelgroep&layout=demo&fh_location=//donateursvere
niging/nl_NL/_doelgroep%3E{Kinderen}/_doelgroep%3E{Vrouwen}&fh_start_index=15#top
```

This request specifies 6 of the URL parameters: *fh_sort_by*, *fh_refpath*, *fh_reffacet*, *layout*, *fh_location* and *fh_start_index*. These are by internet standards separated by ampersands (&) and both parameters and their values are URL encoded. This URL encoding encodes special characters in %XX values.

Common techniques that encode parameters in the URL, simply transform the query string as a whole into a path. For example: `index.jsp?colour=purple&start=15&end=50` could be simply be literally encoded as either `index.jsp/colour=purple/start=15/end=50` or `index.jsp/colour/purple/start/15/end/50`. To crawlers these last two paths obviously look much more agreeable that the first.

There is however one serious issue with this technique when applied to FAS. The parameters may appear in any order in the URL. This means for every request with `N` parameters there are `N!` possible encoded paths: semantically identical, visually unique. The problem here is that it is not desirable to have all these semantically identical pages appear in the index of search engines (as their contents will be the same!). When we add to this the same ordering issue that appears in the location string (refer to paragraph 5.2.1 and Figure 18), the risk of redundancy is clear: for every `N` parameters of which 1 always is the location string containing `L` steps, there are `N! * L!` encoded paths possible.

Essential parameter encoding

An alternative to the encoding of all parameters is the encoding of just those parameters that actually navigate to the desired subset of items; showing crawlers the items of the current location is all that matters in the end. As can be seen in the example URL form the previous paragraph, most of the parameters present in this URL do not actually specify what is to be displayed. There are many parameters involved that are used for reporting and statistical purposes (*fh_refpath* and *fh_reffacet*). Additionally there are some parameters that specify in what order and what part of the subset to display (*fh_sort_by* and *fh_start_index*). There is even a parameter that specifies what set of XSLT stylesheets to use (*layout*). As has been explained in chapter 0, the only parameter that actually defines what subset of items was requested, is the *location string*, specifying a location in the catalogue through the parameter *fh_location*.

For a solution it may be considered to only actually encode this location string in the URL. However, the location string for `L` queries can take on `L!` different forms. So there will never be a mapping from location string to URL that identically encodes location strings defining the same subset - simply because a mapping back will not be able to restore the location string to its original form. Review Figure 19 for a visualization of this situation:

```
Encoding a location string with 3 query parameters

       Location_String(Qa,Qb,Qc)
       Location_String(Qa,Qc,Qb)
       Location_String(Qb,Qa,Qc)     →      URL (Qa,Qb,Qc)
       Location_String(Qb,Qc,Qa)
       Location_String(Qc,Qa,Qb)
       Location_String(Qc,Qb,Qa)



Decoding the query parameters back: in what order?

       URL (Qa,Qb,Qc)                →      Location_String(Qa,Qb,Qc)  ?
```

Figure 19: Visualizing the impossibility of a 1 to 1 generic location string encoding

It is useful now to peek ahead a little. Later in this chapter a technique is revealed that might help us to define what site structure crawlers perceive. This technique would allow us to define a univocal mapping for just the location string as required, by knowing and even choosing in advance the possible location string values that will be presented.

Essential parameter encoding: detail-pages

The detail pages allow an easier solution. All information a detail page really needs to be generated is the *secondid* of the item it displays. Although a location string is present in the detail URLs in current FAS implementations, this is only of use to human users that may navigate further after visiting a detail page. As we will see later, this is of no importance to crawlers, since they are expected to crawl no further from a detail-page. This allows the solution to simply encode the *secondid* in the URL. Naturally, this id is unique for every item

Selective parameter showing

Thus it seems that there exists a promising solution to circumvent the parameter problem. Crawlers may find all contents we want them to find, independent of parameters. By browsing the URLs with encoded location information, they will be exposed to every part of the catalogue.

However, FAS does rely on parameters for many other purposes, as was discussed previously. Apart from the valuable reporting and statistics information, navigation experience is also dependent on the parameters: the sort order and page sizes, for example, are transferred through the query string. Clearly, FAS relies on its parameters for any human visitor, while crawlers can be served predefined views without trouble.

This gives birth to the next consideration: wouldn't it be possible to serve both crawlers and users the encoded URLs, and include the parameters only for human visitors? This would result in both crawlers and users visiting the same real or virtual physical files; the human visitor however transfers additional information through parameters, which is not required for crawlers. For two examples, see Figure 20.

Users see	Crawlers see
/Price=50-100/index.jsp *?fh_start_index=70&…*	**/Price=50-100/index.jsp**
/Women/Lingerie/index.jsp *?fh_sort_by=price&…*	**/Women/Lingerie/index.jsp**

Figure 20: Example of location string URL encoding with human/crawler distinction

Chapter 3 discussed the detection of crawlers and has shown this needs not be a complicated procedure. However, it was also stressed that search engines don't like to be fooled. Consistently serving their crawlers different pages than users for the same URL will result in some form of punishment, as it is considered a bad practice. Fortunately for our solution, the parameters might only slightly alter the contents of the delivered page (in comparison with the naked URL).

5.3.3 Site structure solution

The reason of the complexity of the location string simply is the navigational freedom that FAS offers to its users. They can navigate from any point in any direction. These directions are configured as *categories* and *facets* as was explained in chapter 0. Along the way, users will encounter *summary-*, *lister-* and *detail*-pages.

Since the ultimate goal is to showcase specific items and provoke action, detail-pages can be considered the endpoints in the navigational structure – even though further navigation usually will be possible. This is also visualized in Figure 18. As paragraph 5.2.1 concluded: our solution should focus on the inclusion of detail-pages in search engines. Therefore it should make all detail pages appear after as few navigational steps as possible, to minimize the perceived site depth, because this perceived site depth is an important ranking factor as we saw in chapter 3.

Site structure simulation

So, it is required that we force crawlers over a specific path on a FAS implementation to allow the encoding of location strings in the URL: as long as we know in which forms the location strings will appear, it will be theoretically possible to find a 1-to-1 mapping. It is also required that the path we show to crawlers always ends with a detail page. This prevents crawlers from getting stuck in loops or finding identical pages with different URLs.

The techniques investigated in chapter 3 provide the means to actually fulfil these requirements. Although we should not fool crawlers and risk penalties, we can without danger tell them what links to follow, and which not. This way, our solution will be able to force a certain view on the FAS site structure, without changing anything about this structure in FAS itself. The two most important tools in this respect are:

- Robots META tags: allowing us to define within each individual document whether crawlers may index it (or not) and whether crawlers may follow the links within this document (or not).
- Link attribute `rel="nofollow"`: allowing us to specify for each individual link that crawlers must not follow it.

Category only view

The choice left to make, is what site structure to force upon the crawlers. There are many navigational elements to choose from, both categories and facets. However, there are several reasons to choose a category-only approach.

First of all, categories are **always** collectively exhaustive. This means **all** items in the catalogue will be included somewhere in the view presented. Naturally, all facets combined should be collectively exhaustive as well, but this would require just the multi-dimensional location string encoding that we rejected previously.

Secondly, categories are to a very high degree mutually exclusive. This means redundancy of contents will be low. With any Fredhopper implementation there are usually only very few items that appear in multiple categories, if any at all.

Third, categories are – in contrast to specific facets – present in any implementation, which makes a solution more generic and thus re-usable, which is a big advantage if no requirement.

The result of such an implementation will force a view on the FAS catalogue that goes from the homepage, to all category and subcategory lister-pages, and ultimately ends with a detail-page for every product which its own unique URL. Such a structure is modelled in Figure 21.

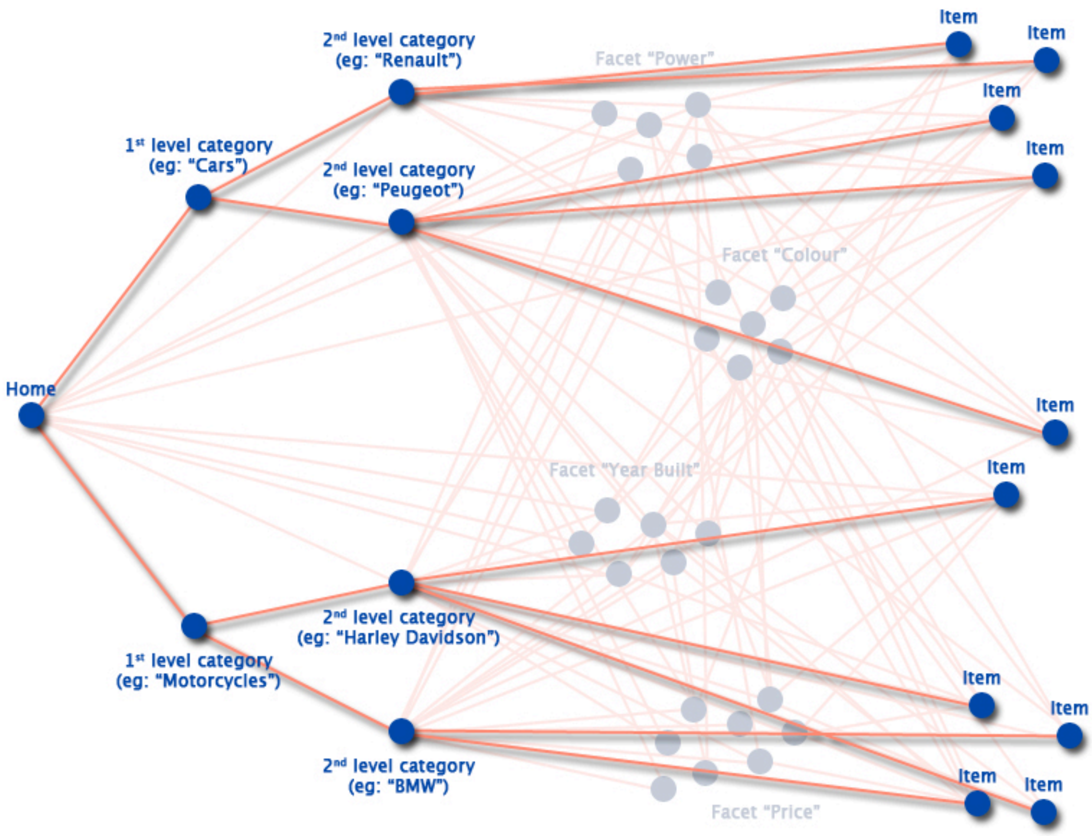

Item
Item
2nd level category (eg: "Renault")
Facet "Power"
Item
1st level category (eg: "Cars")
2nd level category (eg: "Peugeot")
Item
Facet "Colour"
Home
Item
Facet "Year Built"
Item
2nd level category (eg: "Harley Davidson")
1st level category (eg: "Motorcycles")
Item
Item
2nd level category (eg: "BMW")
Item
Item
Facet "Price"

Figure 21: Model of the forced crawler view on navigational structure

Variable pagination size

A small, but related issue that requires attention with such implementation is the pagination of the lister-pages. After browsing categories to a certain depth, crawlers will be shown a list of items. However these lists are divided in multiple pages. To optimize the number of clicks required to reach every item, and thus minimize perceived site depth, it should be tried to show crawlers a considerable larger selection of items on each page.

Search engines are known to be less trustful towards pages with too many links. Although it is nowhere defined what *many* or *too many* means in numerical sense, it should be considered. In practice, sites documents with hundreds of links have been shown to be excluded, while many accepted pages exist that feature up to one hundred links unpunished.

5.3.4 Keyword considerations

If everything previously described works out well, FAS pages will start to be included in the search engine indexes. An important issue left now, is to make sure they have every opportunity to appear high in the results.

Although big parts of the ranking algorithms regard external factors like inbound links and historical data, we should make sure that the FAS documents have every potential in them to rank high for their keywords. This paragraph dedicates attention to the possibilities of keyword optimization, both what's in the name as within the document.

Keyword presence in URLs

While it is relatively easy to stuff the contents of a document with keywords, changing the document names for a ton of pages is already harder. Search engines are known to give considerable weight to words contained in the physical path of a page.

Since FAS does not feature any physical documents, no advantage is taken here. Crawlers are only shown the basic JSP pages: `index.fh` and `detail.fh`. However, since our solution will use URL rewriting already, we may as well use this to allow some keyword presence in the URLs.

An example: instead of the default FAS detail URL

```
http://fas.quelle.fr/detail.fh?fh_secondid=29247&fh_refpath=facet_1&fh_reffacet=_cat
egorie&layout=demo&fh_location=//boutique/FR_fr/_marque%3E{Sony}/_categorie%3E{Jeux}
&&fh_ref=lister#top
```

which will be shown to crawlers (with our designed solution) as:

```
http://fas.quelle.fr/detail.fh/id=29247/
```

might as well pretend to have the following URL:

```
http://fas.quelle.fr/d-29247-le_jeu_pro_evolution_soccer_5.html
```

The last URL is much more convenient, features essential keywords, looks like a physical file, that is even located in the root of this site. According to what was accounted for in chapter 3, crawlers will find it easier to index such documents, and the ranking algorithms should generate a higher ranking.

Keyword stuffing

Although trivial and easy to implement, keyword optimization within a document should receive attention from any SEO solution. Chapter 3 discussed which placements are most valuable in this respect. For the solution designed in this thesis, keywords should be placed:

- in the page title;
- enclosed in valuable tags within the document (`<h1>` and `<b>` for example);
- repeatedly within the document (several times in different places, at least once near the top of the document).

Since detail pages are the main focus of a SEO solution for FAS, the generated detail pages should receive this treatment. Although the implementation of such optimization is simple HTML formatting, this specific outline of the actions required is presented for completeness.

5.4 Summary

This chapter discussed the Fredhopper specific design of a solution to tackle the issues identified with its implementation. These issues can be grouped in three categories, which have been separately discussed, and for each a solution was found after some considerations:

- parametric browsing;
- perceived site structure;
- keyword use.

Parametric browsing will be solved by encoding the essential *location string* within the URL, and leaving all other parameters out. To allow univocal encoding and decoding of this *location string* its possible forms have been clearly defined, cooperating closely with the solution to site structure issues. Using crawler detection by user-agent, a distinction is made between crawler and human visits. Human visitors will still be presented all parameters to maintain the continuity of the environment. The decoding of encoded parameters will be done through URL rewrites.

To solve the issue with the perceived site structure, which exposed crawlers to the possibility of infinite paths, a specific view was forced to these automated visitors. Using techniques of link hiding and META tags, a specific path was revealed to crawlers, not allowing them to deviate. This way, we automatically limit the combinatorial possibilities of the *location string*, required for effective *locating string* encoding.

To use the potential of keywords for ranking, several measures have been proposed. Apart from the more trivial strategic keywords placement and presentation within documents (pagetitle, header tags, and with an increased frequency), we will use the technique of URL rewrites, which will be present in this solution for the sake of parameter decoding, to insert keywords within the URLs. Although these convey no information internally, crawlers will perceive a higher relevance of these documents for the presented keywords.

6 Implementation

Fredhopper offered the opportunity to implement the solution in a real case. One of their clients, Quelle France, wanted to see what could be achieved by SEO, and asked for integration in their new website. This chapter will discuss the technical details of the SEO solution created.

6.1 Quelle France

For a proper understanding of the relevance of this case, this paragraph will discuss some characteristics of the Quelle France case, and review its representativity.

6.1.1 Representativity

Among open Fredhopper implementations, the only real difference is content and the presentation of this content. The URLs, underlying techniques, XHTML generation, navigational structure, and the related issues are all the same. The actual names of the facets, items, categories are different, as are the promotions, and the XHTML presentation. This means that any implemented solution is highly generic, and applicable (with some minor modifications) to all open Fredhopper implementations.

6.1.2 Case characteristics

Some interesting characteristics about Quelle France may help visualize the case. Quelle France is an online shop for anything from clothing to electronics, from cosmetics to furniture. From the launch in late March 2006 up to this very moment, the website has been online[18] using FAS and the solution described in this chapter. The Quelle France office is located near Paris, France.

6.2 Implementation process

The remainder of this chapter will discuss the implementation of the physical (hardware) configuration (paragraph 6.3), and the software and code required for the rewriting module (paragraph 6.4) and the front-end XSLT (paragraph 6.5).

6.2.1 Hardware

With regards to the hardware, its setup was largely determined by the client's existing architecture, the FAS server configuration and the requirements of the solution. The actual considerations that contributed to the final setup are described in the following paragraph.

[18] http://www.quelle.fr/

6.2.2 Software

Regarding the software, the implementation naturally evolved over time. Although an initial format could be readily deduced from the modelled solution, there were much additions, corrections and new insights over time. The implementation process was thus strongly **iterative** in nature, as expected: the model provided an initial setup, which was known to require additional efforts to adapt it to a specific configuration.

The specifics of the evolution of the software have not been monitored during implementation. However, the incremental approach ultimately provided an effective way of creating an implementation that covered all unforeseen cases with extensive testing after the creation of each new version. This testing consisted of:

- a review of all the URLs of generated hyperlinks in the output;
- a review of all the attributes of generated hyperlinks in the output;
- verification of the rewrite rules by following all generated hyperlinks in the output;
- verification of the rewrite rules by following all relatively linked externals in the output;
- a review of META tags for all Fredhopper pagetypes.

6.3 Physical configuration

6.3.1 Global layout

FAS is usually installed in an environment which consists of multiple servers. While many websites run on a single shared or dedicated server, most Fredhopper clients depend on their online sales channel for business, and downtime or a slow website may seriously harm turnover, thus ultimately costing money. In such environments, one or more production servers may be running in parallel to allow load balancing and have backup systems readily available. A clear visualization is given in Figure 22, showing the designed Quelle France server park architecture.

In such architectures it greatly helps reduce complexity and risks to have each FAS server available individually with full functionality. For this reason it was an important prerequisite for the SEO solution to implement it around FAS and have it deployable as one package.

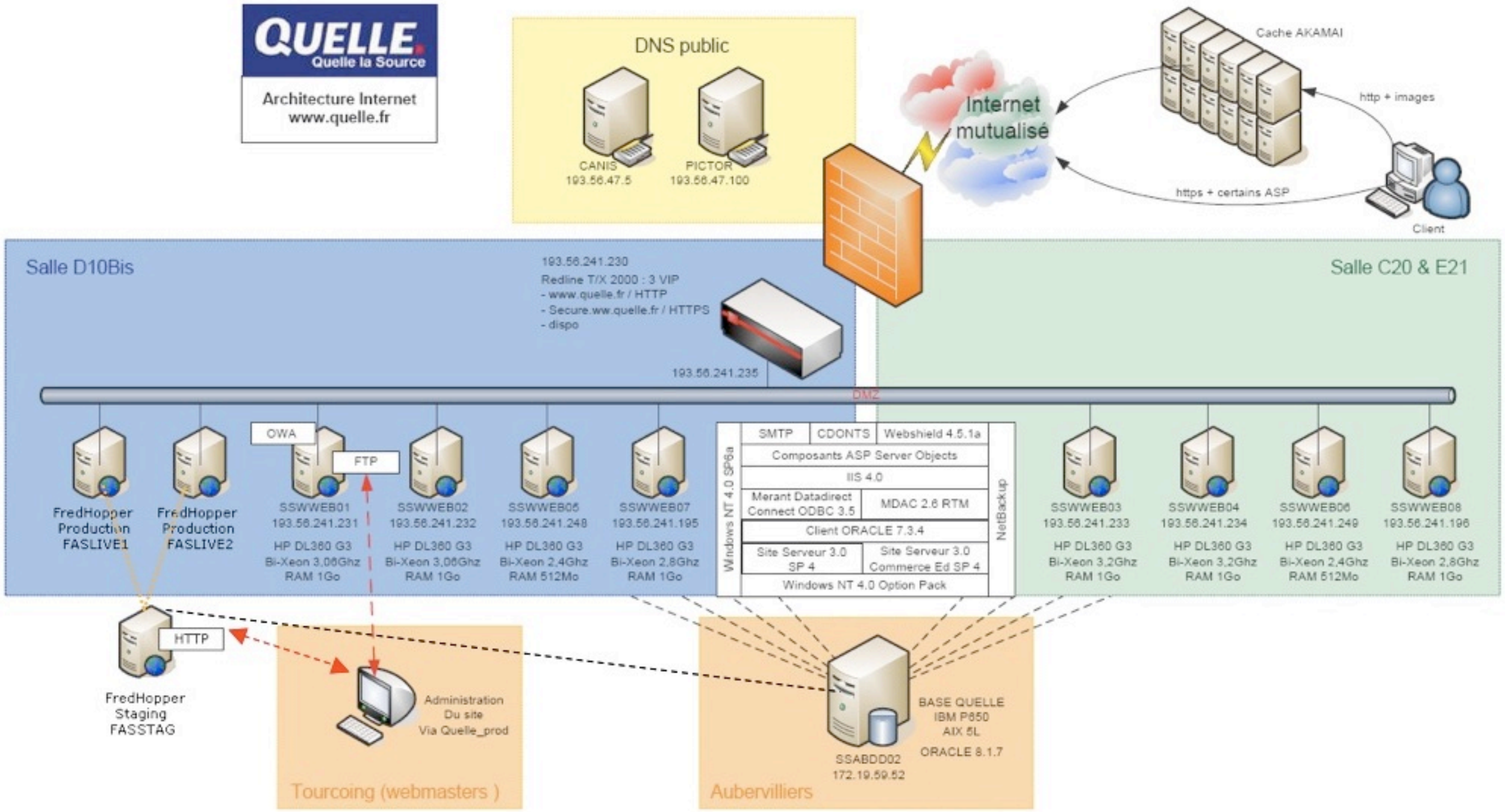

QUELLE
Quelle la Source
Architecture Internet
www.quelle.fr
DNS public
CANIS
193.56.47.5
PICTOR
193.56.47.100
Internet
mutualisé
Cache AKAMAI
http + images
https + certains ASP
Client
Salle D10Bis
Salle C20 & E21
193.56.241.230
Redline T/X 2000 : 3 VIP
- www.quelle.fr / HTTP
- Secure.ww.quelle.fr / HTTPS
- dispo
193.56.241.235
DMZ
OWA
FTP
FredHopper Production FASLIVE1
FredHopper Production FASLIVE2
SSWWEB01 193.56.241.231 HP DL360 G3 Bi-Xeon 3,06Ghz RAM 1Go
SSWWEB02 193.56.241.232 HP DL360 G3 Bi-Xeon 3,06Ghz RAM 1Go
SSWWEB05 193.56.241.248 HP DL360 G3 Bi-Xeon 2,4Ghz RAM 512Mo
SSWWEB07 193.56.241.195 HP DL360 G3 Bi-Xeon 2,8Ghz RAM 1Go
SMTP
CDONTS
Webshield 4.5.1a
Composants ASP Server Objects
IIS 4.0
Merant Datadirect Connect ODBC 3.5
MDAC 2.6 RTM
Client ORACLE 7.3.4
Site Serveur 3.0 SP 4
Site Serveur 3.0 Commerce Ed SP 4
Windows NT 4.0 Option Pack
Windows NT 4.0 SP6a
NetBackup
SSWWEB03 193.56.241.233 HP DL360 G3 Bi-Xeon 3,2Ghz RAM 1Go
SSWWEB04 193.56.241.234 HP DL360 G3 Bi-Xeon 3,2Ghz RAM 1Go
SSWWEB06 193.56.241.249 HP DL360 G3 Bi-Xeon 2,4Ghz RAM 512Mo
SSWWEB08 193.56.241.196 HP DL360 G3 Bi-Xeon 2,8Ghz RAM 1Go
HTTP
FredHopper Staging FASSTAG
Administration Du site Via Quelle_prod
Tourcoing (webmasters)
BASE QUELLE IBM P650 AIX 5L ORACLE 8.1.7
SSABDD02 172.19.59.52
Aubervilliers

Figure 22: Quelle France server architecture for FAS implementation

6.3.2 Rewrite module

While the solution discusses URL rewriting as if it was readily available in the FAS environment, it really is not. URL rewriting is a complex process request rewriting and forwarding, for which several specific programs have evolved over the past decade. As these requests need to be processed before being processed by a webserver, most of these programs are available as webserver modules. These modules typically allow the URL rewrite configuration to be expressed as a set of regular expression rules with additional configuration flags.

Tomcat rewrite module

Having these modules available as webserver modules implicitly requires the SEO solution to use a webserver. This would seem to pose no particular problem, as the JBoss platform includes its own Tomcat webserver, which is used by FAS to deliver its pages. However, there are several serious drawbacks to using the available rewrite modules for Tomcat:

- The available Tomcat module will only work when deployed in the same application context as the J2EE application it communicates with: FAS. However, the module is a commercial product and may therefore not simply be included in the FAS product, which is sold by Fredhopper.
- The configuration of the available Tomcat module is stored in an XML file. Configuring a complex set of rewrite rules as required by our solution in this XML turned out to seriously pollute the overview, like programming Java in Microsoft Excel. In addition, to test the configuration the complete application must be restarted, which may take several minutes to one hour for every test run.
- The available Tomcat rewrite module is still in developmental status. While the product may work indefinitely without issues, there is no guarantee for stability. This is not acceptable in an environment where stability is the highest priority.

Apache rewrite module

These issues obviously pose some serious problems with using the Tomcat rewrite module. The alternative was using a standalone separate webserver like Apache of Microsoft's IIS. Since Apache's server runs on both UNIX and Windows platforms and is freely available, this was the alternative of choice. In addition the Apache rewrite module, mod_rewrite, has a respectable state of service and is extensively documented.

However, there is a disadvantage of using Apache besides JBoss running FAS. It means functionality is divided over 2 applications and these need to be tightly integrated. The following paragraph discusses the technique of connecting Apache to JBoss.

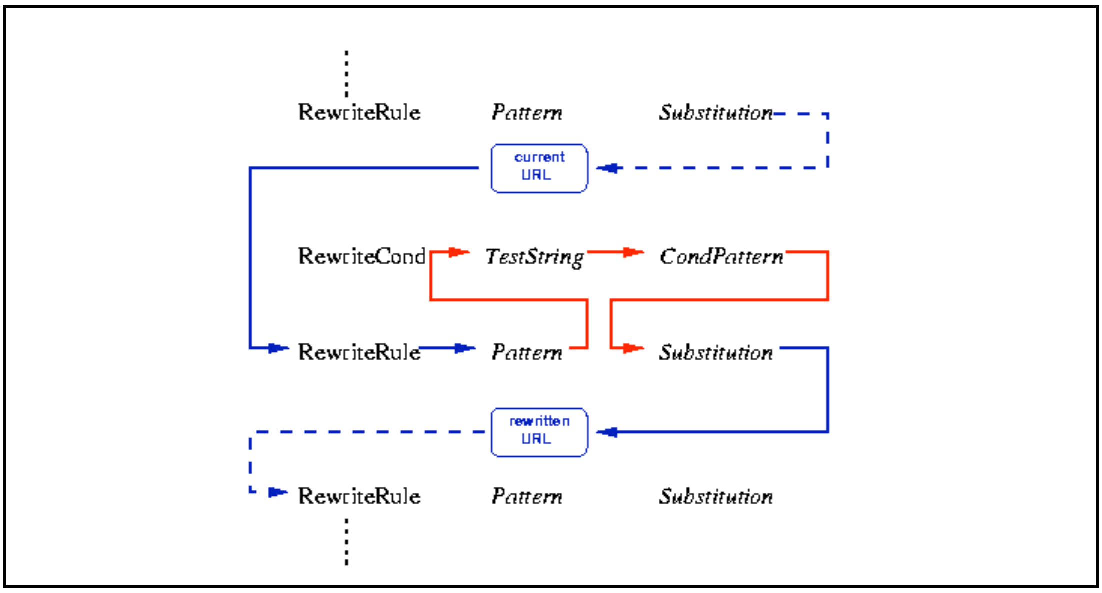


Figure 23: Internal rewrite process flow

6.3.3 Connecting Apache webserver and Tomcat

As the considerations previously described are no isolated case, much work has already been done to allow connections between Apache and Tomcat. In fact, for this sole purpose another module was written for Apache: mod_jk.

Mod_jk is an Apache module in which several *workers* can be defined. These workers connect to Tomcat's AJP/13 connector and each one can be mounted onto a separate virtual path in Apache. In our case we need only one worker that connects to the running Tomcat, both located on the same physical server. The simple configuration of mod_jk is given in Figure 24.

```
# Defining a single worker that connects to the Tomcat process
# running on the same physical server (localhost)

worker.list=default

worker.default.port=8019
worker.default.host=localhost
worker.default.type=ajp13
worker.default.lbfactor=1
```

Figure 24: mod_jk worker configuration

After configuring a worker with mod_jk, we may mount this worked on a virtual path in Apache. If we mount the worker "default" created above, onto `/quelle_fr/` for example, all requests that Apache receives for this path, are forwarded directly to the worker "default", ultimately delivering it to Tomcat.

6.3.4 Final FAS server configuration

With all these modules and configurations set up, we can model the configuration for each server for a clearer understanding and desirable visualization. In Figure 25 the configuration for each individual server is displayed. The arrows and numbers indicate the path that every request will follow. The numbers refer to processing phases for the requests and are described in Figure 26.

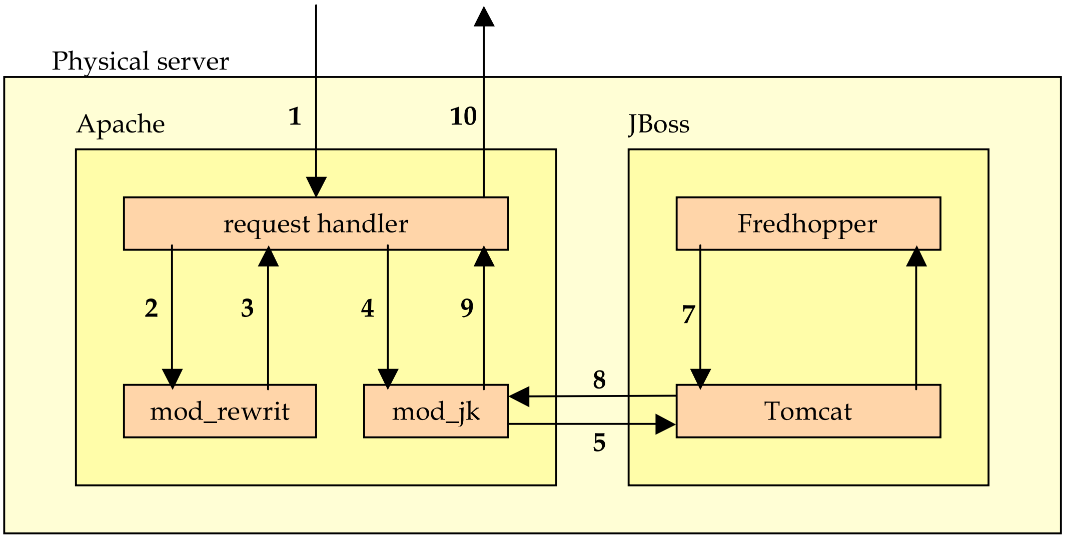


Figure 25: Apache/JBoss configuration on each server

Phase	Description
1	Normal external HTTP request for page (SEO url).
2	The request is given to mod_rewrite, rewriting it to an internal FAS url.
3	The FAS url is returned to the request handler.
4	The FAS url is linking to a path assigned to a mod_jk worker and therefore given to mod_jk for further handling.
5	The request is forwarded through mod_jk's worker, connected to Tomcat.
6	Tomcat makes a request to Fredhopper.
7	The resulting HTML is received from Fredhopper.
8	Tomcat forwards the returned HTML back to Apache's mod_jk.
9	The result is returned to the initial request handler.
10	Apache returns the HTML result to the external requester (user).

Figure 26: Phase descriptions as numbered in Figure 25

6.4 Rewrite implementation

This paragraph will provide an overview of the implementation of the rewriting. The full set of applied rewrite rules and conditions can be reviewed in appendix A.

6.4.1 Rewrite outline

With the environment set up, we can control all rewrites directly from Apache's mod_rewrite configuration file. The rewrite process consists of several steps, in chronological order of processing:

1. Immediate forwarding of requests that are not part of the visible part of the online store.

2. Detecting whether the request comes from a crawler or a human browser, and store this information in the query string.

3. As we will simulate paths, we need to forward relatively linked images and other external files like CSS stylesheets to their true (static) paths.

4. If the request does not originate from a crawler, rewrite the URL to a base Fredhopper URL. Virtual paths are removed and parameters are preserved, as these will take control of navigation.

5. Lastly, we process crawler requests. The crawler information is maintained in the query string to transfer this information to FAS.

6.4.2 Rules and conditions

The actual rewriting is expressed by a regular expression search and replace. These rewrite rules will – for example – look like this:

```
RewriteRule ^/quelle_fr/sample1/(.*)$  /quelle_fr/sample1/$1 [PT,L]
```

Such a rule defines that every request to anywhere within the `/quelle_fr/sample1/` is to be left unchanged and processed no more by the rewrite engine. This specific one is taken from the set of rules that implement phase 1 from the previous outline.

The rewrite engine allows the configuration of conditions that precede the actual rewrite rules. This makes it possible to apply rewrites only under certain conditions. For example:

```
RewriteCond %{QUERY_STRING} !^(.*crawler_visit=x.*)$
RewriteRule ^.*(/[a-zA-Z]+\.fh)$ /quelle_fr/boutique$1 [PT,L]
```

This condition/rule combination forwards any request to a document with the `.fh` extension to the `/quelle_fr/boutique/` path, but only if the original request did not have `crawler_visit=x` in the query string.

Combining these conditions and rules allows much control over the flow of the rewrite process. However, for full control some additional options are required, which are neatly provided by the rewrite module: flags.

6.4.3 Flags

To control the flow of the rewrite process, the rewrite module defines several flags that can be appended to rewrite rules and conditions. The ones that are relevant to this implementation are described:

- [PT]
  Pass-through: tells the Apache webserver to continue processing beyond this module; for this work most relevant, as we need every request to be processed by the Tomcat connector after rewriting. This is required for every rewrite rule defined, except the one that detects crawlers and modifies the query string accordingly.
- [L]
  Last: indicates that the current rule is the last one to be processed by the rewrite module; provides an immediate exit of the rewriting process. Again, this flag is required for every rule except the crawler detection rule.
- [QSA]
  Query string append: indicates that the given query string must be appended to the processed one, instead of replacing it. This will be used to append the crawler flag to the processed query string, plus with those rules that extract encoded information from the URL and need to add this to the existing query string.
- [NC]
  No case: makes the condition case insensitive; this is used to match the user-agents of visitors and find crawlers visits.
- [OR]
  Or next: used to chain a condition to the next based on the logical OR instead of the implicit AND.

6.4.4 URL Format

The specific format of the optimized URLs has not yet been discussed. Although the general idea of the encoding was discussed extensively during design, there are some additional remarks to make about the format of the URLs.

There are two types of optimized URLs, represented by the visible nodes left in Figure 21 (page 63): category URLs and detail URLs. To ease the detection of optimized URLs, a marker is placed in both. The detail page URLs will be of the format:

```
http://fas.quelle.fr/d-{secondid}-{keywords}.html
```

For example:

```
http://fas.quelle.fr/d-29260-la_pochette_de_transport.html
```

Where the `d-` helps our rewrite rules to easily find and rewrite these optimized detail URLs. For the category URLs something similar is practiced. These will be of the format:

```
http://fas.quelle.fr/c/{top_level category_id}/
http://fas.quelle.fr/c/{top_level category_id}/{2nd_level_category_id}/
```

For example:

```
http://fas.quelle.fr/c/homme/
http://fas.quelle.fr/c/homme/1558_street_sport/
```

Where the rewrite rules may rely on the `/c/` prefix to detect category URLs that need to be rewritten.

6.4.5 Rewrite model

The whole rewriting process consists of a complex set of chained rules and conditions and is displayed in appendix A. For a clear understanding of the actual flow and actions of this process, refer to the schema in Figure 27, modelling all rules and actions in a *rewrite flow*.

6.5 XSLT implementation

Naturally, the links that the user clicks on will be generated within FAS. However, these are – with the pages themselves – all dynamically generated. Therefore it is not possible to directly specify the format of the links; this must be accomplished through a small function within the FAS content generator. For the case of open integrations this content is always generated from XML by the set of XSLT stylesheets. This paragraph will give an account of the implementation of this functionality in XSLT: both the encoding of parameters within the URL as the selective showing of parameters to crawlers and human users. For a clear understanding of this implementation, some knowledge of the XSLT language is required.

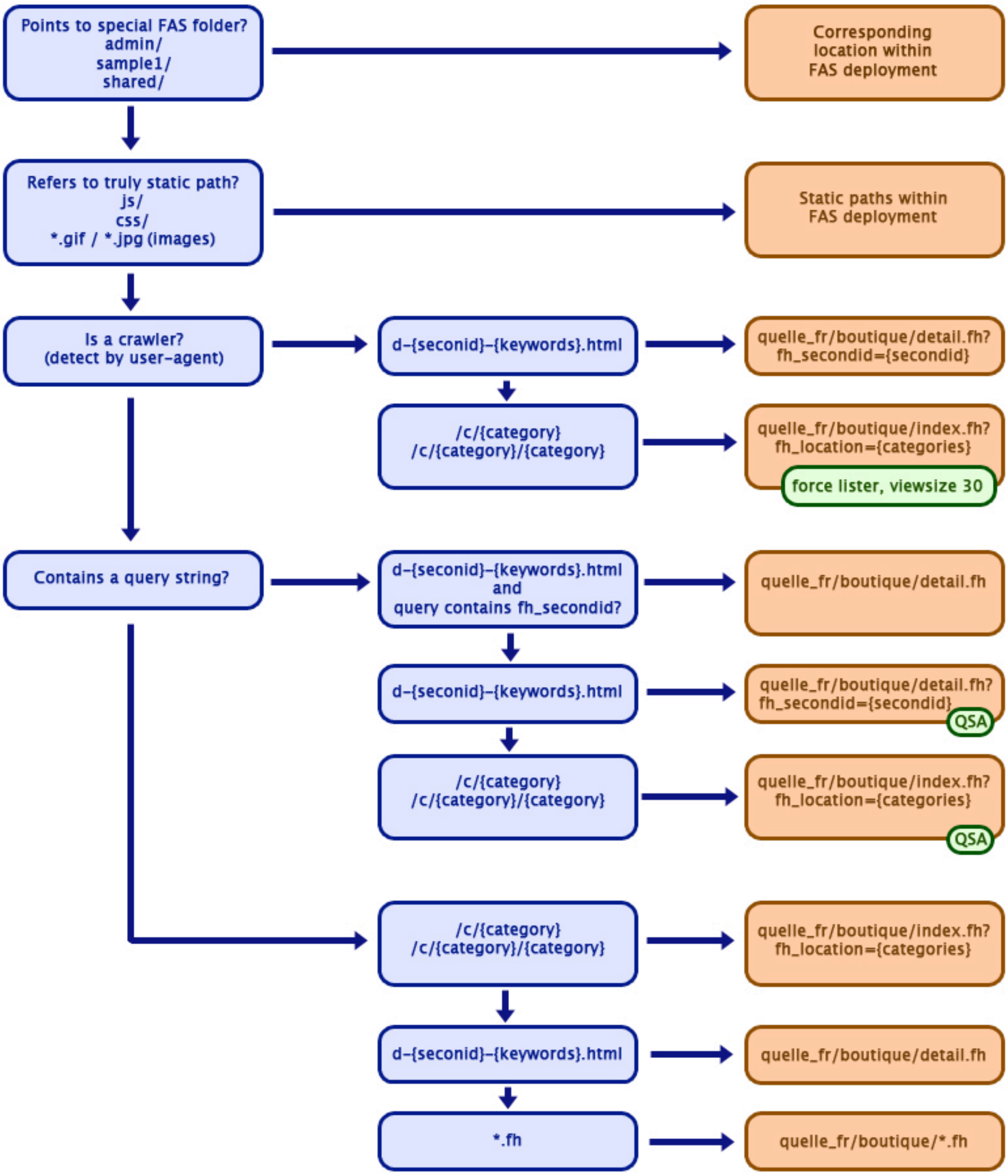

Points to special FAS folder?
admin/
sample1/
shared/
Corresponding location within FAS deployment
Refers to truly static path?
js/
css/
*.gif / *.jpg (images)
Static paths within FAS deployment
Is a crawler? (detect by user-agent)
d-{secondid}-{keywords}.html
quelle_fr/boutique/detail.fh?
fh_secondid={secondid}
/c/{category}
/c/{category}/{category}
quelle_fr/boutique/index.fh?
fh_location={categories}
force lister, viewsize 30
Contains a query string?
d-{secondid}-{keywords}.html
and
query contains fh_secondid?
quelle_fr/boutique/detail.fh
d-{secondid}-{keywords}.html
quelle_fr/boutique/detail.fh?
fh_secondid={secondid}
QSA
/c/{category}
/c/{category}/{category}
quelle_fr/boutique/index.fh?
fh_location={categories}
QSA
/c/{category}
/c/{category}/{category}
quelle_fr/boutique/index.fh?
fh_location={categories}
d-{secondid}-{keywords}.html
quelle_fr/boutique/detail.fh
*.fh
quelle_fr/boutique/*.fh

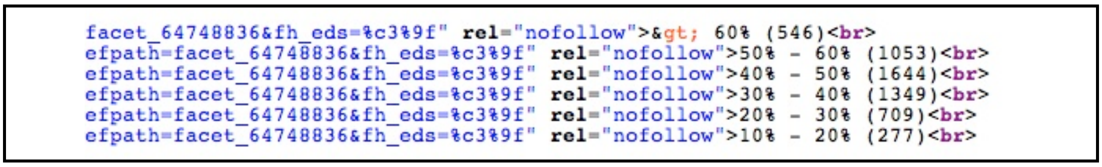
Figure 27: Rewrite flow chart

6.5.1 URL generation module

Within FAS there are 3 base types of URLs that are generated: *facet-*, *breadcrumb-* and *detail-links*. For these types the solution specifies exactly what they must look like and what URL they address. The XSLT implementation provides a template for each of these 3 types, with the addition of a template specifically for the more/less link present below facets. These templates are defined in a separate XSLT sheet within FAS for better overview and easier maintenance: `_urls.xsl`. The full contents of this stylesheet are given in appendix B, but the remainder of this paragraph will outline the main thoughts of the implementations.

Facets

Since we need to force a category only view on the catalogue, all facet-links except categories are given the attribute `rel="nofollow"` to stop crawlers from following these links. When the facet actually *is* a category the shown URL must encode the category and its location within. The actual encoding was designed as `/c/{category}` for top level categories and `/c/{top-level-category}/{sub-category}` for 2nd level categories. With the Quelle implementation the category tree only went 2 levels deep. However, it is obvious this encoding can be extended to whatever depth required. Lastly, the URL parameters which are provided by the FAS output XML, are only included when the page-information indicated this particular visit did not originate form a crawler.



```
facet_64748836&fh_eds=%c3%9f" rel="nofollow">&gt; 60% (546)<br>
efpath=facet_64748836&fh_eds=%c3%9f" rel="nofollow">50% - 60% (1053)<br>
efpath=facet_64748836&fh_eds=%c3%9f" rel="nofollow">40% - 50% (1644)<br>
efpath=facet_64748836&fh_eds=%c3%9f" rel="nofollow">30% - 40% (1349)<br>
efpath=facet_64748836&fh_eds=%c3%9f" rel="nofollow">20% - 30% (709)<br>
efpath=facet_64748836&fh_eds=%c3%9f" rel="nofollow">10% - 20% (277)<br>
```

Figure 28: HTML of the facet links with "nofollow"

Breadcrumbs

None of the URLs in the breadcrumbs are to be followed by crawlers. Just as with most facets, breadcrumb URLs will therefore all receive the `rel="nofollow"` attribute. As will be noticed when reviewing the XSLT, the template does however look rather complex for such a single rule. However, it must be noted that much more logic is involved in the generation of the breadcrumb URLs, that unfortunately had to come along to the URL generating templates.

Detail-pages

The generation of the URLs for detail-pages is the most complex by design, but the most simple to implement in XSLT, as is visualized by its (just slightly over) 20 lines of code. Detail-links do not get the `rel="nofollow"` attribute, as they are most import to be crawled of all. Depending on the type of visitor (crawler or human) they receive the proposed URL parameters as they are found in the FAS output XML. The linked file is by design a simulated physical path, located in the root of the site structure. The actual name of this file consists is `d-{item_ID}-{keywords}.html`, where `{item_ID}` is the only important piece of encoded information and `{keywords}` any piece of text with relevant keywords. The contents of this keyword string is received directly from the FAS output, as this is stored in the database giving Quelle France the opportunity to tweak these strings later whenever they desire to do so.

6.5.2 Detail page optimization

As stressed in the design, the power of keyword optimization within the documents should not be overlooked. Every detail page is optimized for both keywords and site structure in the following ways.

Keyword optimization

The name of the item is placed as the title of the document and repeated within the document both in `<h1>` and `<b>` tags. It is repeated once more later in the detail page. Additionally it appears at the very top of the contents of the document as an element in the breadcrumb.

Site structure optimization

One last simple addition to the contents of the detail pages, which largely affects our solution, is the robots META tag. It is used to communicate to crawlers that the page may be indexed, but that none of the links within the page should be followed. This makes sure that detail pages are actually the end-nodes in the site structure that we designed them to be (refer to Figure 21 on page 63).

6.6 Summary

To summarize the steps taken in this chapter to implement the solutions proposed in chapter 5, these steps are presented here in summary in a table.

	Parametric URLs	Site structure	Keywords
Hardware	Installing and connecting rewrite module to JBoss;	-	-
Software: rewrites	Crawler detection; Decode encoded location string from URLs;	-	Strip "*meaningless*" keywords from URLs;

Software: XSLT	Encode location string in URLs;
	Append full parameter-set to URLs when human visitor;
		Generate META tags;
		Generate *nofollow* links;
			Optimize placement and presentation;
			Encode keywords in URLs

Table 5: Summary of implementation steps and area of effect

7 Results

This chapter will provide experimental data that allows us to judge the effectiveness of the designed solution and its implementation.

7.1 Tests preface

7.1.1 What will be measured?

The implementation was modelled to improve performance in two dimensions: crawlability and ranking. Crawlability describes the ease and ability of pages to be discovered by crawlers and thus be listed on search engines. Ranking determines the position of a document in these lists. High crawlability and high ranking may result in a top-10 position on search engines, while low crawlability alone may result in not being listed at all. These assumptions will be used in this chapter to choose suited tests and interpret the results.

7.1.2 General limitations

Before moving on to the actual tests and their results, it is required to outline some limitations that affect our tests.

Lack of comparative data

To judge the effectiveness of the implemented solution, we would need to define a way to compare the treated and untreated FAS implementations. However, for the implemented solution there is no before/after data available. It was simultaneously deployed with FAS, which Quelle did not run before. The option left is to compare the solution with *other* untreated FAS implementations. The big similarity between FAS implementations will make such *cross-case* comparisons reliable.

External factors

It has been mentioned throughout this work that there are *on-page* and *off-page* factors that influence both crawling and ranking. Although crawling does not depend much on these external (*off-page*) factors, ranking is highly dependent on the inbound links. Unfortunately, the effect of these external factors cannot be eliminated.

Instability of measurements

It should be clear by now that search engines behave much like an organism, in that they are constantly in motion and evolving. Their index is always being updated by crawlers that will never sleep. The ranking algorithms are continually being tweaked, to optimize results relevance.

The effect of this is that the measurements presented in this chapter, may not necessarily yield the same results when repeated. Although this lack of verifiability is regrettable, the cause is completely out of our control.

7.2 Crawlability

7.2.1 Goal

The goal of the crawlability test is to prove that the solution proposed and implemented in this thesis, will improve the crawlability of a FAS implementation.

7.2.2 Procedure

Crawlability can be measured by the number of pages that are included in the search engine indexes. Increased crawlability will result in more documents crawled. We will abbreviate the *number of pages indexed* as NPI.

Theoretically, the NPI equals the number of item plus the number of categories in the catalogue. So, depending on the data in the database this number changes. The actual numbers did meander within certain boundaries: the number of items always counted somewhere between 3,000 and 6,000, while the number of categories was always counted between 80 and 120. The number of categories thus only makes up a few percent of the NPI.

It should be noted that there is a delay between updating the database, and a visible effect in the search engines indexes. This is explained by the delays between a database update, crawler visits, and a search engine index update.

To measure the NPI, the three largest search engines (Google, Yahoo and MSN) have been monitored for a few months after launch. Fortunately, all three engines allowed us to obtain the NPI easily. By using the `site:{domain}`[19] query, supported on all 3 search engines, this number is immediately displayed. This query effectively requests all documents indexed on a certain domain. The FAS implementation for Quelle France was located on `fas.quelle.fr`.

[19] http://www.google.com/help/features.html#domain

The results of the measurements will be compared to those of an untreated FAS implementation: the *donateursvereniging* mentioned in chapter 2. The NPI of this implementation is not as interesting as its message: only 1 page is indexed.

7.2.3 Results

As mentioned the NPI has been measured over 3 months since the actual launch date of the implementation on 21st of March. These measurements are presented in Table 6.

	23-Mar	30-Mar	05-Apr	12-Apr	19-Apr	30-Apr	18-May	06-Jun	28-Jun	10-Jul	29-Jul
Google	12	181	632	1749	2591	3110	3485	3390	3614	3090	3201
Yahoo	19	162	484	781	980	1085	1128	948	1233	1039	1345
MSN	98	393	782	914	719	819	1385	914	789	812	840

Table 6: NPI measurements

At first sight there are three striking properties of this data. First of all, there are a significant amount of pages in the indexes. A promising indication of the success of the solution! Secondly, Google includes much more documents than Yahoo and MSN. Third, all three search engines are slow starters: they take about one month to reach their equilibrium. The measurements are visualized in Figure 29 below.

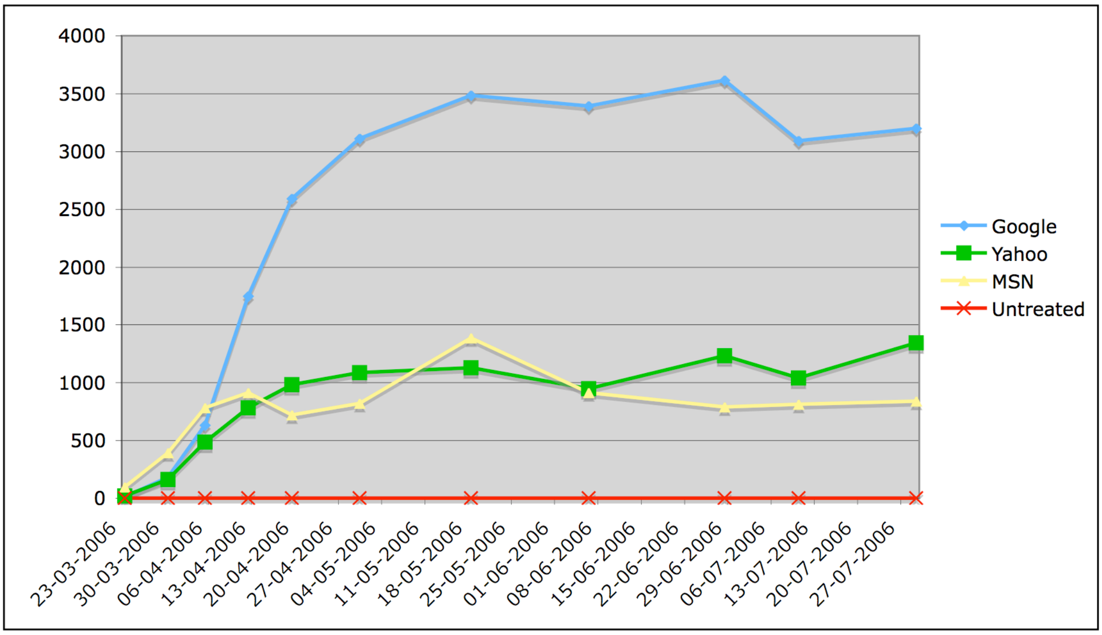


Figure 29: Graph of the measurements over time

7.2.4 Conclusions

The results show that the solution allows search engines to easily crawl and index the pages generated by FAS. The design is scalable to a large degree, although it seems that search engines do limit the number of pages they include in their indexes, as the data from Table 6 and Figure 29 suggests. While Yahoo and MSN only have around 1.000 pages in the index, Google averages above 3.000.

7.3 Ranking

7.3.1 Goal

The goal of the ranking test is to prove that the solution proposed and implemented in this thesis, will result in high(er) rankings on the search engine results pages.

7.3.2 Procedure

The complications with this test are twofold. We suffer from our inability to use before/after data, and the ranking algorithms are too complex to directly relate cause and effect. The issue is that since we cannot detect any improvements over time, how can we judge the measured ranking? Is a listing on position 24 a great result, or is it too low?

It needs mentioning that the Quelle France implementation does have a characteristic that simplifies our interpretation. As can be readily verified online, all FAS pages hosted by Quelle France are designed to be displayed within a frameset. Fortunately crawlers manage to find and index these documents. However, users referred to these pages through the search engine results pages are redirected to the framed setup. The consequence of this, is that *if* anyone desires to link to a Quelle France page, the URL they will link to, will be different from that of the actual document. As a result, ranking algorithms will determine rankings for Quelle France pages based solely on *on-page* factors, weakening its competitive position, since inbound links are known to improve rankings.

The measurements for this test consist of the rankings on the three largest search engines (Google, MSN and Yahoo!). For 50 search terms, based on the items in the Quelle France catalogue, the ranking was determined on these three search engines, and the total number of results was also recorded. This yields a dataset that gives us insight in the ranking performance.

The search terms where picked semi-randomly, making sure there were both very specific searches (*kimono enfant*) as well as very general ones (*armoire*). Subjects were also chosen to vary among different categories (clothing, furniture, *etc.*).

7.3.3 Results

The following table presents the results as were obtained using the procedure described. Positions were only recorded up to 100, anything higher is listed as “>100”. Those search terms that did not yield any results on the Quelle France website, have a position listed as “n.r.” (no result).

Search engine:	**google.fr**		**fr.msn.com**		**fr.yahoo.com**	
Search term	**position**	**# results**	**position**	**# results**	**position**	**# results**
"ceinture mixte minceur"	2	916	1	4	1	348
jupe	>100	6.000.000	>100	8.500.000	>100	1.300.000
"soutien gorge"	>100	1.270.000	66	121.000	31	449.000
"myguide 3500"	59	499.000	n.r.	730	14	46.900
"drap housse"	26	428.000	24	900	6	40.300
armoire	>100	8.650.000	n.r.	350.000	>100	3.510.000
lemmings psp	>100	1.340.000	14	59.000	>100	661.000
matelas	>100	4.490.000	>100	408.600	>100	855.000
"couverture polaire"	48	130.000	n.r.	480	3	10.400
doudoune	>100	669.000	33	393.000	9	94.400
"pantalon maternite"	1	9.660	1	13	1	138
puma tee shirt	48	991.000	9	902	1	269.000
couverture	>100	33.300.000	>100	2.417.000	>100	6.550.000
bureau	>100	197.000.000	>100	31.800.000	>100	134.000.000
horloge	>100	9.930.000	>100	721.000	>100	2.480.000
"home cinema 5.1"	44	299.000	30	11.090	>100	66.500
"blanchisseur de dents"	7	18.800	1	45	93	818
cardigan	>100	8.090.000	37	1.525.000	>100	4.240.000
"lit simple"	>100	405.000	9	23.800	>100	75.400
"mini lave vaisselle"	71	19.900	n.r.	230	5	1.540
sweat adidas	>100	1.570.000	10	951	3	531.000
slip	81	89.300.000	>100	37.635.000	>100	62.200.000
housse matelas	98	679.000	39	22.800	3	62.900
"classeur rideau"	12	12.100	1	24	n.r.	502
"kimono enfant"	15	790	n.r.	21	2	153
taie d'oreiller	30	411.000	19	15.270	7	62.700
vitrine	>100	13.100.000	>100	1.655.000	>100	3.670.000
santiags	>100	114.000	5	3.800	5	21.300
panneau	>100	11.000.000	>100	787.712	88	2.450.000
chiffonnier	47	204.000	9	8.695	7	29.700
philips rasoir	>100	404.000	56	816	4	59.200
nike baskets	>100	1.220.000	21	148.736	28	396.000
carte memoire	>100	4.550.000	>100	886.550	58	4.860.000
vibromasseur	43	402.000	64	1.630.514	>100	171.000
blouson	>100	3.240.000	>100	384.524	45	642.000
cache sommier	9	276.000	8	3.372	2	11.700
epilatuers satinelle	15	34.700	n.r.	49	n.r.	215
robot gourmet	5	1.350.000	>100	175.189	n.r.	760.000
gilet	89	3.830.000	68	336.725	50	964.000
"tapis de bain"	52	221.000	86	943	10	21.300
lecteur dvd	>100	6.040.000	>100	726.473	>100	5.750.000
brosse rotative	40	83.200	20	3.755	n.r.	22.500
"robe de chambre"	93	319.000	>100	20.513	4	69.500
coussinets d'allaitements	4	944	3	28	n.r.	143
bonnet	>100	16.400.000	>100	2.110.008	>100	7.650.000
jean	>100	330.000.000	>100	52.535.400	>100	154.000.000
nintendogs jeu	80	455.000	6	5.509	>100	69.600
gamecube dragon ball z	>100	4.620.000	10	143.402	>100	2.010.000
salon rotin	>100	273.000	>100	9.493	5	32.900
pantacourt adidas	37	206.000	6	391	1	22.700

Table 7: Measured rankings and # results for 50 search terms

At first sight this data seems to reveal one correlation: high rankings are more common for keywords with a lower number of total results indexed. This correlation is visualized by the scatterplot in Figure 30.

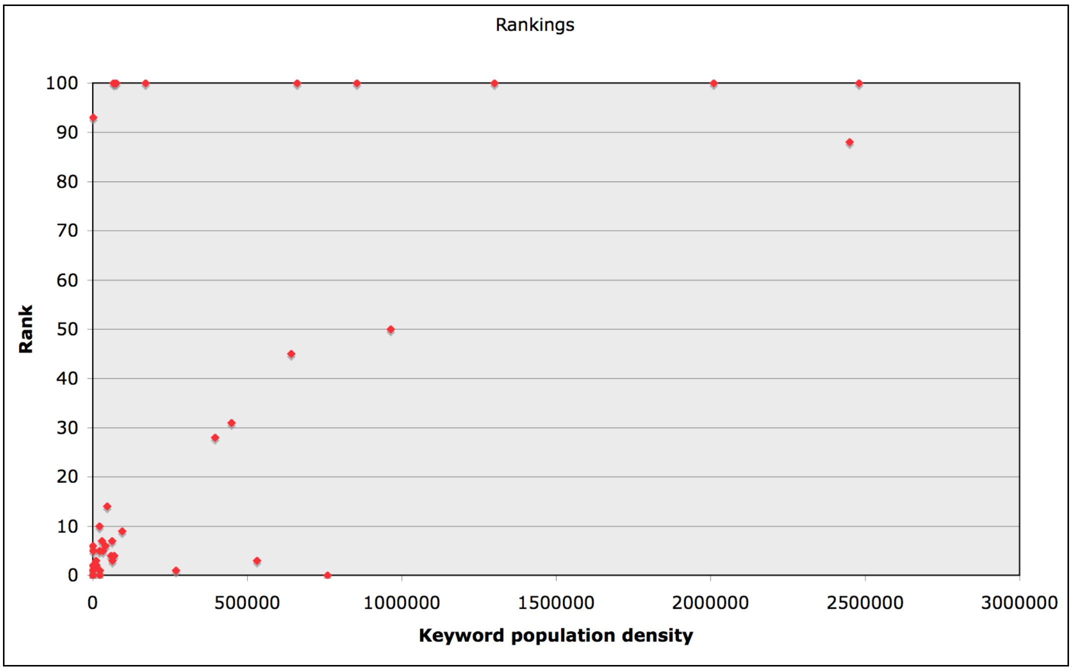


Figure 30: Scatterplot of rankings and keyword population density

By dividing the measurements into 4 categories, based on the total number of results, one can draw the following figure giving even more insight in the data. The 4 categories represent those rankings among less than 1.000, among 1.000 to 100.000, among 100.000 to 1.000.000 and those among more than 1.000.000 other result.

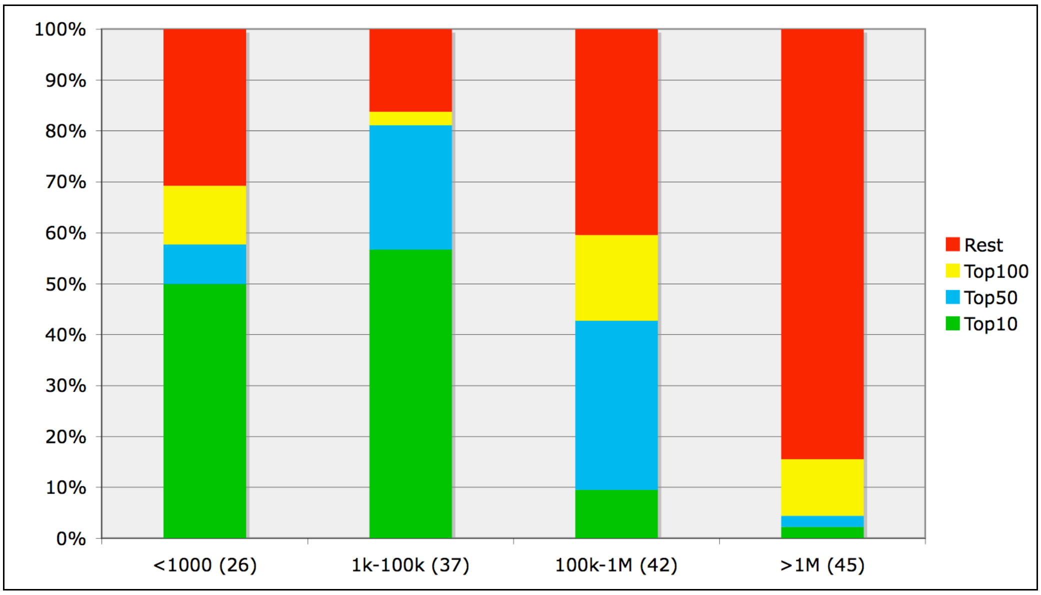


Figure 31: Ranking performance based on number of other results

7.3.4 Conclusions

As the results show, the solution generated many pages that made it into the top results of the search engine results. The applied heuristics from the model seem to have done their work. Especially for those search terms that were not too densely populated on the internet, the results are satisfying. Among the bigger crowd however, performance drops considerable. Theory gives us two possible causes for this: among a bigger crowd it is simply harder to get first; the lack of inbound links becomes substantially more critical.

Restating another point that was previously mentioned, not being able to compare these results unfortunately makes it hard to draw conclusions about the ranking performance. It seems a great result that about 50% of the pages end up in the top 10 for their keywords, even in fields that contain up to 100.000 other documents. However, we can't prove whether this really is cheerful, or is actually pretty moderate. Since there is no apparent solution to this issue, we are left to our personal opinion.

The solution implemented for Quelle thus seems to contain the ingredients for high ranking, but to rank high in a crowded field of popular keywords requires some additional, external factors out of our own direct control. Unfortunately, this specific implementation (in a page design consisting of frames) limits the big advantage that inbound links can provide. However, it is unlikely that for so many items (in an already constantly changing catalogue) a large number of direct inbound links for many of these items would have come to exist. Still, one very popular product may miss its very chance to become an immediate hit all over the internet by using its visitors and their links to gain an ever growing momentum.

8 Conclusions & Future work

8.1 Goals & Objectives

In this chapter we review our initial goals and objectives and show to what extent they were satisfied. These objectives were:

- to analyze the cause of the problem by reviewing literature on the subject;
- to develop a solution model;
- to implement this model, observing the identified constraints;
- to review if the implementation was successful.

8.1.1 Problem analysis and literature survey

We have given an extensive account of existing theory on the problem. The issues that are the cause of the problem were identified. The two most important issues found, are the FAS site structure, and its intensive use of query string parameters. In addition, we paid much attention to keyword optimization. During our literature survey we listed and reviewed several solution strategies and techniques, rating them on several aspects.

Literature told us the issues that FAS faces are common among such large dynamic websites. The cause is technical limitations of crawlers, and their *no-risk* behaviour policy. Unfortunately, the issue will therefore not be solved soon on the search engine side. Literature also offered several promising solutions, that among other techniques used URL rewriting to improve the perceived site structure.

8.1.2 Development of a solution model

Based on the strategies and techniques found in literature, we modelled a solution that promised to solve all identified issues. The strategies employed are generic and any of them may be applied to similar cases.

The modelled solution proposes the simulation of a transparent and straightforward site structure, with few steps required to reach the detail pages (refer to Figure 32). This simulation is achieved by link attributes and robots META tags. In addition, the model suggests to improve crawlability along this path by using URL rewrites to tackle the issue of parametric browsing. This requires the encoding of those parameters that are essential for navigation to be encoded within the URLs. Last, the model pays attention to keyword optimization by giving directions for keyword presence and placement. The model thus covers all issues, and features heuristics for both crawlability and ranking optimization.

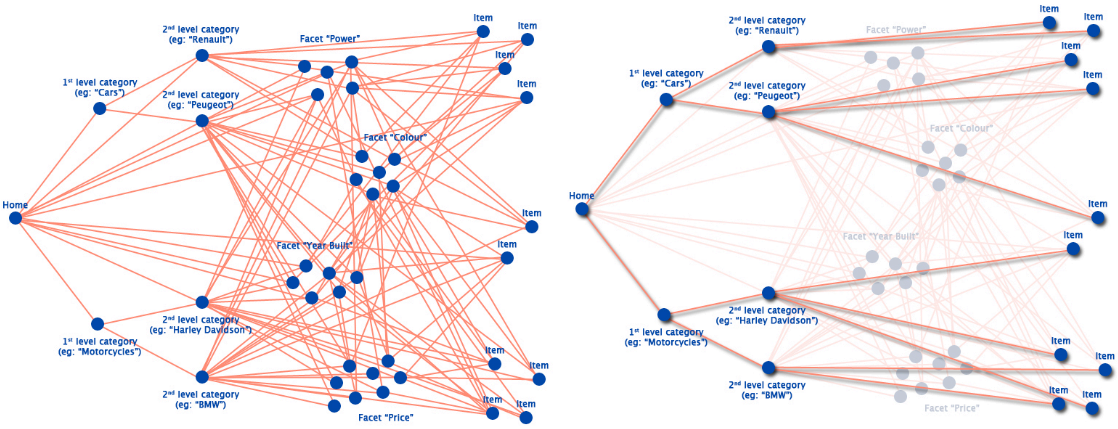

Home
1st level category (eg: "Cars")
1st level category (eg: "Motorcycles")
2nd level category (eg: "Renault")
2nd level category (eg: "Peugeot")
2nd level category (eg: "Harley Davidson")
2nd level category (eg: "BMW")
Facet "Power"
Facet "Colour"
Facet "Year Built"
Facet "Price"
Item

Figure 32: Site structure simulation model

8.1.3 Implementation

During implementation both hardware and software challenges were faced. A hardware setup was designed that featured a combination of FAS deployed through JBoss and Apache with its rewrite module on one server, which was found to function as desired and scalable to any number of servers.

Regarding software, a rewrite configuration was iteratively implemented along with its XSLT counterpart for generating the optimized URLs. With trial and error, this evolved over time and ultimately proved to work flawlessly on both our test- and live-servers.

8.1.4 Tests & Results

To test the effectiveness of the solution, both crawlability and ranking were reviewed in an empirical test. Regarding crawlability we concluded that the approach had been very effective. Cross-case comparison showed an improvement from a no *number of pages indexed* (NPI) to an NPI between 1.000 and 3.000, depending on the search engine.

It was harder to say anything conclusive about the ranking. Although the test showed hopeful results, the lack of comparative results strikes us dumb. However, considering the absence of inbound links, ranking performance *seems* to be respectable, with many pages appearing in the top results, even for the more common keywords. This can be attributed to the strong keyword focus: keywords are presented in URLs, pagetitle and header tags, as well as being repeated a few times within the document.

8.2 Future work

There are some issues that would be interesting to subject to further research. As a last part of this work, we will discuss these here.

8.2.1 Research NPI upper limit

From a theoretical point of view, there is an interesting property of the NPI graphs presented in paragraph 7.2.3. There seems to be some sort of asymptotic behaviour, defining an upper limit for the three search engines. Figure 33 depicts these perceived upper limits.

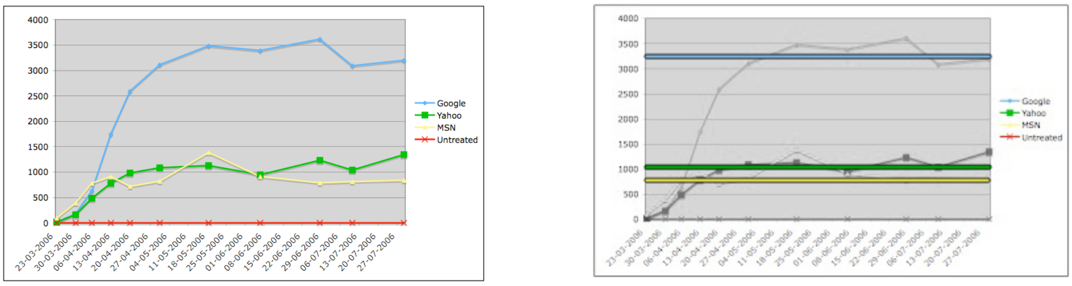


Figure 33: Perceived NPI upper limit

Further research into the field of search engine optimization might conduct an experiment to investigate the cause of this upper limit and why it varies so strong among different search engines. It should be noted however, that this could very well be the result of algorithms, which may change at any moment without notice.

8.2.2 Generic FAS module

From a more commercially interesting perspective, it would be desirable to develop the FAS specific solution into a generic FAS SEO solution, which can effortlessly be moved and installed among different FAS implementations.

The current implementation has Quelle France specific code in both the rewrite rules and the XSLT, which could easily be separated from it and placed in an external configuration file. The Apache/rewrite package could then be packed as a single module, which may circumvent the standard strenuous installation procedure.

9 References

[1] E. Akerboom, *Designing and building an indexing infrastructure to aid Internet resource discovery*, thesis paper, Delft University of Technology, the Netherlands, 2004

[2] M. Bianchini, M. Gori, and F. Scarselli, *Inside PageRank*, Tech. rep., University of Siena, 2003

[3] O. Brandman, J. Cho, H. Garcia-Molina, and S. Shivakumar, *Crawler-friendly Web servers*, Performance Evaluation Review, 28(2), Sept. 2000, Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, June 2000

[4] S. Brin, and L. Page, *Anatomy of a large-scale hypertextual web search engine*, In Proceedings of the 7th International World Wide Web Conference (Brisbane, Australia, Apr. 14 –18), 1998

[5] A. de Carvalho Fontes and F. Soares Silva, *SmartCrawl: a new strategy for the exploration of the hidden web*, Proceedings of the 6th annual ACM international workshop on Web information and data management, Washington DC, USA, 2005

[6] C. Castillo, *Effective Web Crawling*, PhD thesis, University of Chile, 2004

[7] J. Cho, *Frontiers in Web Data Management*, In Information and Communication Technology Symposium (ICTS'03), August 2003

[8] J. Cho and S. Roy, *Impact of search engines on page popularity*, In Proceedings of the WWW 2004, May 2004

[9] J. Cho, H. Garcia-Molina, and L. Page, *Efficient crawling through url ordering*, In Proceedings of the Seventh International World-Wide Web Conference, 1998. Available at http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1999-0103

[10] K. Cool, C. Harford and M. Oppenrieder, *Google and the Online Search Industry in 2005*, INSEAD, Fontainebleau, France

[11] F. Crestani, M. Sanderson, M. Theophylactou, and M. Lalmas, *Short queries, natural language and spoken document retrieval: experiments at Glasgow University in Voorhees*, E.M. and Harman, D.K. eds. Sixth Text Retrieval Conference (TREC-6) (NIST SP 500-240), NIST, Gaithersburg, MD, USA, 1998

[12] Deloitte Touche Tohmatsu: Winners Deloitte Technology Fast 50 2003, September 2003

[13] Deloitte Touche Tohmatsu: Winners Deloitte Technology Fast 50 2004, September 2004

[14] I. Drost and T. Scheffer, *Thwarting the Nigritude Ultramarine: Learning to Identify Link Spam*, Proceedings of the European Conference on Machine Learning, 2005

[15] Fredhopper: Technical Reference Guide, internal report (non disclosure)

[16] J. Green, *Google PageRank and Related Technologies*, Research report Whitepaper, Greenbuilt Research http://www.greenbuilt-research.com/, September 2005

[17] GlobalSpec, GlobalSpec Whitepaper: *An Interview with Search Engine Expert Sarabjit Singh*, June 2005

[18] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, Combating web spam with TrustRank, In Proceedings of the 30th International Conference on Very Large Databases (VLDB), 2004

[19] D. Hawking, Trystan Upstill, and Nick Craswell, Toward better weighting of anchors, In Proceedings ACM SIGIR 2004, July 2004

[20] T. Joachims, *Optimizing Search Engines using Clickthrough Data*, Proceedings ACM SIGKDD 2002, Edmonton, Alberta, Canada, 2002

[21] J. Köhne, Search Engine Optimization: *The theory and practice of optimizing websites for higher search engine rankings*, Delft Technical University report, March 2006

[22] A. Langville and C. Meyer, *A reordering for the PageRank problem*, SIAM Journal on Scientific Computing, 2004

[23] S. Lawrence and C. L. Giles, *Searching the world wide web*, Science, April 1998

[24] S.W. Liddle, S.H. Yau, and D.W. Embley, *On the automatic extraction of data fromthe hidden Web,* in Proceedings of the International Workshop on Data Semanticsin Web Information Systems (DASWIS-2001), Yokohama, Japan, November 2001

[25] Lycos Insite, *Search Engine Marketing Tutorial*, white paper, available online http://insite.lycos.com/tutorial.asp

[26] Brian Pinkerton, *Webcrawler: Finding what people want*, Ph.D. dissertation, University of Washington, Seattle, WA, 2000

[27] S. Raghavan and H. Garcia-Molina, *Crawling the Hidden Web*, Technical Report 2000-36, Computer Science Deptartment, Stanford University, December 2000

[28] G. Salton, Automatic Text Processing: *The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989

[29] G. Salton, A. Wong, and Yang, C.S. *A vector space model for automatic indexing*, Communications of the ACM

[30] V. Shkapenyuk and T. Suel, *Design and implementation of a high-performance distributed web crawler*, In Proceedings of the International Conference on Data Engineering, February 2002

[31] L. Vaughan, and M. Thelwall, *Search engine coverage bias: evidence and possible causes*, Information Processing & Management, 2004

[32] A.M. Wall, *Search Engine Optimization Book*, available at http://www.seobook.com/, January 2005

[33] B. Wu and B. Davison, *Identifying link farm spam pages*, In proceedings of the WWW conference 2005, May 2005

[34] W. Xing and A. Ghorbani, *Weighted pagerank algorithm*, in Proceedings of the 2nd Annual Conference on Communication Networks and Services Research (CNSR 2004), 2004

10 List of Tables & Figures

List of tables

List of figures

A. Appendix A - URL rewrite set for Quelle France

```
#####################################
# Fredhopper SEO                    #
# --------------                    #
# URL rewrite rules for search      #
# engine optimized Quelle.fr        #
#                                   #
# februari, march 2006              #
#####################################



# First redirect admin/sample1/shared, and prevent them from
# being processed further by setting L (last) flag

RewriteRule ^/quelle_fr/admin/(.+)$
        /quelle_fr/admin/$1 [PT,L]
RewriteRule ^/quelle_fr/([a-zA-Z0-9_]+/)*shared/(.*)$
        /quelle_fr/shared/$2 [PT,L]
RewriteRule ^/quelle_fr/sample1/(.*)$
        /quelle_fr/sample1/$1 [PT,L]



# Crawler detection
# Detect crawlers by http_user_agent filtering. If detected,
# set the crawler_visit=x parameter to the URL and continue
# processing rewrite rules.

RewriteCond  %{HTTP_USER_AGENT}  googlebot                  [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  msnbot                     [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  slurp                      [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  ask.jeeves                 [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  fluffy                     [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  ia_archiver                [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  teoma                      [NC,OR]
RewriteCond  %{HTTP_USER_AGENT}  webcrawler                 [NC]


RewriteRule ^(.*)$     $1?crawler_visit=x [QSA]




# Rewrite images/css requested from any (simulated) path
# to the boutique root

RewriteRule ^.*(/css/[a-zA-Z0-9_]+\.css)$
        /quelle_fr/boutique$1? [PT,L]
RewriteRule ^.*(/js/[a-zA-Z0-9_]+\.js)$
        /quelle_fr/boutique$1? [PT,L]
RewriteRule ^.*(/img/[a-zA-Z0-9_]+\.gif)$
        /quelle_fr/boutique$1? [PT,L]
RewriteRule ^.*(/img/[a-zA-Z0-9_]+\.jpg)$
        /quelle_fr/boutique$1? [PT,L]





# If query string does not indicate a crawler,the visit is
# from a normal parametric-browsing user. Rewrite these URL's
# (if pointing to *.fh fredhopper pages) to base URL's.

RewriteCond %{QUERY_STRING} ^$
RewriteRule ^(/quelle_fr/boutique)?/c/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$
        /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2} [PT,L]

RewriteCond %{QUERY_STRING} ^$
```

```
RewriteRule         ^(/quelle_fr/boutique)?/c/([^/]*)/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$
       /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2}/categori
es<{$3} [PT,L]

RewriteCond %{QUERY_STRING} !^(.*crawler_visit=x.*)$
RewriteRule ^.*(/[a-zA-Z]+\.fh)$
               /quelle_fr/boutique$1 [PT,L]



# The same query-string size filter is executed on rewritten
# detail page URL's. Instead of rewriting to a non-parameter
# detail page (crawlers), just rewrite to fredhopper detail
# page with the parameters from the URL.

RewriteCond %{QUERY_STRING} !^$
RewriteCond %{QUERY_STRING} !^.*(crawler_visit=x).*$
RewriteCond %{QUERY_STRING} ^.*(fh_secondid=).*$
RewriteRule ^(/quelle_fr/boutique)?/d-([0-9]+)-.*\.html$
       /quelle_fr/boutique/detail.fh [PT,L]

RewriteCond %{QUERY_STRING} !^$
RewriteCond %{QUERY_STRING} !^.*(crawler_visit=x).*$
RewriteCond %{QUERY_STRING} !^.*(fh_secondid=).*$
RewriteRule ^(/quelle_fr/boutique)?/d-([0-9]+)-.*\.html$
       /quelle_fr/boutique/detail.fh?fh_secondid=$2 [PT,L,QSA]


# Requests to the detail page are rewritten to fredhopper
# detail pages. This is a crawler visiting: non-crawler
# detail rewrites are filtered earlier.

RewriteRule ^(/quelle_fr/boutique)?/d-([0-9]+)-.*\.html$
       /quelle_fr/boutique/detail.fh?fh_secondid=$2 [PT,L]



# Rewrite any other *.html request to the boutique root. This
# prevents requests into (simulated) paths.

RewriteRule ^.*(/[a-zA-Z]+\.html)$
               /quelle_fr/boutique$1 [PT,L]


# Any request to optimized category URL's are rewritten
# to fredhopper request. Show a lister page (size 30)
# directly to crawlers, to help retrieve detail pages.

# 1st level categories
#  -- crawlers
RewriteCond %{QUERY_STRING} ^.*(crawler_visit=x).*$
RewriteRule ^(/quelle_fr/boutique)?/c/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$

       /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2}&fh_view=
lister&fh_view_size=30 [PT,L]

#  -- normal users
RewriteRule ^(/quelle_fr/boutique)?/c/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$

       /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2}
[QSA,PT,L]

# 2nd level categories
#  -- crawlers
RewriteCond %{QUERY_STRING} ^.*(crawler_visit=x).*$
RewriteRule ^(/quelle_fr/boutique)?/c/([^/]*)/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$
       /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2}/categori
es<{$3}&fh_view=listerfh_view_size=30 [PT,L]

#  -- normal users
RewriteRule ^(/quelle_fr/boutique)?/c/([^/]*)/([^/.]*)[/]?([a-zA-Z]+\.[a-zA-Z]+)?$

       /quelle_fr/boutique/index.fh?fh_location=//quelle_fr/fr_FR/categories<{$2}/categori
es<{$3} [QSA,PT,L]
```

B. Appendix B - XSLT for generating optimized SEO URLs

The following XSLT can also be found online in the productive environment at http://fas.quelle.fr/quelle_fr/boutique/_urls.xsl

By viewing this online, a much clearer view will be offered.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" encoding="UTF-8"/>

<!-- Configure the prefix between the domain and the seo optimized paths -->

<xsl:variable name="boutique_root">/</xsl:variable>



<xsl:template name="url-facet-filter"><a>

  <xsl:variable name="is_category" select="contains(link/url-params,
'fh_reffacet=categories')" />
  <xsl:variable name="crawler_visit" select="contains(link/url-params, 'crawler_visit=x')"
/>

  <xsl:attribute name="href">

   <xsl:if test="$is_category">

    <xsl:if test="not(contains(/page/server-info/page-url,
'fh_location=%2f%2fquelle_fr%2ffr_FR%2fcategories%3c%7b'))">
     <xsl:text>/c/</xsl:text>
    </xsl:if>

    <xsl:value-of select="value" /><xsl:text>/</xsl:text>
   </xsl:if>

   <xsl:if test="not($crawler_visit)">

    <xsl:choose>
     <xsl:when test="@nr = 1">
      <xsl:text>detail.fh</xsl:text>
     </xsl:when>
     <xsl:otherwise>
      <xsl:choose>
       <xsl:when test="$current_page = 'detail.fh'">index.fh</xsl:when>
       <xsl:otherwise>
        <xsl:value-of select="$current_page" />
       </xsl:otherwise>
      </xsl:choose>
     </xsl:otherwise>
    </xsl:choose>

   </xsl:if>

   <xsl:if test="not($crawler_visit) or not($is_category)">
    <xsl:text>?</xsl:text><xsl:value-of select="link/url-params" />
   </xsl:if>
  </xsl:attribute>

  <xsl:if test="not($is_category) or /page/user/view = 'detail'">
   <xsl:attribute name="rel">nofollow</xsl:attribute>
  </xsl:if>

  <xsl:value-of select="link/name" />
  <xsl:text> (</xsl:text>
  <xsl:value-of select="@nr" />
  <xsl:text>)</xsl:text>
 <br/></a></xsl:template>

<xsl:template name="url-facet-filter-moreless">
 <a rel="nofollow">
  <xsl:attribute name="href">
   <xsl:text>index.fh?</xsl:text><xsl:value-of select="link/url-params" />
  </xsl:attribute>

  <xsl:choose>
   <xsl:when test="link/@name = 'more'">
    <xsl:text>plus (</xsl:text><xsl:value-of select="link/@nr" /><xsl:text>)</xsl:text>
   </xsl:when>
   <xsl:otherwise>
    <xsl:text>moins</xsl:text>
   </xsl:otherwise>
  </xsl:choose>

 </a><br/>
</xsl:template>

<xsl:template name="url-breadcrumb">

 <xsl:choose>

  <xsl:when test="position() = last() - 1 and ../crumb[position() = last()]/name =
'search'">

   <xsl:if test="position() &gt; 1"><xsl:text> &gt; </xsl:text></xsl:if>

   <font class="last">
    <xsl:choose>
     <xsl:when test="/page/user/view = 'detail'">
      <xsl:value-of select="//items/item/attribute[@name='_pv_titre']/value" />
     </xsl:when>
     <xsl:when test="name/@type = 'search'">
      "<xsl:value-of select="name" />"
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="name" />
     </xsl:otherwise>
    </xsl:choose>
   </font>

  </xsl:when>

  <xsl:when test="position() = last() and name = 'search'">
   <!-- do not display -->
  </xsl:when>

  <xsl:when test="position() = last()">

   <xsl:if test="position() &gt; 1"><xsl:text> &gt; </xsl:text></xsl:if>

   <font class="last">
```

```
    <xsl:choose>
     <xsl:when test="/page/user/view = 'detail'">
      <xsl:value-of select="//items/item/attribute[@name='_pv_titre']/value" />
     </xsl:when>
     <xsl:when test="name/@type = 'search'">
      "<xsl:value-of select="name" />"
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="name" />
     </xsl:otherwise>
    </xsl:choose>
   </font>

  </xsl:when>

  <xsl:when test="position() &gt; 1">

   <xsl:if test="position() &gt; 1"><xsl:text> &gt; </xsl:text></xsl:if>

   <a rel="nofollow" class="other">

    <xsl:attribute name="href">
     <xsl:value-of select="$boutique_root" /><xsl:choose><xsl:when test="$current_page =
'detail.fh' and contains(url-params, 'action=search')">search.fh</xsl:when><xsl:when
test="$current_page = 'detail.fh'">index.fh</xsl:when><xsl:otherwise><xsl:value-of
select="$current_page"/></xsl:otherwise></xsl:choose><xsl:text>?</xsl:text><xsl:value-of
select="url-params" />&amp;fh_session=<xsl:value-of select="$session-id"
/>&amp;fh_host=<xsl:value-of select="$fh_host" />&amp;<xsl:text>layout=boutique</xsl:text>
    </xsl:attribute>

    <xsl:choose>
     <xsl:when test="name/@type = 'search'">
      "<xsl:value-of select="name" />"
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="name" />
     </xsl:otherwise>
    </xsl:choose>

   </a>

  </xsl:when>

  <xsl:otherwise>

   <xsl:if test="position() &gt; 1"><xsl:text> &gt; </xsl:text></xsl:if>

   <a rel="nofollow" href="http://{$fh_host}/?fh_session={$session-
id}&amp;fh_host={$fh_host}">

   <xsl:attribute name="class">
     <xsl:choose>
      <xsl:when test="position() = last()">
       <xsl:text>last</xsl:text>
      </xsl:when>
      <xsl:otherwise>
       <xsl:text>other</xsl:text>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:attribute>

    <xsl:value-of select="name" />

    </a>
  </xsl:otherwise>

 </xsl:choose>

</xsl:template>


<xsl:template name="detail-url">

 <xsl:param name="url-params" />
 <xsl:param name="seo-optimization-term" />

 <xsl:variable name="seo-term" select="translate($seo-optimization-term,
'ABCDEFGHIJKLMNOPQRSTUVWXYZ +&lt;&gt;&amp;.:/\', 'abcdefghijklmnopqrstuvwxyz____n____')"
/>


 <xsl:variable name="crawler_visit" select="contains(/page/server-info/page-url,
'crawler_visit=x')" />
 <xsl:variable name="secondid" select="substring-before(substring-after($url-params,
'fh_secondid='), '&amp;')" />

  <xsl:value-of select="$boutique_root" /><xsl:text>d-</xsl:text>
  <xsl:value-of select="$secondid" />
  <xsl:text>-</xsl:text>
  <xsl:value-of select="normalize-space($seo-term)" />
  <xsl:text>.html</xsl:text>

  <xsl:if test="not($crawler_visit)">
   <xsl:text>?</xsl:text><xsl:value-of select="$url-params" />
   <xsl:if test="not(contains($url-params, 'fh_session='))">
    <xsl:text>&amp;fh_session=</xsl:text><xsl:value-of select="$session-id" />
    <xsl:text>&amp;fh_host=</xsl:text><xsl:value-of select="$fh_host" />
   </xsl:if>
  </xsl:if>

 </xsl:template>

</xsl:stylesheet>
```