My Personal Search-Engine

Mark A.C.J. Overmeer

AT Computing bv
Toernooiveld 104, 6525 EC,
Nijmegen, The Netherlands
markov@ATComputing.nl

Abstract

We have to reconsider whether the current structure of search engines provides us with useful results. A modular approach to constructing spiders will facilitate more development, hopefully resulting in new information retrieval techniques. A layered architecture will improve our access to data on The Net, which is crucial in our information driven society.

Key words: Restructuring search engines. Open search engine infrastructure.

1 Search Engines

Back in 1994, when World-Wide Web was just emerging, the first index of sites in The Netherlands appeared. It was named the *Dutch Home Page*[1]. Of course, today there are large number of such indexing sites, such as Yahoo![2] — at that time there were only 10 sites in the whole of the Netherlands and they could easily be displayed on one small page!

Within a year, the amount of work to maintain the index overwhelmed this initiative. Software was developed to ease the manual administrative task. New sites were first registered by their web master, by submitting a form. The site was then visited by our maintainer, who checked it for credibility and importance: personal home-pages where only included when they contained information which was useful for other people than family and friends. When these tests passed successfully, the site was added to the database from which we produced our own pages.

However, with the number of registered sites growing exponentially the drawbacks of adding those hundreds of new sites by hand increased to such a level that it became harder and harder to manage. It is not usefull to put many thousands of sites into one page, so you had to classify the sites into categories. As the number of sites within each category grew the category definitions had to become narrower to limit the num-

ber of sites in each of them. This strategy worked for three years but the number of categories grows exponentially too.

For an example of a site that indexes sites by category see *Yahoo!*, which is the largest such site on the Internet. It claims to have half a million sites registered in twenty five thousand categories. A single organization or company will fit in a huge number of detailed categories but, is usually listed under a small number of major subjects, and will not be listed under all of its other activities.

Categorization has its own problems, which are best illustrated by an example. An university will be active in many specialized fields, but is not listed under all of those hundreds of distinctive subjects. This means that someone looking for information about, say super-conductors, in the index ends up with companies specialized in that area, but not with this university which may be performing leading research in this area.

Specialized subject indices which are maintained by people 'in the field' are useful, so long as they are updated regularly (which is often not the case) and well known by its target-community. In such an index, a university department will get its place. However, for many subjects these indices are not well maintained, and they often not easy to find. As the Internet grows even these lists can become overwhelming. Additionally, knowing about the existence of a site related to a subject, does not give one the knowledge about what is actually present there.

As the amount of information on WWW grew, textual search systems were introduced. These *search engines* (also called *spiders* or *crawlers*) do not try to categorize sites, but use brute-force methods to scan for pages where certain keywords can be found. With search-engines, the effort for the builders of the indexing site has shifted from manual administration to writing smart software.

Search-engines try to apply natural language techniques, which can assist searchers in finding knowledge which is does not the highest priority of the site-publisher's point of view. A spider will find this knowledge whereas an index will not. Thus, spiders that categorise sites have more control over the quality of the information they show to their visitors because of the intelligence used in their construction. Spiders can be more useful to the searcher because they find items which are not in the site's description and hence transcend the restrictions imposed by an index.

However, a spider will return many results — a clear case of *quantity* and not *quality* and the searcher is subject to information overload.

Most people use search-engines nowadays, instead of indexes. This is not because their results are satisfying, but because the indexing system give less useful results.

I assert that the usefulness of general indexing sites has come to an end. This paper suggests a layered archi-

tecture which improve the access of search-engines to data on The Net. This will improve their quality, and thereby benefit everyone who has interest in Internet.

2 Current Strategies

Spiders are certainly not delivering a good quality result. Various smart algorithms have been developed to find the pages which best serve the query of the user (you can find more about the current strategies at *Search Engine Watch*[3]). However, everyone will have experience about the huge list of unwanted hits you get in response to your simple question. Current text-retrieval algorithms are designed to work on large amounts of data but not for the huge quantity on Internet.

Real spiders

Going through all the references returned by a spider takes a lot of time. The human interfacing is usually a disaster; you can see that the programmers found "getting the thing to work" was complicated enough, and they are happy that they can at least deliver some results.

Apart from the shear number of references returned a lot are outdated because the network is too large to get a timely overview. AltaVista[4] currently can only visit a site once every seven weeks and they retire sites from their database after a few failed fetch attempts, So it takes months before their database is correct.

The best part of current search engines is that they retrieve their data very quickly. The performance of computers in not the bottle-neck.

The *real* spiders, fetch all their pages themselves. They are built on expensive hardware (huge quantities of disk-space and memory is required) and complicated software. Figure 1

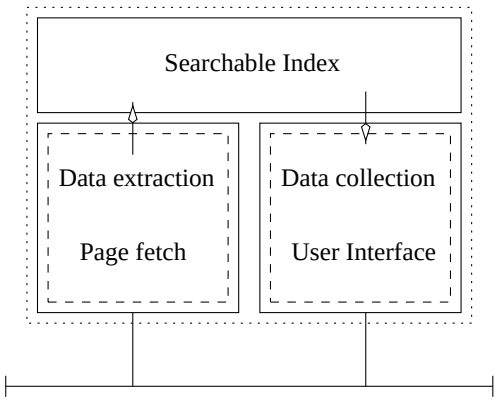


Fig. 1. Basic (*classic*) real spider.

shows the structure of the simplest of those spiders. Typically, this functionality is spread over a few computers to get the required performance.

To solve the drawbacks of 'real'-spiders, three variations were introduced:

specialization, improving results by reducing the number of searched-sites,

meta-spiders, improving results by combining the results of a few spiders or,

meta-information, adding information to (HTML) pages which is to used by spiders for building indexes.

Specialized spiders

Specialized spiders try to improve the results of a spider by manually adding a preselection of subject material. This results in something between a real spider and a manual index. As long as the list of related sites is not too large and well maintained, this will work better than a normal indexing-site with categories. However, a lot of manual interference is required.

Meta-spiders

Meta-spiders call many 'real'-spiders and combine their results, as shown

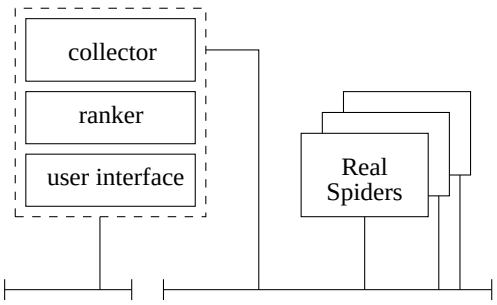


Fig. 2. A meta-spider.

in figure 2.

The meta-spider passes the users request to many real-spiders which query in their own databases and return appropriate results. The meta-spider counts how many real-spiders return a certain page. Then those pages are ranked and returned to the user. This way, the collective result might be better than each separate result.

This procedure may result in less optimal coverage than that of a single search engines: the combined performance leads to the average result, a single engine may be better balanced.

Meta-information

Steps are taken by a sites maintainer to add information to documents to facilitate the spiders. HTML meta-tags `keywords` and `description` are examples for this extensions. In this situation, the publisher of the document is adding human intelligence.

An example of HTML with meta-information:

```
<HTML>
<HEAD>
<TITLE>Wildlife in Holland</TITLE>
 <META NAME=keywords
  CONTENT="bears, apes, rabbits">
 <META NAME=description
  CONTENT="An overview of wildlife
   in The Netherlands.">
<BODY>
 ....
```

The `NAME`-parameters of the `META` are not officially standardized, but are commonly used. Most spiders treat them specially. However, this method relies upon accurate abstraction of the content of a page, which is reliant on the person developing the page. These fields are often abused by sex or car-selling companies to lure visitors.

The Web is also changing from pages mainly being written in HTML into providing pages in multiple formats.

More and more, the documents are published in doc, XML, pdf, etc. In most of those formats, there is no way to specify meta-information.

For all these reasons, this way of adding meta-information has only limited use.

The librarian Way

A different method to providing extra information is by setting-up a library like structure. A few projects try to find solutions for the massive amount of information this way, for instance Harvest[10], CHOICE[11], and DESIRE[12]. These systems add BibTEX-like information to the pages. whilst adding META-tags to HTML-pages is relatively easy, the librarian way requires more effort to capture the content of the page. A site has to run a special service alongside its web-server that supplies the meta-information to the indexing-systems.

But, is everyone willing to become a librarian? For institutes, universities, and such, the effort to comply to this system is not too high, but it is a lot to ask small companies and private persons to use this system. A large part of the knowledge on Internet will never be contained in this information-structure.

3 Identifying the problems

Are we giving-up on real-spiders? Should we try to focus on the quality of 'real'-engines by solving their problems directly?

One of the main reasons why the 'real'-spiders are not able to fulfill our needs to discover information is that their development is slowed down by their current structure. Everyone who wants to experiment with improving them has to re-implement all parts of the system — each time. Bright new ideas can not be experimented with unless a lot of money is invested. Building a useful search engine takes a lot of effort and time.

When 'real' spiders are improved, both meta-spiders and specialized spiders become superfluous. When implementation is easier, more variation can be tested and *"My personal search engine."* is born.

Fetching the pages

Severe problems exist with the way spiders behave with respect to obtaining their raw data. Some of the worst I will mention here:

- Each search-engine needs to retrieve the contents of pages held on web sites on regular bases to build their search tables. Engines which span the whole WWW are taking more and more time to visit all the pages to check their existence and to index their content. As mentioned earlier AltaVista needs

about seven weeks to scan the Net once. Some pages, however, change hourly. Quite some delay!

- Each Spider uses its own method to fetch pages. The first versions are typically a *rapid-firing* [1] one, which is easier to implement but destructive for the visited sites; blocking all access to them for real users of the site. The `HTTP/1.1`-protocol[6] performs much better then its `1.0` predecessor, but seems too complicated for some implementors to utilise successfully.
- Some producers of server-software (no names here) are stimulating web site designers to put all of their pages into a private database. Access to these pages demands a lot from the underlying (operating?) system. These systems are heavily loaded even by a few normal users and any superfluous access should be avoided to keep the system stable.
- There are many spiders. The *Big Search-Engine Index*[7] lists 420 search-engines. Happily not all those spiders cover the whole Internet; some are related to a country, a language, or a subject. No-one is happy when all of these spiders extract all of the pages from their databases over and over again. It can seem that only spiders are interested in your content.
- A large number of interesting sites require registration. Some even require payment. So these sites are not indexed, even if the person who is looking for facts might be willing to register or pay.

The badly-implemented spiders and the exhaustive database-retrievals have forced many sites to *block access* to every spider. All spiders obey the `robot.txt` convention, which is described in [9]. From own count I estimate about 30 percent of all sites currently block all access by spiders. This then looses that site a lot of visitors: because about 35 percent of the people find ones — averaged sized — site via an engine. They may decide to bookmark your site and become a regular visitor. This only works if you are listed!

In summary, there are many reasons for the very *bad hit-rate on pages*. An investigation made by *Search-Engine Watch*[3] shows that even largest engines of today only get their hands on only 8 to 27 percent of the existing pages. And this number is decreasing.

Spider Functions

Each Spider has to develop a page-fetching mechanism before it can concentrate on how to work with the raw data they retrieve. Ignoring for the moment page-fetching, what kinds of data are does the spider supply to the 'user' and how will the searching work?

- Some spiders build indexes based HTML fetched over HTTP alone. Other spiders also supply search facilities on other document formats (XML, PDF, doc), other

[1] A *rapid-firing* spider fetches pages site-by-site. The site it is working on is fully occupied serving that spider, hence not able to serve 'normal' visitors.

protocols (FTP, Usenet), or hostnames. Even some experimental indexing of images is available, for example from AltaVista.

- A number of Spiders do a blunt search on keywords, possibly with some boolean algebra (which few people understand). Others try to build artificial intelligence (or fuzzy-search, or whatever you want to call it) into their indexing.
- Some know about synonyms, and include those in their search. Unfortunately this is usually only implemented in English.
- The ability to specify the language a page is in is common, however often you have no possibility to restrict your search to the languages you know. The meaning of a word can differ between two languages: even if you do speak both languages you need to be able to restrict to one language.
- Most spiders show results by displaying a part of the page which contains the keyword. In some cases, the first few lines are shown. Other spiders show the actual place where the word was found. When the page contains the meta-tag 'description', this is usually displayed, which is often a better representation of the content of a document then one paragraph is.

Many variations on many themes. But... I never find the answer to my questions fast. Why?

Blocked Development

Many bright ideas have been implemented in search engines. Some spiders have a good collection of such functions (as are described in the previous section) and contain a relatively good set of data to apply them to. But at the same time a lot of features are left-out because of a lack of time and financial means, or intellectual property problems. Many promising ideas relating to small parts of the search-process are not implemented because one needs to build a complete search-engine, from page-fetch to display, or have nothing at all.

If we are able to remove some of the blocks to implementing new ideas, we will see a better behavior, hence a more valuable World Wide Web.

4 Redesigning the Engines

Only open architectures can help to extend the current search engines. This way, we might be able to spend time on improving our search techniques instead of reinventing the same thing all the time.

The main parts of a Spider are

(1) the page-fetching mechanism;
(2) the index-building and related search-facility; and
(3) the user-interface.

Why should they be so tightly connected as in current implementa-

tions? We can decouple these parts. See figure 3, where the parts are

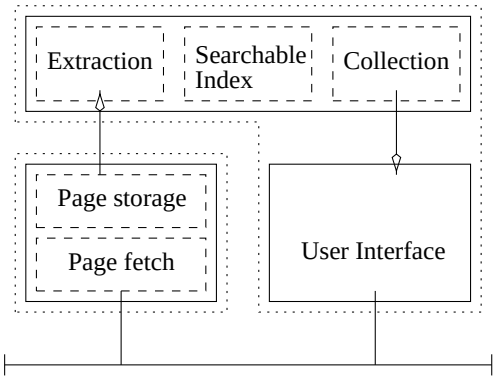


Fig. 3. Restructured spider.

split. In the figure, the index and user-interface are coupled into one machine, but this need not be the case.

When we look at the index-building ('page inventory') and search part, we see a lot of easy-separable functions which can be modularized. The modularization is *the* opportunity to develop new ideas. Modules can be released to the Public Domain, but there may also be commercial products. To 'build' a new Spider you combine a few Public Domain modules, your own modules, and maybe some commercial modules. Run them over the publicly available set of pages, and pass the results to your preferred interface.

This is easy to say but not that easy to implement. Let us now focus on the separate parts of a spider, and see how they could be structured to fit into a m modular system.

The Page-Fetch

We must solve the problem of page-fetching for once and for all. It is totally ridiculous that hundreds of engines access each site. However, it is not feasible to forbid spiders to be built. What *is* feasible is to collect all data on central machines, placed in tactical locations, and allow everyone to attach to that machine to build their spider.

It is not practical to put all of the data on one machine at one point in 'The Web'. The exploding amount of information to be found has resulted in existing (centralized) spiders taking too long between visiting a site. One new huge central place will have the same performance problems.

It will be better to arrange things per backbone, per language, or per country (a combination will work best). [2]

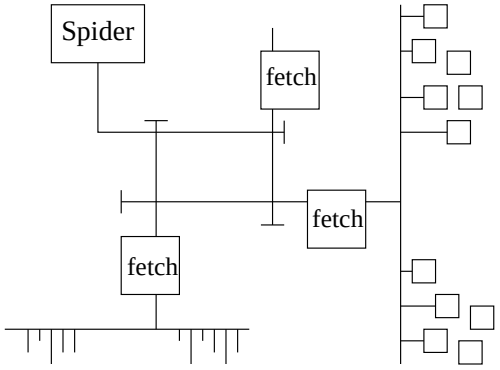


Fig. 4. Localized Fetchers.

Localized fetching facilities (figure 4) will improve the quality of the retrievals because they can be tuned

[2] Of course these are 'virtual servers', possibly hosted on the same machine.

and influenced by people who know about the local situation. For instance, about what hours are best to scan the sites; night-time is different everywhere. The revisiting frequency should be determined by the change-rate of data, not by the size of 'the Net'.

Next to this, more and more pages are made in other languages than English. We have to take into account that more and more Internet users are not able to understand English (and that average American is not able to understand Chinese). The requirement to search "the whole wide World" will decrease. Spiders will localize with it.

Each server (fetcher) repeatedly scans sites on its controlling domain. New spider implementations are informed by the fetchers when they find a changed page which is in the target range of the specific spider. These then fetch their personal copy of this page from the fetcher's storage to process and build their private index with.

Each locally active central fetcher knows were to find the other fetchers. If the storage-server finds pages in a language other than it serves, it passes this data on to an other server which *is* capable of understanding that language correctly.

To be successful, the fetchers must be open systems. They do not require to be very fast: the update to the spiders will be asynchronous with the site-scan. Techniques can be based on current retrieval software.

Some commercial sites require registration of visitors or even payments. Of course, they want to have as many visitors as they can, but they cannot be searched through by current public spiders. In the structure proposed here, we can build applications who deliver the keywords and description of pages of a closed site to the nearest fetcher, which passes this on. Of course, spiders will ask users if they want to pay for information when a search is made, to avoid disappointments.

Another extension can be made to reduce superfluously checking huge sites with mainly static data. A simple application scans the site locally (so on the system where the data resides) and informs the fetcher which pages changed.

This structure has the following advantages over today's techniques:

- Sites can open their doors to being searched again: the current reasons for closing (too many requests by spiders, poor implementations) are resolved. Therefore, participating spiders will get a much better coverage.
- It is easy to build facilities so that web masters can add information to their sites which pleases them: how often to scan the site, which authorization should be used, which parts should be skipped, where it is located physically... When they have to enter this data only once, they will be more willing to provide useful data, than when it has to be entered at hundreds of places. This adds to the trustwor-

thiness of the spider's results.
- Engines are cheaper to build, because they do not need to store all pages themselves, only to process them into indexes. It is practical to locate Spiders physically close to the fetchers, but not required.
- On Internet the traffic will be reduced. Don't worry, no doubt it will be filled again in no time with other applications.

The ISPs and backbone-providers profit when sites are indexed better, because they will be used more. It is not inconceivable that they will be willing to provide the central-storage for their network.

The search-index

The searchable index is the main playroom for commercial and non-commercial developers. Three functions are used in implementing the index. At first, *the extractor* selects the fetched pages which are of interest. This processes the contents of the pages and passes it into *the search-able index* that builds tables to be searched. On the user's demand, data from these tables is assembled by *the collector*, and then passed to the user-interface.

Some implementations will prefer to do a lot of work in the extractor, so the collector has less work and is faster. Some implementations will put more effort into developing the collector, and avoid wasting computer-power because no-one ever asks for the results.

Basically, the extractor and collector will contain a lot of modular functions. A short description of what is likely to be there:

- Translations from HTML, PDF, Postscript, and friends into indexable text.
- Search algorithms: plaintext search, fuzzy search, phonetic search, regular-expression search, ...
- Language detection. Language dependencies.
- Knowledge of synonyms in many languages.
- Translations of searches between languages.
- Site-recognition and site-structure discovery.

There are a lot more functions that could be applied to the data which have no direct relationship to search activities:

- HTML, XML syntax checking.
- Spell-checking.
- Dead-links checks.
- Language scans for dictionary writers.

Of course, we cannot force everyone to give away their own implementations of these modules, but a more open attitude will improve the overall spider's quality.

The User-Interface

The final step is the Human-Computer Interface. This interface accepts requests from the user, translates them into calls for the collector and presents the results received.

The increasing performance of systems enable a complicated exchange of information between the search-engine and the searcher. This is required to reduce the number of wrong answers we get from our initial questions. One experimental interface trying this is described in [14].

Generalization

In figure 5, a generalization is shown of the private versus open model of spiders. Each spider designer needs to find a balance between private development, public domain parts, and commercial services. For each level of public service, there can be many different implementations.

5 Who Benefits

Who benefits from search-engines which use as many public parts as possible? Everyone. The benefits are numerous but most valuable to our modern society is finding the right information. More public facilities to develop spiders will improve development.

But what is the consequence for the parties involved?

The site-maintainers: More sites will open up their pages, because they are less bothered by the engines. With minimal expense, their data can be found. One central administration point is provided where new sites are registered. Access will be optimally implemented, because the software is only written once. Even data on pay- or registration-required sites can be looked-up in all public spiders.

The users: People searching for information profit most: more sites will open up their pages, so more data is searched. New algorithms can be tested easily, as well in interfacing as in retrieval methods. This will give better search engines.

Spider-developers: Organizations which develop spiders save a lot of money and effort, when they easily can hook-up to the fetching system. They can concentrate on what makes them special and spend more money on that.

Service providers: The companies involved in selling Internet-bandwidth (ISPs) will see a decrease in network capacity requirements for the search-facilities, However, when the search works better more people will use the Net (they always proclaim that "usage grows as fast as they can deliver").

A new role will emerge for them to develop and maintain the fetchers and possibly hosting the new spiders, too.

6 Phased Implementation

The need for a new setup is acute. A proposed schedule to transform the way we search for information on the web is presented below:

(1) Start some fetchers. Especially for small languages, this is

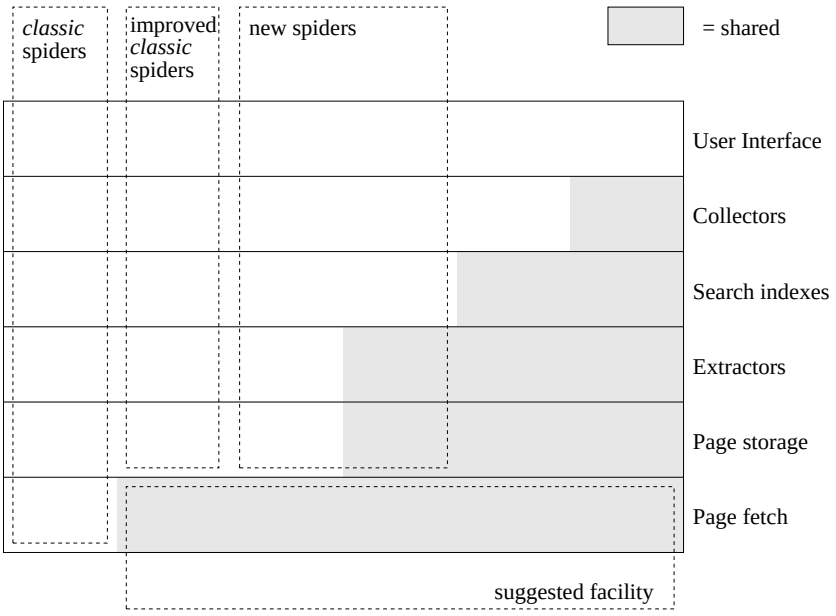


Fig. 5. Generalized spider designs.

rather inexpensive and easily made. For the English language, more work has to be done: one server is not enough to scan all web sites Some free software developers have already produced packages which can be deployed without too many changes.

(2) The storage-servers have to exchange information, for instance when they find pages in a language they do not serve. This protocol has to be developed, but can be based on normal HTTP for the communication.

(3) The interface where administrators can register and specify how their site should be downloaded should make sites to open-up their doors for the central fetcher. This shall be combined with promotional activities.

(4) Modules for the public and private extractors and collectors are already wide-spread. In the Perl libraries (CPAN [13]), for instance, quite a lot of useful modules can be found. The interface to the storage can be very simple on file-by-file bases.

(5) The user-interface based on existing modules can start with a simple plaintext search and textual output although alternatives are being developed. There will be simple interfaces with rough results, which can be used by anyone, but also complex search methods designed for trained librarians or other specialists. An experimental version of such an interface is described in [14].

(6) When a first simple implementation is ready, new spiders should be stimulated to use the configuration. It might be easy to attract smaller spiders: just because they save a lot of work writing code and costs for disk-space and network-access.

(7) When more and more sites open-up for the central fetching, the existing spiders will be more willing to change. Large engines might never be willing to commit to the proposed structure.

7 Conclusions

Naturally, this overview is extremely brief. Many more details are obvious, and some difficult design work and then implementation has to be performed for the distributed fetching of pages. However, this has been done before.

The aim of this paper is to call for reconsideration of whether the current developments in search engines are leading us anywhere useful. Workgroups need to be formed to investigate details.

References

[1] H.C.M. Withagen and M.A.C.J. Overmeer. Internet site of *The Dutch Home Page*. `http://www.dhp.nl/`

[2] Yahoo! Inc. Internet site *Yahoo!*. `http://www.yahoo.com/`

[3] D. Sullivan. Search Engine Watch, Mecklermedia, 1996-98. `http://SearchEngineWatch.com/`

[4] AltaVista Inc. The AltaVista Search Engine. `http://www.AltaVista.com`

[5] Trans-European Research and Educational Network Association (TERENA). `http://www.terena.nl/`

[6] HyperText Transfer Protocol version 1.1, rfc 2068.

[7] Big Search-Engine Index. `http://www.merrydew.demon.co.uk`

[8] BotSpot Inc. *BotSpot:* The Spot for all the Bots on the Net. `http://www.botspot.com`

[9] M. Koster. A Standard for Robot Exclusion, 1994. `http://info.webcrawler.com/mak/projects/robots/norobots.html`

[10] The Harvest Information Discovery and Access System. `http://harvest.transarc.nl`

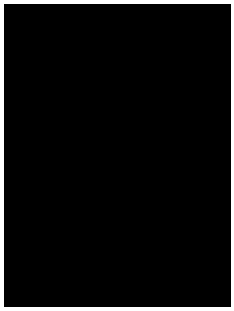
[11] TERENA Task-force CHIC. Cooperative Hierarchical Object Indexing and Caching for Europe (CHOICE) `http://www.terena.nl/task-forces/tf-chic/`

[12] A. Ardö and S. Lundberg. A regional distributed WWW search and indexing service – the DESIRE way. `http://nwi.dtv.dk/www7/`

[13] Comprehensive Perl Archive Network. `http://www.CPAN.org`

[14] M.A.C.J. Overmeer. A Search Interface for my Questions. Proceedings of the *TERENA-NORDUnet Networking Conference 1999*, June 1999.

My gratitude to Duncan Barclay for corrections and improvements.



Mark Overmeer got his MSc in Informatics from the University of Nijmegen, The Netherlands in 1990. Since then, he gained professional experience in maintaining a large variety of UNIX-systems, from tiny to super-computers.
In his current occupation, this knowledge is taught to system-developers and -maintainers at AT Computing bv, a leading training institute on UNIX and UNIX-related languages in the Netherlands.
Next to his professional activities in computers, he maintains a very popular Dutch Internet-site since 1995, and actively participates in development of Public Domain software.
`http://www.dhp.nl/~markov/`