



SCHOOL OF COMPUTING, INFORMATION SYSTEMS & MATHEMATICS

B.Sc. (Hons) Computing Studies

Project 143

iSeeker

A client-side Internet search application

DANIEL FARINHA

May 1999

ACKNOWLEDGMENTS

Firstly I would like to thank Tony Flower, my supervisor, for all the help and enthusiasm during the past ten months. Obrigado!

Secondly I would like to thank Dave Inman for giving me the opportunity of presenting my project ideas to the class. These presentations were highlights in this project, thanks very much!

Thanks also to all the nice people that I worked with during my industrial placement at DEC, especially Ajay Mahorta, Bill Layne, Pete Goodwin, Emma Martin, Grant Lills, Pete Livesey, and Colin Bradbury.

A special thanks to Darren Male for all the Red Alert games that we played while at Digital, but especially for the shared thoughts on our projects.

I am grateful to my flatmate Paul Plumridge, who had to put up with me for the last months. Please do not reveal my secret plans for conquering the world... 😊

Thanks to everyone who showed interest in this project and wished for its success.

And finally, I'd like to thank my parents and grandmother for all the help and support that they give me. I dedicate this work to them.

PREFACE

This report covers all the research and implementation details of the iSeeker project. It is assumed that the reader has a basic knowledge of the Internet, and the World Wide Web (Web) in particular, but no specific knowledge of Intelligent Agents (IA) or any other Artificial Intelligence (AI) area is required.

The report is divided into seven parts:

1. Introduction

Project overview. This section also introduces the proposed solution and how it differs from the existing ones. Finally it briefly describes what technologies were researched and why.

2. Intelligent Agents

After a brief introduction to intelligent agents (IAs), the section describes in detail the main characteristics of agents and their different architectures. It also explains the chosen IA architecture for this project and expands on the different underlying AI technologies involved.

3. Internet Information Retrieval

Basic and advanced web search techniques are given here, followed by a description of the techniques used to rank a web page, including the study of *web-communities*.

4. User Interfaces

A small section that explains the importance of interface design in information retrieval applications.

5. Development Process

A detailed walkthrough of the development stages. Planning, analysis & design, implementation, and testing are all described.

6. Future Developments

The product delivered by this project is the corner stone of a larger project that involves a massively distributed search engine. The main ideas involved in this are described, as well as some technologies that have the potential to improve the product, such as natural language processing (NLP).

7. Conclusion

This is the overall conclusion to the project and will address questions such as: Were the technologies appropriate and well implemented? Did the project achieve its goals?

CONTENTS

1. INTRODUCTION	9
1.1 Introduction	10
1.2 Search Engines and Directories	13
1.3 Meta-Search	14
2. INTELLIGENT AGENTS	15
2.1 Introduction	16
2.2 What Are Intelligent Agents?	17
2.3 Agent Components	18
2.4 Agent's Mobility	19
2.4.1 Fixed Agents	19
2.4.2 Mobile Agents	20
2.5 Agent Communication	20
2.5.1 Message Passing	20
2.5.2 Blackboard Architecture	20
2.6 AI Techniques	21
2.6.1 Fuzzy Logic	21
2.6.2 Genetic Algorithms	23
2.7 iSeeker's Agent Architecture	24
2.7.1 Proposed Architecture	25
2.8 Agent Benchmarking	28
2.8.1 The Problem	28
2.8.2 The Proposed Solution	28
3. INTERNET INFORMATION RETRIEVAL	29
3.1 Introduction	30
3.2 Web Search Techniques	31
3.2.1 Start general	31
3.2.2 Refine after First Attempt	31
3.2.3 A Good Hit is Great	31
3.2.4 Browse Your Findings	32
3.3 Other Tools	32
3.3.1 AltaVista's Babelfish: Online Translations	32

3.4	Web-Communities	33
3.4.1	<i>What is a Web-Community?</i>	33
3.4.2	<i>Web Rings</i>	35
3.4.3	<i>Free Web Space Providers</i>	36
3.4.4	<i>Why Are Web-Communities Important?</i>	36
3.4.5	<i>Proposed Web-Communities Solution</i>	37
3.5	Web Crawling	42
3.5.1	<i>Exclusion Methods (Robots.txt)</i>	42
3.6	Potential Problems	42
3.6.1	<i>Search Engine Exploitation</i>	42
3.6.2	<i>Copyright Issues</i>	43
4.	USER INTERFACE	44
4.1	Introduction	45
4.2	User Interface Guidelines	45
4.3	Boolean Query Interface	46
5.	DEVELOPMENT PROCESS	48
5.1	Introduction	49
5.1.1	<i>Programming Language</i>	49
5.1.2	<i>Development Methodology</i>	49
5.2	Project Plan	50
5.3	Prototypes Specification	51
5.3.1	<i>Prototype 1</i>	51
5.3.2	<i>Prototype 2</i>	51
5.3.3	<i>Prototype 3</i>	52
5.3.4	<i>Prototype 4</i>	52
5.3.5	<i>Prototype 5</i>	53
5.3.6	<i>Prototype 6</i>	53
5.3.7	<i>Release</i>	54
5.4	Development Description	55
5.4.1	<i>Prototype 1</i>	55
5.4.2	<i>Prototype 2</i>	59
5.4.3	<i>Prototype 3</i>	61
5.4.4	<i>Prototype 4b</i>	61
5.5	Implementation Problems	65
5.5.1	<i>Local Internet Files Access</i>	65
5.5.2	<i>Multiple Threads: Worker versus Interface</i>	66
5.5.3	<i>Header Files Redefinition</i>	67
5.6	HTML Parser	68
6.	FUTURE DEVELOPMENTS	70
6.1	Other Search Options	71
6.2	Natural Language Processing	71
6.3	Agents as Plug-Ins	72

6.4	Distributed Search Engine	73
7.	CONCLUSION	74
7.1	Research	75
7.2	Development	75
7.3	Project Plan	76
7.4	Further Progress	76
7.5	Achievements	78
7.6	Personal Comments	78
8.	BIBLIOGRAPHY	79
8.1	References	80
8.1.1	<i>WWW Resources</i>	80
8.1.2	<i>Printed Resources</i>	80
8.2	Bibliography	80
8.2.1	<i>WWW Resources</i>	80
8.2.2	<i>Printed Resources</i>	81
9.	APPENDIX A - SAMPLE SEARCH TOOLS	83
9.1	Sample Server-Side Search Tools	84
9.1.1	<i>General Search Engines</i>	84
9.1.2	<i>General Directories</i>	84
9.1.3	<i>Geographically Local Directories</i>	84
9.1.4	<i>Specialised Directories</i>	85
9.1.5	<i>Meta-Search Engines</i>	85
10.	APPENDIX B - PROJECT PLAN: GANTT CHART	86
11.	APPENDIX C - HTML PARSER STACK EXAMPLE	88
12.	APPENDIX D - EMAIL AND NEWSGROUP POSTS	90
12.1	WinInet: local (file://) files with spaces in the path	91
12.1.1	<i>Post 1 - Question</i>	91
12.1.2	<i>Post 2 - Reply to Post 1</i>	91
12.1.3	<i>Post 3 - Reply to Post 2</i>	92
12.1.4	<i>Post 4 - Reply to Post 1</i>	92
12.2	IA Questions Email	92
13.	APPENDIX E - SAMPLE SEARCH ENGINE COMPONENT (SEC)	95
14.	APPENDIX F - SOURCE CODE	97

1 INTRODUCTION

1.1 Introduction

The aim of this project is to produce a tool that will help people to search the Internet more efficiently.

In opposition to most search engines, the proposed solution is a client-side application, which means the program will run on the user's own PC and not on a remote server.

In order to achieve our goal, research was conducted on the current problems involved in web search, the existing solutions, and technologies that might enhance the proposed solution.

The three main technologies/areas covered in this report are:

- Intelligent Agents
What agent technology is and why it is suited to information retrieval.
- Internet Information Retrieval
Techniques used by experts in web search and how these can be applied as heuristics in an agent.
- User Interfaces
The importance of user interfaces specially designed to Internet information retrieval and existing research in this field.

Other subjects covered within the main areas are *fuzzy logic*, *web-communities*, and *live-update technology*.

As part of the development, an attempt has been made to implement some of the ideas and technologies covered in this report, with the aim of demonstrating the overall viability and expected success of such a tool. Due to the short period of time available to this project, it was not intended for the implementation to reach the quality of the existing commercially available tools, but a minimum to demonstrate the main concepts.

Incremental prototyping was chosen as the development methodology since it is considered to be the most flexible.

The initial plan was to divide the development into seven prototypes but this proved to be unrealistic due to the short time available. Only two prototypes were successfully completed, with another two partially implemented.

The report concludes by discussing in detail all the features that could not be implemented for lack of time or other problems, as well as the planned future developments. This section also covers the ambitious idea of a large-scale distributed search engine, as a sequence to this project.

The Internet is considered the most important technology in the recently established information age. Today, many of us already see the Internet as natural as the Television and the Telephone, and use it for the most varied of reasons and with the most varied of objectives in mind. All the information that one could wish for is available at the end of the fingertips, and every day the amount of information available grows exponentially.

However, such rapid growth is also proving to be a problem. Since there is no organised way in which to store the information on a macro-scale, the task of finding the relevant information is becoming harder every day. As opposed to a library, where all the documents are indexed and stored in well-defined places, the Internet is generally unstructured. There is not a 'librarian' who keep records of all the documents published on the Internet. Anyone can publish anything at anytime without letting anyone know about it. This means that people who might be interested in such resources will not know where to look for them, or even if they exist at all.

Some attempts are being made to solve this problem. Search engines and directories are useful attempts at indexing some of the Internet's content, but they cannot cope with the rate at which the Internet is growing.

ARCHIE Since 1990 with Archie* until nowadays with massive search engines like AltaVista and Excite, and equally impressive directories such as Yahoo and the About.com, Internet users still keep on struggling to find the best resources in the minimum amount of time. I will briefly discuss the differences between search engines and directories later in this chapter.

A study[†] by Steve Lawrence and C. Lee Giles from the NEC Research Institute [NEC WWW] reveals that the search engine coverage is also very limited.

The experiment revealed that by combining six of the most popular search engines we still just get a coverage of about 60% of the "publicly indexable Web", which was also

* For more information on Archie visit <http://www.bunyip.com/products/archie/>

† <http://www.neci.nj.nec.com/homepages/lawrence/websize.html>

estimated to be 320 million pages. By publicly indexable it is meant all the pages that can be indexed by a search engine. This excludes all the password protected pages and dynamically generated pages (e.g. server side scripts).

HotBot, the best-ranked search engine of the six, had an estimated coverage of less than 35%. This means that combining the results of the various known search engines is not only a good idea but also an essential requisite for a good search.

META-SEARCH This combining of multiple search engines may be achieved with a meta-search. We will go into more detail later in this Section.

Lawrence and Giles also mentioned another problem: the poor freshness of search engines. Many pages change location or simply disappear without the knowledge of the search engines, meaning the engines will also keep a number of invalid records, also

BROKEN LINKS known as broken links.

On the study for recency, HotBot reached the top place again, but this time as the worst search engine, with over 5% of the index containing broken links.

1.2 Search Engines and Directories

Both Search Engines and Directories aim to index the web, but they are very distinct in their approaches.

A directory relies on people who manually inspect web pages and add them to the directory under the most appropriate category/section. Since humans inspect all the pages before being adding them to the directory we can normally expect pages of a good standard.

The bad side is that the manual procedure is very slow and cannot cope with the rate at which the web is growing.

Search engines on their turn do not rely on humans but on a computer to do the indexing.

WEB CRAWLER The computer, commonly known as a web spider or crawler, automatically looks for new pages by following the links from the pages that it already indexes. This process is much faster than that of directories, resulting in far larger and up-to-date index.

However the quality of the pages may not always be the best, since the web-crawler lacks the evaluative capacity of a human.

As a summary, directories offer quality while search engines produce quantity. Each will be useful in different types of search, as we shall discuss in Section 3.

1.3 Meta-Search

As I mentioned earlier, a meta-search combines more than one source in a single search. For example, instead of submitting separate queries for different search engines, we can perform a search that uses all the known search engines and directories at once (Fig. 1)

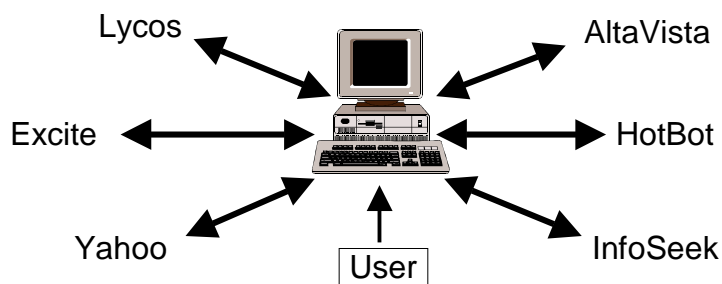


Fig. 1 – meta-search

This saves the user a lot of time, since she can search all the search engines with a single query.

One obvious problem that occurs in a meta-search is that some pages will be indexed by more than one source, since search engines and directories overlap in their mapping of the web, resulting in duplicate results for a search. A good meta-search program should identify such duplicates and keep only one occurrence of each page. Most of the online meta-search engines (server-side) do not allow this. Most are very archaic and only append the output of each search engine on a single page (including the advert banners!)

2

2 INTELLIGENT AGENTS

2.1 Introduction

Intelligent Agents (IA) technology is based on autonomous entities with a personal state, a set of beliefs and a behaviour.

IAs are a part of the research in Distributed Artificial Intelligence (DAI) which combines Distributed Computing with Artificial Intelligence (AI).

The research in IAs dates back to the late seventies, focusing on the macro issues of collaborative agents. However it wasn't until the early nineties that the research in IAs boosted, with the diversification in the types of agents being investigated.

Now, during the late nineties, IAs enjoy some of the greatest hype ever surrounding an area of research in AI. Such hype is partially fuelled by the Internet, the perfect world for agents to exist, multiply, co-operate, and roam freely in the vast computer networks.

The present hype surrounding IAs, while raising awareness to the field, has some disadvantages. A part of the AI community seems to be against the recent popularity surrounding IAs. Either because they prefer to keep AI as an 'underground' subject, or simply to prevent the mistake that happened in the past with expert systems from happening again, with the initial hype being followed by a global disappointment, resulting in bad reputation to AI.

Another disadvantage is the fact that many companies want to 'force' this technology into their products, most of the times just to serve as a marketing slogan. This results in badly designed systems, which will not meet the customer needs. IAs will then be blamed for the failures. In other situations, developers will claim that they developed IAs all along, when this is not entirely true, also leading to a number of different flavoured definitions of Intelligent Agents.

This section will address the general concepts of Intelligent Agents, existing architectures, and some of the AI techniques that provide the *intelligence* to agents.

2.2 What Are Intelligent Agents?

“Intelligent Agents are self-contained entities which interact with their environment in order to achieve their goals.”

This is our very own definition of intelligent agents, and it is a very simplistic view, but it illustrates the most important concept: autonomy.

An agent is:

- An autonomous system
 - It has total control over its own state and external behaviour
- Situated in an environment
 - Receives perceptual information from environment and acts upon it

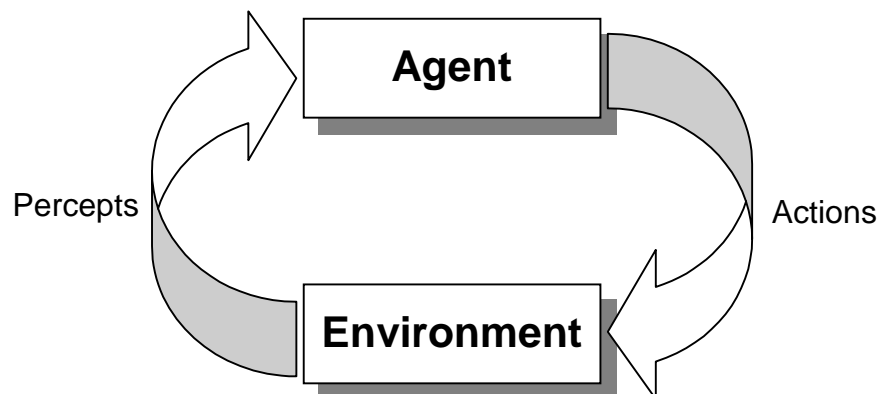


Fig. 2 - agent-environment relation

As mentioned before, agents have become the most recent artificial intelligence related hype; therefore the definitions of agents are becoming as diverse as one can imagine. Agents can be whole software applications working on behalf of their master, the user. In other situations, agents can exist in the physical world (i.e. a thermostat fits all the requirements of an agent and is purely physical; autonomous robots also fall into this category).

The tool implemented in this project, iSeeker, can be seen as an information retrieval agent. However we would like to go into a lower level and focus on intelligent agents as a development methodology used in the internal structure of the application.

Agents can also be seen as logical extension of objects in Object-Oriented Development. Not only does an agent have an internal state, but also a set of personal goals and behaviour.

2.3 Agent Components

Regardless of the agent architecture chosen for a system, agents should typically contain a set of internal components. These define the agent's type, state and behaviour.

Knapik [KNA 98] lists several of such typical components. Here are a few:

- Agent name
Unique ID
- Owner
User name, parent process or master agent name
- Goal
Goal statements, measures of success
- State
A single or a combined set of values which represent the agent's overall state
- Authorisation
Authorisation required by agent to access protected resources

These general components can be included in general agent classes, since most of the derived agents will contain them.

2.4 Agent's Mobility

2.4.1 Fixed Agents

Fixed agents are perhaps the less exciting kind of agents since they cannot physically move from machine to machine. With a bit of imagination we can fit most of the self-contained and autonomous systems in this category. A UNIX daemon, for example, is a good example of a fixed agent. Daemons only sit listening to a particular port. When something happens, the daemons wake up and execute their mission (whatever that may be).

Even when agents are said to 'crawl' the web, most of the time they are fixed agents, accessing the web pages from the client where they were created. iSeeker's agents fall into this category.

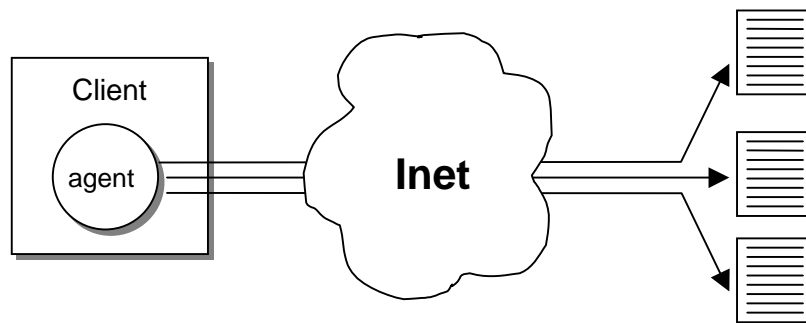


Fig. 3 – fixed agent crawling the web from local client

2.4.2 Mobile Agents

Mobile agents can physically move from one host to another. This means the executing code in the origin host will upload an exact copy of itself onto a remote destination host and then destroy the original. The agent then resumes running on the new host.

This category of agents allows users to upload their search agents to dedicated servers on the web (also called ‘agent nurseries’) to carry out the search, while the user is offline.

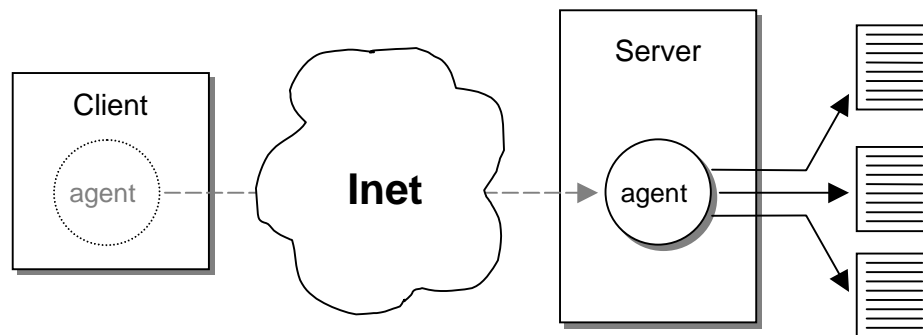


Fig. 4 – mobile agent crawling the web from remote server

2.5 Agent Communication

2.5.1 Message Passing

In an object-oriented architecture, message passing is the typical communication method, since it is very efficient. The agents involved are probably internal components of the same system, and know exactly which other agents they need to interact with in order to achieve their goals. This is the typical communication architecture for fixed agents.

2.5.2 Blackboard Architecture

However when it comes to co-operation between agents from different sources (i.e. agents produced by different companies, with different skills and goals) message passing is not a feasible solution.

This last scenario applies when mobile agents are uploaded to an agent server and need to communicate in order to share information.

In these cases the blackboard architecture is the typical solution.

While message passing in OOD is similar to a group of people working in separate rooms and sending private messages between one another, the blackboard architecture resembles a group of people in the same room scribbling information on the same blackboard. When one person/agent writes something, the others can see it.

This way, an agent can post advertisements both for what knowledge it has and what it needs, allowing further negotiation between agents that benefit from sharing knowledge.

This architecture also allows the collaborative solution of problems, with different agents solving different parts of the problem.

2.6 AI Techniques

Intelligent Agents can only be called *intelligent* if they perform a task, or achieve a goal, that would require intelligence if performed by a human.

In order to supply a degree of intelligence to agents, developers can employ most of the well-established AI techniques as part of the agent's internal architecture.

We discuss some of these techniques in the following sections.

2.6.1 Fuzzy Logic

CLASSICAL LOGIC Classical logic is limited to Boolean values such as *true* or *false*, *yes* or *no*, *on* or *off*. However the real world cannot be represented in such a simplistic way.

For example, representing people's height:

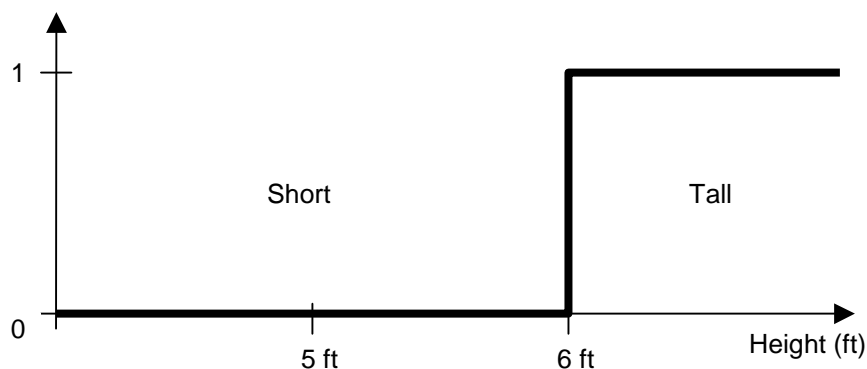


Fig. 5 – classical (Boolean) logic representation of height

In this representation, only those 6 ft. and higher will be considered 'Tall'. One inch short of 6 ft. and the person will be categorised as 'Short' along with people below 5ft. This is clearly not a suitable representation to model a world as ambiguous, and imprecise as ours.

FUZZY LOGIC Fuzzy logic (also known as fuzzy theory) allows a system to cope with the vague and uncertain information that characterises our world.

It does so by attributing degrees of *truth* or *membership* to the elements being analysed.

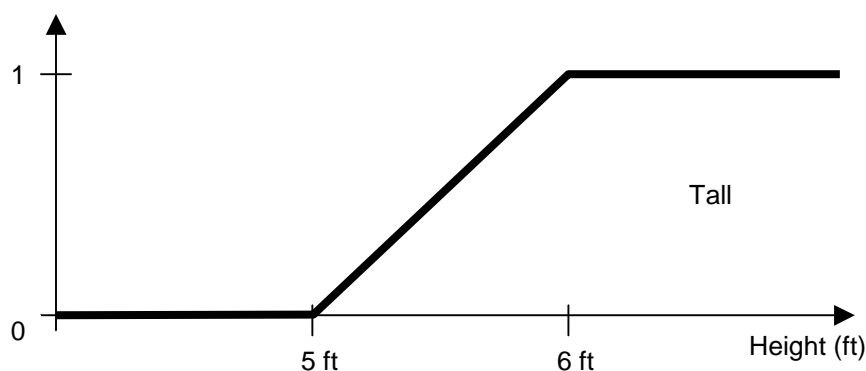


Fig. 6 - fuzzy logic representation of height

As we can see from the figure above, persons with height between 5 and 6ft. have a degree of membership between 0 and 1. The following is the common mathematical description:

$$\mu_{\text{Tall}} = \{(x, i) \mid \text{Tall}(x) = i, 0 \leq i \leq 1\}$$

Every x is an element of the fuzzy set *Tall*, with a membership value i between 0 and 1.

**MEMBERSHIP
FUNCTION**

Some of the element's membership degrees may be defined by a membership function.

The membership function depends on the particular implementation of the fuzzy set. In the example above, the membership function used to calculate the height is linear. This is not necessarily always the case.

In the section 3.4.5.1, 'Fuzzy' Membership, fuzzy set theory is used in the proposed representation of Web-Communities.

2.6.2 Genetic Algorithms

One of the main benefits of using agent architectures is being able to produce slightly different agents to solve a common task, and decide at run time which type of agent is having the most success.

Genetic Algorithms (GA), an area of AI which is based on the principle of natural evolution, may be a good solution for the breeding of successive better agents, based on the user preferences and overall success in searches.

By applying a pre-determined set of heuristics to an initial generation of agents, and with the progress of a search, the best agents can be combined to create better generations.

In order to achieve this, the application must choose the fittest agents, so that crossover operations using these *parents* will result in improved *offsprings*.

A way of selecting the fittest agents is described in section 2.8, Agent Benchmarking.

2.7 iSeeker's Agent Architecture

Before describing the agent architecture chosen for this project, we will start by outlining the high-level architecture of the application:

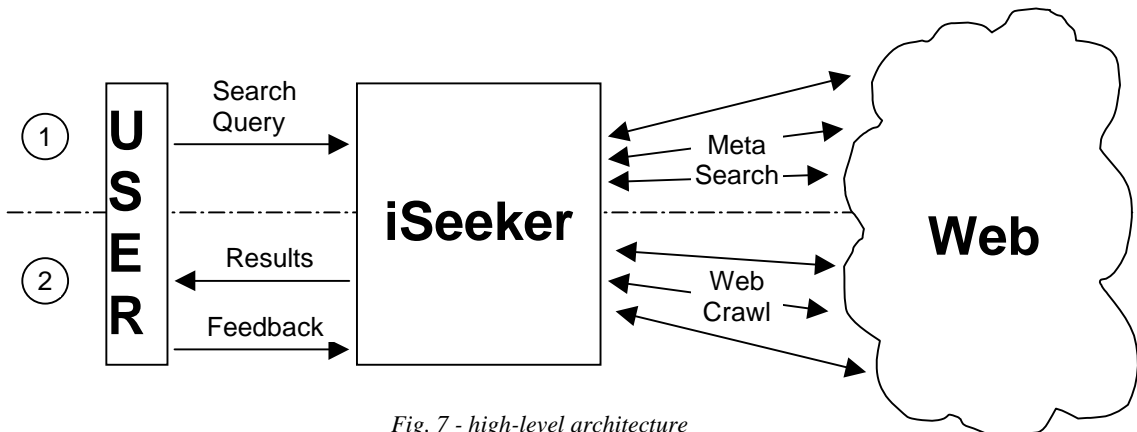


Fig. 7 - high-level architecture

In Fig.7 we can see the two basic steps that occur in a typical search:

1. Meta-Search

The user supplies a single search query.

The program sends the same query to the different search engines.

After receiving all the results from the search engines, the program removes any duplicates, opens the pages and applies own criteria to rank the pages accurately.

2. Web Crawl

While opening the pages in step 1, the program builds a list with all the links present in those pages. The links should be ranked just like the pages are.

The program opens the links and adds the new pages to the results list.

During step 2 the user can give feedback which the program uses to refine the search and rank the results more accurately.

2.7.1 Proposed Architecture

In an early architecture, the number of agents was limited to the number of concurrent Internet connections allowed by the user.

This happened because the *InetConnection* ('connect' in the figure below) was a component of each agent, as shown in the following diagram:

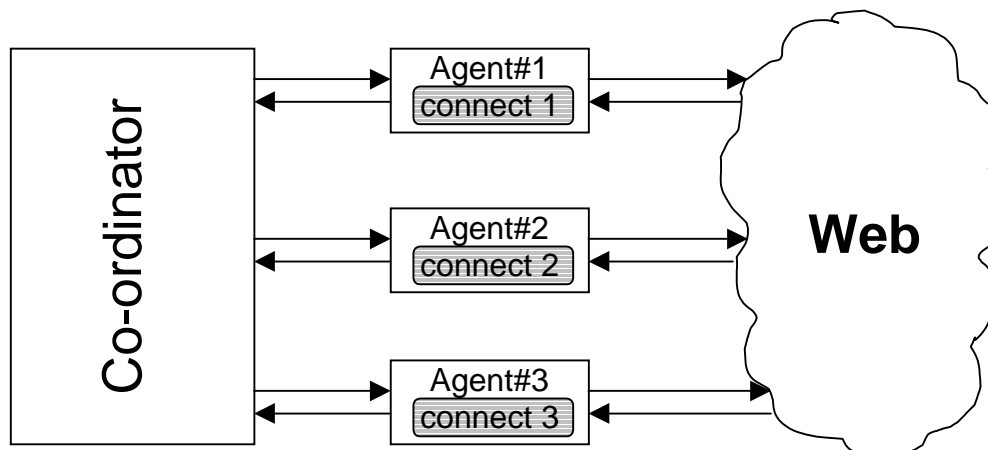


Fig. 8 - three agents with 'built-in' connections

This early approach has the disadvantage of creating a limit in the number of agents, which is not necessary. In practice, each agent does a lot more work than just fetching pages from the web: they must parse the HTML file, search for keywords and other search criteria, apply user-specified filters, calculate a score for the page, 'decide' on which link to follow, communicate with the co-ordinator, etc.

With this architecture, each agent will spend a large amount of time without using the Internet connection, therefore wasting a useful resource.

Ideally, we should have as many agents as we wish and share the limited Internet connections by all the agents. This allows for an optimised use of the maximum concurrent connections pre-set by the user.

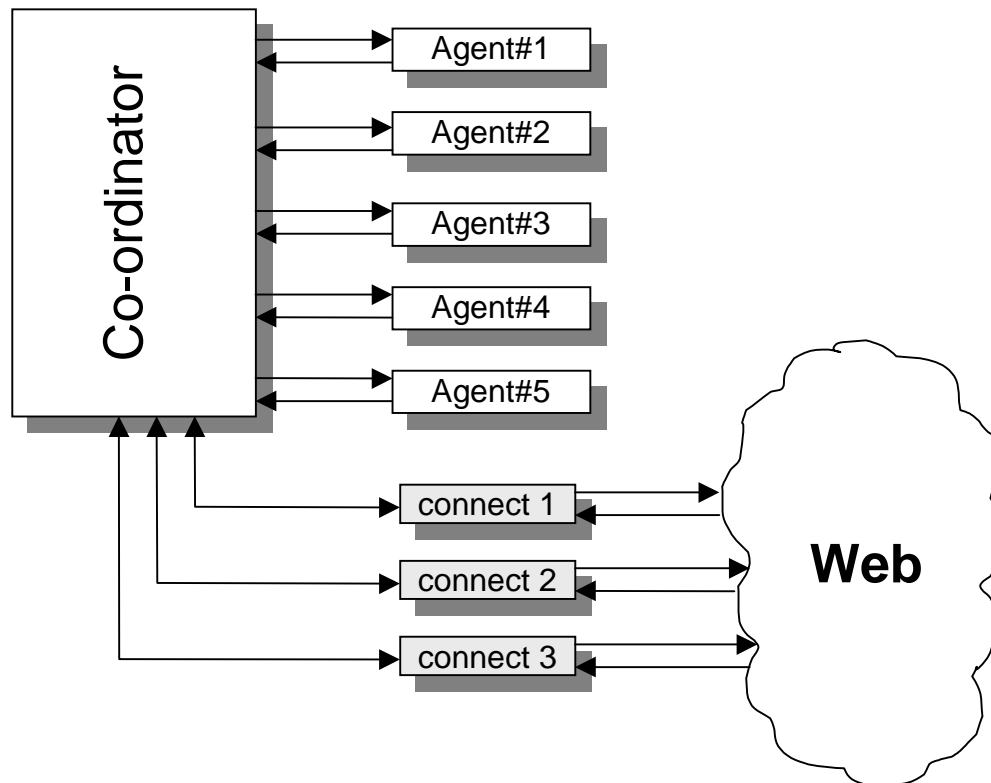


Fig. 9 - revised agent architecture

Now each agent can request an Internet connection to the co-ordinator. The Co-ordinator will manage the available connections, placing agents on a queue if all connections are busy.

The main structure of this architecture was implemented in the revised prototype#4, described in section 5.4.4.

2.7.1.1 Why multiple connections?

There are two main reasons for using multiple connections to the web: Firstly, if the client is accessing the web via a T1 connection, or other equally fast medium, the more connections will mean a larger combined access bandwidth.

Secondly, in the case of a slow connection via a modem, multiple connections will also speed up the overall access time. Even though the overall bandwidth is divided by the multiple connections (meaning each connection will have a reduced speed), the fact that a significant amount of time is spent waiting for server responses and not actually downloading data justifies the multiple connections. If only one connection was used, large amounts of time would be wasted without actually making use of the available bandwidth.

The optimal number of connections depends on the connection speed available. The larger the bandwidth, the more connections should be used.

In iSeeker, the user should be able to set the maximum number of concurrent connections.

2.8 Agent Benchmarking

In order to evaluate the performance of each agent in a fair way, agent benchmarking seems to be a good solution. This could be the basis for selecting the fittest agents in a Genetic Algorithms implementation.

2.8.1 The Problem

Knowing that the co-ordinator agent has to decide which agents are performing well or not, we had to come up with a fair way of evaluating the agents.

For example, a typical problem is that an agent who has the best ‘approach’ for a particular search (in the form of heuristics or ‘genetic code’) may suffer from having a poor starting URL and retrieve equally poor results. On the other hand, a ‘not so good’ agent may have got lucky with the initial URL and score higher.

2.8.2 The Proposed Solution

The proposed solution to this problem is *Agent Benchmarking*:

The co-ordinator should perform regular benchmarks, by submitting **the same** URL to all agents at a given time.

Such URL should be one, which the user has already given relevant feedback to (either positive or negative).

The agents will be asked to rate the URL and the ones who get the closest match to the user’s feedback will score higher.

An interesting idea is then to assign the best URLs already found to the best-rated agents, as well as duplicating the ‘genetic code’ of the best agents and destroying the weakest ones.

This benchmarking process is another very good reason for user feedback to be considered important in this application.

3 INTERNET INFORMATION RETRIEVAL

3.1 Introduction

Any intelligent Internet information retrieval tool such as iSeeker should contain a set of good search techniques that the agents can apply during a search.

As described in the previous section, different agents should be trained for different functions, so it is only natural that different search techniques should be devised to fulfil the multiple search needs of the users.

Reva Basch in an interview* to About.com (old MiningCo) stated that "*What [technique] works beautifully for one kind of search is often worthless for another.*"

In this section we cover some of the search techniques used in a general web search.

Most of these techniques were acquired by my own experience.

After many hours of research into web search and cognitive psychology work in this area I arrived at the conclusion that each one of us builds a significantly different mental model of how to search the web. However, different mental models may be equally successful since the meaning of *success* is itself very subjective, depending uniquely on the user.

For this reason the reader is free to disagree with the techniques mentioned below. These are my techniques, which evolved during approximately four years of personal experience, and are by no means the magical answer to web search. Since I am not an expert in introspection I may also miss out on small details or procedures that became automated with the years.

Ideally, different mental models should be *coded* into different sets of heuristics to be used by different agents. This way the application will be able to select those agents with the mental model closest to the user by applying the benchmarking described in the previous section.

Towards the end of the chapter some of the existing tools that may be useful in web search are described, as well as web-communities and potential problems such as search engine exploitation and copyright issues.

* http://websearch.miningco.com/library/weekly/bl_990108d.htm?pid=2825&cob=home

3.2 Web Search Techniques

3.2.1 Start general

This contradicts many experts, who prefer to narrow searches from the beginning and expand as they go along.

Personally I start with a general search because I can immediately identify problems with the initial query, such as keywords that I should exclude, by looking at the top hits. This would be impossible in a narrow search, which generates no hits.

3.2.2 Refine after First Attempt

If you got all the pages that you need in the first attempt then consider yourself very lucky. Most of the times this does not happen.

Usually I refine the search by excluding keywords from irrelevant pages that topped the first search, and perhaps throwing in a few more keywords of my own.

If after three to four searches I still get only pages that do not relate to my subject, then adding more exclusion words will not make things much better. I stop and think again about alternative keywords to start a new query with.

However, if I get at least one or two pages which relate to my subject (even if only remotely), I examine those pages for useful keywords that I could use in my query.

3.2.3 A Good Hit is Great

When I find a really good hit, I feel a lot happier. This is because I can then use the *link:* field in my query to find pages that link to that hit. (E.g. `link:www.sbu.ac.uk` will retrieve only the pages which link to South Bank's home page).

Most of the times you will find pages very similar to the original one.

The more pages I find the best. For each page I do a 'link:' search and find other related pages, in a vicious circle.

3.2.4 Browse Your Findings

At some stage I stop the search engine queries and start browsing through the pages that I found so far. My aim then is to find pages which haven't been indexed by the search engine. This means I can keep using the 'link:' search in the search engine while I browse the pages.

This is another chance to find synonyms of words that I didn't think of. Further searches will take place until I am happy with the results.

This iterative process is not infallible. Some times I do not find any relevant pages to start with and will eventually get bored.

I then go back and use one of the old techniques. This is to include the '+links' string in my queries. It is very different from the 'link:' field search. It is just a search for the 'links' word on the pages. The reason is that pages with '*Other links to blahblah*' are good starting points if you are searching for *blahblah*. These are like mini-directories created by people, which means the links will be of good quality.

A variation is to use '+title:"links"'. This looks for the word *links* on the title of the documents.

3.3 Other Tools

3.3.1 AltaVista's Babelfish: Online Translations

With the increased quality in translation software, the conducting of searches in multiple languages could result in a wealth of information that otherwise would be inaccessible to most users.

With a service like AltaVista's Babelfish^{*}, an online multiple language translation system powered by Systran[†], any search tool can expand a search to a number of different languages.

A search tool could use Babelfish to translate the initial search query keywords to all available languages, perform the search in those languages, and then use Babelfish again to translate the resulting pages back to the original language.

The system suits these two types of translations very well because it allows translation of separate words (useful for search keywords) and also translates whole web pages, leaving the original layout intact (useful for translating the search results).

However at present multi-language search is not a reality. This is because single keyword translation is bound to be inaccurate, since it lacks the context of a full sentence or paragraph. Without the context, ambiguity cannot be resolved, and serious errors in the translation will happen too often.

Searching for a wrongly translated keyword would generate an unacceptable amount of irrelevant hits.

In the future, when queries in natural language become a reality, multi-language search may become feasible.

3.4 Web-Communities

3.4.1 *What is a Web-Community?*

WEB-COMMUNITY The term *web-community* defines a group of people present on the Internet that has one or more interests in common.

^{*} <http://babelfish.altavista.com/cgi-bin/translate?>

[†] <http://www.systransoft.com/>

In practice web-communities are identified by a number of web pages that are very closely related, either by reciprocal links or by being hosted in the same server, dedicated to the particular interest/subject.

Good examples of both types of communities are *web rings* (chain of web pages) and free web space providers Tripod and Geocities. These will be described in detail.

Web communities play an important role in Internet search, as explained below in section 3.4.4.

3.4.2 Web Rings

WEB RING The first web ring was created in 1995 by 17-year-old Sage Weil [GAT WWW], and consisted of a number of web pages linked in a chain by a common web ring *browser*.



Fig. 10 – ‘Elvis Remembered’ web ring browser

In Fig. 10 above, we can see an example web ring browser used by a ring dedicated to Elvis Presley*. Each site in the ring will contain this browser (normally on the main page) allowing users to navigate through the ring using the links provided.

Ideally the users should be able to circle the whole ring by clicking the ‘Next’ link, however sometimes rings are broken (pages disappear, or don’t include the browser). In

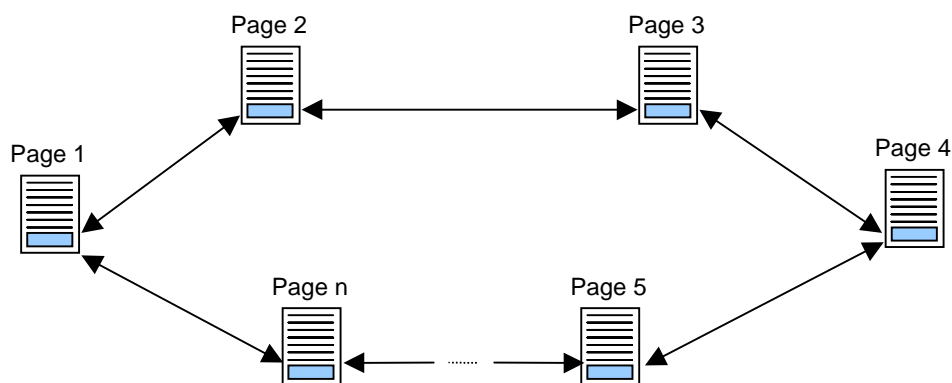


Fig. 11 – a typical web ring

* Elvis Remembered web ring home page: <http://www.geocities.com/SouthBeach/Port/8910/Elvis.html>

these cases, the links '*Skip*', '*Random*', and '*Next5*' allow the user to jump existing gaps.

3.4.3 Free Web Space Providers

Tripod and Geocities are popular free web space providers. People register for free and are able to build their web sites with little effort, since most of these large servers provide web page building tools.

When registering, the user has to choose a community within the server where to place the web page. This way, many related sites will be clustered in the same community.

These large servers also normally keep a directory of all the sites hosted, organised by communities/categories, allowing the easy browsing of pages related to the same area.

3.4.4 Why Are Web-Communities Important?

Web-communities are important in Internet search because they provide a higher level of abstraction of the search space. Instead of treating all the pages as individuals, the Internet search tool can create larger clusters and assign a relevance value to each one.

The clusters allow the distinguishing of subjects with common keywords and, combined with user feedback, the tool can *decide* which cluster(s) are more valuable for each particular search. For example:

The user types the query *dolphins*, with no other keywords.

Most search engines will retrieve many pages related both to the American football team *Miami Dolphins*, and pages related to the mammal.

A good search tool should identify the two distinct communities by the way the pages are linked. If the user supplies positive feedback to one of the pages, the other pages in the same cluster should also be positively affected. The same goes for negative feedback.

So if the user is looking for information on the American Football team, the *Miami Dolphins* cluster will raise in the rank (due to positive feedback) while pages related to the mammal will drop.

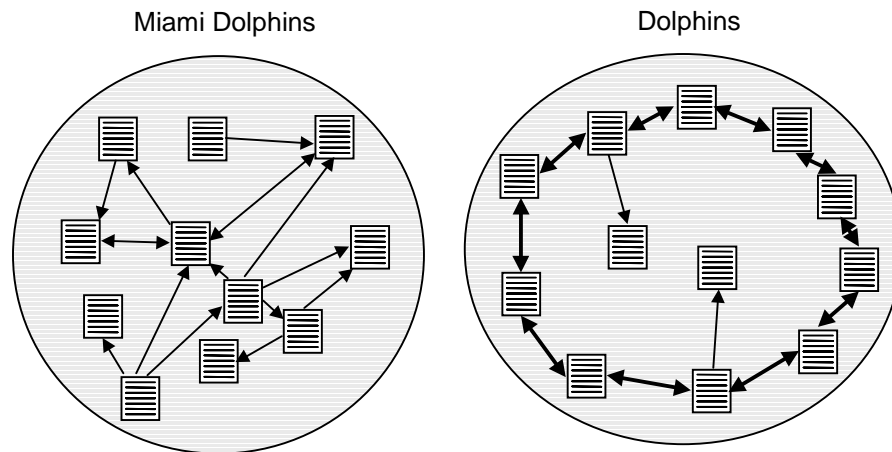


Fig. 12 - example of web-communities

In the example above, we can identify the two distinct clusters by the way the pages are linked. In the right we can even distinguish a probable web ring (marked with thicker arrows).

It is important to emphasise that the identification of web-communities is only useful when allied with iterative user feedback. Without such feedback, the tool cannot decide by itself, which clusters interest the user.

3.4.5 Proposed Web-Communities Solution

A tool such as iSeeker will benefit from identifying and analysing web-communities. This section describes the proposed solution for this project.

3.4.5.1 'Fuzzy' Membership

In larger communities one can find sites that are very popular (many outgoing and incoming links) and other less popular which may only contain a single link. In such a case the rating of the community should have a stronger influence on the most popular sites while affecting less the weaker members.

This means that when deciding whether a site is a member of a community we should not be limited to the classical Boolean logic values *true* or *false* (in this case *member* or *not member*). Instead we should calculate a membership value within a certain range (i.e. weak member, average member, strong member, etc).

Fuzzy logic, previously described in section 2.6.1, is a technique well suited to this problem, since it allows the representation of an uncertain degree of membership.

Each web-community found is represented by a fuzzy set '*communityN*'. Every web site '*siteN*' has a degree membership '*i*' in each community, between 0 and 1.

A site with membership degree 0 is said *not* to be a member of that community (meaning it doesn't have any links with the community).

$$\mu_{\mathbf{F}communityN} = \{(siteN, i) \mid communityN(siteN) = i, 0 \geq i \geq 1\}$$

In order to calculate the membership degree of *siteN* for the *communityN* fuzzy set we need to know how *siteN* links to other members of the same community:

$$LnkOut(siteN) = \{\sum x \mid x \text{ is linked by } siteN\} \text{ (sum of outgoing links)}$$

$$LnkIn(siteN) = \{\sum x \mid x \text{ links to } siteN\} \text{ (sum of incoming links)}$$

$$member(x) = \{x \mid x \in X, \mu_{membership}(x) > 0\}$$

member(x) is the subset of all the members of a particular community.

Now we can devise the membership function for our fuzzy set:

$$\mu_{\text{membership}} = \frac{\text{LnkOut}(\text{siteN}) + \text{LnkIn}(\text{siteN})}{\sum \text{member}(x) - 1}$$

Using this function we can calculate some fictitious membership values:

LnkOut	LnkIn	Total Members*	Membership Value
15	5	201	0.05
100	100	201	0.50
5	0	201	0.01
200	200	201	1

Table 1 – sample membership values

As we can see from the table above, a higher number of links means a higher membership value. The maximum membership, 1, can only be achieved if the site has reciprocal links with all the other community's members (by reciprocal we mean, both outgoing and incoming links).

3.4.5.2 Web Page & Web-Community Rating

The calculation of the site's membership degrees is only useful when combined with the community's own weight. Each community's weight is calculated from the feedback supplied by the user.

The overall score of a page is calculated from 3 values:

1. Internal Score
Based on the internal content of the page, i.e. keyword occurrences
2. User Relevance Feedback
Direct feedback supplied by the user (either positive or negative)
3. Community Influence
Product of the page's degree of membership with the community's weight.
Each page may be a member of multiple communities. In these cases, the sum of all community's influences is calculated.

* Total Members includes the page being evaluated

PageX Score				
Internal Score	User Feedback	Community Influence		
		Membership Degree	Community. Weight	Total Influence
15	0	0.04	-5	-0.2

Table 2 – example page score

In the example above, *PageX* scored 15 points with its internal content. The user has not supplied feedback to the page so the *User Feedback* value is 0. The page belongs to a single community, with membership degree 0.04. This value is multiplied by the community's Weight (-5 as a result of negative user feedback to community) resulting in a *Community Influence* of -0.2.

Adding this value to the page's Internal and User scores we arrive at the page's total score: **14.8**.

The weight of a community is the average of all its member's *User Relevance Feedback*.

3.4.5.3 Potential Problems

The proposed web-communities solution does have problems, for which there are no obvious answers.

For example, when should we identify a cluster of pages as a community? Should one site linking to another be considered a community?

And how should we define a site in the first place? Individual pages should be considered sites? Obviously not since one site may contain many pages.

One possible solution is to consider the domain host as a site. (i.e. www.neuron.co.uk is one site, and www.microsoft.com is another). However, there are domains which host many distinct sites (ie. www.tripod.com and www.geocities.com), and we do want to identify every site within the same domain.

Another question for which there is no easy answer is the boundaries of a community. What happens if, for example, a site links to two opposed domains (i.e. a site, which links both to the Miami Dolphins, and the National Geographic sites). Should we merge

the two communities? The answer is clearly not, but this answer doesn't apply to all the cases.

Concluding, web-communities are with not doubt a useful tool in web search, but further research must be undertaken to answer some of these fundamental questions.

3.5 Web Crawling

3.5.1 Exclusion Methods (*Robots.txt*)

One way to prevent current indexing web crawlers from indexing whole sites is to include meta information on web pages of the type

```
<meta name="robots" value="nofollow">
```

or other crawler exclusion methods such as the `robots.txt` file which lists web files to be ignored by the crawler.

As a web crawler, should iSeeker identify such exclusion methods?

Perhaps it could be left to the user to decide, as a setting (with the default being compliance with the standard exclusion procedures).

3.6 Potential Problems

3.6.1 Search Engine Exploitation

The problem of using a search engine as a 'hidden' source of information without displaying any sort of banners on the main application interface is well known. Search Engines get their revenue from such advertising.

If iSeeker makes use of a certain search engine, it should provide the same with some sort of 'payment'. The implementation of a 'Properties...' dialog where the user can customise each Search Engine, should display in an 'About' tab information about the Search Engine, with a link to its main site. However this might not be enough if an increasing number of users stop loading the Search Engine's page and use only iSeeker's front end.

On the other hand, iSeeker would get extremely slow if it was to display all search engine's banners during a search, as well as being annoying for the user.

A possible solution: iSeeker would indeed display each search engine's banners in the freeware version. If a user wants to discontinue the adverts, she will have to buy a registered version of the software. For every registered user, iSeeker would have to pay the Search Engines. Perhaps, pay-per-view, or other sort of agreement.

3.6.2 Copyright Issues

iSeeker, similarly to web browsers, must make a local copy of a web page in order to be able to display/index it, even if the copy is only stored in volatile memory and not cached onto a file on a local drive, which it normally is.

This raises an interesting problem concerning copyright.

The recent debate over web page caching by ISPs means that some time in the future, web page caching may become illegal.

Will this affect caching performed by web browsers?

People seem to be most interested in tools that download whole web sites, for viewing while off-line. This is indeed one of the features that a tool such as iSeeker could implement. But wouldn't this be a serious violation of copyright?

The debate is not over yet, and personally I feel just as divided as most people.

4

4 USER INTERFACE

4.1 Introduction

Graphical User Interfaces (GUI) have an important role to play in information retrieval. The complexity in formulating queries (i.e. Boolean logic) and the large amount of data retrieved in searches are problems that could be minimised if appropriate user interfaces were used.

The development of an application such as iSeeker requires a great deal of attention devoted to the user interface.

The following section describes some rules of thumb that can guide engineers in designing and evaluating a user interface.

4.2 User Interface Guidelines

When designing and evaluating a GUI a set of guidelines should be followed. Such guidelines should ensure that every aspect of the interface's usability is covered both in the design and implementation stages. The same guidelines should then be used to evaluate the interface in experiments with real users, normally in the form of checklist questionnaires.

Ravden and Johnson [RAV 89] produced a comprehensive checklist, which covers the following sections:

1. Visual Clarity: organised and easy to read information
2. Consistency: the system should look and work in a consistent manner
3. Compatibility: the system should be compatible with the user's expectations
4. Informative Feedback: status, and user triggered feedback should be given
5. Explicitness: the way the system works should be clear
6. Appropriate Functionality: user requirements should be met
7. Flexibility and Control: the interface should be adaptable to different users
8. Error Prevention and Correction: the system should minimise user error
9. User Guidance and Support: on-line help and other documentation should be provided

This checklist is general and suitable to most systems involving a GUI.

In a project like iSeeker, *Informative Feedback* assumes a greater importance than in other applications, due to the long delay involved in accessing pages from the Internet.

Once a user starts a search, the system should provide regularly updated feedback on the progress of the search (i.e. accessing what search engines or pages, status of connections, etc).

Another equally important factor is the *Flexibility and Control*. Different users will have different search skills and will use different levels of search techniques. For example, a novice user may feel happy using a friendly interface with pull-down menus for choosing Boolean operators, however an expert user may want to input the Boolean and other advanced operators directly into the search string in the shape of +s and –s.

The guidelines described above are normally used as general heuristics. However, according to the rules of a good Usability Engineer, such heuristics are often considered a *cheap* way of implementing usability, and not the best way. Only a proper design and constant evaluation with the end users during the various prototypes will ensure a usable system.

4.3 Boolean Query Interface

The difficulty in representing queries in Boolean prevents novice users from fully exploring the functionality of most search tools.

Jerry Yang [YAN WWW] produced a simple but effective interface, which simplifies the creation of Boolean queries. The user types the keywords in the text fields provided and then clicks on the area, which interests him. In the example below, the user is interested in the pages relating Bill Clinton with NATO but excluding any material related to Kosovo.

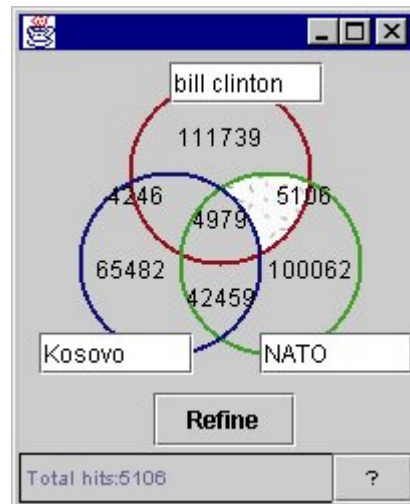


Fig. 13 – Venn Search Interface*

This query in the classic Boolean notation would be:

```
("bill clinton" AND NATO) NOT kosovo
```

or

```
+"bill clinton" +NATO -Kosovo
```

The user can then click on the **refine** button to apply further operations to the first resulting set.

The interface constantly refreshes the results of the search in a browser window, by sending the query to the desired search engine.

* Courtesy of Jerry Bo-chieh Yang. Homepage: <http://www.andrew.cmu.edu/~yang/>



5 DEVELOPMENT PROCESS

5.1 Introduction

This project started in February 1998, while I was doing my industrial placement at Digital Equipment Corporation. Much of the initial research and some early development took place during the next 6 months, while at DEC.

During these early stages there was no formal project plan. The main goal was to conduct research into the various relevant areas, and build a prototype to test some of the aspects of the artefact, such as Internet connection and downloading of web pages.

The earliest prototype, originally called GetWeb, dates back to April 1998. This early experiment tested the suitability of Visual C++ with MFC to the project.

5.1.1 Programming Language

Since I was learning Visual C++ at DEC, I decided to choose that programming language for this project.

I was aware that my inexperience with the language would make my task very difficult, but C++ is a very important skill and I thought the extra effort would pay out in the future.

A good side of programming in Visual C++ is the usage of the Microsoft Foundation Classes (MFC). This library of classes allows one to concentrate on the high-level aspects of the application. For example, the MFC WinInet classes allow the development of Internet client applications without worrying about WinSocks and other low-level details of Internet connections.

5.1.2 Development Methodology

Object-Oriented Design (OOD) is the methodology chosen for this project, since it suits the development of agents, especially when using C++ as the programming language. However, during the earliest prototypes the methodology was kept very loose. Typical OOD programming techniques such as data encapsulation were implemented, but no formal object diagrams were produced.

Prototyping was used because it is flexible and allows for significant changes in the development plan, a necessary feature since the plan was not well defined in the early stages.

The use of prototypes also allows one to backtrack to an earlier version of the software in case something terrible happens with the working version. This is particularly useful for a C++ newcomer like me.

5.2 Project Plan

Even though prototype#1 started as early as April 1998 it was not until September of the same year that a project plan was produced, after consulting my project supervisor.

It was during the design of the project plan that a formal specification for the application was produced.

Before producing the plan I had just a vague idea of how to tackle the development. The making of the project plan was useful since it made the objectives very clear.

It was at this stage that I decided to produce seven prototypes. Each prototype is supposed to fix any bugs present in the previous version and add some functionality. The original specification for the seven prototypes follows this section.

Unfortunately, the goals set in the plan were over ambitious, partly because my inexperience with the programming language was underestimated. In practice, none of the deadlines in the plan were met, with the development running increasingly late from prototype#2.

Due to the problems in implementation, only prototypes 1 and 2 were implemented successfully.

Prototype#3 proved to be the stumbling block in the development, due to the HTML parser.

After most of the time spent in this prototype, and without success, some last minute effort was put into implementing some of the requirements of the next prototypes.

Therefore prototype#4b (renamed due to change in specification) is partially implemented with some extra features (i.e. SEC plug-ins from prototype#6's specification).

Prototypes 5, 6, and Release were not implemented.

5.3 Prototypes Specification

The proposed seven prototypes have the following specifications:

5.3.1 Prototype 1

(Implemented before project plan)

Single-thread HTTP connection

Prototype#1 will take a URL as input from the user and will fetch the corresponding web page, displaying the HTML source in the window.

Single threaded application, GUI freezes while downloading page.

This prototype was successfully implemented, as described in section 5.4.1.

5.3.2 Prototype 2

Multi-threaded application

HTTP connection will run on a separate thread allowing a connection status to be displayed dynamically.

(Create dedicated class for the HTTP connection)

Deliverables:

- Multi-threaded application
- HTTP connection functionality built into a dedicated class

This prototype was successfully implemented, as described in section 5.4.2.

5.3.3 Prototype 3

Multi-connection application (simple web-crawler)

Based on previous prototype, follow links from original page until pre-set depth saving all pages locally.

Maximum number of concurrent connections should be easily set from the GUI.

Deliverables:

- Multi-threaded application
- Dynamic (run-time) creation of multiple connection objects

This prototype was not successfully implemented due to problems in the development of the HTML parser, as described in section 5.4.3.

5.3.4 Prototype 4

Multi-connection application using agents (advanced web-crawler)

Based on previous prototype, follows some links from original page until pre-set depth saving all pages locally.

Maximum number of concurrent agents (connections) should be easily set from the GUI.

Deliverables:

- Multi-threaded application
- Implementation of an 'agent' class
- Dynamic (run-time) creation of multiple agents

This prototype's specification was changed due to the lack of time to fully implement all of the future prototypes. The revised specification and development details are described in section 5.4.4.

5.3.5 Prototype 5

Simple search tool using agents

Based on previous prototype, this tool will start crawling web pages looking for a set of keywords.

Deliverables:

- Construction of a dynamic database containing all the URLs opened so far
- Agent heuristics, for page relevance decision

This prototype was not implemented due to lack of time.

5.3.6 Prototype 6

Meta-search tool using agents

This major prototype will do a meta-search using various search-engines and then uses agent technology to crawl the web for better hits.

Deliverables:

- SEC (Search-Engine Components) hold info about external search engines
- MSG (Meta-Search Generator) builds multiple queries using SEC plug-ins

- Advanced Agent class, with built-in internet search techniques knowledge base

This prototype was not implemented due to lack of time.

The MSG, also entitled SECLoader, implements the automatic plug-in technology which allows the tool to be updated every time a search engine changes.

5.3.7 Release

Meta-search tool using agents & web-communities

This final prototype will improve on the previous at the web-crawl stage. It will examine and organise the hits database into web-communities, using user feedback to control the communities weights.

Deliverables:

- SEC (Search-Engine Components) hold info about external search engines
- MSG (Meta-Search Generator) builds multiple queries using SEC plug-ins
- Advanced Agent class, with built-in internet search techniques knowledge base
- User-level adaptable GUI

This prototype was not implemented due to lack of time.

5.4 Development Description

5.4.1 *Prototype 1*

5.4.1.1 Description

The first prototype was produced shortly after I decided on the general idea for the final year project, on early February 1998, whilst still on my industrial placement.

The aim of the prototype was to perform a simple web page download, using the MFC classes available for that purpose.

There was no real effort in a formal design at this stage. My objectives were to learn the basic steps involved in accessing a page via the Internet as well as performing very simple operations with the retrieved page.

The AppWizard facility available in Visual C++ was used to create a basic Single-Document Interface (SDI) template program. This is a very easy way of starting a program, since the basic structure of a Windows program is automatically set up, allowing the programmer to focus on the application details.

After 5 minutes spent creating the SDI template with AppWizard, and another 10 minutes browsing the Help system for the appropriate classes, the task was almost completed.

Within 30 minutes prototype#1 was accessing AltaVista's main page (www.altavista.com), and displaying the HTML code locally.

During the following week, a very basic HTML parser was implemented. After downloading each page, the parser counts the number of links in the page and separates the HTML code into 3 distinct 'containers': normal tags, inline tags, and pure text. See section 5.6,

HTML Parser, for an explanation of the difference between the three types of HTML code.

5.4.1.2 Bugs/Limitations

5.4.1.2.1 GUI hangs

I was surprised to notice the hanging of the user interface while downloading a page.

I was initially convinced that the SDI template created by the AppWizard would handle the user interface and the application on different threads, but this was visibly not the case.

It was decided that the next prototype, Prototype 2, should fix this problem.

5.4.1.2.2 Limited HTML Parser

The HTML parser written for this prototype is very limited, and was only implemented as a way of testing very simple operations with the returned HTML code.



Fig. 14 – prototype 1: results from HTML parser

Fig. 14 above shows the results from the HTML parser in a simple statistics dialog. This dialog automatically appears when the application finishes downloading a page.

Using `CString` search methods, the parser counts the total number of links on the page, and divides the code into the three categories: Tags, Inline Tags and Text.

5.4.1.2.3 Web Page Testing

Prototype 1 was tested against a number of different types of web pages.

- Static Pages with plain HTML: handled with no problems.
- Pages with client scripts: HTML parser gets confused with unknown tags, resulting in inaccurate parsing (tags are placed in wrong containers)
- Server-side script pages: handled as Static Pages. For example, by using the URL resulting from a query in AltaVista, the application retrieves the exact same page as Netscape or Internet Explorer do.
- Automatic forwarding domains: forwards to new page correctly.

It was surprising that the application handled automatic web forwarding. This seems to be handled within the `WinInet` class.

No other major bugs were found. The application did not crash at any time.

5.4.1.3 Screen Shots

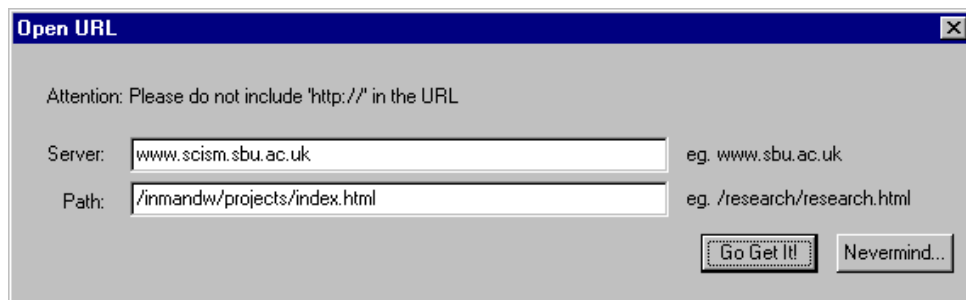
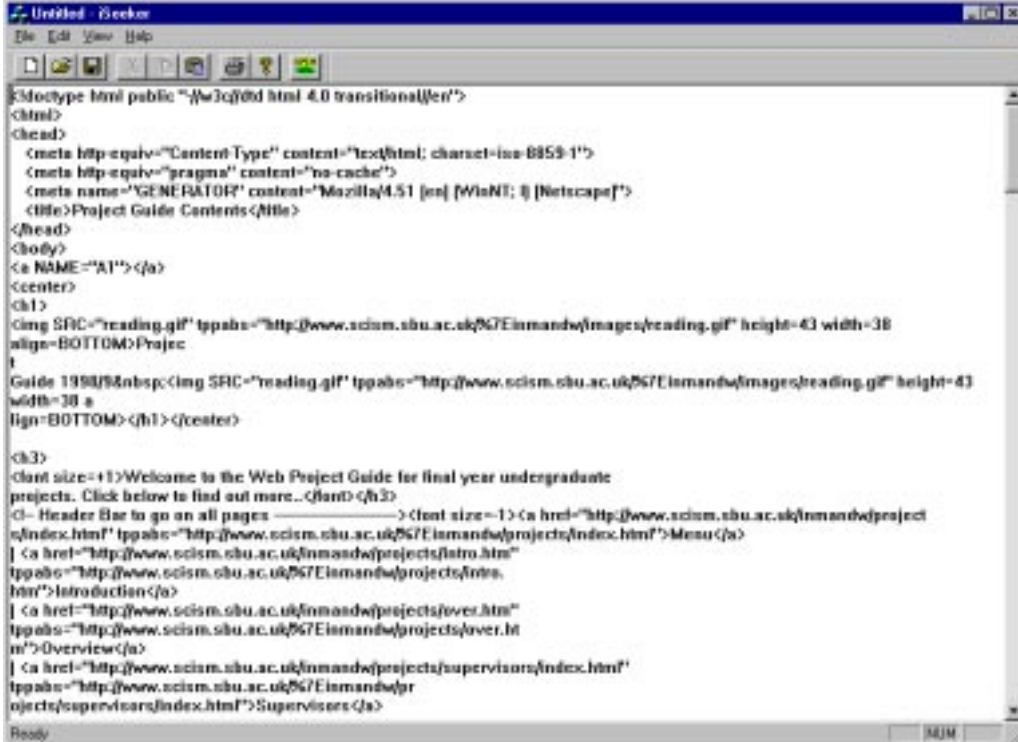


Fig. 15 – prototype 1: Open URL dialog



```

Untitled - Notepad
File Edit View Help

<!doctype html public "-//w3c//>html 4.0 transitional//en">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta http-equiv="pragma" content="no-cache">
  <meta name="GENERATOR" content="Mozilla/4.51 [en] [WinNT; &] [Netscape]">
  <title>Project Guide Contents</title>
</head>
<body>
<a NAME="A1"></a>
<center>
<h1>
<img SRC="reading.gif" tppabs="http://www.scism.sbu.ac.uk/567Eismandw/images/reading.gif" height=43 width=38
align=BOTTOM>Projec
t
Guide 1998/9&nbsp;   <img SRC="reading.gif" tppabs="http://www.scism.sbu.ac.uk/567Eismandw/images/reading.gif" height=43
width=38 a
lign=BOTTOM></h1></center>

<h3>
<font size=+1>Welcome to the Web Project Guide for final year undergraduate
projects. Click below to find out more...</font></h3>
<ol>
<li>Header Bar to go on all pages _____</li>
<li>font size=-1><a href="http://www.scism.sbu.ac.uk/inmandw/project
s/index.htm" tppabs="http://www.scism.sbu.ac.uk/567Eismandw/projects/index.htm">Menu</a>
</li>
<li><a href="http://www.scism.sbu.ac.uk/inmandw/projects/intro.htm"
tppabs="http://www.scism.sbu.ac.uk/567Eismandw/projects/intro.
htm">Introduction</a>
</li>
<li><a href="http://www.scism.sbu.ac.uk/inmandw/projects/over.htm"
tppabs="http://www.scism.sbu.ac.uk/567Eismandw/projects/over.ht
m">Overview</a>
</li>
<li><a href="http://www.scism.sbu.ac.uk/inmandw/projects/supervisors/index.htm"
tppabs="http://www.scism.sbu.ac.uk/567Eismandw/pr
ojects/supervisors/index.htm">Supervisors</a>
</li>
</ol>
Ready

```

Fig. 16 – prototype 1: main window

5.4.2 Prototype 2

5.4.2.1 Description

Prototype 2 fixes the problem with the hanging of the GUI by implementing a simple working thread, which handles the download while the GUI continues to run on the original thread.

Prototype 2, while not adding any major extra functionality, took quite a longer time to implement than what was planned. My inexperience with multithreading programming proved to be the cause for this delay. The problem with threads is described in detail in section 5.5.2.

5.4.2.2 Bugs/Limitations

5.4.2.2.1 Local file access

When opening local files, problems were found in paths that contain spaces (i.e. file://C:\My Documents\test.html). The manual fixing of the path (i.e. file://C:\mydocu~1\test.html) fixes the problem but this is naturally a bug that must be fixed.

The description of this bug is given in more detail in section 5.5.1.

5.4.2.2.2 Thread Testing

The testing of this prototype focused on the main functionality introduced, the extra worker thread used to download the web page.

After starting the download of a page, I immediately tried to use all the GUI buttons and menus. These worked as normal, while the download was still in process.

I also tried downloading a second web page, while the first was still being processed. Surprisingly, the application started the download of the second page in parallel with the first one. Whichever page is downloaded first gets displayed first. When the second page is downloaded, its contents then replace the previous one.

5.4.2.3 Screen Shots

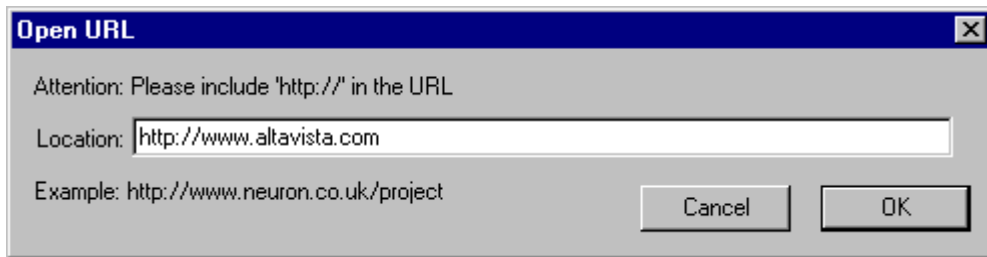


Fig. 17 – prototype 2: Open URL dialog

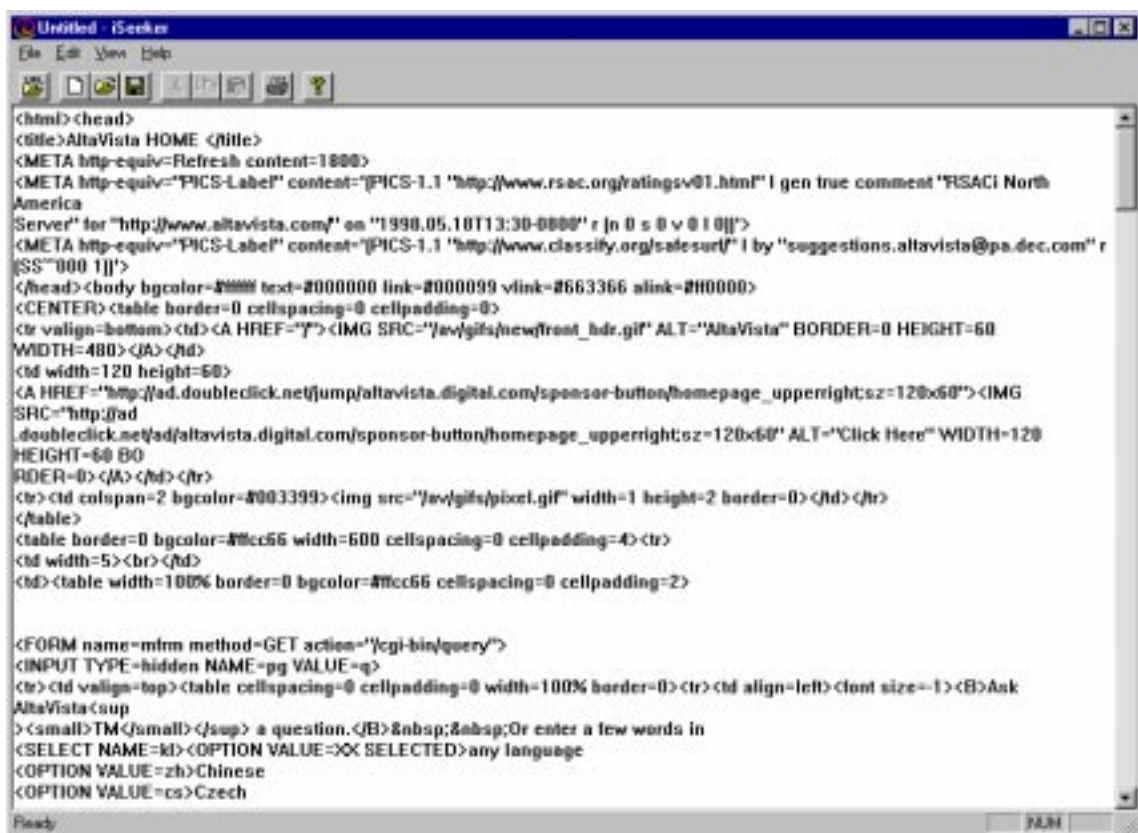


Fig. 18 – prototype 2: main window

5.4.3 Prototype 3

5.4.3.1 Description

Prototype 3, while not adding much functionality to the previous prototype, depended on the implementation of an HTML parser, which was planned as a parallel task to the prototypes.

Most of the project's development time was spent in the HTML parser, but little progress was made. Prototype#3 was never completed due to the problems with the parser. Details on the implementation of the parser are described in section 5.6.

5.4.4 Prototype 4b

5.4.4.1 Description

After the problems with prototype#3, and with little time left to continue with the other planned prototypes, I decided to abandon the HTML parser and tried to implement some of the further prototype's features. It was imperative that this project implemented at least a simple set of agent classes.

For this reason, the four prototypes scheduled were scrapped and a modified prototype#4 was implemented (named prototype#4b). This prototype implements some of the features planned for the later stages of development.

5.4.4.2 Prototype#4b Specification

Simple implementation of agent classes and Search Engine plug-ins

The revised prototype should implement a simple set of agents to demonstrate the proposed agent architecture and search engine plug-ins.

Deliverables:

- Implementation of the three main types of agents (co-ordinator, meta-search and crawler)
- Implementation of experimental Search Engine plug-in technology

5.4.4.3 Search Engine Components (SEC)

The idea behind search engine components is that search engines are dynamic and may change the way in which they display the results or the query syntax.

In order for iSeeker not to become obsolete after a few days, it needs to allow the *live-updating* of this search engine information.

The solution for this is the storing of all the information necessary to use a particular search engine in a special file called 'Search Engine Component' (SEC).

In the case of iSeeker, the SEC is a text file including all the necessary information to query a search engine and extract the resulting links. An example SEC for AltaVista can be found in Appendix E - Sample Search Engine Component (SEC), page 95.

When the application is initialised, it tries to locate a directory with all the SECs and loads every file.

If during a meta-search one of the search engines fails (possibly because the search engine changed in some way), the application will connect to a SEC server on the web, looking for an updated SEC for that search engine.

In Prototype#4a, some test SECs were produced. The prototype loads all the SECs in the directory while initialising.

When the user starts a new search, the list of loaded search engines will appear in the dialog. The user can then select the ones to be used in the search.

This technology is also useful for local or specialised search engines, which are not the interest of the general public. When setting up the tool for the first time, the user can ask for any geographically local search engine SECs to be automatically downloaded, as well as any specialised search engines. A sample list of different search engines can be found in Appendix A - Sample Search Tools, page 83.

5.4.4.4 Agent Architecture

This prototype implements the proposed agent architecture, even if only in a general structural way.

A main agent class `CAgent` derived from `CWinThread`, servers as the parent class for all the other agents. It is derived from `CWinThread` because each instance of the agent class will run on a separate **interface** thread.

The CCoordinator is the main agent which co-ordinates both the CSEAgents and the CCrawlerAgents, as planned.

The CSEAgents are only used to communicate with the search engines, during the meta-search stage. Each one of these should be assigned to a different search engine, by being passed the information of a single SEC.

The CCrawlerAgents are the real *intelligent agents* which will do the exhaustive search on the web, opening and indexing page after page.

5.4.4.5 Screen Shots

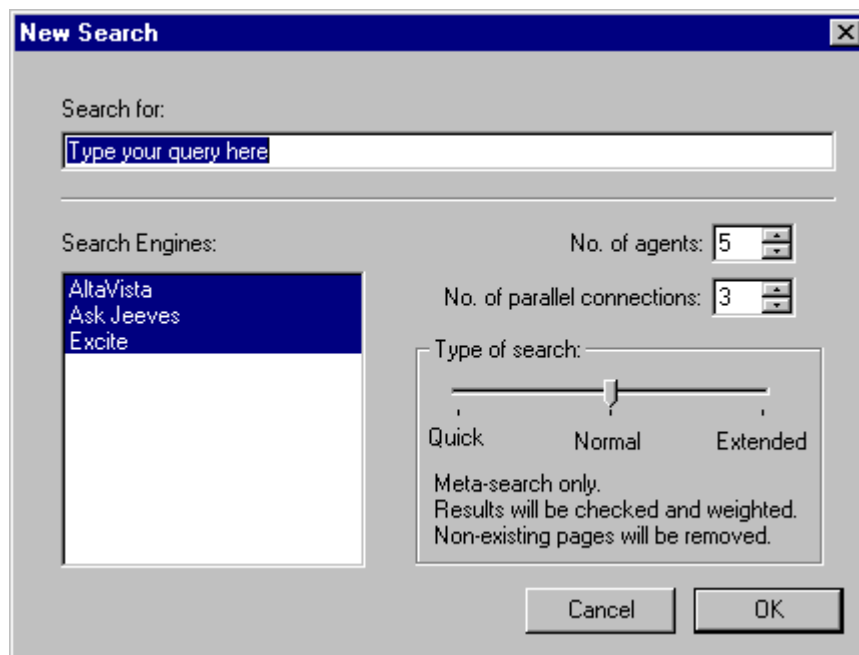


Fig. 19 – prototype 4: new search dialog

The figure above displays the *New Search* dialog that the user is presented with every time she starts a new search.

Note how the information read by the `CSECLoader` class is automatically included in the 'Search Engines' list box. The user can easily select which search engines should be used for the search.

It is also possible to set the number of agents, and concurrent Internet connections.

The slider control allows the selecting of search type. The three types are:

- Quick
Meta-search only. The results are simply collected without any extra validation. Duplicate results are removed.
- Normal
Meta-search only. The results will be downloaded and weighted according to internal criteria. Broken links are removed.
- Extended
Meta-search and Web Crawl. After the normal meta-search, crawler agents will start crawling the web for more pages.

5.5 Implementation Problems

5.5.1 Local Internet Files Access

Module: `inetConnect.cpp`

Function: `CinetConnect::OpenFILE(CString& HTML)`

There is a problem when opening local Internet files (`file://`) which have spaces in the path. E.g.:

```
file:///C:\Program Files\web\index.html
```

It triggers an error message ('invalid path') and does not open the file.

Note: It works well in paths with no spaces.

I noted that the URL with the spaces gets properly parsed by the `AfxParseURL` function (it substitutes the spaces by `%20`) but it still fails when opening...

Possible solutions:

1. Try opening the file (URL) without parsing (`AfxParseURL`), therefore without the `%20`s in the string. **(Doesn't work)**.
2. Open local files using standard MFC file functions (not `WinInet`).

Note: This problem was posted in two MFC related newsgroups. Had three replies, but none with a valid solution. (*See the posts in the appendix, page 91*)

The solution 2 described above is not ideal, but will ensure the safe crawling of local pages.

5.5.2 *Multiple Threads: Worker versus Interface*

Module: ISeekerDoc.cpp

Function: IseekerDoc::OnOpenURL()

One of the greatest design problems in the project was the choice of thread type for each of the objects used in iSeeker.

During the implementation of prototype#2, a decision had to be made for the type of thread to use in the *InetConnection* objects. The choices: worker thread or interface thread.

Originally, worker threads seemed to be the obvious choice: the thread does not need to maintain its own window, since it has not direct interaction with the user.

However I did not find a suitable way of passing messages between 'Main' thread and 'Child' threads. The 'Child' thread's controller function seems to be only suitable for batch jobs, where the work is done in parallel and the thread automatically finishes when the function returns.

For prototype#2, I used **worker thread**, therefore avoiding multiple inheritance of the class *InetConnect* (from *CInternetConnection* & *CWinThread*)

However, in prototype#4b, **interface threads** were used for all the objects in the application. The reason for the choice is simple: only the interface thread allows the whole object to be associated with the thread, facilitating the communication between objects, either by normal method calling or message passing.

5.5.3 Header Files Redefinition

Files containing *popular* class definitions such as *iAgent*, the base class for all the agents in iSeeker, are referenced to from several other header files. This caused a `'class' type redefinition` compile error, since the compiler was reading the same header file multiple times.

In order to tackle the problem, I firstly rearranged the order in which the `include` statements appeared in all the files, but this resulted in a very untidy chaining of includes. I knew there had to be a better way of fixing this, after all the files generated by the AppWizard didn't have this problem.

After a quick search on the web for “`'class' + "type redefinition" + include`” I found the right solution: `#ifndef` and `#define`. By including these statements in the beginning of the header file, the compiler only reads the class definitions once. Any subsequent includes will not generate a redefinition error. For example, the `iAgent.h` class now has the following lines in the beginning:

```
#ifndef IAGENT_H
#define IAGENT_H
[..file body here..]
#endif
```

After discovering the right solution, all the code was revised and the `includes` were restored to the most logical structure (i.e. all the classes deriving from `iAgent` have an `include` to that header file), which resulted in a much more readable and easier to maintain code.

It was also nice to understand what those three lines of code meant in the AppWizard generated files.

5.6 HTML Parser

For any information retrieval system to be successful, the use of efficient parsers is of great importance. Whether the application only works with HTML files or also with XML, PostScript, Word or any other common document format, if the corresponding parser is not of a good standard the system will be very limited.

As I found out, even though building a parser is not supposed to be the hardest of tasks, the lack of experience with MFC and the tendency for perfectionism led me to very hard times.

I was committed to building an efficient HTML parser, which would be the cornerstone of each agent's ability to rate a web page, but this soon proved to be the hardest task of the whole project.

The main problem is the way in which I chose to store the parsed information. A list of pointers to a list of pointers to a list of pointers (I think I got this right) is not the easiest of tasks for a C++ newbie.

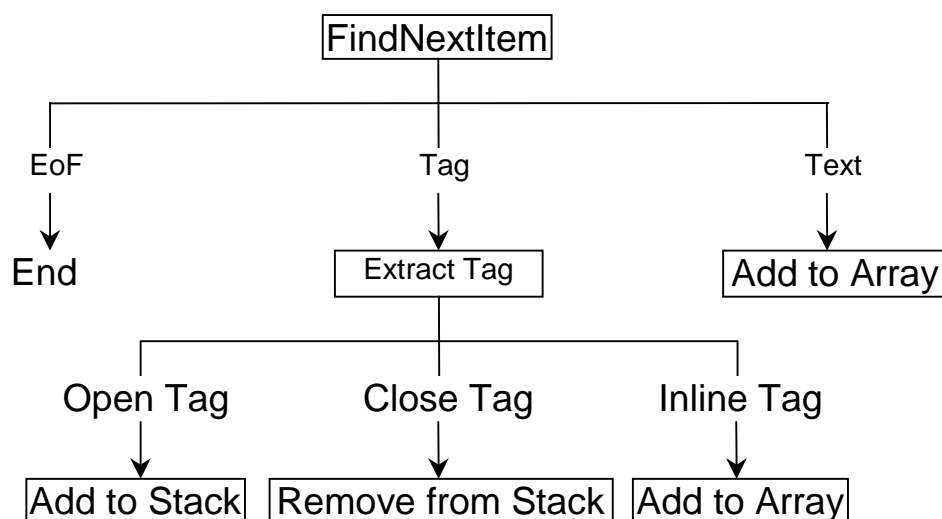


Fig. 20 - html parsing procedure

An example of the way in which the HTML parser stacks the information is present on the appendix, page 88.

In Fig. 20 we can see that the procedure for parsing a file is in itself quite simple.

The parser identifies normal tags, inline tags (with no closing tag) and normal text.

When it finds a normal tag, it adds it to a stack. Normal tags are always added to or subtracted from the stack. Only inline tags (such as image tags) and text is added to the array. Each time a record is added to the array, the stack at the moment is also added.

This enables the system to know in which situation a paragraph of text or image is (e.g. centred, text in bold, header1, header2 or any other style)

Please note that the array notation has been modified to a list of pointers for increased performance and better memory management.



6 FUTURE DEVELOPMENTS

Assuming that all the planned features were implemented in this project, there are some other that were initially decided as ‘future developments’. This section describes the features that would improve iSeeker.

6.1 Other Search Options

At the moment, iSeeker only allows searches on the Internet, using the meta-search as a starting point.

A useful feature is to allow the user to specify a starting URL, from which the tool will start crawling, without resorting to the meta-search.

This would allow a local search (ie. Intranet web site, locally saved web pages, etc)

6.2 Natural Language Processing

One of the AI techniques that would be of most importance to any information retrieval system is Natural Language Processing (NLP). NLP would allow the system to understand queries from the user in natural language instead of just single keywords, therefore constructing a mental model closer to that of the user.

Not only that, each agent could have a real ‘understanding’ of a web page content and be able to decide on the true relevance of the page to the user.

The combination of NLP with speech recognition would also improve the user interface, allowing the user to communicate the query and give feedback via speech instead of the normal keyboard and mouse.

A system with true NLP capacity would allow the user to communicate with the system as if she was querying a human librarian.

6.3 Agents as Plug-Ins

Similarly to the Search Engine Components (SEC), the agents could be implemented as plug-ins. This would allow for users to develop special agents for particular types of search (i.e. sports specialised) and share the agents over the Internet. Special servers could be used for users to upload their agents, as well as freely download other people's agents. This free trade of agents would most certainly become popular with the users^{*}

^{*} This already happens with some applications, such as the Artificial Life game *Creatures2*, developed by CyberLife. Players share the *Norms* which are based on genetic algorithms.

6.4 Distributed Search Engine

Normal search engines rely on their powerful spiders and indexers to do all the hard work of indexing the web. As mentioned early in the report, even though these indexers are state-of-the-art very powerful computers, they cannot keep up with the rate at which the web is growing.

However, if there was a way of distributing the workload to every user on the web, the task of indexing the whole of web could be a reality.

In the future, hundreds if not thousands of users could use tools such as iSeeker, at home or workplace. Since these tools are doing the same basic work as the large spiders, why not combine the efforts of everyone onto a single resource?

If millions of users contribute with their searches to a single centralised database, the process of indexing the whole of the web would go much faster. A tool like iSeeker could be the way to do it...

The user benefits from the personalised search tool, which is indexing a fraction of the web for its own personal use, and by allowing the tool to send the indexed results to a central search engine, everyone would benefit from the largest and most up-to-date index in the world.

The more users contribute with their searches the best the future searches will be because the centralised search engine will cover a wider range.

The number of 'dead' links supplied by the search engine would decrease dramatically, since it is constantly being refreshed by thousands of users.

In addition to this, since the users supply feedback during the searches, this extra information could be used to help ranking the best pages on the web.

With the increasingly popular distributed computing efforts in the world (i.e. RC5 and SETI@Home), distributed indexing could be the next big thing.



7 CONCLUSION

7.1 Research

The research conducted as part of this project reveals the usefulness of Intelligent Agent technology applied to information retrieval.

The web search techniques described in the report were used to represent one of many human mental models in web search. Such techniques should be further documented and formally implemented into a set of heuristics to be used by the agents.

Since every search is different, the usage of different agents with different heuristics allows the selecting of the most appropriate ones for each particular search. Genetic Algorithms can play an important part in this run-time selection.

The complexity involving web search can be minimised if suitable user interfaces are developed specifically for this type of task. Special interfaces to simplify the formulation of queries as well as the visualisation of the results are essential features in a well-designed web search tool, and these should be addressed early in the development.

The evaluation of web-communities is a feature that may offer any search tool a competitive edge over the competition. The research in this area described how this can be achieved, but also revealed some problems which will have to be solved before a realistic implementation is feasible.

7.2 Development

The development was left rather short from the initially planned. This was mainly due to my lack of experience in C++ programming, which is far from sufficient to achieve such ambitious plans, as well as some poor project management decisions.

At this stage I believe a more flexible approach to project management would have allowed further progress. At some stages I was stuck with implementation problems and could have worked on further prototypes but decided to persist with solving the problem instead. This proved to be a wrong decision and one that I certainly learned from.

In the future I will get on with other parts of the implementation, which are not dependent on the previous prototypes.

7.3 Project Plan

Another mistake was the over ambitious project plan itself. Seven prototypes were just too much for such a short period of time. This is naturally linked with my lack of C++ knowledge, since I believe a good C++ programmer would be able to tackle the proposed plan without many problems. I think I was misleadingly confident after the first prototype which was fairly easy to implement. Now I think three or four prototypes would be more realistic, but this was hard to calculate before being well into the project. Still, the experience will be present when designing future plans.

7.4 Further Progress

Prototype#4a, the most advanced of the implemented prototypes, already contains the main structure that the release version should have, namely the proposed agent architecture.

In order to complete the development of the project, the HTML parser should be implemented. This is still the main missing module preventing the further development of the tool.

Even though the HTML parser's design seems to be correct, the implementation is not. The main task is to construct a new data structure in which to hold the parsed HTML tree, since this was the essential problem with the attempted implementation.

Once the HTML parser is in place, the crawler agents can be implemented.

The implementation of web-communities analysis will depend on the decisions made about the potential problems associated with them.

The interface should be improved so that the list of results can be displayed. The interface should also allow the user to supply feedback to each page in the list.

7.5 Achievements

Overall, I feel this project covered a reasonable breadth of areas that are relevant to the development of the proposed tool. It was intended, as part of the research, to propose solutions for each of the areas researched, and I feel this was achieved.

The under achievement is obviously related with the development. Even though the main structure of the finished product is in place, the functionality was not achieved. The tool is supposed to search for information on the web, and this was not implemented. The reasons for this were already explained.

Nevertheless, the partial failure in the implementation serves as a lesson for future projects, and that must only be a good thing.

7.6 Personal Comments

Even though I do feel disappointed with the unfinished product, I do not regret having chosen such ambitious goals in the beginning.

At this stage I feel that I learned a great deal from this project. The fact that I stumbled on a problem which I could not fix, and that ultimately prevented me from going much further in the development is an invaluable lesson.

Thinking back, it would take just a few hours to develop a very simple HTML parser that would only extract the existing links and count the occurrences of a keyword in the page. I actually did something very similar in the very first prototype. And even though it would not be the best HTML parser in the world, it would do the essential job, allowing me to continue with the really important part of the development.

With this lesson in mind, I can now face the challenge of completing the implementation and, eventually proceed with the further developments described in the previous section.

I also do not regret having chosen C++ for the development. Even though I still have a lot to learn to master this language, I have come a long way since the beginning of the project, and hopefully will continue to do so in the future.



8 BIBLIOGRAPHY

8.1 References

8.1.1 WWW Resources

[GAT WWW]	Gates, D. [1998] Web Rings / Will the Circle Be Unbroken? [web page] May 1998 http://home.microsoft.com/reading/archives/business-5-4-98.asp [Visited 13 Mar 1999]
[NEC WWW]	Lawrence, S., and Giles, C. L. [1998] Size of the Web, Web Size, Search Engine Coverage and Recency [web page] Apr 1998 http://www.neci.nj.nec.com/homepages/lawrence/websize.html [Visited 9 Mar 1999]
[YAN WWW]	Yang, J. B. [1999]. <i>Venn Search Interface</i> . [web page] May 1999 http://www.andrew.cmu.edu/~yang/work/vsi.html [Visited 19 May 1999]

8.1.2 Printed Resources

[RAV 89]	Ravden, S. and Johnson, G. (1989), <i>Evaluating Usability of Human-Computer Interfaces – A Practical Method</i> , Ellis Horwood Limited, West Sussex, United Kingdom
[FIN 96]	Finlay, J., and Dix, A. (1996). <i>An Introduction to Artificial Intelligence</i> . UCL Press, London. p.236-237
[KNA 98]	Knapik, M., and Johnson, J. (1998). <i>Developing Intelligent Agents for Distributed Systems</i> . McGraw-Hill.

8.2 Bibliography

8.2.1 WWW Resources

Baldonado, M. Q. W., and Winograd, T. [1996]. *SenseMaker: An Information-Exploration Interface Supporting the Contextual Evolution of a User's Interests*. [web page] Sep 1996
<http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0048>
[Visited 13 May 1999]

- Bunyip Information Systems Inc. [1999]. *Archie Home Page - Bunyip Information Systems Inc.* [web page] Feb 1999
<http://www.bunyip.com/products/archie/>
[Visited 9 Mar 1999]
- Ring, P. [1998]. *Tip of the Month 9706: Citation of URLs.* [web page] Nov 1998
<http://www.prc.dk/user-friendly-manuals/ufm/tip-9706.htm>
[Visited 9 Mar 1999]
- Sipper, M. [1996]. *A Brief Introduction to Genetic Algorithms.* [web page] Nov 1996
http://lslwww.epfl.ch/~moshes/ga_main.html
[Visited 1 May 1999]
- Sonnenreich, W., and Macinta, T. [1998]. *A History of Search Engines.* [web page] Oct 1998
<http://gsd.mit.edu/~history/search/engine/history.html>
[Visited 10 Mar 1999]
- Stefani, A., and Strapparava, C. [1998]. *Personalizing Access to Web Sites: The SiteIF Project.* [web page] Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98, Pittsburgh, USA, Jun 1998
<http://www.wis.win.tue.nl/ah98/Stefani/Stefani.html>
[Visited 13 May 1999]
- WebRing Inc. [1999]. *Welcome to WebRing!* [web page] Apr 1999
<http://www.webring.com/>
[Visited 13 Mar 1999]
- Waern, A. [1997]. *Links to material on Intelligent Interfaces.* [web page] Apr 1997
http://www.sics.se/~annika/ii_links.html
[Visited 12 May 1999]
- Yang, J. B. [1999]. *My Research Topic: Search on the Web.* [web page] May 1999
<http://studioserver.pc.cc.cmu.edu/51742/jerry/research.html>
[Visited 19 May 1999]

8.2.2 Printed Resources

- André, E., and Wahlster, W. (1998). *Intelligent Multimedia Interface Agents.* Tutorial notes from the 13th Biennial European Conference on Artificial Intelligence, Brighton, UK, Aug 1998.
- Bradley, N. (1998). *The XML Companion.* Addison-Wesley, Essex.
- Burrell, P. (1998). *Fuzzy Systems.* Lecture notes from Hybrid Knowledge-Based Systems unit, South Bank University, London.
- Collins, D. (1995). *Designing Object-Oriented User Interfaces.* Benjamim/Cummings Publishing Company.

Connolly, D., editor (1997). *XML: Principles, Tools, and Techniques*. O'Reilly.

Faulkner, C. (1998). *The Essence of Human-Computer Interaction*. Prentice Hall.

Finlay, J., and Dix, A. (1996). *An Introduction to Artificial Intelligence*. UCL Press, London.

Glossbrenner, A., and Glossbrenner, E. (1999). *Search Engines for the World Wide Web*, 2nd Edition. Peachpit Press.

Hahn, U., and Mani, I. (1998). *Automatic Text Summarisation*. Tutorial notes from the 13th Biennial European Conference on Artificial Intelligence, Brighton, UK, Aug 1998.

Jennings, N. R., and Wooldridge, M. (1998). *Applying Agent Technology*. Proceedings of 13th Biennial European Conference on Artificial Intelligence, Brighton, UK, Aug 1998.

Knapik, M., and Johnson, J. (1998). *Developing Intelligent Agents for Distributed Systems*. McGraw-Hill.

Leavens, A. (1994). *Designing GUI Applications for Windows*. M&T Books, New York.

Leventhal, M., Lewis, D., and Fuchs, M. (1998). *Designing XML Internet Applications*. Prentice Hall, New Jersey.

Light, R. (1997). *Presenting XML*. Sams.net Publishing.

Pereira, F. B. (1998). An Artificial Life Model for Information Retrieval in a Distributed Environment. In, *Proceedings of 13th Biennial European Conference on Artificial Intelligence*, Brighton, UK, Aug 1998.

Russell, S., and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

Schneider-Hufschmidt, M., Kühme, T., and Malinowski, U., editors (1993). *Adaptive User Interfaces: Principles and Practice*. North-Holland, Netherlands.

Sullivan, J., and Tyler, S. W., editors (1991). *Intelligent User Interfaces*. ACM Press.

Tracy, K. W., and Bouthoorn, P. (1997). *Object-Oriented Artificial Intelligence Using C++*. Computer Science Press, W. H. Freeman, New York.

Watson, M. (1996). *Programming Intelligent Agents for the Internet*. McGraw-Hill.

9 APPENDIX A - SAMPLE SEARCH TOOLS

9.1 Sample Server-Side Search Tools

9.1.1 General Search Engines

Name	URL
AltaVista	http://www.altavista.com
Excite	http://www.excite.com
GoTo	http://www.goto.com
HotBot	http://www.hotbot.com
InfoSeek	http://infoseek.go.com
Lycos	http://www.lycos.co.uk
Magellan	http://magellan.excite.com
Northern Light	http://www.northernlight.com
PlanetSearch	http://www.planetsearch.com
Starting Point	http://www.stpt.com
Thunderstone	http://www.thunderstone.com
WebCrawler	http://www.webcrawler.com

9.1.2 General Directories

Name	URL
About.com	http://www.about.com
Dogpile Open Directory	http://opendir.dogpile.com
Galaxy	http://galaxy.einet.net
LookSmart	http://www.looksmart.com
Lycos Top 5%	http://point.lycos.com
NetGuide	http://www.netguide.com
Open Directory	http://dmoz.org
Rex	http://rex.skyline.net
Snap	http://www.snap.com
Switchboard	http://www.switchboard.com
Yahoo	http://www.yahoo.com

9.1.3 Geographically Local Directories

Name	URL	Location
Alcanseek	http://www.alcanseek.com	Alaska & Canada
Answers	http://www.answers.com	Australia & New Zealand
Austrian Internet Directory	http://www.aid.co.at	Austria
Webbel	http://www.webbel.be	Belgium
Cross	http://cross.carnet.hr	Croatia
Yiasou	http://www.yiasou.com.cy	Cyprus
Seznam	http://www.seznam.cz	Czech Republic
CyberCity	http://search.cybercity.dk	Denmark
NETI	http://www.neti.ee	Estonia

EuroFerret	http://www.euroferret.com	Europe
Ihmemaan haku	http://www.fi	Finland
Echo	http://www.echo.fr	France
AllesKlar	http://www.allesklar.de	Germany
GoGreece.com	http://www.gogreece.com	Greece
123India	http://www.123india.com	India
Swift Guide to Ireland	http://swift.kerna.ie	Ireland
Arianna	http://www.arianna.it	Italy
NetLondon	http://www.netlondon.com	London, UK
Betsy	http://www.betsy.nl	Netherlands
Otigosøk	http://www.origo.no/sok	Norway
Sapo	http://www.sapo.pt	Portugal
Rambler	http://www.rambler.ru	Russia
Dónde?	http://donde.uji.es	Spain
Spaystart	http://www.spray.se	Sweden
Heureka	http://www.heureka.ch	Switzerland
Arabul	http://arabul.dominet.com.tr	Turkey
searchUK	http://www.searchuk.co.uk	United Kingdom

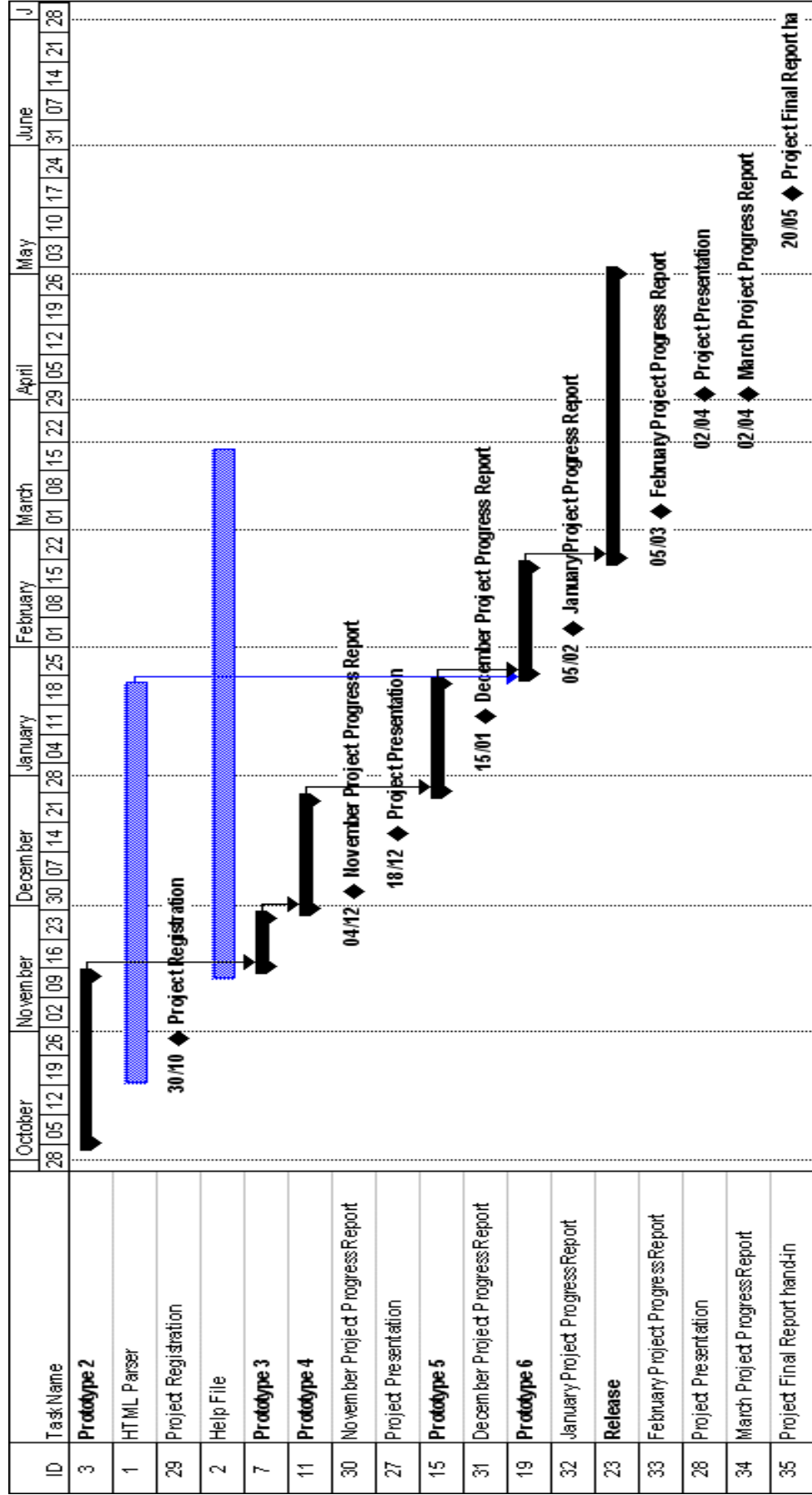
9.1.4 Specialised Directories

Name	URL	Subject/Category
StudyWeb	http://www.studyweb.com	Academic Research
Adam	http://adam.ac.uk	Art & Design
Neuron AI Directory	http://www.neuron.co.uk	Artificial Intelligence
LiveLink Pinstripe	http://pinstripe.opentext.com	Business
AskAlex	http://askalex.co.uk	Business, UK
AntiSearch	http://www.AntiSearch.com	Hackers & Security
HealthWave	http://www.healthwave.com	Health
FindLaw	http://www.findlaw.com	Law
1212	http://www.1212.com	Music Production
Yahoo! People	http://people.yahoo.com	People
Bigfoot	http://www.bigfoot.com	People, emails
Deja	http://www.deja.com	Usenet
EarthCam	http://www.earthcam.com	Webcams

9.1.5 Meta-Search Engines

Name	URL
A1 DigiSearch	http://www.digiway.com/digisearch
Cyber 411	http://www.cyber411.com
Dogpile	http://www.dogpile.com
Metadog.com	http://www.metadog.com
WhatUseek	http://www.whatuseek.com

10 APPENDIX B - PROJECT PLAN: GANTT CHART



11 APPENDIX C - HTML PARSER STACK EXAMPLE

Source Code	Stack		inline tags & text (content)	
<html>	html			
<head>	html	head		
<title>	html	head	title	
My home page	html	head	title	My home page
</title>	html	head		
</head>	html			
<body bgcolor=#FFFFFF text=navy>	html	body		
<h1>	html	body	h1	
Welcome	html	body	h1	Welcome
</h1>	html	body		
<p>	html	body		<p>
This is my very first web page. I hope you like it here	html	body		This is my very first web page. I hope you like it here
	html	body		
 	html	body		
This is me! :)	html	body		This is me! :)
<p>	html	body		<p>
Please check out my links:	html	body		Please check out my links:
 	html	body		
	html	body	a	
CNN home page	html	body	a	CNN home page * see example below
	html	body		
 	html	body		
	html	body	a	
BBC online	html	body	a	BBC online
	html	body		
 	html	body		
<p>	html	body		<p>
Hope you enjoyed, and please visit again!	html	body		Hope you enjoyed, and please visit again!
</body>	html			
</html>				

* example record



12 APPENDIX D - EMAIL AND NEWSGROUP POSTS

12.1 Wininet: local (file://) files with spaces in the path

12.1.1 Post 1 - Question

Author: Daniel Farinha
Email: daniel_farinha@hotmail.com
Date: 1998/12/03
Forums: microsoft.public.vc.mfc, microsoft.public.inetsdk.programming

Hello,

I'm having a problem in opening local internet files (file://) which contain spaces in the path, eg. (file://F:\test one\index.html)

When I call this function:
pFile = session.OpenURL(pstrURL, 1, INTERNET_FLAG_TRANSFER_ASCII, NULL, 0);
where pstrURL is "file://F:\test one\index.html" it produces the following error:
"F:\test%20one\index.html contains an invalid path."

I tried using the AfxParseURL(pstrURL,...) function, and then using the returned strObject in the OpenURL function but it didn't work either.
AfxParseURL adds the %20 itself, so then the error becomes:
"F:\test%2520one\index.html contains an invalid path."

OpenURL doesn't seem to understand the %20 as a space...

Files with no spaces in the path work just fine...

What am I doing wrong?

Cheers

Daniel Farinha

12.1.2 Post 2 – Reply to Post 1

Author: Juergen Fiedler
Email: juergen@greenberg.org
Date: 1998/12/03
Forums: microsoft.public.vc.mfc, microsoft.public.inetsdk.programming

Wow, that was colorful :)

Anyway: Your problem is with the backslashes in the path names. A single backslash starts an escape sequence (like \t, \n...). If you want to use a backslash in a string, you have to double it, i.e. "file://F:\test one\index.html" should turn to "file://F:\\test one\\index.html".

HTH,
Juergen

12.1.3 Post 3 – Reply to Post 2

Author: David Lee
Email: xl0@enr.uark.edu
Date: 1998/12/03
Forums: microsoft.public.vc.mfc, microsoft.public.inetsdk.programming

No. He said it works for paths without spaces.

12.1.4 Post 4 – Reply to Post 1

Author: Igor Pronin
Email: Igor.Pronin@Elma.Net.Nospam
Date: 1998/12/03
Forums: microsoft.public.vc.mfc, microsoft.public.inetsdk.programming

I have used such local path/filenames which contain spaces with the ftp-functions included in wininet.dll (GET and PUT) and they work without problems.

I have had problems with case-sensitivity - depending on the utility program and if using intranetwork connection you may see the sharenames with capitals although it really isn't etc And I have received just similar errors.

regards

Igor Pronin

12.2 IA Questions Email

From: "Marcus P. Zillman, M.S., A.M.H.A." <zillman@botspot.com>
Organization: BotSpot, Inc. (<http://www.botspot.com/welcome/>)
To: Daniel Farinha <Daniel.Farinha@digital.com>, team@botspot.com
Subject: Re: IA questions

Top of the morning Daniel....

Thanks for the questions and here are the answers...

1) Intelligent agents and bots cruise the _entire_ Internet searching for information from selected or all sites (URLs). They then bring back this information to you their master :-). They do not search Alta Vista or HotBot or another database of links. They create their own database and bring it back to you. I am working on ARCBot which brings back automated resource compiled links (like Yahoo) for specific requests. Cornell has done the initial research with interesting results..... Bots and Intelligent agents do not need the search engines to bring back their information. The meta search engines you are discussing are really very shallow bots that have been scripted just to search specific databases like alta vista etc...really not a true bot or intelligent agent as their universe is limited to the capability of the data base they are

searching. If the database does not have .ps or .pdf resources then you loose out on some of the very best information on the Net....Yes it would hurt these little meta-search engines if alta vista blocked them but it would not hurt any of the _real_ bots and intelligent agents and they would continue their searching of all the Internet's URLs and bringing back this information to you.

2) This is an excellent question that is being worked on in a number of areas at the present time. Bots and Intelligent agents need to communicate with each other and to communicate with humans. In BotSpot® we have a number of excellent resources about this:

http://www.botspot.com/language_and_code1.htm

This section on Language and Code covers many many pages that discusses the latest attempt and creation on Language and code for bots and Intelligent agents. FIPA is working on a number of incentives. KQML, AIML, WEDL and others languages have been discussed. You will find many resources in this section.

<http://www.botspot.com/faqs/faqs1.htm>

The FAQs and Libraries also has some excellent resources for Human Computer Language, FIPA, KQML resources as well.

<http://www.botspot.com/robotguidance/>

This is BotSpot® initial stage for the creation and direction of robot guidance and communications. Remember this is just a staging area and we are constantly looking for input and volunteers. Intokmi folks have show a very strong interest lately as well as others.

So you can see there is no little answer to your questions. I hope that I have added fuel to your fire and that you may continue your thirst for continued education in the fields of bots and intelligent agents and BotSpot® be your well that runneth over :-)

Cheers

Marcus

P.S. You might want to subscribe to our free monthly newsletter if you have not done so already. It is available at <http://www.botspot.com/newsletter/> or you may subscribe directly by email at join-botspot@botspot.com

Daniel Farinha wrote:

> Hello Marcus,
>
> I asked for a chat yesterday at ICQ (my nick: Drakkula) but my net
> connection went down half way my question.
> I think you might be the right person to ask a couple of questions about
> agents:
>
> As I said in the chat, I had a word with some guys in the AltaVista group
> about meta-search agents, and they told me that if these tools get too
> popular they could deny access to the search engine. They say that the
> agents act as a front end for the search engine, preventing their sponsor's
> adds from reaching the users.
> It's true that without the income from the adds the search engines would
> just close doors.

> Does this mean that search agents have no real future?
>
> My other question is about this idea I had some time ago.
> With the increasing number of agents on the web, wouldn't it be a good idea
> to create a standard which allows all agents to communicate, and negotiate
> online, creating a world wide co-operation system?
> And if this was possible, which entity would be responsible for setting the
> standard?
>
> Thanks in advance for your time.
>
> Cheers
>
> Daniel Farinha

13 APPENDIX E - SAMPLE SEARCH ENGINE COMPONENT (SEC)

```
[secinfo]
sec=altavista
file=altavista.txt
version=0.1
date=19990324

[general]
name=AltaVista
home=http://www.altavista.com/
hitsperpage=10

[query]
url=http://www.altavista.com/cgi-bin/query?pg=q&kl=XX&q=
url={query}
url=&search=Search

[morepages]
url=http://www.altavista.com/cgi-bin/query?pg=q&q=
url={query}
url=&stq=
url={starthitno}
url=&c9k

[totalhits]
html=<b><font face=arial size=-1><b>AltaVista found
html={totalhits}
html= Web pages for you.

[hit]
html=<dl><dt><b>
html={hitno}
html=. </b><a href="
html={hitlink}
html="><b>
html={hittitle}
html=</b></a><dd>
html={hitdesc}
html=<br><b>URL:</b> <font color=gray>
html={hiturl}
html=<br>
html=Last modified
html={hitlastmodified}
html= - page size
html={hitpagesize}4
html=K - in
html={hitlanguage}
html=</font>
html={ignorethis}
html=</dl>

[vardesc]
query=Query string supplied by user
starthitno=Number of first hit to display in page (Select other pages)
totalhits=Total number of hits in search
hitno=Number of a hit
hitlink=Hit URL with protocol header (ie http://www.cnn.com/)
hittitle=Hit(page) title
hitdesc=Description of hit
hiturl=Hit URL without protocol header (ie www.cnn.com)
hitlastmodified=Date when hit was last modified
hitpagesize=Total size of the page in Kb
hitlanguage=Language that page is written in
ignorethis=Normally a link to the BabelFish translation of the hit
```

14 APPENDIX F - SOURCE CODE

