A New Model for C# Client-Side Search Agent

Ebada A. Sarhan                   Iraky H. Khalifa                   Wael S. Tawah
Faculty of Computers & Information  Faculty of Computers & Information  Zagazig University
Helwan University

Abstract
The World Wide Web (Web) offers an uncountable number of documents which deal with information from a never ending list of topics. Thus the question of whether to find information turned into a question of how to find relevant information. This paper presents the design and implementation details of a C# client-side search agent that assists the user in the process of finding information. This search agent contacts multiple web search engines as a seed for starting pages to locate related URLs. Before sending the user’s keyword(s) to the different search engines the synonym of keyword is returned from a lexicon. After receiving a list of possibly relevant sites, the agent contacted the individual web sites and downloading their respective contents. For future search, the URL and the body of the retrieved web sites are stored in a SQL database.

Keywords: Information retrieval, search agent, web crawling and search engine.
1. Introduction

The problems of information retrieval on a large and distributed scale have become highly apparent during the last six years. The information explosion that has taken place, particularly with the exponential growth of the Web, has shown that users treat information, especially hyperlinked information, as a valuable resource. However, the electronic information community is already showing signs of what has been commonly termed “information overload”.

Currently, Internet search engines are one of the commonly used tools for information retrieval. Although these engines are of valuable services, they are attached to several limitations. For example, they only provide the user with the address of the information (location) rather the information itself. The search for information is limited and typically biased towards indexing more “popular” information, and each search engine covers a small portion of information resources on the internet. The task of information retrieving and integration is the user responsibility.

Recently, there is a growing interest in using software agent approach for designing systems that assist users on the Web. Because of the flexible and dynamic characters of software agents, they are being used widely as an interface system between the user and the Web for different applications. For example, an agent that finds web pages for the users, helping users to browse the Web or assisting the user on scientific literature search.

The main objective of this work is to build a client-side search agent that is capable of helping users to locate and retrieve information from distributed resources in an open environment. This agent is based on the C# (pronounced “C sharp”) language and the .NET framework as a novel framework from Microsoft Corporation.

2. Search agent

Searching is an automated process, where the user only gives his/her requirements and the system will try to find the best matches. Several searching tools (known as Network based Information Retrieval Systems) have been developed.

There were many types of Web agents proposed in recent years, which range from domain-dependant agents, like e-commerce agents and information gathering agents, to function-dependant agents, like negotiation agents[12]. Among these agents deployed on the Web, search agents who can query multiple search engines simultaneously are gaining popularity among users. Hence search agents are a form of information gathering agent, providing an integrated interface to accessing multiple search engines.

Web search agents can be divided into two types: server-side search agents that operate from a single server for multiple users, and client-side search agents that operate from the client machine for the benefit of a single user [3].

2.1 Server-side Search Agents

Server-side search agents operate from a single centralized system for the benefit of multiple users. Users initially connect to the agent's home page and retrieve the query form written in HTML (Hyper Text Markup Language). When the form is filled out by the user and submitted to the agent, the agent maps
the query to a form suitable for each search engine it plans to contact.

The agent, then submits the translated form to the search engines and retrieves the results, often performing the task in parallel by multithreading the network connections. Finally, the results sent from the search engines are gathered and sent back to the user who initially requested them. Some of the well known server side search agents are SavvySearch, ProFusion, and MetaCrawler [3].

2.2 Client-side Search Agents

Client-side simply means that it is a program that is installed on the client's machine and most of the processing and transformations are performed on the client-side. This means that the client will share in the data processing tasks, potentially alleviating the heavy load on the server.

Client-side search agents have advantage over the server-side search agents in that network usage because retrieval and filtering process are performed on the client's machine, scalability, and personalizability [8]. Examples of client-side search agents are: Copernic6.11, WebFerret, EasySeeker, Bullseye2, JOC Web Finder1.10, and Fish Search. Unfortunately, there are several drawbacks to client-side search agent [5].

3. The .NET Framework

The Microsoft .NET Framework (.NET for abbreviation) is a great leap forward in the evolution of computing from PCs connected to servers through networks such as the Internet, to one where all manner of smart devices, computers, and services work together to provide a richer user experience. It can be considered a support infrastructure enabling the development of distributed applications [2].

3.1 The .NET Architecture

.NET sits on top of the operating system. Currently, the .NET Framework exists only for the Windows platform, although a version is under development for the FreeBSD and Linux operating systems [8, 10]. It consists of a number of components [9], these are:

- Four official languages: C#, VB .NET, Managed C++, and Jscript .NET.
- The common Language Runtime (CLR), an object-oriented platform for windows and Web development that all these languages share.
- A number of related class libraries, collectively known as the Framework Class Library (FCL).

Figure 1 shows the .NET architecture components. For more details see [4].

3.2 The C# Language

A short introduction to the language used along all of the presented work is presented, focusing on the novelties introduced by such a language with respect to other object-oriented programming languages (Java and C++). C# provides tools and facilities to build the presented search agent. Some of the important features of this language include classes, interfaces, delegates, boxing and unboxing, namespaces, properties, indexers, events, operator over loading, versioning, attributes, unsafe code, and XML documentation generation. C# is similar to Java and to C++ in many concepts and solutions, yet it is different in some fundamental ways.
and behaviors [11]. For example, the C# language does not permit functions to exist outside of class declarations, which is, on the contrary, allowed in C++. It also enables a new degree of language interoperability: software components from different languages can interact as never before. In addition, C# applications interact via the Internet, using industry standards such as the Simple Object Access Protocol (SOAP) and XML [1].

4. Client-side Search Agent Architecture

In this section, we implement a client-side search agent tool called MyAgent, based on C# language and .NET. MyAgent connects to 3 general-purpose search engines, and parses the Web sites obtained from these engines by using regular expressions.

4.1 MyAgent architecture

MyAgent system architecture is shown in Figure 2. There are 6 main components, namely (1) User Interface, (2) Utility, (3) Formatter, (4) Parser, (5) Database, and (6) search agents.

User Interface: it is the means of interacting with the user, through a friendly graphical interface. It consists of two forms, one for users to submit their queries parameters and the other for listing results that are returned by each query. After submitting the query, users have the ability to choose a synonym for their keyword to be searched. A browser window is existed to display the selected URL from the result list. Figure 3 shows the User Interface form that has a keyword (search agent), a synonym (hunt), Max number of links (20), and search engines (Yahoo, AltaVista, and Web Crawler). Figure 4 shows a typical result form for the input parameters in figure 3.

Formatter: it adapts the user's query format to match the format of the search engine with which the search agent interacts. Every search engine has a different query format. All these formats are located in the formatter.

Search agents: users can deal with search engines and their results throw search agents. They use HTTP (Hyper Text Transport Protocol) to communicate with remote Web-accessible search engines.

Utility: a synonym for a keyword is fetched by the Utility component from the Lexicon; synonym is one of the problems that search engines face [7]. It accepts the format for each search engine from the formatter then sends this query format and the synonym to search agents if the database has not that keyword stored in it. After the parsing process, the Utility obtains the URLs, it sends it to the User Interface to be displayed.

Parser: parsing HTML pages is much harder because an HTML page is petty unstructured, it contains different types of information including images, text, and hyperlinks. The Parser component uses regular expressions to process the pages collected by search agents from engines and collects links as well as to filter links that may not be interesting to follow.

Database: in this architecture, we use a SQL server 2000 database management system to store all the keywords submitted by the users and the URLs names and the body of these pages in order to avoid performing the same query.
Figure 1: .NET Framework architecture

Figure 2: MyAgent architecture
Figure 3: User Interface form

Figure 4: A typical result form
4.2 MyAgent Implementation

We have developed MyAgent, a client-side search agent, which queries and gathers information from multiple search engines distributed over the Web. The principle design goal of MyAgent is to assist users in locating relevant information resources available on the Web while minimizing network usage and server loading that is running the engine.

It's developed using regular expressions, `httpWebRequest`, `httpWebResponse`, `webHeaderCollection`, `streamReader`, `stringBuilder`, `Uri`, `Hashtable`, `sqlConnection`, Threading, and delegates. Snapshots of the C# code are presented below.

4.2.1 Getting web pages as string

Microsoft .NET Framework SDK already provides the necessary classes to fetch a Web page and convert the result into a string. To get a page whose URL is given, we use the `httpWebRequest` and `httpWebResponse` classes from System.Net. The result is a streamReader that can then be converted into a string using `stringBuilder`.

```csharp
// Creates an HttpWebRequest for the specified URL.
HttpWebRequest req = (HttpWebRequest) WebRequest.Create(addr);
// Sends the HttpWebRequest and waits for a response.
HttpWebResponse resp = (HttpWebResponse) req.GetResponse();
StreamReader sr = new StreamReader(resp.GetResponseStream(), System.Text.Encoding.UTF8);
StringBuilder pageBuffer = new StringBuilder();
String line;
while ((line = sr.ReadLine()) != null)
{
    pageBuffer.Append(line);
}
sr.Close();
```

r = new Regex("href=\s* (?:'(?<1>[^']*)'|"(?<1>[^""]*)"|(?<1>[^>\s]+))", RegexOptions.IgnoreCase|regexOptions.Compilited);
for(m=r.match(page);m.Success;m=m.NextMatch())
{
    link = m.Groups(1).ToString().ToLower();
    link = FilterLinks(link);
    if(link.Length!=0) bageOfLinks.Add(link);
}

4.2.2 Parsing a Web page using regular expressions

After a page is fetched, we have an HTML page in the form of a string. There are many possible ways to parse it. We use regular expressions to collect all the links in it.

Although regular expressions look rather cryptic (except to seasoned Perl programmers), they have the advantage of being powerful and efficient. Microsoft .NET regular expressions are compatible to Perl regular expressions, and in addition have an object-oriented outer wrapping [4]. The above classes, properties, and methods (`Match`, `Groups`, `Success`, and `NextMatch`) are used together to process the results of the matching.

```csharp
r  = new Regex("href\s*=\s* (?:'(?<1>[^']*)'|"(?<1>[^""]*)"|(?<1>[^>\s]+))",
regexOptions.Compilited);
for(m=r.match(page);m.Success;m=m.NextMatch())
{
    link = m.Groups(1).ToString().ToLower();
    link = FilterLinks(link);
    if(link.Length!=0) bageOfLinks.Add(link);
}
```

5. Evaluation
In this section, we present various statistics and measurements obtained from comparing MyAgent and other free and available client-side search agents like (Copernic 6.11, WebFerret, Web Seeker, JOC Web Finder, and BullsEye). Based on [9] a twenty five (25) standard test searches which were devised in different subject areas and related words, we measure the overall performance of the six (6) selected client-side search agents. In general, single term search queries were presented to the search agents using their own search forms interfaces with default setting using exact phrase search mode.

5.1 Ranking Protocols and Computations

The search agents were rated on the selected search queries in four ways: (1) The total number of hits retrieved, (2) The position in the search output list of the first relevant hit, (3) The number of relevant items in the first five hits listed, and (4) An Ordinal Ranking for each search agent within each query based on an evaluation of the first three factors where: 6 = top of 6 tested, 1 = bottom of 1 tested, and 0 = failed the test. The following assumption symbols will be used during the evaluation process.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TH</td>
<td>The Total number of Hits retrieved by an agent.</td>
</tr>
<tr>
<td>PFH</td>
<td>The Position in the search output list of the First relevant Hit.</td>
</tr>
<tr>
<td>N5H</td>
<td>The Number of relevant items in the first 5 Hits listed.</td>
</tr>
<tr>
<td>RR</td>
<td>The Relative Ranking assigned to each search agent on the basis of a comparison of the absolute ranking computed for each search agent on each query.</td>
</tr>
<tr>
<td>OR</td>
<td>An Ordinal Ranking for each search agent within each query based on the first three factors where: 6 = Top of the test, 1 = bottom of the test, 0 = failed the test.</td>
</tr>
<tr>
<td>HR</td>
<td>Hit Ranking. An assigned index value based on TH where: 3 = less than 20 hits retrieved, 2 = from 21 to 100 hits retrieved, 1 = over 100 hits retrieved.</td>
</tr>
<tr>
<td>AR</td>
<td>The Absolute Ranking measurement.</td>
</tr>
</tbody>
</table>

AR is achieved by each search agent was computed using three figures: HR, PFH, and, N5H. Returns of less than or equal 20 hits were considered to be positive for users and were given the highest HR of 3. Returns of more than 100 hits were considered to be the most negative for users and were given the lowest HR of 1.

AR was computed as follows, \(( HR * ( 10 – PFH ) * N5H )\). This formula yielded a number for AR of each agent for each query. The AR was used to compute the RR, with the agent receiving the highest AR at 100%.

OR from 6 (highest) to 1 (lowest) and 0 (failed) were assigned on the basis of the computed AR. Where computed AR were identical the highest OR was given to the search agent with the best overall retrieval results. Where all else was equal search agents with identical computed AR were given the same OR. Table 1, Shows an example result from the
search query "Abelard and Heloise Letters".

In the example result Table 1, MyAgent receives the highest OR even though the computed AR for BullsEye is identical because the TH retrieved is higher. WebSeeker receives OR (0) because it failed the search.

The speed of returned search results was not significant in the overall ranking processes since all of the tested search agents generally returned results within less than twenty, and usually within ten, seconds even at times of peak Internet usage. But the total number of hits returned was significant in the overall ranking processes. Moreover search agents which either do not return total hits counts, or which make discovery of total hits difficult were ranked lower.

**5.2 Evaluation Summary**

Table 2, summarizes all the obtained results from all the twenty five search query results(Table 1 like). In that table, Cumulative Absolute Ranking (CAR) column shows a cumulative AR for the 25 test queries for each tested agent.

Cumulative Ordinal Ranking (COR) column shows a cumulative OR for the 25 test queries. The highest achievable is 6 X 25, or 150. No search agent tested achieved this score. It is clear from these results that by this measure of performance, based on a computed AR, MyAgent is ranked at the top of the tested agents.

Relative Absolute Ranking (RAR) column shows that the top ranked search agent by this measure is MyAgent. RAR is computed as follows, (CAR / ( largest CAR ) *100 ) for each test query. Clearly, strong performers are also Copernic and BullsEye. WebSeeker is the relatively poor performance agent.

Times in Top 3 (TTT) column shows the number of times each search agent was ranked in the top three; that is, received an OR of 4 or higher, on each test query. The maximum achievable score is obviously 25. MyAgent is, the clear leader, having been ranked in the top three 23 times out of the 25 test queries. Close behind are BullsEye and the Copernic at 19 and 15 times. Also relatively strong compared to the lowest ranked agents are WebSeeker and JOC WebFinder.

The Average Relevancy computed for the first 5 hits returned by the search agents on each test query is presented in ARF5H column. It is computed as follows, ((total of N5H / 25) *100). The search agents consistently ranked among the top four, MyAgent, JOCWebFind, BullsEye and Copernic, are the top ranked by this measure, as well.

ARR column shows the Average Relative Recall computed for each search agent over the 25 test queries. Bear in mind that recall is not necessarily a measure of good performance, especially with regard to the Web search agents. Some consistently produce wildly inflated recall figures in the tens or even hundreds of thousands of items. This kind of return can be very daunting to novice and casual users and could be considered a search failure by some. Interestingly, the WebSeeker agent shows the highest recall. This relatively high recall linked with the demonstrated poor relevancy among the first 5 hits makes WebSeeker a very bad search agent.
### QUESTION

<table>
<thead>
<tr>
<th>Agent name</th>
<th>TH</th>
<th>HR</th>
<th>PFH</th>
<th>N5H</th>
<th>AR</th>
<th>RR</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copernic</td>
<td>26</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>54.00</td>
<td>15.79</td>
<td>3</td>
</tr>
<tr>
<td>WebFeret</td>
<td>72</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>72.00</td>
<td>21.05</td>
<td>4</td>
</tr>
<tr>
<td>WebSeeker</td>
<td>988</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>JOC Web Finder</td>
<td>145</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>36.00</td>
<td>10.53</td>
<td>2</td>
</tr>
<tr>
<td>BullsEye</td>
<td>22</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>90.00</td>
<td>26.32</td>
<td>5</td>
</tr>
<tr>
<td>MyAgent</td>
<td>33</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>90.00</td>
<td>26.32</td>
<td>6</td>
</tr>
<tr>
<td><strong>TOTALS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>342.00</strong></td>
<td><strong>100.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: An example result

### SUMMARY

<table>
<thead>
<tr>
<th>Agent name</th>
<th>CAR</th>
<th>COR</th>
<th>AR</th>
<th>TTT</th>
<th>ARF5H</th>
<th>ARR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copernic</td>
<td>1393</td>
<td>96</td>
<td>76</td>
<td>15</td>
<td>12.3</td>
<td>5.3</td>
</tr>
<tr>
<td>WebFerret</td>
<td>1251</td>
<td>89</td>
<td>68</td>
<td>13</td>
<td>11.7</td>
<td>8.8</td>
</tr>
<tr>
<td>WebSeeker</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>61.4</td>
</tr>
<tr>
<td>WebFinder</td>
<td>1152</td>
<td>74</td>
<td>63</td>
<td>5</td>
<td>14.6</td>
<td>12.8</td>
</tr>
<tr>
<td>BullsEye</td>
<td>1783</td>
<td>113</td>
<td>97</td>
<td>19</td>
<td>13.3</td>
<td>4.7</td>
</tr>
<tr>
<td>MyAgent</td>
<td>1830</td>
<td>128</td>
<td>100</td>
<td>23</td>
<td>16.5</td>
<td>7.1</td>
</tr>
</tbody>
</table>

Table 2: Summary of results.

**6. Advantages of MyAgent**

According to MyAgent architecture it has the following advantages: downloads all the pages that the search engines return and eliminates those pages that have changed and no longer contain the user's keywords, uses multiple search engines at the same time and handles the syntax differences between the various different search engines, removes all of the dead links saving user's time, removes duplicates by Title and by URL not only for one search engine, but across all of the search engines (This can be a huge time saver), saves the results of the user's search and runs the same search again later, an easy to use interface that becomes second nature through repeated use, the ability to choose search engines to be used and ability to specify the maximum number of results to be returned, works with a proxy or with a single client, provides an internal browser [6], and retrieves relevant information.

**7. Conclusion**

The Web is potentially the richest database of human knowledge ever constructed, and yet we are consistently frustrated by our inability to query that database in an efficient manner. In this paper, we have seen MyAgent architecture for searching the Web, which bring us closer to that goal of efficient data retrieval. On the basis of the results from twenty five (25) test queries the free and available five (5) and the proposed C# Web search agents are ranked here as follows, (1) MyAgent, (2) BullsEye, (3) Copernic, (4) WebFerret, (5) JOC WebFinder, and (6) WebSeeker.

MyAgent search agent is clearly superior to all others on the basis of several test measures. The ease of use and simplicity of the MyAgent user interface also make it the
strongest candidate for recommendation as a good search agent.

8. Future Work

Problems will arise when the search engines change their query syntax, making the queries sent by the search agents obsolete, and not all users will realize this misbehavior. Moreover these search engines rely on centralized indices that are updated periodically yielding outdated results and network overload. It is more efficient to send an agent to the search engine's server, where it can locally search for links. It can also follow these links and do a full text search in each link host and return results to the user either he/she is connected to the Internet or not (The results of the search may then be emailed to the user).

Future work will focus on intelligent mobile agents for Web retrieval based on C# language and EtherYatri toolkit[10].

8. References


