Searching the Web via Mobile Agents

Corrado Priami, Matteo Lonardi and Stefano Martini
Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie 1, I-37134 Verona, Italy
priami@sci.univr.it

Abstract

We describe an approach to information retrieval and filtering on the Web based on mobile agents technology, allowing users to be disconnected from the net while the query is being resolved by some agents. We reduce the space solution of queries by including some knowledge in the code of the agents, making users save time in searching the Web. These ideas are implemented in a prototype called MAWS (Mobile Agents for Web Searching).

Introduction

The amount of information stored in the Web is rapidly growing and it lacks structure. Many tools (*search engines*) that search the Web according to keywords have been implemented (Altavista, Lycos, Magellan, Excite, HotBot, Arianna). They are essentially indexes of the Internet. The server sites of these engines scan the network daily or weekly and build a keyword index. When the user submit a query to them, they return the set of links available for the keywords specified.

Search engines usually return hundreds or even thousands of links (sometimes with duplications) for a simple query, and the user cannot check all the links. The association of a quantity (usually a percentage) to any link expressing an importance rate (*relevance*) may help users. Unfortunately, the relevance of a document is computed on purely syntactical grounds. For instance, the number of occurrences of the keyword in the document is a typical measure. This measure can be misleading due to the many different meanings that a keyword may assume according to the context in which it occurs. When sentences are used in place of keywords, the problem is even more evident.

Finally, the different user interfaces of search engines influences their users in the choice of the tool preferred. Also, once an engine has been adopted it is difficult to pass to another even if more suitable for the query at hand. There exists some interface agents that accomplish similar tasks (Lieberman 1997; Armstrong *et al.* 1995; Balabanovic & Shoham 1995; Rhodes & Starner 1996; Maes & Kozierok 1993; Lashkari, Metral, & Maes 1994; Chen & Sycara 1997; Yan & Garcia-Molina 1995) (see Sect. for further details). However, none of them relies on mobility of code: the next feature we describe.

Our main goal is to save user's time for searching the Web by improving the query solution and by limiting the activities that need a stable connection to the net.

We apply here mobile agent technology. We mean by mobile agent an entity that is made up of a piece of code, some data and an internal state, and that can migrate from site to site while computing. Hence, such an agent can be remotely executed on different systems at different times possibly interacting with other agents. More abstractly, an agent is something that acts for its creator by trying to accomplish its delegated task.

As far as performance is concerned, we show in Sect. that mobile agents usually generate less network traffic than classical approaches thus optimizing the usage of the bandwidth available. Also, the search precision is not affected by mobile agent technology because it is simply a different paradigm in which to implement the same algorithms.

We spawn some agents over the net to look for documents that satisfy the query of the user. While the agents are roaming the net, the user performs other tasks or even disconnects himself. To allow disconnection each agent sends the results of its search to the mailbox of the user either as link or as full document according to the choices of the user. The agents include an analyser to limit the number of documents satisfying the query. We implemented these ideas in a prototype called MAWS (Mobile Agents for Web Searching). Since we implemented MAWS relying on a Java environment (AWB by IBM), we limit the need of installing a runtime support on any machine because almost all of them already support Java applets.

MAWS logical architecture

We now describe our search prototype that is under implementation at the University of Verona. MAWS interacts with the user helping him in finding the appropriate logical structure of his query, possibly reformulated for successive refinements. We aim at designing and implementing

- an user interface to query heterogeneous data management systems;
- a module that interprets and optimizes the queries of the users;
- the computation of the relevance of documents with respect to a query.

We encode in the agents the access methods to the search engines, and the morphological structure of the user's natural language to derive synonyms, variants of the keywords and their context.

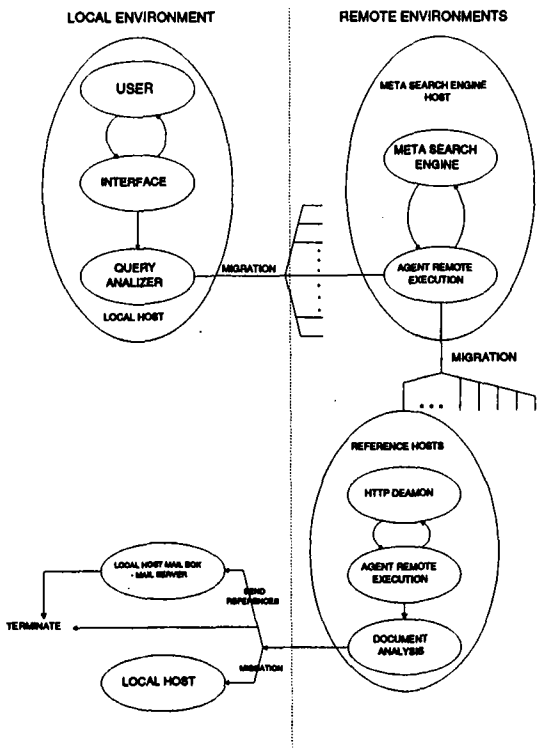

Figure 1: MAWS logical architecture.

MAWS is made up of four main modules: the user interface, the query analyzer and optimizer, the query mobile agent, and the analyzer mobile agent. A scheme of the logical architecture is in Fig. 1 where we also highlight the environment in which actions are performed. The local environment is the user machine, while the remote environments can be the host of a search engine or the host containing a relevant document.

The *user interface* permits to select a list of search engines to which address the query, to select a maximum number of documents to be analysed and their minimum relevance, to choose some domains and patterns to be excluded in the search. This resembles an advanced query interface of a search engine. The advantages of our solution are that the user can select a list of search engines to be queried simultaneously, and once the queries have been delivered to the search engines via agents, the user can disconnect himself. The answers to the queries will be sent back via `e_mail`.

The second module of MAWS is the *query analyzer and optimizer*. It gathers as much knowledge as possible on the interests of the user.

The analyzer tries to disambiguate the context of the keywords inserted in the query by relying on dictionaries and thesaurus of the language, and on interaction with the user. This task is performed in the environment of the user (see Fig. 1). The analyzer suggests a list of contexts or domains for the keywords to help the user. The list of candidates is built according to a statistical method called *Trigger Pairs Model* (hereafter, TPM for short) (R. 1994). A data structure specifies the correlation between the words of the natural language in which it is expressed the query. The data structure is dynamically updated according to the solutions of the queries and their effective relevance for the user. In other words this data structure records some user's interest profiles. Finally, since the larger is the number of words selected to refine a keyword, the slower can be the solution of the query, the user can set in the interface a bound to the number of words to be used. More details are in the section on the implementation issues.

The third module is the *query mobile agent*. Once the query has been formalized in the languages of the search engines selected, we obtain a set of URLs[1]: one for any search engine. The structure of URLs for queries is quite standard and logically similar for most of the search engines. We only need to change the format of parameters and the path of the CGI [2] programs. Due to this similarity, we have a single module that can interact with many different search engines and can dynamically vary the set of engines enabled.

Note that any browser works by interacting with a CGI program of the server hosting the search engine. Therefore, we only need to know the format of the parameters for this CGI program to properly format the query and directly submit it to the CGI without relying on a browser. Hence we do not need an open browser window. Some interface agents instead (e.g., (Armstrong *et al.* 1995; Balabanovic & Shoham 1995)) needs to cooperate with a browser to work.

Now as many mobile agents as the number of search engines selected are spawn to the hosts of the engines,

[1]The Uniform Resource Location (URL) is a standard to identify resources over the net (Berners-Lee, Mastiner, & McCahill 1994; Fielding 1995).

[2]The Common Gateway Interface (CGI) (Felton 1997) is a standard to interface applications with Information Servers like Web servers or HTTP servers. The programs that handle input and output between the applications and the information servers run on the machine in which they are located on demand by a remote client. They can receive input parameters as a single ASCII string, and they can return results.

and the user can disconnect his device from the network, provided that he has a mailbox to which the agents can send the results of their work. This step makes our approach very different from a standard search engine, to which one must be connected for all the time of the search. These agents wait locally to the host of the search engine for the result of the search. Then they erase links duplicated. Finally, they spawn an analyzer agent towards the site of any link in the result of the query that will analyse the corresponding documents. After all the agents are sent, the query agent kills itself.

The last module is the *analyzer mobile agent.* It checks whether a document returned by a search engine is effectively interesting for the user. According to the TPM approach, this agent includes a profile of the user and the implementation of the multiple TF-IDF algorithm that extracts from documents their representative sets. Representative sets are then matched against the profile.

Once the evaluation is completed, the links of the interesting documents (or the documents themselves according to the choice of the user in the interface module) are sent by `e_mail` to the address of the user.

According to Fig. 1 we partition the operations of our prototype into the ones performed in the user's environment and the ones performed in other (remote) environments.

First MAWS interacts with the user to set up the query mobile agent, and handles the information on the profile of the user. Therefore our prototype acts as an *interface agent.*

The second phase concerns the analysis of documents on remote environments. Our agents are now *autonomous*, i.e. they act without human intervention. Since MAWS operates both as interface and autonomous agent, it meets the requirements singled out in (Lieberman 1997) for being a useful Web search agent.

Implementation

To make MAWS as *portable* as possible we implemented it in *Java.* We used the visual environment AWB by IBM built on top of Java to design and implement network applications based on mobile agents. The AWB provides the user with an *Aglet class* that includes all the functionalities of mobile agents. In particular, the aglet class assigns a unique name to an agent, defines a migration plan to specify multiple destinations, automatically deals with agent failures, allows agents to cooperate in environments with shared resources, allows agents to exchange messages, and has a class loader to make Java agents travel. The objects of the Aglet class, called *aglets*, are an evolution of the Java applets. Applets are pieces of code that can be downloaded, instantiated and executed locally by a browser. Thus applets by no means have the properties of autonomous migrating agents. Aglets instead provide their users with the notion of mobility as it appears from the features of the Aglet class listed above. Migration relies on the Agent Transport Protocol (hereafter ATP for short) (Lange & Aridor 1997) provided by AWB and proposed for standardization. It is developed at the application level and it is platform independent.

To make MAWS as *flexible* as possible we implemented a module that dynamically updates or builds descriptors of search engines. These descriptors are then used to format the ASCII string of parameters for the corresponding CGI programs. The descriptor of a search engine is a table whose first two rows store the ATP and the HTTP address of the search engine including its CGI, respectively. The other rows describe the format of the CGI parameters. The last three rows specify the pair of HTML tags delimiting the URLs which are the solution of the query and the URLs that must be excluded from the list of results.

To deal with *contexts* or *information domains*, we rely on statistical language modelling (Brown *et al.* 1990; Cerf-Danon & Elbeze 1991). These models encode statistical properties of languages and are used to predict some behaviour of documents given their prefix. Statistical language models are usually built by automatically analyising a large amount of data (the training set) in a phase called *training.* Its result is a set of vectors of words that identify a profile of the user. We then compute a measure of the correlation among these words called *mutual information* (hereafter, MI for short). The meaning of MI can be given in terms either of classical probability theory relying on the Bayes' rule (as done in (Balabanovic & Shoham 1995; Gauch & Futrelle ; Chen & Sycara 1997)) or of information theory relying on the notion of entropy (Shannon 1948) (as done in (Shannon 1951; R. 1994)). In terms of textual documents, we can say that the mutual information $MI(h, w)$ represents the amount of information that the sequence h of words in a document provides on the next word w.

We instantiate the above framework to MAWS. The training phase is completely local to the environment of the user. He needs to provide MAWS with a quite large set D of documents that he his interested in. Following (Salton & McGill 1983) the documents $d_i \in D$ are represented as vectors of words within a vector space. The basic idea is that similar vectors denotes documents with similar semantics. We associate a weight to any word in a vector and we sort it by decreasing ordering of weights.

To build the vector V_i representing a document d_i, we first filter d_i by deleting the HTML tags, by deleting the words with very high frequency such as articles, prepositions and conjunctions, and by putting words in their most general forms (verbs in their infinite form, nouns in their singular form, etc.). Then, we apply the multiple TF-IDF approach (Salton & McGill 1983) to compute the weight $v^{(i)}$ of a word w_i in the vector V_i through the equation

$$v^{(i)} = TF(w_i, d) \times IDF(w_i)$$

where $TF(w_i, d)$ (*term frequency*) is the number of occurrences of the word w_i in the document d, and $IDF(w_i) = \log(|D|/DF(w))$ (*inverse document frequency*) with $DF(w)$ (*document frequency*) being the number of documents in which the word w_i occurs at least once.

The vector V_i is called the *profile* of the document, or more generally a profile of the user. We impose a fixed size (say m) to the profile vectors and we allow at most n profiles per user. The set of profiles V_i is arranged in a $n' \times m$ matrix V called *profile matrix* with $n' \leq n$.

The user decides when a document is particularly interesting and must be considered for updating the profile matrix. (We are currently investigating the possibility of updating the profile matrix with documents that the user marks as particularly insignificant.) The profile matrix is dynamically updated in the local environment of the user according to the multiple TF-IDF algorithm proposed in (Chen & Sycara 1997). Let n be an upper bound to the number of storable profiles. Let V_j be the profile of an interesting document that is not yet recorded in V. If the rows of V are less than n, we add a new row V_j to the profile matrix. Otherwise, we merge the two vectors having the highest *cosine similarity* computed as

$$Sim(V_i, V_j) = \frac{V_i \cdot V_j}{|V_i| \times |V_j|}, \quad i, j \in [1..n],\ i \neq j$$

where $\cdot$ is the scalar product of vectors. Let V_x and V_y be the two vectors for which $Sim(V_x, V_y)$ is maximum. We now define a vector of dimension less than or equal to $2m$ by joining V_x and V_y. Note that we can obtain a vector of dimension strictly smaller than $2m$ due to multiple occurrences of the same word in the two vectors multiplied. Then, we sort the new vector according to the weights of words and we only keep its first m components to make it fit the matrix size.

To end the training phase, we must compute the MI between the words of the profile vectors. MAWS only computes the MI between the words of the same row because each profile vector represents a cluster of semantically similar documents. The values obtained give raise to m vectors of dimension $m-1$. We add a new dimension to the profile matrix to store the mutual information values. Thus $V \in \mathbb{R}^{n \times m \times (m-1)}$. We define MI as in (Chen & Sycara 1997), i.e.

$$MI(w, w') = P(w, w') \log \frac{P(w, w')}{P(w)P(w')}$$

where $P(w, w')$ is computed as the ratio between the number of documents in D that contains both w and w' and the total number of documents $|D|$ (D represents here the space of suitable documents). More precisely,

$$P(w, w') = \frac{|\{d \in D \mid w \in d \wedge w' \in d\}|}{|D|}$$

and

$$P(x) = \frac{|\{d \in D \mid x \in d\}|}{|D|}.$$

The profile matrix is used for the *query refinement*, in the local environment of the user, and for computing the *relevance* of documents, in a remote environment.

We implemented both an automatic and a semi-automatic strategy to refine queries. The two strategies differ in that the semi-automatic one asks the user for confirmation or suggestions in presence of critical choices. We here describe the automatic solution and we point out where the semi-automatic one requires human intervention.

Let r be the maximal number of words allowed for refinements. For any keyword in the query we define a new profile vector as the intersection of all the ones in which the keyword occurs, and we sort the new vector by decreasing values of mutual information. If the intersection contains more than r words, we keep the first r words of the vector as refinement of the query.

The intersection computed above and taken as the refinement of the query could be initially empty. In such a case, we discard a keyword of the query and recompute the intersection. We proceed this way until a non empty intersection is found. Note that the semi-automatic solution provides the user with a list of words that could be deleted from the query for refinement and the user chooses one of them. The automatic solution decides to discard the word with the smallest value of total mutual information. The total mutual information of a word is the sum of all its mutual information with respect to the other words in the same profile vector. A particular situation arises when all the keywords in the query occur in no profile vector. We handle this case by resorting to interaction with the user that can provide directly refinement words.

The agents in charge of computing remotely the *relevance* of documents need to embody the set of keywords in the query, the minimal relevance value set by the user in the query interface, the profile matrix V and the refinement vector with the corresponding mutual information values.

To take the context of keywords into account, we only consider the five words preceding and following the keyword k in the document (hereafter, we write $C[k]$ for these words). This number of words is a tradeoff between accuracy and computation time (R. 1994). We then check whether these context words occur in the set of keywords or in the refinement vector. If this is the case, to compute the relevance of the keyword k we use

$$R(k) = \frac{\sum_{w_i \in C[k]} VR(w_i)}{\{w_i\}}$$

where the set below the fraction line is made up of the w_i which are keywords or they occur in the refinement vector. Furthermore, $VR(w_i)$ is 0 if the word w_i is neither a keyword nor it occurs in the refinement vector, and it is $MI(w_i, k)$ otherwise. If a keyword occurs more than once in a document, we are investigating if it is better to take an average weighted sum of its relevances or the maximum of its relevances.

The next step is the computation of the relevance of a document. Once we have the relevance $R(k)$ for any keyword in the query, we can again take an average weighted sum of the relevances or the maximum value of them. Then, we normalize the value to compare it with the percentage of minimal relevance set by the user. If the document is relevant, the corresponding link or the document itself is sent by `e_mail` to the user.

Performance considerations

To compare MAWS with a classical search engine with respect to the network traffic generated, we assume that both the user and MAWS download all the documents returned by the search engine. Also, all documents reside on different sites. As far as MAWS is concerned, we assume that it sends by `e_mail` the non relevant documents as well. Of course, we assume the same network load in both approaches, and that MAWS interacts with a single search engine.

Let hereafter i be the bytes of the interface page of the search engine; l be the bytes of a page of links; t be the total number of links returned by the engine; b be the number of links per page; $\overline{d}$ be the average size of an html document; and a be the size in bytes of the mobile agents spawn over the network. Then, the total amount Q_e of bytes transmitted by the engine and the total amount Q_M of bytes transmitted by MAWS are

$$Q_E = i + \frac{t}{b}l + t\overline{d} \qquad Q_M = (t+1)a + tp\overline{d}.$$

We take $(t+1)$ times the size of the agents because we send one of them to the site of the engine, and then one to any site of the documents. Since it is usually $i \geq a$, we have that MAWS performs better when $a < l/b$, which is almost always the case.

Note that the above evaluation makes sense whenever MAWS is used as an interface agent to search engines, and its full capabilities (filtering via users' profiles, disconnection from the network while searching, multiple engine interaction) are not of interest.

Related work

Many agents are available for assisting the user in filtering information. We briefly survey here these kind of agents with which MAWS shares the statistical language modelling for building profiles.

We distinguish two kind of agents usually called *interface agent* and *autonomous agents*. The first kind is based on an interactive interface to dialog with the user while working. Instead autonomous agents works without human intervention. Our prototype MAWS covers both aspects. It works as an interface agent while refining queries and as an autonomous agents while scanning the net to find relevant documents.

Another agent that is both interface and autonomous is *Letizia* (Lieberman 1997). It makes real-time suggestions for Web pages that a user might be interested in browsing. Letizia consider Web searching as a continuous venture between the user and a computer search agent. Similar agents are *Webwatcher* (Armstrong *et al.* 1995), *Lira* (Balabanovic & Shoham 1995) and *Remembrance Agent* (Rhodes & Starner 1996). Webwatcher and Lira differ from Letizia because the last one runs on the client machine, while the first two run on the server. Letizia and Remembrance Agent differ because the former tries to provide new information to the user, while the latter recall to the user some relevant information that he already stored onto his disk. Instead MAWS runs partly on the user site, partly on the server hosting the search engine and partly on the sites that store the relevant information, thus implementing the mobile computation paradigm. Furthermore, MAWS provides the user with new information on demand.

Another work direction is the one of filtering the incoming information to save user's time. Notable work in this direction is in (Maes & Kozierok 1993; Lashkari, Metral, & Maes 1994) where the agents build a model of the user to organize his data. To establish trust with the agent, the user must perfectly know the model of the agent to recover eventual errors. Another agent of this kind is *Web Mate* (Chen & Sycara 1997) that aims at choosing relevant information on the basis of a profile describing the user's interests. The *SIFT system* (Yan & Garcia-Molina 1995) tries to continuously informing the user. However this system requires the user to submit to it a profile describing his interests. MAWS shares with this class of agents the maintenance of a profile of the user based on the notion of mutual information.

At the best of our knowledge no agent system for Web searching is based on the mobile paradigm as MAWS. This is therefore an original feature of our prototype that permits to disconnect a portable device running MAWS after the query has been refined and the agent has been spawn.

Conclusion and further work

We presented a prototype (MAWS) for Web searching based on mobile agents technology. It permits to search the Web being disconnected for most of the search time. The user only need a connection to spawn the agents and then the answer will be received by `e_mail`. Some user's knowledge is encoded in the agents that prune the space of the query solution.

We plan to extend the relevance computation with semantic information encoded in the HTML tags. For instance, a keyword occurring in a title should be considered more relevant than one occurring in a footnote.

MAWS is a brick of a larger project that aims at developing a set of assistants for tasks involving extensive net accesses. We plan to use MAWS as the basis to edit domain-oriented electronic newspapers. The profile of the user drives daily or weekly a search on the Web for new documents potentially useful. The search is performed via mobile agents thus letting the user free to perform other tasks concurrently. Once a set of interesting documents has been collected, an editorial agent put them together in a greater document and submit

it to the attention of the user. Of course, the profile of the user interests must be updated on the basis of the effective relevance of the documents collected in the newspaper.

References

Armstrong, R.; Freitag, D.; Joachims, T.; and Mitchell, T. 1995. Apprentice for the World Wide Web. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Hetereogeneous, Distributed Environments.*

Balabanovic, M., and Shoham, Y. 1995. Learning information retrieval agents: Experiments with automated Web browsing. In *Proceedings of the AAAI Symp. on Information Gathering from Hetereogeneous, Distributed Environments*, 13–18.

Berners-Lee, T.; Mastiner, L.; and McCahill, M. 1994. Uniform resource locators (URL). Technical report, RFC 1738, CERN.

Brown, P.; Cocke, J.; Della Pietra, S.; Della Pietra, V.; Jelinek, F.; Lafferty, J.; Mercer, R.; and Roosin, P. 1990. A statistical approach to machine translation. *Computational Linguistics* 16:79–85.

Cerf-Danon, H., and Elbeze, M. 1991. Three different probabilistic language models: Comparison and combination. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 297–300.

Chen, L., and Sycara, K. 1997. WebMate: A personal agent for browsing and searching.

Felton, M. 1997. *CGI Internet Programming.* Prentice-Hall.

Fielding, R. 1995. Relative uniform resource locators. Technical report, RFC 1808, UC Irvine.

Gauch, S., and Futrelle, P. Experiments in automatic word class and word sense identification for information retrieval. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval.*

Lange, D., and Aridor, Y. 1997. Agent transfer protocol - ATP/0.1. Technical report, IBM - TRL.

Lashkari, Y.; Metral, M.; and Maes, P. 1994. Collaborative interface agents. In *Proceedings of the 12th National Conference on Artificial Intelligence.*

Lieberman, H. 1997. Autonomous interface agents. In *Proceedings of ACM Conference on Human-Computer Interface.*

Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of the 11th National Conference on Artificial Intelligence.*

R., R. 1994. *Adaptive Statistical Language Modelling: A Maximum Entropy Approach.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

Rhodes, B., and Starner, T. 1996. The remembrance agent. In *Proceedings of AAAI Symposium on Acquisition, Learning and Demonstrations.*

Salton, G., and McGill, M. 1983. *Introduction to Modern Information Retrieval.* McGraw-Hill.

Shannon, C. 1948. A mathematical theory of computation. *Bell Systems Technical Journal* 27:379–423 (Part I), 623–656 (Part II).

Shannon, C. 1951. Prediction and entropy of printed English. *Bell Systems Technical Journal* 30:50–64.

Yan, T., and Garcia-Molina, H. 1995. SIFT-a tool for wide area information dissemination. In *Proceedings of the USENIX Technical Conference.*