SHARK, a Multi-Agent System to Support Document Sharing and Promote Collaboration

Antonella Di Stefano[1], Giuseppe Pappalardo[2],
Corrado Santoro[1], Emiliano Tramontana[2]

University of Catania
[1]Dept. of Computer and Telecommunication Engineering
[2]Dept. of Mathematics and Computer Science
Viale A. Doria, 6 - 95125 - Catania, Italy
{adistefa,csanto}@diit.unict.it
{pappalardo,tramontana}@dmi.unict.it

Abstract

This paper describes SHARK*, a document-sharing multi-agent application providing users with a means to share documents and allowing keyword-based search.* SHARK*'s agents operate transparently—with respect to the user—by analysing (*i*)* user's personal documents *and (*ii*)* user's actions *performed during web browsing. Activity (*i*) aims at categorising users' documents, in order to allow keyword-based search. Activity (*ii*) aims instead at determining user's interests. Both activities provide an improved documents searching ability and help finding users who share common interests, thus promoting potential collaboration among them.* SHARK *is designed to run in* AgentCities*, a world-wide network of multi-agent platforms based on the FIPA standard.*

1 Introduction

Nowadays several technologies are used to remotely find documents of interest over a network. Alongside web browsing in search of resources published on web servers, an increasing popularity has been gained by software systems based on the peer-to-peer (P2P) concept, whereby users can search documents directly on other users hosts and download those found interesting [17, 37, 35, 34, 36].

P2P and file sharing systems are the subject of intense research. Since the birth of Napster, researchers have spent much effort in providing effective approaches for supporting file distribution and query in a P2P system [27, 3, 38, 9, 4]. Such approaches aim at finding the best way to distribute knowledge about the files to be shared, in order to speed up searching and downloading. Basically, the objective is to improve performances by exploiting replication, while ensuring information consistency.

However, in document sharing and searching, another important aspect is the *quality of search*, i.e. the ability to find the documents with the *right relevance* with respect to a user-provided query. This issue depends not only on the algorithms used to perform the search, but, above all, on the technique used to extract the meaning—e.g. the most relevant keywords—from shared documents (unless of course, as in "semantic" P2P [2], documents already carry ample descriptive metadata). For this information to be optimally exploited, it is important to appropriately disseminate it among the P2P hosts. Finally, it would be highly desirable to make document search more effective by tailoring it to the user, dynamically adapting it to her/his interests.

In view of this *personalisation* goal, it struck us as a waste of useful information that, usually, the activities of browsing the web and using P2P software systems are carried out independently from one another. In this work, we propose an infrastructure that can benefit from data gathered from the documents a user chooses to share, and the observation of his/her web browsing activity. From this information, we build *user profiles*, useful to personalise document search in accordance with user's interests and observed behaviour. Our P2P search support is, in this respect, somewhat similar to techniques used by web search engines that try to match a known user profile with current requests.

We have designed the SHARK (SHARing Knowledge) document sharing system with a view to the three noted

objectives: *extraction* of metadata from documents, their *dissemination*, and search *personalisation*. Important design features accomplished by SHARK are: (*i*) maintaining a high autonomy between its "intelligent" information handling and typical P2P activities, i.e. document sharing, searching and downloading, and, at the same time, (*ii*) providing a component-based framework that allows not only reuse of components but also integration of new functionalities. Indeed, the model of software agents particularly fits the design and implementation of *component-based*, *autonomous* and *intelligent* activities, like those related to metadata extraction and searching. In addition, personalisation can be easily obtained by means of specialised *personal* agents [18, 21, 22, 13, 10, 12], which are able to observe user behaviour, infer the goals the user wants to achieve with her/his activities, and help him/her in reaching these objectives. Such an "assistance" capability is exploited in SHARK to integrate agents with existing client/server applications (like web browsers) in order to transparently understand user preferences. The knowledge on these user preferences is then used by SHARK agents, which, by means of cooperation, are able to discover relations between users interests and promote collaboration among users.

Our prototype application for SHARK is a system for the dissemination and retrieval of scientific and educational documents; it has been designed to be used by *professors* and *students*. *Professors* are SHARK users that publish their scientific and educational documents, while *students* exploit SHARK services only to perform document retrieval by means of the web-based search support.

The remainder of this paper is structured as follows. Section 2 gives an overview of the proposed infrastructure and outlines the work of the agents. Section 3 details the behaviour of each proposed agent. Finally, conclusions are collected in Section 4.

2 Overview of SHARK

2.1 System Architecture

The working scheme of SHARK is strongly based on the topology and the architecture of the network hosting it, i.e. the *AgentCities* project [33], a world-wide network of hosts—we call them *AgentCities Servers*—running FIPA-compliant [16] agent platforms (such as JADE [16], FIPA-OS [1], etc). The aim of AgentCities is to provide the execution environment for distributed multi-agent applications. In particular, the testbed for SHARK is the Italian branch of AgentCities, which is the ANEMONE network [30].

In SHARK, computers, connected through the Internet, may play two different roles (see Figure 1): *client machines*, which are the user's PCs, and *AgentCities Servers*, which act as access nodes for document gathering and query. A SHARK network is structured hierarchically, for AgentCities Servers are responsible for supporting client machines (preferably belonging to the same AgentCities site). An AgentCities Server and the associated clients run each an instance of an agent platform (e.g. JADE); moreover, AgentCities Servers must also run a web application server (e.g. Tomcat [31]), in order to allow interaction among agents and WWW pages.

2.2 Activities and Services of SHARK

The essential activities performed by SHARK are the following:

1. Analysing and categorizing users' (public) documents, in order to extract the information needed for document search.
2. Analysing activities performed by the user in everyday web browsing in order to understand his/her preferences and interests.
3. Providing a web-based interface in order to assist the user not only in finding the documents related to his/her query, but also in contacting other SHARK users who share similar interests.

The services listed above are managed and supported by a set of agents executing on the AgentCities Servers and on each client machine. The activities performed by these agents aim at offering a support for sharing and searching documents, highlighting potential areas of collaboration among users with common interests. In detail, the agents' activities can be described as follows.

- Agent Cruncher is responsible to gather and analyse user's documents, in order to extract a set of *keywords*.
- Agent Categoriser, on the basis of the keywords extracted, autonomously associates some *categories* to the documents analysed by the Cruncher.
- Agent UserProfiler is connected with the user browser to sense his/her activity and continually update the user profile accordingly [11, 10].
- Agent Searcher performs document searches on the basis of a user-provided query, also aiming at finding other users whose profile may be of interest with respect to the given query.
- Agent Correspondent handles document download requests originating from other users.
- Agent Advertiser autonomously finds users whose profiles match each other (to a certain degree).

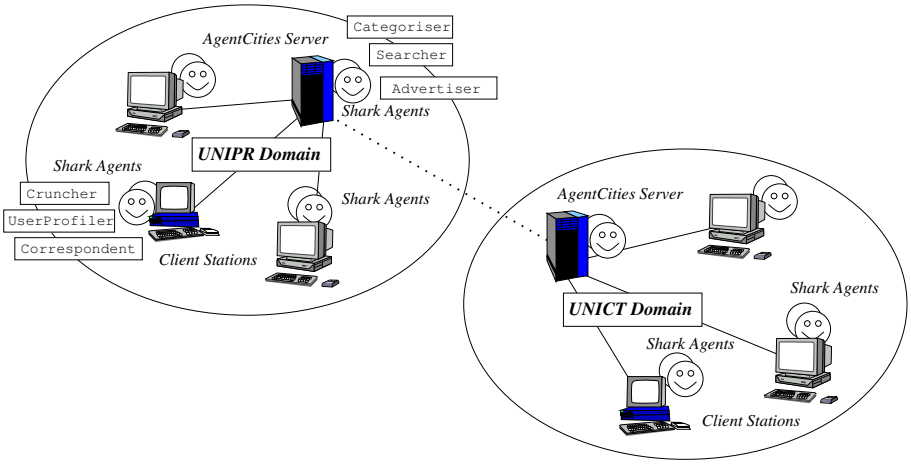

Figure 1. Overview of the Architecture of SHARK

As Figure 1 illustrates, Agents Cruncher, UserProfiler and Correspondent run on each client machine, while agents Categoriser, Advertiser and Searcher run on each AgentCities Server.

3 The Agents of SHARK

In this Section we provide a full description of the purposes and activities performed by each agent of SHARK.

3.1 Gathering Knowledge about User's Documents

Generally, the P2P document sharing model aims at making publicly available a set of documents chosen by the owner. Any SHARK user is thus required to specify which directories, on his/her client, contain the documents that s/he wants to share.

According to such a setting, the Cruncher agent periodically scans the specified directories and analyzes each document found. This aims at identifying a set of keywords that is made available to the rest of the system in order to allow document categorization. The Cruncher employs data extraction techniques reported in the literature [22, 19, 14, 21], some of which have been used by the authors in other agent-based applications [13, 10, 12].

As depicted in Figure 2, the type of the document is determined by strategies modelled on that of the Unix "`file`" utility. Depending on the file type, a suitable *content filter* is applied, aiming at removing all formatting instructions and any other information that is not strictly related to the text contained in the document. For example, for a LaTeX document, the filter removes the comment lines and the LaTeX commands (i.e. `\title`, `\author`, `\begin`/`end{document}`, etc.); while for an HTML

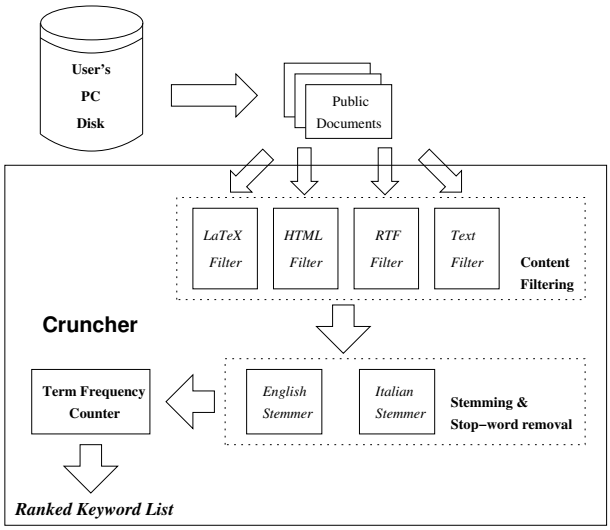

Figure 2. Tasks of the agent Cruncher

document, the filter removes comments, scripts and tag strings. In the current SHARK prototype implementation, the filters are only provided for LaTeX, HTML, RTF and plain-text[1] files.

The result of the filtering phase is the plain text of the document and some additional metadata that qualify the document itself. In particular, these metadata are the *title*, the *language* and the *keywords* given by the document's writer. Indeed, these metadata can be extracted only for some kinds of documents. LaTeX and HTML documents have this information encapsulated in proper tags[2], while, for other document types, these metadata are unavailable.

The output of the filtering phase is then sent to the *stemmer*, a Cruncher's module entrusted with the task of extracting the stem of each word by applying Porter's algorithm or a derived one [26, 32]. A different stemming submodule is employed according to the document language—if provided in the metadata—while, if the language is unknown, the default English stemmer is used. The stemmer sub-module also performs stop-word identification by removing all the words (i.e., articles, conjunctions, adjectives, etc.) that are useless for categorising a document. The list of stems is finally given to a module that performs a frequency count of each term. The result of this last phase is a list of stems, ranked accordingly to their frequency in the document.

At this stage, two kinds of data are available for further processing:

1. the list of keywords proper, or *explicit keywords* (e-keywords for short), i.e. those originating from metadata provided by document authors;

2. the ranked list of stems, or *implicit keywords* (i-keywords for short), as determined by the Cruncher.

All this information, together with file names and user personal data[3], is sent by the Cruncher to the Categoriser agent running in the user's AgentCities server, and will be stored in the server database (cf. Section 3.3).

3.2 Understanding User's Preferences

The objective of SHARK is not only to provide a document sharing system but also to promote collaboration among users on the basis of common research and educational interests. For this to be possible, user's preferences should be known to the system that, in this way, can perform the adequate matching among users' interests.

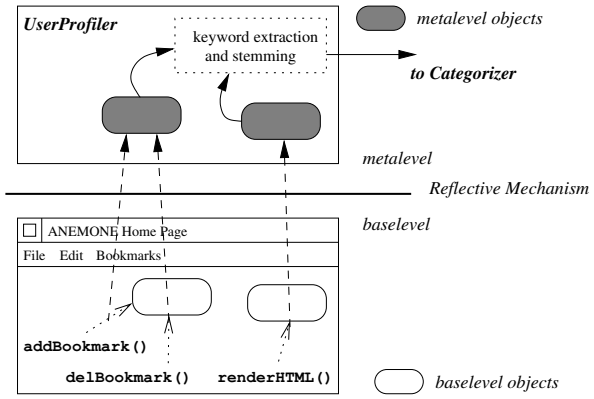

Figure 3. The Reflective UserProfiler **Agent**

To this aim, the user's browser on the client host is interfaced with the UserProfiler agent, which observes user's actions as s/he surfs the Internet, trying to determine his/her interests. The approach employed aims at analysing both the *visited web pages*, in order to extract a set of keywords, and the *actions performed* by the user during browsing (e.g. adding a page to the bookmarks or removing it), in order to understand the relevance of a page for the user.

To make such activities possible, the UserProfiler needs to be interfaced with the browser using a mechanism able to intercept both visited pages and users actions [13, 10]. To this aim, the UserProfiler exploits *computational reflection* [23], a software engineering technology that allows events taking place in an application to be intercepted by some of its components. Thanks to this mechanism, an application can be transparently *extended* by appropriately influencing its behaviour. Essentially, a *reflective system* consists of a *baselevel*, hosting an application, and a *metalevel*, hosting the components interacting with and extending the application. The reflective approach adopted is the *metaobject model* [15, 7, 8, 28, 29]. Given an application written in an object-oriented language, the metaobject model may associate to a selected application (baselevel) object a *metalevel object*. Each operation made on the baselevel object (i.e method invocation, attribute get, attribute set) will then be intercepted by the associated metalevel object, which can choose to modify the behaviour of the baselevel.

This scheme is used in SHARK for the UserProfiler agent, which observes from the metalevel user's interaction with the browser (i.e. the baselevel). Following the approach described in [11], user's actions captured by the UserProfiler are (cf. Figure 3):

1. *Downloading a new web page*. By intercepting the method invoked to render a newly arrived page, the UserProfiler extracts page contents and performs key-

[1] The plain text filter performs only encoding conversions, e.g. Unicode to UTF-8, depending on configuration decisions.

[2] The title is determined by tags `\title` in LaTeX and `<title>` in HTML. Writer-assigned keywords are extracted by analysing tag `<meta name="keywords">` for HTML documents, while, for LaTeX documents, this is possible only if they have a "`keywords`" environment. The document language is also encoded in proper HTML/LaTeX tags/commands (e.g. `<body lang="it-IT">`).

[3] User name, contact info and email address.

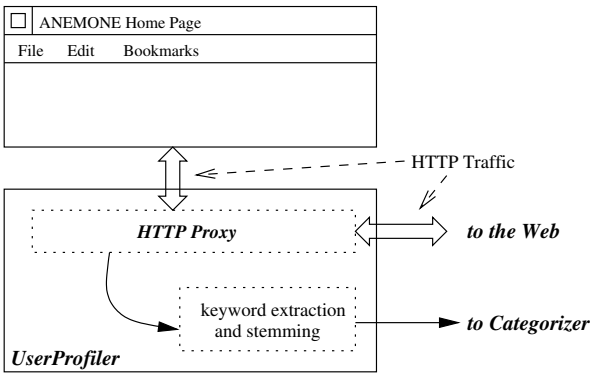

Figure 4. The Non-Reflective UserProfiler **Agent**

word determination and ranking using the same approach as Cruncher (cf. Section 3.1). Keywords from all visited pages are thus collected together (frequencies of the same keywords belonging to different pages are all summed) and ranked, obtaining a single ordered list of keywords. This information is then passed on to the Categoriser.

2. *Adding a URL to the bookmarks*. In this case, the page is considered "very interesting". Therefore, after intercepting the method invoked by the "Add to Bookmark" menu item selection, the page is analysed as described above and its keyword frequency counts are *doubled*.

3. *Removing a URL from the bookmarks*. The page is now considered "not interesting". Therefore, after intercepting the method invoked by the "Delete item" option selection, the associated page is retrieved, analysed and the frequency counts of its keywords are halved.

In our prototype implementation, we extended a Java web browser using the reflective extension Javassist [8] to perform load-time association between UserProfiler's metalelevel objects and the browser's baselevel objects.

Reflective techniques are not always applicable, because they need specific hooks, which consist of e.g. bytecode manipulation libraries for Java, appropriate libraries or precompilers for C++, or reflective languages. To overcome this issue, we designed a non-reflective UserProfiler agent that interacts with the browser by acting as a *web-proxy* between the browser itself and the Internet (see Figure 4). In this way, the UserProfiler is able to intercept and analyse all the web pages visited by the user—thus performing keyword extraction based on downloading activity (item (1) of the above list)—but is unable to intercept operations on bookmarks.

3.3 Categorising and Storing Documents

As discussed in Section 3.1, SHARK users' documents are automatically characterised by the lists of e-keywords and i-keywords. The task of agent Categoriser is to classify documents and user profiles accordingly.

Agent Categoriser uses an ontology consisting of a number of categories, say `Computing`, `Law`, `Medicine`, `...`, each having a certain number of descendent categories. Each category is characterised by a list of weighted keywords. For a known category, say `Law`, there is a list of ordered keywords with their own weight, e.g. `0.12 act, 0.11 rule, 0.10 code, 0.09 statute, ...`, where the words that better characterise the category have a bigger weight.

In our experiments, we have built an ontology by choosing a suitable list of categories, and a training set of documents for each category. Using the algorithm described in Section 3.1, we have then extracted the keywords occurring in each document set and used them as the list characterising the associated category.

As in [24], once an ontology has been provided, Categoriser determines which category a document belongs to, by calculating a *similarity* value between the keywords of the known categories and the keywords of the document. As for the keywords characterising categories, each document has a list of ordered keywords whose weight is proportional to the number of occurrences. We have calculated the similarity as the sum of the products between the weights of each keyword found on both lists. This represents a scalar product in a n-dimension space, where n is the number of keywords in each category. The categories whose similarity exceeds a threshold value are chosen as the categories for the document; this association, expressed as $document \rightarrow (Category_1, SimValue_1), \ldots, (Category_m, SimValue_m)$, is stored in the *local document database*, present in the AgentCities Server (cf. Section 3.4).

Categoriser is asked to determine the categories for user documents by agent Cruncher, every time this finds that new documents are made publicly available. By grouping documents into categories it is possible to easily find all the documents that are in some relation. E.g. the user can request a list of the documents that belong to the same categories as his/her document, or can choose to execute queries about some categories only.

Moreover Categoriser periodically analyses the keyword list built by each UserProfiler from the web pages visited by the relevant user. Thus, Categoriser can determine which categories the keywords belong. These categories represent what we call the *user profile*.

3.4 Sharing Knowledge among AgentCities Servers

The information on documents' categories and user profiles gathered in an AgentCities Server must be made available to the other servers of the environment in order to allow document finding. To make this possible, two different techniques could be employed by the infrastructure: (*a*) to route each query made by a user on a AgentCities Server to all the other servers; or (*b*) to broadcast information on documents and users from a AgentCities Server to all the other servers, in order to perform searches locally.

Both these approaches feature some disadvantages. On one end, routing a query implies to contact servers that could not have any document matching the criteria, thus burdening the search activity with useless transactions; on the other end, broadcasting information could imply a large amount of network traffic when the number of documents and users handled by the system increases and/or changes frequently.

Our solution aims at providing a trade-off between the above two techniques. Each Categoriser agent, after performing the activities described in Section 3.3, computes a weighted sum of the ranks of each category for all the documents processed. The result, that is a ranked list of the categories handled by the AgentCities Server hosting that Categoriser, is then stored in a database that is distributed among all the AgentCities Severs using a Distributed Hash Table (DHT) [27]. A similar approach in used for user profiles: the ranked list of categories relevant to all the user profiles managed by a AgentCities Server is stored in a DHT spread over all the servers of the SHARK system. In summary, the following databases are managed:

- The *local document database*. It is in each AgentCities Server and stores the associations among documents and categories for all the documents held by the users belonging to that AgentCities site.
- The *site category database*. It is based on a DHT and stores the associations among each AgentCities Server and the categories of all the documents handled by the server.
- The *local contact database*. It is in each AgentCities Server and stores the categories of profiles for the users of the AgentCities Server.
- The *site contact database*. It is handled as a DHT and stores the association among each AgentCities Server and the categories of all the profiles for its users.

The choice of adopting different techniques to organise and distribute SHARK's databases is due to two reasons. First of all, a local database for domain's documents seems a natural choice to favour collaboration among users belonging to the same domain. Moreover, the use of a local document database allows users of a certain domain to find and access local information also when the network is partitioned. Secondly, when document searching and accessing is performed through different domains (thus requiring the network to be connected), the DHT provides a valid support for data distribution among the various SHARK Servers.

Search is performed by using all these databases, as detailed in Section 3.5. For this to be possible, all AgentCities Servers must share the same ontology for document categorisation; this is not a strong constraint but instead an usual requirement for distributed systems that must share meaning and semantics.

3.5 Searching for a Document

Users can search documents by pointing their web browsers to one of the AgentCities Servers of the SHARK system and then by typing the desired keyword(s) in the web form that will appear. To support such a web-based interface, each AgentCities Server involved in SHARK runs a Tomcat application server [31] equipped with suitable libraries for activating AgentCities agents by means of JSP code [5]. The browser is an interface for users to access the P2P system provided by the infrastructure and agents described hitherto.

Agent Searcher, running on the AgentCities Server, receives the keywords and starts the search process by performing the following tasks:

1. The *local document database* is first queried, to find the documents whose keywords match those requested by the user.
2. The *site category database* is then queried, to find the SHARK servers (AgentCities Servers) that refer documents with matching keywords.
3. The Searchers agents of such sites found are then queried; these agents will return a list of matching documents.
4. Tasks 1–3 above are repeated for *local-* and *site-contact* database, in order to find people sharing interests that are common to those of the querying user.

As reported in the example screenshot in Figure 5, the results of search tasks 1–4 are two ranked lists.

The first list (depicted on the right side of Figure 5) gives all the documents matching the requested keywords and reports, together with the document's name, the document's relevance and the name of user who holds that document. By clicking on the document's name, the Correspondent agent running in the client hosting the requested document

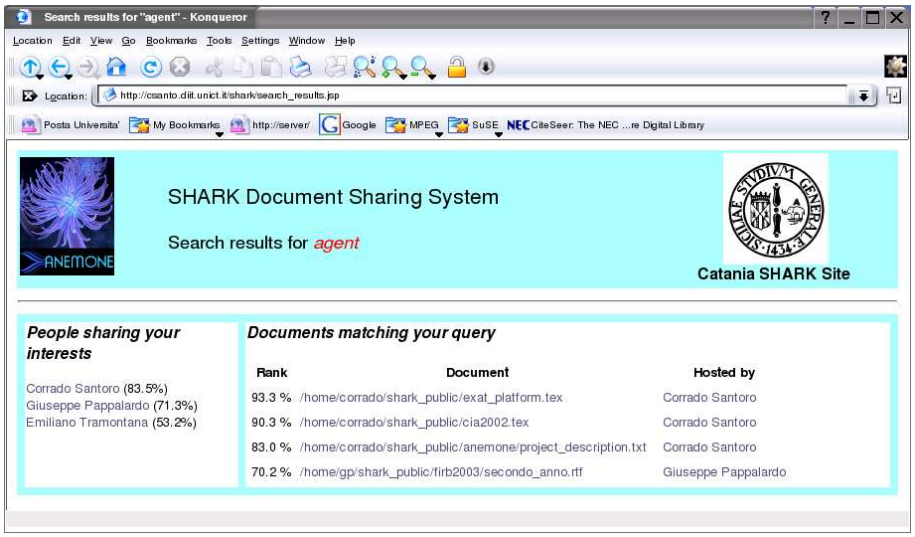

Figure 5. Search Results Example

is contacted, and the document is finally downloaded. Document's relevance is determined by using the frequency in the document (in percentage with respect to all the document's keywords) of the keyword queried—if only one keyword is provided. If more than one keyword is given, the total relevance is computed as the average of each single keyword relevance[4].

The second list (depicted on the left side of Figure 5) reports the name of the users who have, in their user profile, the keyword(s) queried. The percentage near the name is the relevance of the user profile with respect to the keyword(s) queried, and is computed using the same method employed for documents. By clicking on a user name, a possible contact with that person can be established, as it is detailed in the following SubSection.

3.6 Promoting Collaboration

SHARK provides two different mechanisms for promoting collaboration among users: the former is *user-triggered* and the latter is performed *autonomously*.

As introduced previously, the first mechanism—*user-triggered*—is activated by a user query and when one of the names of the list of matching users is clicked. Basically, the action performed by the Searcher to promote collaboration is to automatically send an email to the selected user announcing that "user x is interested in these arguments ....". It is up to the receiver, then, to decide whether to contact the user who performed the query or to give up, if not interested in such a collaboration.

Since SHARK's agent Searcher can be also contacted by unknown users, the type and the form of the message sent depends on the fact that the user performing the query is known or not. In the former case, the email message sent contains not only the keywords typed by the user in the search form, but also his/her interests (extracted from his/her profile), in order to give a better indication of common interests. The UserProfiler agent recognises the contacting user and sends to the Searcher a message containing his/her identity and profile. If the user is unknown to the SHARK system, clicking on the contact will be not recognised by any UserProfiler and, as a consequence, a pop-up window appears to request the user email address before sending the message.

The second mechanism—the *autonomous* one—is handled by the Advertisers agents running in the various AgentCities Servers. Each Advertiser knows the profiles of the users relevant to its AgentCities Server[5] and periodically (once a month) contacts the other Advertisers in order to determine the users with common interests. This is done by performing a matching between the profiles of the various users, checking that the matching degree is above a given threshold. As a consequence, users with common interests are notified with an email message, leaving then the choice on the opportunity to collaborate to the users themselves. This matching task is performed periodically because profiles may change due to users' activities and/or change of interests.

[4]This means that an implicit "and" is considered placed among keywords. Currently, this is the sole search semantics provided by SHARK. More flexible search expressions—i.e. allowing the use of "and" and "or" operators—will be provided in the future releases of SHARK.

[5]The agent has access to the user profile database.

4 Concluding Remarks

The research field of agents and P2P provides many works essentially ranging from supporting information dissemination and discovery [34, 9, 3], to organizational issues related to the use of "mediator agents" among network peers [25]. Other works consider semantic concerns, and deal with formal grammars for expressive query specification [20] or propose P2P approaches for the semantic web [2, 6].

With respect to such approaches, SHARK differs in both architectural and semantic aspects. The presence of AgentCities Servers, that directly derives from the structure of the AgentCities network on which SHARK operates, allows a form of clustering for document information, thus reducing the network activities needed during dissemination and query. This also allows treating a larger amount of information for each document—i.e. the extracted/implicit keywords—rather than the document's title or the filename as in [34, 3]. The latter aspect, i.e. the ability to automatically extract the implicit keywords, does not require semantic annotation for the documents, as in [2, 6], but allows SHARK to operate with the current web environment and with the documents generally handled by any user.

Above all, the main contribution of SHARK is to tie users' web browsing behaviour to P2P document sharing principles, i.e. profiling the user and allowing not only to find documents dealing with certain keywords but also users having such keywords among their interests. This is made possible thanks to the joint use of computational reflection—for browser interfacing and user profiling—with the agent technology—used to autonomously perform user profile matching. Such a combination, in the light of improving (scientific) knowledge dissemination and sharing, can constitute an effective support for document and user finding, and thus for promoting collaboration among people/communities.

References

[1] `http://fipa-os.sourceforge.net/`. FIPA-OS Web Site., 2003.

[2] M. Arumugam, A. Sheth, and B. Arpinar. The Peer-to-Peer Semantic Web: A Distributed Environment for Sharing Semantic Knowledge on the Web. In *Proc. of Intl. Workshop on Real-World PDF and Semantic Web Applications (WWW2002). Honolulu, Hawaii, USA*, 2002.

[3] O. Babaoglu, H. Meling, and A. Montresor. Anthill: a Framework for the Development of Agent-based Peer-to-Peer Systems. In *Proc. of* 22^{th} *Intl. Conference on Distributed Computing Systems (IDCS2002)*, Vienna, Austria, 2002.

[4] F. Bagci, J. Petzold, M. Trumler, and T. Ungerer. Ubiquitous Mobile Agent System in a P2P-Network. In *UbiSys-Workshop at the Fifth Annual Conference on Ubiquitous Computing, Seattle, October 12-15*, 2003.

[5] D. L. Berre and O. Fourdrinoy. Using JADE with Java Server Pages (JSP). In *JADE Documentation available at JADE Web Site* `http://jade.cselt.it/`, 2002.

[6] S. Castano, A. Ferrara, and S. Montanelli. H-MATCH: an Algorithm for Dynamically Matching Ontologies in Peer-based Systems. In *Proc. of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany*, 2003.

[7] S. Chiba. A Metaobject Protocol for C++. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'95)*, pages 285–299, 1995.

[8] S. Chiba. Load-time Structural Reflection in Java. In *Proceedings of the ECOOP 2000*, volume 1850 of *Lecture Notes in Computer Science*, 2000.

[9] P. Dasgupta. Incentive Driven Node Discovery in a P2P Network Using Mobile Intelligent Agents. In *Proc. of the Intl. Conference on Artificial Intelligence (IC-AI2003)*, Las Vegas, Nevada, USA, 2003.

[10] A. Di Stefano, G. Pappalardo, C. Santoro, and E. Tramontana. A Multi-Agent Reflective Architecture for User Assistance and its Application to E-Commerce. In *Cooperative Information Agents (CIA 2002)*. LNAI Springer, Sept. 18-20 2002.

[11] A. Di Stefano, G. Pappalardo, C. Santoro, and E. Tramontana. Extending Applications using Reflective Assistant Agents. In 26^{th} *Annual International Computer Software and Applications Conference (COMPSAC'02)*, Oxford, 2002. IEEE.

[12] A. Di Stefano, G. Pappalardo, C. Santoro, and E. Tramontana. Coordinating Multi-Agent Assistants with an Application by means of Computational Reflection. In *Design of Intelligent Multi-Agent Systems Human-Centredness, Architectures, Learning and Adaptation*, volume 162. Springer, 2004.

[13] A. Di Stefano and C. Santoro. NETCHASER: Agent Support for Personal Mobility. *IEEE Internet Computing*, 4(2), March/April 2000.

[14] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data Knowledge Engineering*, 31(3):227–251, 1999.

[15] J. Ferber. Computational Reflection in Class Based Object Oriented Languages. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89)*, volume 24 of *Sigplan Notices*, pages 317–326, New York, NY, 1989.

[16] Foundation for Intelligent Physical Agents. `http://www.fipa.org`, 2002.

[17] R. L. Graham. Peer-to-peer: Toward a Definition. In *Proc. of IEEE Intl. Conference on Peer-to-Peer Computing (P2P2001)*, Sweden, 2001.

[18] J. Bradshaw et al., editor. *Software Agents*. AAAI Press, Cambrigde, Mass., 1997.

[19] Y. Kitamura, T. Yamada, T. Kokubo, Y. Mawarimichi, T. Yamamotom, and T. Ishida. Interactive Integration of Information Agents on the Web. In *Proceedings of CIA 2001*,

volume 2182 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.

[20] M. Koubarakis, C. Tryfonopoulos, P. Raftopoulou, and T. Koutris. Data Models and Languages for Agent-Based Textual Information Dissemination. In *Cooperative Information Agents (CIA 2002)*. LNAI Springer, Sept. 18-20 2002.

[21] H. Lieberman. Letizia: An Agent That Assists Web Browsing. In *International Joint Conference on Artificial Intelligence*, Montreal, August 1995.

[22] H. Lieberman, P. Maes, and N. Van Dyke. Butterfly: A Conversation-Finding Agent for Internet Relay Chat. In *International Conference on Intelligent User Interfaces*, Los Angeles, January 1999.

[23] P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, volume 22 (12) of *Sigplan Notices*, pages 147–155, Orlando, FA, 1987.

[24] H. Mase. Experiments on Automatic Web Page Categorization for IR system, 1998. Technical Report, Stanford University.

[25] L. Penserini, L. Liu, J. Mylopoulos, M. Panti, and L. Spalazzi. Cooperation strategies for agent-based P2P systems. *Web Intelligence and Agent System*, 1(1):3–21, 2003.

[26] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Service for Internet Applications. In 2^{nd} *International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, 2003. Springer.

[28] E. Tramontana. Managing Evolution Using Cooperative Designs and a Reflective Architecture. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *Lecture Notes in Computer Science*. Springer-Verlag, June 2000.

[29] E. Tramontana. Reflective Architecture for Changing Objects. In *Proceeding of the ECOOP Workshop on Reflection and Metalevel Architectures (RMA'00)*, Nice, France, June 2000.

[30] WWW. `http://aot.ce.unipr.it:8080/anemone/index.jsp`, 2004.

[31] WWW. `http://jakarta.apache.org/tomcat/`, 2004.

[32] WWW. `http://snowball.tartarus.org/`, 2004.

[33] WWW. `http://www.agentcities.net`, 2004.

[34] WWW. `http://www.freeproject.org`, 2004.

[35] WWW. `http://www.gnutella.com`, 2004.

[36] WWW. `http://www.jxta.org`, 2004.

[37] WWW. `http://www.napster.com`, 2004.

[38] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proc. of* 22^{th} *Intl. Conference on Distributed Computing Systems (IDCS2002)*, Vienna, Austria, 2002.